University of Zurich[UZH]

BACHELOR THESIS – Communication Systems Group, Prof. Dr. Burkhard Stiller

Distributed Ledger-based Gossiping for Decentralized Federated Learning

Sandrin Raphael Hunkeler
Zurich, Switzerland
Student ID: 18-253-815

Supervisor: Dr. Alberto Huertas Celdrán, Jan von der Assen, and Chao Feng
Date of Submission: June 25, 2024

University of Zurich
Department of Informatics (IFI)
Binzmühlestrasse 14, CH-8050 Zürich, Switzerland

ifi

Bachelor Thesis
Communication Systems Group (CSG)
Department of Informatics (IFI)
University of Zurich
Binzmühlestrasse 14, CH-8050 Zürich, Switzerland
URL: http://www.csg.uzh.ch/

Declaration of Independence

I hereby declare that I have composed this work independently and without the use of any aids other than those declared (including generative AI such as ChatGPT). I am aware that I take full responsibility for the scientific character of the submitted text myself, even if AI aids were used and declared (after written confirmation by the supervising professor). All passages taken verbatim or in sense from published or unpublished writings are identified as such. The work has not yet been submitted in the same or similar form or in excerpts as part of another examination.

Zürich, June 25, 2024

Signature of student

Abstract

Klassisches maschinelles Lernen (ML) hat mehrere kritische Nachteile. Dazu gehören einen zentralen Ausfallspunkt, wenig Datenprivatsphäre und einen Mehraufwand bei der Sammlung und Speicherung von Daten. Verteiltes Föderales Lernen (DFL) geht diese Probleme an, indem es die Notwendigkeit einer zentralen Datenerfassung überflüssig macht und so den Datenschutz wahrt. Dies wird dadurch erreicht, dass verteilte Knoten ein ML-Modell lokal trainieren, wobei nur die Modellparameter und nicht die Rohdaten periodisch ausgetauscht werden. DFL ist jedoch anfällig für Vergiftungsangriffe, bei denen schadhafte Teilnehmer ihr lokales Modell oder ihre Trainingsdaten verfälschen, um das gemeinsam trainierte Modell zu korrumpieren. Es gibt verschiedene robuste Aggregationsmethoden, bei denen die Modelle nach bestimmten Kriterien analysiert werden, um ihre Qualität vor der Aggregation zu bewerten. Die bestehenden Verfahren in DFL beschränken sich jedoch auf die lokale Sicht der einzelnen Teilnehmer, um böswillige Teilnehmer zu identifizieren. In dieser Arbeit wird ein Ledger-basiertes Reputationssystem vorgestellt, das es den einzelnen Teilnehmern ermöglicht, ihre lokalen Beurteilungen der Vertrauenswürdigkeit der anderen Teilnehmenden zu teilen. Die kombinierten Bewertungen werden in eine Reputation umgewandelt und von einem ergänzenden robusten Aggregationsalgorithmus verwendet, um eine gewichtete Aggregierung der Modelle durchzuführen. Experimente haben die Wirksamkeit der Implementierung bei der Eindämmung von Model Vergiftungsangriffen und Angriffen auf das Reputationssystem selbst gezeigt. Darüber hinaus wurden durch Messungen der Nutzung von Rechen-, Zeit- und Finanzressourcen begrenzende Skalierbarkeitsfaktoren und Kompromisse ermittelt. Die Bewertung zeigt, dass das Reputationssystem die Teilnehmer mit effektiven Informationen versorgt, die Modellvergiftungsangriffe abschwächen und gleichzeitig die Latenzzeit auf einem wettbewerbsfähigen Niveau halten.

Traditional machine learning (ML) has several critical disadvantages, such as a single point of failure, data leakage, and overhead to collect and store data. Distributed Federated learning (DFL) addresses these issues by omitting the need for central data collection, thereby preserving data privacy. This is achieved by having distributed nodes train a ML model while only periodically sharing the model parameters instead of the raw data. However, DFL is vulnerable to poisoning attacks where malicious participants tamper their local model or training data to corrupt the jointly trained model. Various robust aggregation methods are available, which analyze models according to specific criteria to assess their quality before aggregation. Yet existing schemes are limited to the individual DFL participants' local view for identifying malicious participants. This work proposes a ledger-based reputation system enabling the individual participants to share their local opinions. The combined opinions are transformed into a reputation and utilized by a complementary robust aggregation algorithm to perform weighted aggregation. Experiments have shown the implementation's efficacy in mitigating model poisoning attacks and attacks targeting the reputation system itself. Further, measurements regarding the utilization of computational, time, and financial resources identified limiting scalability factors and trade-offs. The evaluation demonstrates that the reputation system provides the federation's participants with effective information mitigating model poisoning attacks while keeping the introduced latency at a competitive level.

Acknowledgments

I would like to express my sincere gratitude to my supervisors, Dr. Alberto Huertas Celdrán and Jan von der Assen. Their consistent support, valuable feedback, and generous time investment have been crucial throughout this thesis. I'm also thankful to Enrique Tomás Martínez Beltrán for his technical expertise that greatly aided integrating my work into FedStellar.

Additionally, I'm grateful to Prof. Dr. Burkhard Stiller for the opportunity to complete my Bachelor's thesis at the Communication Systems Group (CSG) of the University of Zurich. This experience has been personally enriching and enhancing my scientific skills. Thanks to everyone who has been part of this journey.

Contents

Chapter 1

Introduction

Traditional machine learning (ML) has several critical disadvantages, such as single point of failure, data leakage, and overhead to collect and store data [1]. Federated learning (FL) addresses these issues by omitting the need for central data collection, thereby preserving data privacy [2]. This is achieved by having distributed nodes train a ML model while only periodically sharing the model parameters instead of the raw data [3]. However, FL is vulnerable to poisoning attacks where malicious participants tamper their local model or training data to corrupt the jointly trained model [4]. The major objective of this thesis is to propose a ledger-based reputation system for improving the detection and mitigation of poisoning attacks. The following chapter motivates the system and provides an overview of the structure of this work.

1.1 Motivation

Federated learning allows a swarm of mutually trusted and potentially untrusted clients to jointly train a ML model. Its potential has led institutions worldwide to adopt the technology and utilize it for sensitive tasks such as credit risk predictions [4]. The reliance on potentially untrusted participants and its application in sensitive tasks make it a vulnerable target for attacks. Untargeted attacks are designed to compromise the integrity of the model in general, while targeted and backdoor attacks seek to manipulate the model's predictive behavior in specific ways [4]. Verifying the authenticity of training data while preserving data privacy presents a challenge in FL [5]. This increased the likelihood of malicious attackers successfully tampering with their local model while acting undetected as insiders of the federation. Therefore, detecting malicious behavior and preventing the aggregation of poisoned model updates is essential to FL [4].

A literature review conducted in this thesis revealed a focus in current research on robust aggregation using distributed ledger technology. Robust aggregation analyzes models based on certain criteria to estimate their quality of contribution [4]. Additionally, utilizing distributed ledger (DL) technology enhances the security of the FL process by leveraging the technology's security properties.

However, the use of DL technology for implementing a reputation system and using the individual participants reputation for robust aggregation has been identified as a research gap. This motivates the implementation of a robust aggregation algorithm for FL that uses individual participants' reputation for weighted aggregation.

1.2 Description of Work

This thesis proposes and implements a DL-based reputation system for robust aggregation in DFL. Unlike existing robust aggregation algorithms for DFL, this thesis introduces an aggregation algorithm that utilizes the combined local views of all participants to detect and mitigate poisoning attacks. Instead of directly adjusting the aggregation based on locally computed trust metrics, the insights are first reported to a reputation system. The reputation system performs statistical analysis and anomaly detection on the combined local views of the behavior of the individual participants. It then provides aggregation weights for all models the participants intend to aggregate with.

The reputation-based aggregation algorithm and the reputation system are evaluated in terms of resource utilization and efficacy against poisoning attacks. This evaluation provides insights into the trade-off between utilizing computational resources for increasing the DL's security and for training the models. Additionally, the impact of the inherited latency from the underlying blockchain on the reputation-based aggregation algorithm is evaluated. The best performing settings are then compared to the performance of existing aggregation algorithms. To estimate the financial costs of deploying the reputation system onto a public blockchain, measurements of the gas costs are taken and analyzed by current market prices. Furthermore, the system's robustness to attacks aimed at undermining the reputation system is examined.

1.3 Thesis Outline

This thesis is structured as follows. Chapter 2 introduces the fundamental concepts of distributed ledger technology, federated learning, and reputation systems that form the basis for this work's research objectives. Chapter 3 provides an overview and comparison of existing frameworks and approaches utilizing DL technology to enhance security in federated learning. In Chapter 4 the architecture of the reputation system is proposed, including its aggregation and reputation algorithms. Chapter 5 provides detailed insights into the implementation of the proposed architecture. In Chapter 6 experiments are conducted for evaluating the performance and efficacy of the implementation. In Chapter 7 the evaluations are summed up and discussed to conclude this work's contribution to research.

Chapter 2

Background

The following section summarizes the theoretical foundation relevant for this work. The initial subsection introduces federated learning and common poisoning attacks. The second subsection introduces FedStellar, which forms the basis for this work's implementation. The third subsection provides knowledge about the relevant aspects of distributed ledgers and blockchain technology. The final subsection introduces basic functionalities of reputation systems and common threat models.

2.1 Federated Learning

Traditional machine learning has several critical disadvantages, such as central point of failure, data leakage, and overhead to collect and store data [1]. Federated learning reduces the overhead of data collection while maintaining data privacy in traditional machine learning [2]. This is achieved by having distributed nodes train a ML model while only periodically sharing the model parameters instead of the raw data [3]. The shared model parameters of participating nodes are aggregated until convergence is reached or some terminating criterion is met [6].

Federated learning is classified by the topology of how the locally trained model parameters are shared and aggregated [3]. With centralized federated learning (CFL), at each round, the distributed nodes communicate and send their locally trained model to a central node, which aggregates the parameters and distributes it back to the nodes [6]. CFL still suffers from disadvantages such as the lack of scalability and the single point of failure at the aggregating node [7]. With semi-distributed federated learning (SDFL), the role of the central aggregator is taken on by one or more different nodes in each round.

As for distributed federated learning (DFL), the nodes share the model parameters among the other nodes and aggregate the models independently of a central node [3]. Whereas DFL has the advantage of no single point of failure and no bottleneck at the central aggregating entity, it suffers from other disadvantages such as overall increased communication overhead and training optimization [3].

A key component in FL is the method used for aggregating multiple trained models into a single one. Federated Average (FedAvg) as introduced by [8] combines local stochastic gradient descent on each node while model averaging is used for aggregation. Various aggregation algorithms exist, such as Stochastic Gradient Descent (SGD) and Batch Gradient Decent. Recent research has shown that certain aggregation algorithms like SGD are vulnerable to inference attacks targeting to recover private data from an individual nodes' updates [9].

2.1.1 Poisoning Attack

Poisoning attacks aim to compromise the final model's usefulness by targeting its integrity [10]. Corrupted participants can tamper their local model or training data to poison the jointly trained final model [4].

Data poisoning merges intentionally wrong-classified data points with correctly classified ones during the training process [5]. Data poisoning can be categorized into targeted attacks, backdoor attacks, and untargeted attacks [4]. Targeted attacks tamper only a specific range of labels, while backdoor attacks tamper data points having a specific attribute. Untargeted attacks, on the other hand, aim for the whole data distribution.

Model poisoning makes direct changes to the model's parameters instead of tampering the local training data [5]. Model poisoning implies that the attacker has access to the model's parameter as well as its prediction behavior [4]. Model poisoning can be classified into untargeted and targeted attacks [5]. Targeted attacks aim at the prediction behavior for a specific range of labels, while untargeted attacks aim for corrupt the general prediction accuracy.

2.2 FedStellar

FedStellar is an extensive platform for creating and running various FL scenarios with highly configurable properties [11]. The platform's architecture comprises three major components: front end, controller, and core.

2.2.1 Architecture

The front end allows to configure various properties of a scenario, such as the network topology, the architecture, algorithms, and datasets. Key performance indexes (KPIs) about computational resources, communication, and the learning process allow for detailed and efficient monitoring of ongoing scenarios. Furthermore, the collected monitoring data can be exported to be post processed.

The controller is responsible for executing commands from the user interface for the orchestration and setup of a given scenario. It interprets user settings and bootstraps the federation network by assigning algorithms, datasets, and network topologies to the cores.

The cores are responsible for executing the FL-task and communication. They manage the model training by processing data, handling storage, and aggregating. They further continuously monitor the ongoing process and report the KPIs to the front end.

2.2.2 Implementation

Scenarios are deployed in up to 20 docker containers, which are container applications managed and run by the software Docker Engine. Scenarios mainly focus on image classification on common public datasets, such as MNIST and CIFAR-10, while training LeNet5, CNN or MobileNet models. Further, the deployments allow setting up scenarios for DFL, CFL, and SDFL. The computational resources can be individually provisioned for each core. A YAML file is used to specify the individual limits of a core's resource allocation.

2.3 Distributed Ledger

Distributed ledger technology (DLT) can be described as a distributed application that allows for tamper-resistant, transparent, and verifiable transactions in a dishonest environment [12]. DL was first introduced in 2008 by the pseudonym *Satoshi Nakamoto* with the white paper about Bitcoin (BTC) [13]. It aims to solve the underlying *Byzantine generals' problem* (BGP), that describes a situation where multiple honest generals have to reach an agreement by exchanging messages while not being disrupted by dishonest generals [14]. As shown, the problem is solvable if no more than one-third of the generals are dishonest [14].

DLT aims to solve the BGP by establishing trust among participants by relying on a consensus protocol instead of a trusted third party [12], [15]. Consensus protocols combine rules that are used by participating nodes to validate transactions [16]. This ensures that all nodes agree on transactions and maintain a synchronized ledger.

DLT runs on a *peer-to-peer* (P2P) network within a distributed eco-system which forms an append-only database (ledger) which is simultaneously maintained and stored by distributed nodes [12]. This removes any single point of vulnerability and makes the ledger resilient against breakdowns of single nodes [15].

2.3.1 Blockchain

A blockchain is a public ledger where all transactions are stored permanently and cannot be tampered [17]. With BTC, the first distributed ledger using a blockchain was created [13]. The distributed ledger holding all valid transactions is stored in batches named blocks, which have a header containing the root of the blocks Merkle tree, the hash of the prior block and parameters related to the consensus protocol [18]. The linked block hashes gave the blockchain its name and are used to identify the blocks uniquely while ensuring

their integrity [15]. If a transaction would be modified, the hash of the transaction's block as well as the hashes of all succeeding blocks would change [19]. Combined with the highly duplicated and distributed ledger, blockchains are highly resistant to tampering and therefore seen as immutable.

Consensus Protocol

New changes to the ledger are initially validated by the nodes. If the nodes find the transaction to be legitimate, they store it in a preliminary block and share it with other peers [15]. Consensus protocols are used to determine the peer to add the next block to the chain and the criteria by whom the new block is validated by the other peers [17]. Various types of consensus protocols using voting or proofing based schemes exist to reach consensus about the next block to be added to the chain, where Proof-of-Work (PoW) and Proof-of-Stake (PoS) are among the most common ones [18]. Generally, if a mining or validating node has successfully finalized a block, it sends it to other peer nodes, which validate and append the block to their local ledger [18]. This ensures the synchronization and resilience against malicious participants if the majority of miners are honest.

With PoW, the miners compete to solve a puzzle that is solvable by computational power faster than by logic, while with PoS, the so-called validators get selected randomly with higher chances the higher their token balance on the current chain [18]. The Practical Byzantine Fault Tolerance (PBFT) protocol originated from permissioned blockchains. It uses a three-stage process of communication with every node to determine the next primary node, which makes it inefficient but more suitable for small permissioned networks [20]. There exist multiple variations and extensions of PBFT, such as Delegated Byzantine Fault Tolerance (dBFT) is an extended version with improved properties such as the total ordering of events, reduced message overhead, and improved resilience against malicious participants [21]. Proof-of-Authority (PoA) is used in permissioned networks where each trusted node gets time slots allocated for being selected as a leader for the block validation [22]. Each leading node has to be identified as honest and reliable by the majority of participating nodes to be able to validate a block.

Blockchain Type

In public blockchains, anyone can participate in the consensus protocol, while private and consortium blockchains require admission (permissioned) by a trusted third party [18]. Public blockchains are fully decentralized and without any trust for participating peers [23]. Since anyone can participate in the consensus protocol, participants are rewarded with incentives to take part in the resource-intensive consensus protocols [24]. The energy consumption and computational effort to synchronize and secure a public blockchain can become immensely high [23]. Consortium and private blockchains, on the other hand, are run by selected and identified participants [25]. This makes the private blockchains rely more on the accountability and trust of the accepted nodes while having less competitive and resource-intensive consensus protocols [24]. The gained efficiency and the need for a trusted third party handling the admission to the network sacrifices the decentralization of the network [23].

Smart Contract

Smart contracts are software running on a blockchain whose correct execution is ensured by the underlying consensus protocol [15], [26]. Smart contracts are duplicated and stored throughout all participating blockchain nodes [27]. This level of redundancy makes deployed contracts nearly immutable against tampering. Ethereum was a pioneer by first introducing the development and deployment of smart contracts, which can be used to create distributed applications [28]. Each call to a smart contract changing the ledger is recorded as a transaction and stored permanently on the blockchain [17]. Compared to a traditional contract, does the execution of smart contracts not require a trusted third party [12].

Smart contracts are used to build distributed applications (dApps), which host parts of their back-end and database on a blockchain [16]. Ideally, dApps should not rely on any human interaction and have all policies encoded in its smart contracts [19]. Centralized applications use login credentials for authorization at a server, whereas dApps use wallet addresses and private keys of blockchains for authentication. While dApps provide more transparency and improved identity verification, they also introduce other challenges, such as execution efficiency and irreversibly of exploited vulnerabilities [27]. Moreover, are DApps limited by the latency and the transaction throughput of the underlying blockchain [19]. If the underlying blockchain network is currently overloaded, dApp users might have to wait up to minutes to get a confirmation on their transaction.

Popular Implementations

Multiple open-source implementations of blockchains allow for the deployment of smart contracts to create decentralized applications (DApp). The following frameworks were used by analyzed frameworks in the *related work* chapter: Hyperledger Fabric, Exonum, and Ethereum.

Hyperledger Fabric (HLF) is an open-source permissioned blockchain framework maintained by the Linux foundation [25]. Fabric is extensible and allows for modular consensus protocols to adapt the blockchain to specific scenarios and needs. Fabric is not limited to domain-specific languages but allows distributed applications to be written in general-purpose languages. A channel of the blockchain is a private subnet on the blockchain which has its own peers, ledger and ordering service for the consensus protocol. This allows a group of organizations with a common goal to still have separate subnetworks for exchanging information without trusting the other organizations. In contrast to Ethereum, which runs smart contracts in virtual machines, does Fabric use docker for encapsulation [17]

Exonum is an open-source permissioned blockchain framework maintained by the company Bitfury [29]. Exonum uses a PBFT-based consensus algorithm, which includes improvements such as increased resilience in unstable networks. The framework is based on the Rust programming language, which comes with a high degree of memory safety [30]. It was built for flexibility and performance. Smart contracts come with REST endpoints, which can be used to create transactions. The smart contracts can be written in Java or Rust.

Ethereum is a public open-source blockchain maintained by a global developer community [23]. Ethereum allows to deploy turing-complete smart contracts and decentralized applications on the network. Nodes running the Ethereum network maintain an Ethereum Virtual Machine (EVM), which serially executes transactions and smart contracts. Geth and Parity are both independent open-source implementations of Ethereum nodes. Both can be used for participating in the public blockchain or to run a private Ethereum blockchain.

2.4 Reputation Systems in a P2P - Network

Reputation systems allow entities in distributed decentralized environments to rate each other after a transaction and aggregate a public reputation score [26]. In P2P systems, the public reputation is formed by the peers' opinion about the other peers [31]. Reputation systems for P2P networks have three general components [32]:

1. **Information Gathering:** Peers rate each other and submit their opinions to the reputation system
2. **Scoring and Ranking:** The reputation system aggregates the peers' opinions and provides a public reputation metric
3. **Response:** The reputation system punishes misbehaving peers directly, or the honest peers are restrained from interacting with low-ranked peers

The aggregated reputation metric can be in binary, discrete, or continuous form while being scaled linearly or non-linearly [31]. Common mathematical models range from simple summation, averaging to dedicated statistical methods [26]. Different scaling approaches are further used to increase the resilience to attacks while simultaneously allowing to recovery from previous misbehavior [31].

2.4.1 Vulnerabilities and Attacks

Centralized reputation systems are vulnerable to denial-of-service (DoS) attacks, while hosting the system distributed among the peers is vulnerable to manipulating scores or switching identity [26]. Using smart contracts on a blockchain resolves those vulnerabilities while still being vulnerable to other attacks such as discrimination without evidence [26]. Therefore, smart contracts should, have logic implemented to detect and deal with reputation fraud [26]. There exist multiple threat scenarios targeting P2P reputation systems as mentioned by [31], [33]:

- **Individual Malicious Peers:** Uncoordinated malicious peers providing bad service.
- **Malicious Collectives:** Coordinated malicious peers providing bad service while maximizing each other's trust values.

- **Malicious Collectives with Camouflage:** Coordinated malicious peers providing bad service with probability p while maximizing each other's trust values.
- **Malicious Spies:** Coordinated malicious peers providing bad service and maximize each other's trust values. Joined by malicious peers providing good service while as well maximizing the malicious peers reputation.
- **Driving Down the Reputation of Honest Peers:** Coordinated malicious peers providing bad service while down-rating honest peers while up-rating malicious peers reputation.

Attacks on P2P reputation systems may have several goals, namely, increasing malicious peers' reputation while decreasing honest peers' reputation or destabilize the reputation system as a whole so it becomes unusable [33]. Depending on the attack, in-depth knowledge about the underlying reputation system and its current state is necessary to disrupt the system effectively [33]

Chapter 3

Related Work

This chapter discusses previous research conducted on the utilization of distributed ledger technology for FL. Initially, dimensions underpinning the literature review are introduced, followed by an analytical comparison of the current state of research across these dimensions.

3.1 Usage of Ledger

The potential of using a distributed ledger for FL has been explored for various architectural and logical components. Mainly, five use cases of distributed ledger technology were found to be in focus of the current research. Multiple research projects made use of DL technology for multiple purposes simultaneously, which makes the following application categories not necessarily disjoint.

3.1.1 Aggregation by Smart Contract

The reliable and consistent execution of chain code is guaranteed by the underlying consensus protocol. By transferring centralized aggregation to the blockchain network, several advantages emerge. These include mitigating the risk of a central point of failure and reducing the potential for data leakage associated with centralized trust [20]. Once deployed on the blockchain, the dApp cannot be modified and provides trustworthy and fair aggregation to the participants [34].

Quality metrics of individual contributions can further be stored on the blockchain and utilized in the aggregation process for increasing the model quality. [35] makes use of an aggregating contract *gateway* which itself calls a *defender* contract to compute quality metrics of a model. The quality metrics are stored on the blockchain along the model and are used to detect model poisoning. The decoupling of the two contracts aims to further leverage the semi-honest relationship between smart contracts. [36] has split the process steps and its responsibilities into four smart contracts without further discussing the implications of this design decision.

3.1.2 Incentive System

Participants of a federation face various risks, such as having their private data leaked from the shared models, from which they should be compensated if the participation is voluntary [10]. Reward systems can incentivize users to join the federation while providing compensation proportional to the contribution [37]. On the other hand can incentives also be used to encourage honest training behavior or to punish attempted model poisoning [35].

[38] used a trusted task publisher to verify the worker node's contributions and to reward them accordingly by transferring a token reward to the worker's wallets. The tokens enable the owner of the worker node to spend the reward on services from the task publisher, such as maintenance of household appliances from which the training data originated. However, monetized reward systems introduce new opportunities for malicious behavior and increase the overall attack surface of a system [37].

3.1.3 Reputation System

Reputation metrics about the training behavior and the quality of the contribution of worker nodes, enable the detection of malicious participants. Initially, worker nodes start with a neutral reputation score and get rated for each contribution whereas the reputation is reset after a fulfilled task [36]. In contrast, [1] reuses the reputation system over multiple unrelated task publisher.

While the majority of analyzed approaches use performance metrics about the model, does [1] take dimensions such as communication quality and reliability into account. Moreover, published metrics of participants about the federation are used for anomaly detection and time series analysis. [36] takes a node's history of reputation into account, to select the top-k performing models on a label basis to compute the aggregation.

[1] and [38] use the reputation history to filter out malicious nodes completely, and to select only the top-k trusted worker nodes for a task. Whereas more recent opinions are weighted higher to allow nodes to recover [1]. Experiments have shown that filtering worker nodes increases the model quality but suffers from false positives [1].

Other frameworks such as [39] proposed to include the amount of training data used in the current round into the trust metric. While the former framework trusts the workers to submit valid metadata, does [1] make use Proof-of-Elapsed-Time for validating training time.

Computing a global reputation value from various local opinion does not inherently rely on distributed ledger technology. However, implementing the reputation logic on a blockchain can further leverage the technology's advantages. Reputation metrics exchanged by peers in a P2P network, are vulnerable to tampering and coordinated reputation attacks. Blockchains provides the worker nodes with a platform for publishing and retrieving reputation metrics with high integrity [1].

Although the following frameworks do not make use of blockchain, they introduce algorithms suitable for being deployed on a smart contract. The Sniper protocol in [40] uses graph topology for measuring reputation. Only the nodes in the largest clique are considered, where an edge between two working nodes exists if the Euclidean distance between two model updates is below a predefined threshold. [7] does not take the model or the topology into account but relies solely on the local view of the participants. The reputation about a target node is computed by the local opinions, weighted by the requester's opinion about the latter. The mechanism aims to reduces the weight of an untrusted node in the computation of the global reputation and to increases the resilience against attacks on the reputation system.

3.1.4 Analytic System

Chain code provides utilities to analyze the ongoing FL process and to validate the model aggregation process. Information stored on the ledger is fully traceable from the initialization up to the finished process, which allows detailed analytics of the aggregation and contributions [39]. Depending on the framework, different artifacts of the FL process are stored and processed by the distributed ledger.

Instead of publishing the model updates on the ledger or having chain code aggregating a new global model, [41] proposes to utilize the ledger for an anomaly detection system. Rolling hashes are used to generate a fuzzy state model of the individual contributions. The state models are stored on the ledger and used to compute the Hamming distance between a contribution and the latest global model. Variations and anomalies are then used for protection against poisoning attacks. In contrast, [38] proposes to use external entities to process the quality of the individual contributions, utilizing the chain code for storage purposes only.

3.1.5 Consensus System

[38], [20] and [37] proposed to integrate the aggregation process into a well-established or newly proposed consensus protocol. They either used an election-based protocol or a computational power-based protocol to chose the next aggregating node.

Voting based algorithms are mainly used for permissioned networks with few validating nodes, where collecting the votes of all participants is practically feasible. Such as in the DS2PM framework [20], where a randomly selected mining node starts the voting process for a new global model update. Two-thirds of all validating nodes are required to agree on the proposal. Whereas, the individual validation nodes vote for an update by locally verifying the proposal on a private test set. Experiments with DS2PM have shown that the training time increases with the number of validating nodes. Although a newly introduced and efficient consensus protocol is used, reaching consensus still generates high computational overhead. Similarly, does [38] rely on the Algorand consensus protocol, which uses verifiable random functions (VRF) to elect the next aggregating node.

[37] proposed a lightweight version of the PoW protocol, which makes the training nodes compete for the aggregation by using their hash-power. The node first solving the cryptographic puzzle is allowed to aggregate the new global model and write it to the ledger. Experiments have shown that the convergence latency is convex with an increased block generation rate, and the hash rate has been found to grow exponentially with an increased block generation rate. The robustness of the PoW protocol in mitigating poisoning attacks remains uncertain.

3.2 Security

The security of a proposed framework refers to the primary techniques by which a framework is protected against poisoning attacks and privacy breaches. It further includes methods to enhance the availability and integrity of a system.

3.2.1 Blockchain

Different aspects of using DL technology have been shown to increase resilience against poisoning attacks in FL. On the other hand, different challenges are inherited, such as scalability, transaction throughput, and high demand for computational resources [42]. Moreover, although nodes interact correctly with the DL and share their model updates with the federation, it is not ensured that the update is useful and accurate. It could also be an attack that aims to compromise the final model [10].

Other properties of a framework have also been shown to increase the vulnerability, such as allowing nodes to freely join and leave the federation [37]. Frequently changing identity allows to maintain a neutral reputation while avoiding the consequences for malicious behavior. This reduces the needed effort for a malicious node to launch a poisoning attack.

3.2.2 Consensus Protocol

Experiments have shown that by relying mainly on consensus protocol related to PoW, the success of an adversary attack depends on the hash-power of the attacker [37]. Furthermore, it was shown that the block generation rate can be used to increase the resilience against poisoning, but at a cost of exponentially increased computational load. Voting-based consensus protocols, which are less reliant on computational power, have also been shown to create high computational load with increased numbers of nodes due to the surge in requests [20]. Findings of [43] indicate that there exists an optimal ratio between mining validator nodes and worker nodes to achieve optimal utilization of computational resources. Specifically, an higher number of validators enhance the system's resilience against poisoning attacks, while concurrently, an increased count of worker nodes contribute to reduce training time. Trough their application of a voting-based consensus algorithm, the optimal configuration was determined to be seven workers paired with three validators.

3.2.3 Model- and Node Filtering

Strict filtering of maliciously behaving nodes has shown to increase the model quality [1]. However, the resulting loss of valuable data by false positives remains a challenge [1]. [35] proposed to filter out the lowest performing nodes in each round rather than permanently. The impact of permanent and strict filtering in scenarios with few nodes remains in question.

3.2.4 Trusted Aggregation

Frameworks using DL technology for aggregation have shown reasonable efficacy in mitigating poisoning attacks [35]. Further, using DL technology removes the central point of failure and reduces the risk of data leakage caused by centralized trust in CFL [20]. Once deployed, the aggregation algorithm cannot be modified and ensures a fair and trusted aggregation process [34]. Moreover, the history of all contributions and aggregations is traceable, which ensures the integrity of the process [20].

3.2.5 Privacy

Partitioning models have been used to increase the privacy of worker nodes [38]. Shared model parameters are still vulnerable to attacks on sensitive data [10]. [34] proposed a novel aggregation algorithm for partitioned models so none of the blockchain peers has access to a complete model, which increases protection from malicious or curios attackers. Other works proposed to add noise to each local model to obfuscate the sensitive data [10], such as in [20] and [35].

3.3 Architecture

The analyzed frameworks introduce various architectures to leverage the security and privacy advantages of DL technology in FL. The following section describes the core components that form the technology independent basis for the analyzed frameworks.

3.3.1 Blockchain Type

While various architectural approaches exist for leveraging DL technology, all the examined platforms utilized blockchains as their underlying architectural basis. [37] stands out as the sole platform explicitly specifying a public blockchain; all remaining platforms made either use of a consortium blockchain, or a private blockchain. The latter provides researchers with the advantage of fully configuration and observability of the network. However, [37] did not assess the monetary costs for maintaining the infrastructure, which are a major disadvantage of utilizing an externally hosted infrastructure. [41] was the only framework proposing to utilize a private blockchain.

3.3.2 Coupling

Both, the blockchain network and the FL network, are hosted by independent nodes. The level of coupling determines whether nodes simultaneously participate in both networks or belong exclusively to one. The networks can be overlapping, partly overlapping or disjoint. Having the working nodes simultaneously host the distributed ledger increases the efficiency and system performance of FL [10]. In contrast, having a separate network for training and hosting the ledger reduces the development complexity. The majority of reviewed platforms proposed a decoupled network, whereas [37] and [42] proposed a coupled network.

3.3.3 Channel

Platforms utilizing configurable consortium blockchains often make use of channels. Channels are subnetworks, partitioned from the underlying blockchain network. Each channel is independently permissioned and consists of a distinct distributed ledger and consensus configuration. This enables semi-trusted consorts to jointly use a blockchain network while maintaining privacy. FLchain [39] uses a multi-channel approach where nodes can register themselves for channels to contribute to an existing federation. Whereas, [34] uses different channels for training and inference.

3.3.4 Trusted-Third-Party

Trusted-Third-Parties (TTP) are external entities trusted by the blockchain and the worker nodes. TTPs are part of multiple reviewed frameworks and are ether independent of the architecture or overlapping with the initial task publisher. Both, [1] and [38], depend on TTPs to compute trust values of contributions or to post analyze contributions to reward incentives.

DL technologies were initially created to remove the need of a TTP to reach consensus. Introducing TTPs again as a major component of a FL network seems therefore questionable. TTPs are still vastly used in blockchain networks, such as for managing the admission to permissioned networks [35], but rarely for providing core features to the framework.

3.4 Implementation

The following section describes the technologies used to implement the reviewed frameworks' core features. The underlying implementations were rarely shared in detail, but most often limited to a description on a architectural level.

[36] and [41] have used simulations for their experiments while [37] and [38] analyzed their framework only from a theoretical perspective. The decision is argued with having persistent behavior while performing experiments.

While the majority of frameworks stored the reputation metrics and model updates directly on the blockchain, does the theoretical setup of [38] solely store the pointers to the location of the files. Pointers to files on the InterPlanetary File System (IPFS) maintain the integrity of data, although the files are not stored directly on the blockchain. [42] used Swarm, a distributed file system similar to IPFS, to reduce the requirements on the computational power and storage.

Research platforms introducing novel consensus protocols or making heavily use of channels mainly used Hyperledger Farbic. HLF's extensibility and use of replaceable modules make it a robust basis for integrating FL into the existing blockchain framework. [34] and [39] used HLF's channels as a core feature for security and privacy in their frameworks. Both rely on a basic blockchain network, whereas new channels are created upon task creation. [39] proposes to store the initial model in the genesis block of a new channel. This allows to verify and analyze the iterative progress during training. Worker nodes are further allowed to dynamically join a federation by requesting admission to a task.

[42] introduces a multilevel blockchain for ledger based FL. Coupled worker nodes host a lightweight microchain using BFT protocol for reaching consensus. Model updates are shared with another independent blockchain for aggregation. The aggregated models are accessed by a web application for inference services. The aggregating blockchain is implemented using Ethereum with PoW as underlying consensus protocol. The microchains are implemented using Flask, a Python webserver framework. The microchains are built for privacy and efficiency, while the Ethereum blockchain provides security and scalability.

[43] used the Exonum blockchain framework because of its template of the PBFT consensus protocol and its Rust-based implementation, both of which were already in use in the project's research. To interact with the blockchain, a JavaScript-based user interface was implemented alongside a Python-based ML process.

3.5 Discussion

Table 3.2 summarizes the literature review on previous work about implementing FL using distributed ledger technology.

Use of DL

The column labeled *Use of DL* in Table 3.2 describes the utilization of the DL for federated learning. The reviewed frameworks implemented various utilities on the distributed ledger to make use of the technology's trusted, secure, and tamper-proof properties. Moving the aggregation of CFL to a smart contract (A) or integrating the process into the consensus system (C), tightly couples the architectures of FL and blockchain. Changes

to the implementation of the blockchain or FL have direct implications for each other. While more loosely coupled approaches such as incentive systems (I), reputation systems (R), an analytic systems (A) are less dependent on the FL process.

Security

The column labeled *Security* in Table 3.2 states a framework's primary technological or logical approach to reduce its attack surface. Proposed frameworks mainly use four approaches to mitigate poisoning attacks or to reduce the possibility of successful attacks in general. Those are consensus protocols, filtering, anomaly detection and weighted aggregation.

Integrating the aggregation into consensus algorithms depends on the nodes' ability to determine the validity of a transaction. However, this approach remains dependent on computing a trust metric for evaluating the quality of contribution. Furthermore, PoW based protocols' ability to mitigate attacks remains questionable. Especially, for small FL networks and low power devices.

Filtering out models or participants has shown to increase the model quality and training performance. The underlying assumption of an uniform data distribution and the filtering out of false positives remains a weakness of this approach. Moreover, strict filtering and weighted aggregation are both heavily reliant on assessing the quality and performance of the individual models. However, both techniques are extendable by using reputation metrics to increase the robustness and fairness of the aggregation algorithms.

Architecture

The column labeled *Architecture* in Table 3.2 states a framework's level of coupling between the worker- and blockchain nodes. Other aspects are of less importance for this work and therefore omitted. The level of overlapping between the blockchain network and the FL network mainly depends on the focus and the requirements of a framework. Coupled nodes (C) are more streamlined and resource saving but challenging to be extended. Decoupled nodes (D) are easier to be modified and extended but more wasteful with computational resources. Semi-coupled (SC) architectures combine coupled and uncoupled approaches and their properties.

Consequently, coupled nodes appear to be more suitable for productive environments where persistence and resources are in focus. Conversely, in development and research settings where experimentation and adaptability are in focus, the decoupled nodes emerges to be a more fitting choice.

Implementation

The column labeled *Implementation* in Table 3.2 states a framework's underlying technological basis. Mainly two popular open-source blockchain frameworks were chosen for

implementation, Ethereum and HLF. Less known frameworks, such as Exonum, were rarely observed during the literature research. In contrast, simulations and theoretical setups were frequently used.

HLF offers full configurability, yet it necessitates a comprehensive understanding of encryption methods and blockchain protocols. This positions HLF a powerful tool for research frameworks focusing on consensus protocols within FL. Projects focusing on chaincode and dApps are more reliant on a platform that is both stable and adaptable. While open-source implementations of Ethereum may lack extensive extensibility and modularity, they offer greater ease of use regarding deployment and configuration.

Table 3.1: Abbreviations for Compact Table Visualization.

Term	**Abbreviation**
Decoupled Nodes	D
Coupled Nodes	C
Semi-coupled Nodes	SC
General Blockchain	BC
Architecture	Arch.
Aggregation by Smart Contract	Aggr.
Aggregation by Consensus Protocol	Consensus
Incentive System	Inc.
Reputation System	Rep.

Table 3.2: Literatur Review of Distributed Ledger Technology in FL.

Work	**Use of DL**	**Security**	**Arch.**	**Implementation**
[20] 2023	Consensus	Con-dBFT	D / CFL	HLF
[37] 2021	Consensus	PoW	C / CFL	Public Ledger
[1] 2020	Reputation	Node Filtering	D / CFL	HLF
[38] 2019	Reputation	Node Filtering	D / CFL	BC, IPFS
[34] 2024	Aggregation	Partitioned Model	D / CFL	HLF
[41] 2021	Analytics	Anomaly Detection	D / CFL	Private BC
[35] 2023	Inc. Rep. Aggr.	Model Filtering	D / CFL	HLF
[43] 2023	Consensus	dBFT	D / CFL	Exonum
[39] 2019	Aggregation	PoW or pBFT	D / CFL	HLF
[36] 2023	Aggr. Rep.	Node & Model filtering	D / CFL	Simulation
[42] 2022	Aggregation	Multi-Layer BC	SC / (S)DFL	Flask and ETH
This 2024	Reputation	Weighted Aggregation	D / DFL	Private ETH

Summary

In summary, previous work shows a research trend towards decentralizing the central aggregating entity in CFL by utilizing DL technology. Various trust- and reputation metrics have been employed to filter out or reduce the weights of low performing models. Nevertheless, computing and storing reputation values was not observed in DFL utilizing DL

technology. Moreover, frameworks proposing a reputation system predominantly considered poisoning attacks, rather than attacks on the reputation system itself. This presents a research gap in developing a secure ledger-based reputation system for reputation based weighted aggregation in DFL.

Given the research gaps identified, this work addresses the following areas:

- Development of a ledger-based reputation system for DFL
- Implementation of reputation based weighted aggregation for DFL
- Establishment of a robust reputation system resistant to adversarial attacks targeting the system's integrity

Chapter 4

Architecture

In the subsequent chapter, the architecture is outlined for creating a reliable blockchain infrastructure for FedStellar, which incorporates a reputation system for DFL. FedStellar and its foundational design choices are thoroughly considered to leverage and preserve its maintainability and extensibility.

The first section presents the underlying reasoning behind the high-level architectural choices. The second section gives an overview of FedStellar's architecture and the proposed changes. The final section goes into details about the individual components of the updated architecture and their roles during the bootstrap and training process.

4.1 Design Decisions

FedStellar's core feature is the fully configurable creation of FL scenarios. Following a loosely coupled and modular approach preserves those unique properties, while tightly coupling architectural components would reduce the platform's flexibility. Therefore, the blockchain network follows a fully decoupled architecture, separating the entities and responsibilities of blockchain and FL nodes.

Moreover, FedStellar stands for process transparency and self-containability. Using an external and public infrastructure for deployments would reduce both properties. Locally hosted services allow rapid prototyping and are cost-efficient, even though the reliability and security of external services are less likely to be matched. Therefore, this work proposes a private permissioned blockchain network instead of a public blockchain to account for FedStellar's focus on experiments and self-containment.

In conclusion, this work proposes an decoupled, private, and permissioned blockchain network to extend the current capabilities of FedStellar with a robust and flexible blockchain infrastructure.

4.2 Architecture Overview

Figure 4.1 visualizes the proposed architecture. Existing architectural components are framed in black, while newly introduced components and extensions are framed in green. Requests to APIs and bootstrapping processes are visualized with arrows originating from the initiator. The numbers in brackets indicate the order in which the requests are executed during the bootstrap and training process.

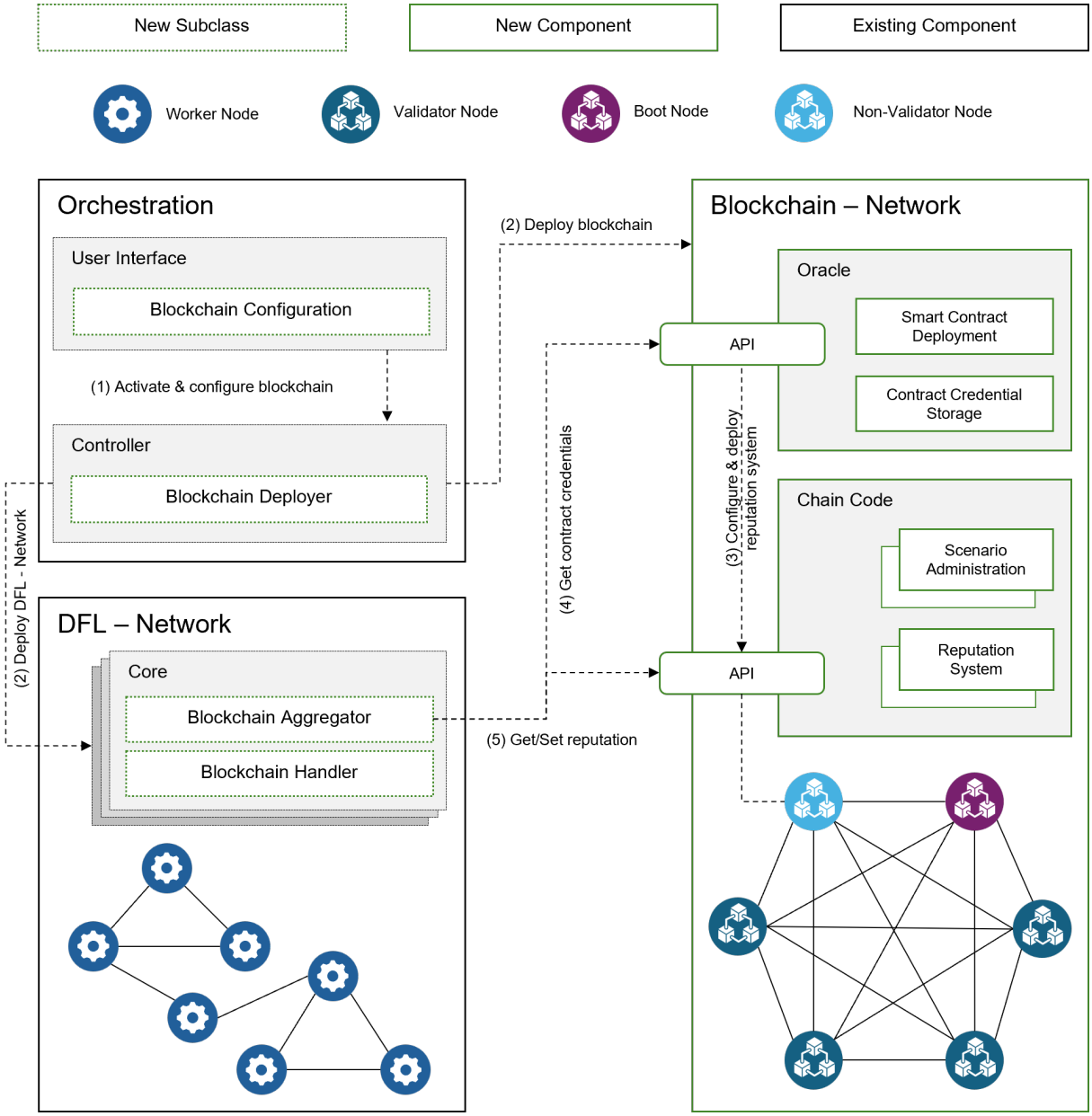

Figure 4.1: Architecture Overview

4.3 Components

FedStellar consists of an *User Interface* and a *Controller*, allowing users to configure and deploy FL scenarios. The *User interface* (UI) registers the user's preferences and sends a deployment request to the *Controller* with the users' defined settings. The *Controller* creates the necessary deployment resources and boots the worker nodes. The worker nodes run on a configured *Core* component, which holds all features for participating in the FL scenario. The *Cores* then start training on their local datasets and gossiping the model updates, while the *Controller* observes the ongoing FL process.

4.3.1 Orchestration

The *UI* is extended by a dashboard element that enables users to activate and configure the blockchain infrastructure and the distributed reputation system **(1)**. The selected preferences are sent along with the other settings to the *Controller*. On receipt of an FL scenario specification **(2)**, the *Controller* generates the deployments for the FL nodes and the blockchain infrastructure according to the scenario's settings. Therefore, the *Controller* is extended by functionalities to configure, deploy, and stop the *Blockchain Network*.

4.3.2 Blockchain Infrastructure

The blockchain infrastructure consists of the *Blockchain Network*, *Chain Code*, and an *Oracle*. The *Blockchain Network* is composed of three types of nodes: *Validator Nodes*, *Boot Nodes*, and *Non-Validator Nodes*. *Validator Nodes* nodes are shielded from external services and execute the consensus protocol. Their responsibility is to validate transactions and maintain the distributed ledger.

Boot Nodes are passive nodes in the network, providing a platform for the other nodes to find and interconnect with other peers. Their address in the network is static and is preset in the deployment configuration of all other blockchain nodes joining the network.

Non-Validator Nodes synchronize the DL with the *Validator Nodes* but do not participate in the consensus protocol. They provide various application programming interfaces (APIs) to external services for interacting with the ledger. Received transactions changing the state of the DL are locally verified and forwarded to the *Validator Nodes*. The *Validator Nodes* synchronize the pending transactions and validate them by the consensus protocol. Upon successful validation, the transactions get appended to the blockchain and synchronized by the *Non-Validator Nodes*, which respond to the initial requester with a transaction confirmation. Transactions not changing the state of the DL are locally processed and returned to the external requesting service without interacting with the *Validator Nodes*.

Simultaneously with the blockchain nodes, the *Oracle* boots up. The *Oracle* is responsible for awaiting the successful bootstrap of the *Blockchain Network* and deploying the *Chain*

Code hosting the *Reputation System*. Once the chain code is successfully compiled and deployed on the *Blockchain Network* **(3)**, the *Orcale* proceeds to be responsible for providing the FL nodes with the *Chain Code's* address on the blockchain **(4)** and funds for enabling the FL nodes to create transactions. Once the FL nodes have received funds and the *Chain Code's* address, the FL nodes are enabled to interact directly with the *Reputation System* and are no longer reliant on the *Oracle*. After having its main purpose fulfilled, the *Oracle* persists as a part of the network and provides analytical functionalities such as recording the gas usage and the *Blockchain Network's* health.

4.3.3 DFL-Network

Worker nodes participating in a FedStellar DFL scenario are deployed by the *Controller* on a *Core* component. The *Core* component is configured by the *Controller* according to the scenario and provides functionalities for the main DFL process, such as gossipping, aggregation, and training.

The *Core* component is extended by a *Blockchain Handler* and an *Blockchain Aggregator*. The *Blockchain Handler* is responsible for setting up the connection to the *Reputation System* on the *Blockchain Network* over the *Oracle* and for pushing opinion values and requesting reputation values for other *Cores* **(5)**. The *Blockchain Aggregator* is utilized for aggregating the received model updates by the other *Cores* by the relative weight of their reputation from the *Reputation System*.

4.3.4 Reputation System

The *Reputation System* provides the *Cores* with a decentralized database for sharing their local view about the behavior of neighboring *Cores*. The combined views of all neighboring *Cores* are used to aggregate a global reputation value for each participating FL *Core*. The reputation values are utilized by the *Blockchain Aggregator* to aggregate models according to the relative reputations.

Aggregation Algorithm

Algorithm 1 shows a detailed view on the proposed logic of the newly introduced *Blockchain Aggregator*. During training, *Cores* receive model updates from their neighboring *Cores* and buffer them locally for aggregation.

Algorithm 1 shows the pseudo code of the `run_aggregation` method. Once the *Cores* start the aggregation process, each received model is evaluated by similarity to the *Cores'* own local models. The computed similarity values are then transformed to increase the resolution on the higher ranges while omitting differences in lower ranges. The transformed similarity values are pushed to the distributed *Reputation System* as opinion values for sharing their local view about the quality of the neighboring *Cores'* contributions.

The *Reputation System* aggregates all the received opinion metrics and computes a global reputation value for every registered worker node. After sharing their local view with the *Reputation System*, the *Cores* request the global reputation of each model's sender it plans to aggregate with. The relative global reputation values are then used for weighting the aggregation of all locally stored model updates.

Algorithm 1: run_aggregation(list<model> models)

```
local_model ← models[self];
metrics ← {};
foreach model ∈ models do
    if model ≠ local_model then
        metrics[model.sender] ← cosine_similarity(local_model, model);
    end
end
local_opinion ← {};
foreach (sender, similarity) ∈ metrics do
    local_opinion[sender] ← transform(similarity);
end
blockchain_handler.push_opinion(local_opinion);
senders ← {};
foreach model ∈ models do
    senders ← senders ∪ {model.sender};
end
reputations ← blockchain_handler.get_reputations(senders);
sum_reputations ← 0;
foreach reputation ∈ reputations do
    sum_reputations ← sum_reputations + reputation;
end
normalized_reputations ← {};
foreach reputation ∈ reputations do
    normalized_reputations[reputation.name] ← reputation / sum_reputations;
end
final_model ← zero_copy(local_model);
foreach layer ∈ final_model do
    foreach model ∈ models do
        final_model[layer] ← final_model[layer] + model[layer] *
          normalized_reputations[model.sender];
    end
end
return final_model;
```

Reputation Algorithm

The *Reputation System* is deployed as *Chain Code* on the *Blockchain Network* and runs distributedly on all *Validator Nodes*. The public methods of the *Reputation System* can be addressed trough the API of the *Non-Validator Node* and are called by the *Blockchain Handler* during the bootstrap and aggregation process.

Algorithm 2 shows the pseudo code of the `get_reputations` method. The method for requesting the aggregated reputation values takes as parameter a list of senders from which the models originated. In an initial step, names unknown to the reputation system are filtered out. In the next step, for every name passed to the algorithm, the average pushed opinion from all neighboring *Core* is computed.

Nodes are confirmed neighbors if both received a model from the other and pushed their local view about the quality of the received contribution to the *Reputation System*. This prevents *Cores* from distributing their models without revealing their contribution or opinion values to the algorithms of the *Reputation System*. *Cores* not reveling themselves to the *Reputation System* are filtered out and excluded from the aggregation.

The average pushed opinion about a *Core* reflects its initial reputation. In a next step the median and standard deviation of all reputation values are computed. High deviations in the opinions reflecting the model similarities indicates an active poisoning attack in the network. *Cores* which have an average reputation highly deviating from the median are suspects to poisoning and have their reputation reduced. Each suspect's reputation is divided by a constant term times the number of standard deviations their reputation differs from the median. In the final stage of the algorithm, the refined reputation values are returned to the aggregating *Core*.

Algorithm 2: get_reputations(list<node> names)

```
Result: Reputations of nodes
filter_out_unknown_nodes(names);
reputations ← list<reputation>;
foreach target ∈ names do
    sum_reputation ← 0;
    n_reputation ← 0;
    foreach node ∈ registered_nodes do
        if confirmed_neighbors(node, target) then
            sum_reputation ← sum_reputation + avg_opinion(of=node,
              about=target);
            n_reputation ← n_reputation + 1;
        end
    end
    reputations.push(node=target, value=sum_reputation / n_reputation);
end
median ← median(reputations);
stddev ← stddev(reputations);
/* High deviation is a symptom of active poisoning in the network */
if stddev > constant_a then
    foreach reputation ∈ reputations do
        n_stddev ← abs(median - reputation.value) / stddev;
        if n_stddev > const_b then
            /* Reduce a node's reputation                          */
            reputation.value ← reputation.value / (n_stddev * constant_c);
        end
    end
end
return reputations;
```

Chapter 5

Implementation

In the following chapter, the implementation of the blockchain infrastructure and the reputation system are outlined. The first section provides an overview of the implementation. The following sections provide a walkthrough of the build time, scenario setup, and the running DFL process.

5.1 Overview

Figure 5.1 shows the implementation as an extended version of the architectural overview in Figure 4.1. Figure 5.1 consists of various Unified Modeling Language (UML) elements, such as class-, deployment- and component diagrams to provide a concise overview. Existing components are framed in black, while newly added components and extensions are framed in green. Requests to APIs and bootstrapping processes are visualized with arrows originating from the initiator. The numbers in brackets indicate the order in which the requests are executed during the setup and running FL processes.

5.2 Build Process

The following section provides a walkthrough of the deployment process of the *Blockchain Network*, beginning with submitting a scenario utilizing the *Blockchain Reputation* algorithm.

FedStellar's web *User Interface* is hosted by the *Front End* docker container. It provides a form for configuring and submitting FL scenarios. Among other properties, the newly introduced algorithm *Blockchain Reputation* is listed as aggregation algorithm. Confirming the configured scenario in the *User Interface* creates a REST request to the Flask REST *App*. The *App* component proceeds by initializing a *Controller* object passing on the scenario settings from the request.

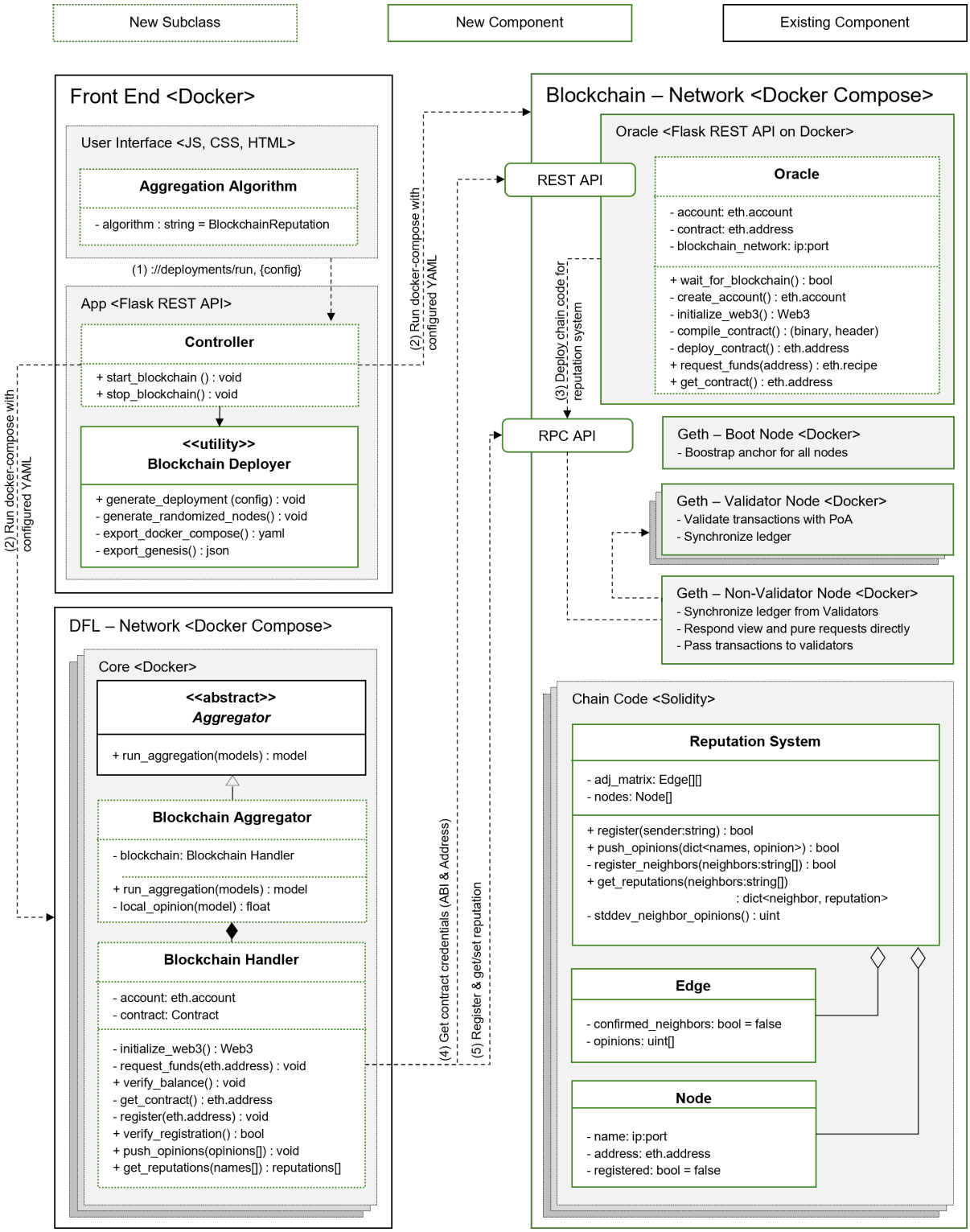

Figure 5.1: Implementation Overview

5.2.1 Controller

The newly initialized *Controller* creates the deployment files for deploying the individual *Cores*. The deployment files consist of a YAML file for Docker Compose and JSON files containing *Core* specific settings. After being deployed by the *Controller*, the *Cores* access the assigned setting files and adjust their configuration accordingly during runtime.

5.2.2 Blockchain

The newly introduced utility class *Blockchain Deployer* is called by the *Controller* to generate the deployment files of the *Blockchain Network*. The class generates a network configuration and an individual service configuration for each individual node of the *Blockchain Network*.

The generated YAML file references for each type of node in the *Blockchain Network* a unique dockerfile. The files contain a collection of build commands configuring the basic node types for their role after deployment. Scenario-specific arguments, such as randomly generated network addresses, are specified in the deployment file and passed as build arguments. This ensures that all private keys, public wallet addresses, and network addresses are uniquely generated for each deployment and not stored in the source code.

Go-Ethereum Nodes

Go-Ethereum (Geth) [44] is used as the underlying framework to create a blockchain infrastructure for FedStellar. Its popularity and simplified deployment using Docker Compose fit into the existing technology stack and deployment process of FedStellar.

Geth's *Validator Nodes* and *Non-Validator Nodes* require a genesis file during bootstrap. The file contains various parameters defining properties of the *Blockchain Network*, such as the consensus protocol and the block time.

Hard coding the public addresses of *Validators* in the genesis file automatically enrolls them in the *Blockchain Network's* Proof-of-Authority (PoA) consensus protocol. This eliminates the need for manual intervention, allowing *Validators* to participate in the consensus protocol immediately upon booting up. Since the public address is derived from the *Validator's* primary keys, the keys are generated during build time by the *Blockchain Deployer* instead of privately during runtime by each individual *Validator*. This ensures the genesis file is created before the individual *Validators'* build time.

Moreover, the genesis file contains the assignment of all existing funds to the wallet address of the *Oracle*. This prevents any external entity from submitting transactions to the *Blockchain Network* without requesting admission and funds from the *Oracle* first.

5.3 Set Up Phase

The following section provides a walk-through of the process of the individual components configuring themselves during runtime. Figure 5.2 visualizes the interactive process with a sequence diagram.

After the *Cores* have booted, they start instantiating all classes required for participating in the federation. Among them is an instance of the *Blockchain Aggregator*, which instantiates a *Blockchain handler*. The initialization process of the *Blockchain Handler* follows an interactive exchange of information between the *Oracle* and the *Non-Validator Node*.

5.3.1 Oracle

The *Oracle's* role during the setup process of the *Reputation System* is deploying the *Reputation System* onto the *Blockchain Network* and providing the *Cores* with the information required to interact with the *Reputation System*. The *Oracle* is implemented as a lightweight Flask [45] server providing a REST interface to the *Cores*. The deployment resources are created by the *Blockchain Deployer* among the other ones of the *Blockchain Network*.

During the build of the *Oracle's* container, the *Reputation System's* solidity file is copied into the image. During runtime, the *Oracle* awaits the *Non-Validators* successful boot before automatically compiling the solidity files and submitting a deployment transaction. The transaction to the *Non-Validator Node* has the generated binary code attached and returns the public address of the deployed chain code.

The individual *Cores* periodically probe the *Oracle's* status until it responds to being ready. The *Cores* then proceed by requesting the public address and the Application Binary Interface (ABI) of the deployed *Reputation System*. Both are required for interacting with the deployed *Reputation System*. In a next step, the *Cores* request funds to be assigned to their public wallet address, which they provide in the payload. The *Oracle* transmits a transfer transaction to the *Non-Validator Node* for provisioning funds from its private account to the *Core's* account.

After having provided the *Cores* with information and funds to directly interact with the *Blockchain Network*, the *Oracle* has fulfilled its main purpose and goes into an idle state. Its secondary role is the provision of funds for *Cores*, which have spent all of their tokens during the aggregation process.

5.3.2 Core

Worker nodes participating in the FL scenario run each on a containerized *Core* component. The *Core* component has implemented all major features required for participating in FL scenarios. The instantiated classes used for training, gossiping, and aggregation are configured according to the deployment files during run time.

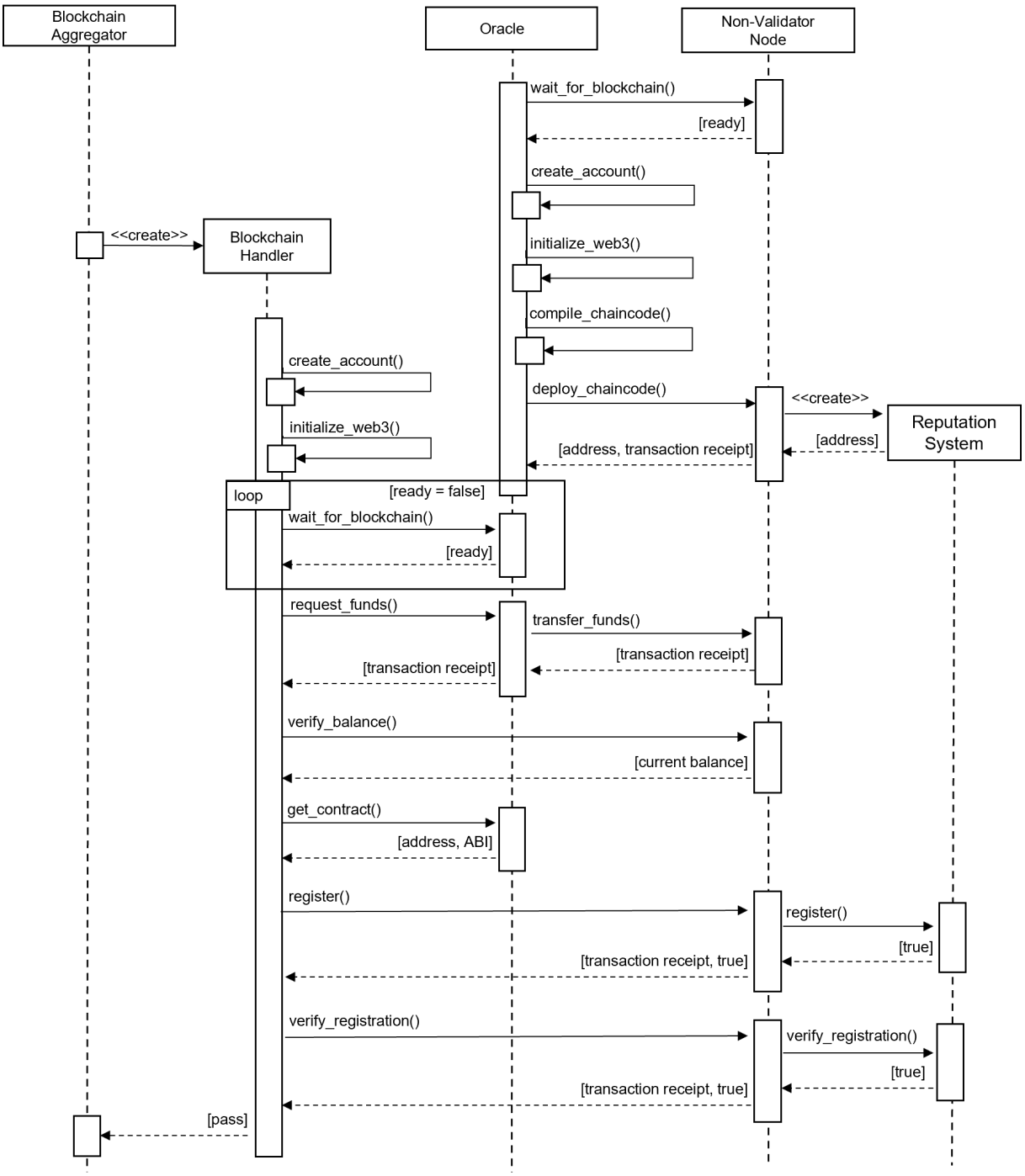

Figure 5.2: Sequence Diagram of Initialization

The individual aggregation algorithms inherit from the abstract class *Aggregator* and implement its abstract method `run_aggregation`. The *Blockchain Aggregator's* realization of the `run_aggregation` method aggregates the buffered models by the weights of their senders' reputation. The reputation values are written and retrieved by the *Blockchain Handler* class, which is instantiated and held by the *Blockchain Aggregator*.

The *Blockchain Handler* generates a private key during its initialization. Compared to the keys generated for the *Blockchain Network* during deployment, the generated keys of the *Cores* are unknown to any other node in the FL scenario, nor are they stored in any deployment file. This implementation ensures that *Cores* can provision themselves independently from the scenario administration and theoretically join the federation at any time. Further, the credentials are generated offline and are not coupled to the private *Blockchain Network* or the internet. During aggregation, the generated private key is used to derive a public wallet address and for signing transactions to the *Blockchain Network*.

After having requested the *Reputation System's* credentials and funds from the *Oracle*, the *Cores* switch to exclusively communicating with the *Blockchain Network*. The final step of the *Cores'* setup process is the registration on the *Reputation System*. By individually sending a signed transaction to the *Reputation system* with their public names in the federation, the *Cores* are permanently registered as scenario participants. The public wallet addresses used for creating the transactions are used to verify the identities during all follow-up requests to the *Reputation System*. After the setup process of the *Blockchain Aggregator* with its *Blockchain Handler* object, the *Core* is all set to utilize the *Reputation Systems* for its aggregation process.

5.4 Federated Learning Process

The following section provides a walkthrough over the aggregation process of the *Cores* and their utilization of the *Reputation System*. The sequence diagram in Figure 5.3 visualizes the aggregation process of the *Blockchain Aggregator* and the interaction with the *Reputation System*.

After successfully setting up the scenario, the *Cores* stay idle until the start of the training process. After receiving the start signal, the *Cores* begin training their local model on their private data. After the training process has terminated, the local model is gossiped to the neighboring *Cores* while received models are buffered.

5.4.1 Blockchain Aggregator

The aggregation process is started after a required number of models were collected or a timeout was reached. Initially, the buffered models are compared to the *Core's* locally trained model. Knowing to be benign itself, the *Cores* use the product of Cosine similarity and average Euclidean distance to assess the trust in other models. However, the *Blockchain Aggregator* class provides multiple similarity metrics and enables the use of any metric-producing values in the unit interval.

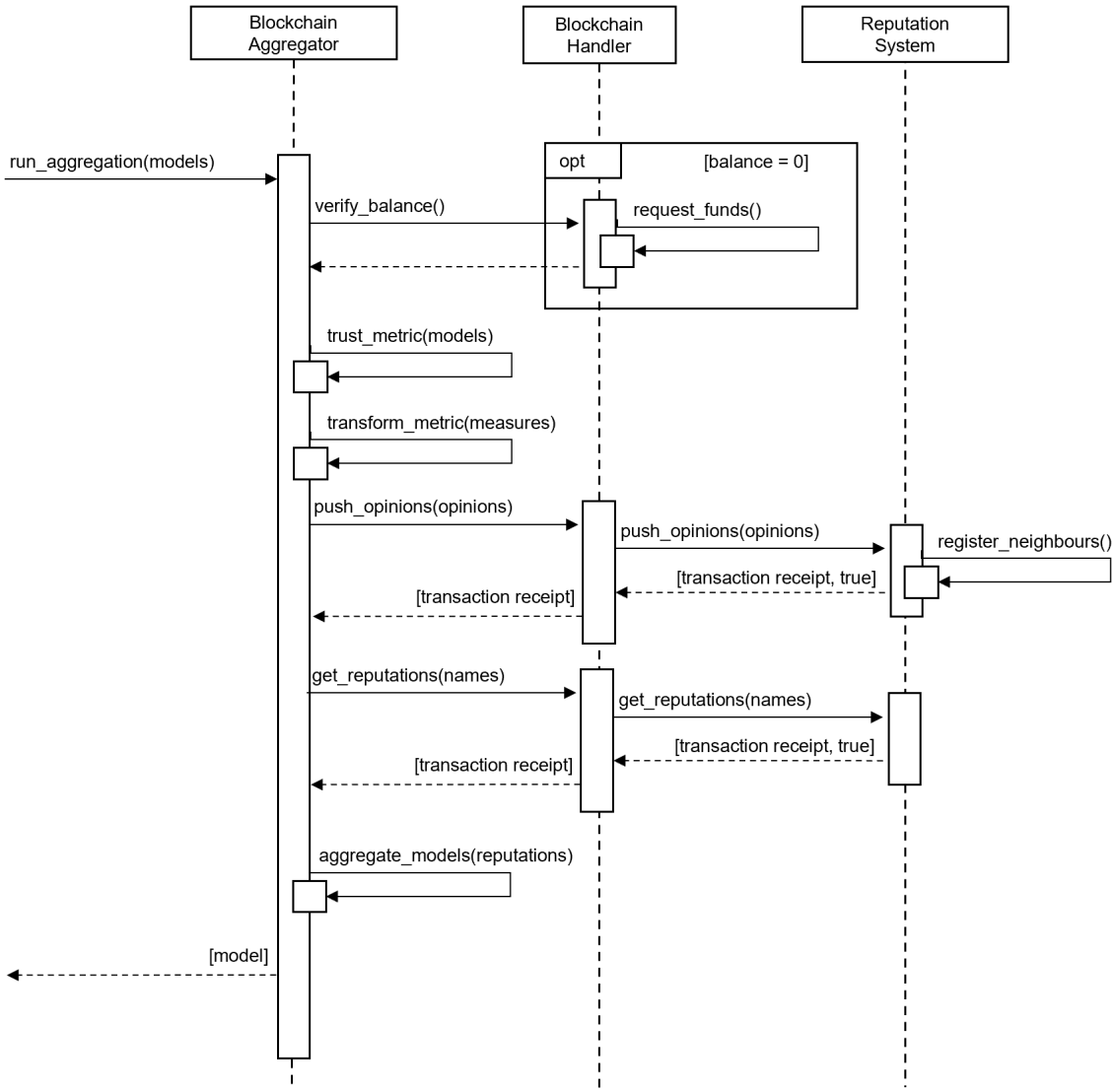

Figure 5.3: Sequence Diagram of Aggregation Process

In the next step, the similarity measures undergo a convex transformation by exponentiation by a constant to generate the local opinions. This enhances the resolution at the higher end, enabling a more distinct differentiation of larger values, while minor differences in lower similarity values are omitted. After transforming the similarity metrics into opinions, a transaction is executed to push the latest local opinion about the creators of the buffered models to the *Reputation System*. After updating the state of the *Reputation System*, it holds the latest view of the aggregating *Core*. In the next step, the aggregating *Core* requests the reputation of each *Core's* model it plans to aggregate with. The *Non-Validator* node returns the reputation values generated by the *Reputation System* to the aggregating *Core*. The received reputation values are locally normalized, to sum up to one and used to aggregate the buffered models by the normalized weights.

5.4.2 Reputation System

The *Reputation System* is implemented on a single smart contract developed in Solidity. Solidity is a statically typed programming language created to develop chain code on the Ethereum blockchain. Its design revolves around the concept of gas from the Ethereum blockchain. Each transaction changing the ledger's state is assessed by the computational resources required to complete. A transaction's runtime and space complexity determine its gas costs, which must be paid when creating a transaction. Transactions not changing but only observing the state of the DL are of type view. Methods entirely independent of the ledger's state are called pure functions. Both types of functions generate no gas costs. However, they are restricted from instantiating dynamic arrays and hash tables. This leads to increased overhead during implementation and an increase in computational complexity.

Further, declaring a variable results in the simultaneous declaration of its default value. Therefore, additional data structures are required to distinguish between variables intentionally set to default values and the newly instantiated default values. Another less common property is the absence of floating point values. Appropriate methods and techniques are required to maintain precision partially lost by integer division. The *Reputation System* is implemented using a high static multiplier, which is applied through all methods.

Push Opinion and Retrieve Reputation

The *Reputation System's* main public methods are `register`, `rate_neighbors`, and `get_reputations`. The methods `register` and `rate_neighbors` change the state of DL and generate gas costs. Although `get_reputations` has the highest runtime complexity among the public methods, its implementation depends entirely on data types kept in memory and does not utilize the DL's persistent storage. This allows defining the method as a type view and results in its execution being free of gas. Overall, the public methods use dictionaries and lists as parameters to minimize the number of transactions to the *Blockchain Network*. This results in reduced gas costs and decreased response time.

The *Reputation System* uses an adjacency matrix for maintaining opinion values and relations. If a *Core* pushes opinion values about another *Core* unknown to the *Reputation System*, the *Core* is registered as a participant. This results in the adjacency matrix being adapted to the new topology. This drastically increases the storage space on the blockchain and is the most expensive method of the *Reputation System*. Each cell of the adjacency matrix contains an array that stores the directed opinion of one *Core* about another *Core*. Algorithm 5.1 shows the resource intensive increase of the adjacency matrix for *Cores* previously unknown to the *Reputation System*.

```
1  // increase the y dimension of the existing adj_matrix's columns
2  while (adj_matrix.length < nodes.length){
3      adj_matrix.push();
4  }
5
6  // increase the x dimension of the existing adj_matrix's rows
7  for (uint64 y=0; y<adj_matrix.length; y++){
8
9      while (adj_matrix[y].length < nodes.length){
10
11         // push an Edge struct which consists of a boolean and a
               list
12         // the boolean indicates if a Core registered another Core
               as neighbour
13         // the list stores the pushed opinion values about another
               Core
14         adj_matrix[y].push(Edge(false, new uint64[](0)));
15     }
16 }
17
18 // confirm all newly registered neighbours for msg.sender
19 for (uint64 j=0; j<neighbours.length; j++){
20
21     // get column index of neighbour
22     uint neighbour_index = names[neighbours[j].key].index;
23
24     // set Edge to neighbour of calling node
25     adj_matrix[node.index][neighbour_index].neighbour = true;
26 }
```

Listing 5.1: Dynamic Increase of the Adjacency Matrix of the Reputation System

The method `rate_neighbors` takes a dictionary of *core* : *opinion* as a parameter and updates the calling *Core's* opinion about each of the other *Cores*. This is implemented by the opinion values being pushed into the individual arrays of the adjacency matrix's edges. Algorithm 5.2 shows the opinion values being pushed to the adjacency matrix associated with the *Core* calling the method.

```
1  // iterate over all neighbour:opinion indexes from the called method
```

```
2 for (uint64 i=0; i<neighbours.length; i++){
3
4     // retrieve the name:opinion from the calling parameters
5     Dict memory neighbour = neighbours[i];
6
7     // retrieve adjacency matrix index of neighboring node
8     uint index_target = names[neighbour.key].index;
9
10    // access cell in adj_matrix for pushing the latest opinion
11    Edge storage edge = adj_matrix[index_sender][index_target];
12
13    // push opinion value to Edge object in adj_matrix
14    edge.opinions.push(neighbour.value);
15 }
```

Listing 5.2: Iterative Storage of Reported Opinion Values in Adjacency Matrix of the Reputation System

The method `get_reputations` takes a list of *Cores* as parameter. For each *Core*, the initial reputation equals the average of the latest pushed opinion of all confirmed neighbors. Algorithm 5.3 shows the iteration over the adjacency matrix and the computation of the average opinion of all confirmed neighbors about a target *Core*. The sum of the *Cores'* opinions is divided by their count and results in the `final_opinion` about a *Core*.

```
1 // iterate over all nodes in the adjacency matrix to retrieve the
     average opinion about a target node
2 for (uint64 i = 0; i < nodes.length; i++) {
3
4     // ignore an edge if not both neighbours did confirm the edge
5     if (confirmed_neighbours(i, index_target) == false){
6         continue;
7     }
8
9     // load all retrieved opinions from storage into memory
10    uint64[] memory participant_history = adj_matrix[i][index_target
         ].opinions;
11
12    // skip averaging the opinion history if there are none
13    if (participant_history.length == 0) {continue;}
14
15    // only include opinion values of the requested round
16    uint64 round_opinion = participant_history[participant_history.
         length-1];
17
18    // scale up sum to reduce integer division error
19    round_opinion *= MULTIPLIER;
20
21    // count the included opinions for computing the average later
         on
22    opinions[i] = round_opinion;
```

```
23      n_opinions++;
24      sum_opinions += opinions[i];
25  }
26
27  // average all final opinions of all neighbouring nodes about a
        target node
28  uint64 final_opinion = n_opinions > 0 && sum_opinions > 0 ? uint64(
        sum_opinions / n_opinions) : 0;
```

Listing 5.3: Computation of Average Opinion about Target Core in Reputation System

After collecting the `final_opinion` about each *Core* in the request, the median and the standard deviation are computed. The statistical values are used to detect outliers. Each initial reputation is assessed by the number of standard deviations it deviates from the median of all requested reputations. If a *Core's* reputation deviates more than a threshold, its final reputation gets divided by a divisor. The divisor depends on the current deviation of all average reputations and the specific *Core's* deviation to the median.

Algorithm 5.4 shows the reduction of deviating *Core's* reputation by dividing its reputation by a divisor. The reduction of the reputation is only applied if the overall deviation is high and a *Core* deviates more equal than one standard deviation from the median. Up to five percentage points in deviation were empirically measured to be highly frequent for both non-IID and IID scenarios of FedStellar without poisoning *Cores*. The polynomial increase of the `divisor` with the increase of the deviation was evaluated to be necessary to minimize the influence of heavily poisoned models during aggregation.

```
1  // if stddev is larger than 5 percentage points, the nodes'
       reputations starts deviating unusually high
2  if (
3           stddev >= 5 * MULTIPLIER &&
4           stddev_count >= 1 * MULTIPLIER &&
5           reputations[i].final_reputation > 0
6      ){
7
8      // reduce the nodes influence to up (2*x)**4 the deviation to
           the median
9      uint64 divisor = (2 * stddev_count)**2 / MULTIPLIER;
10
11     // reduce the node's reputation by the divisor
12     reputations[i].final_reputation = ((reputations[i].
           final_reputation * MULTIPLIER) / divisor);
13 }
```

Listing 5.4: Reduction of Highly Deviating Reputation Values in Reputation System

To account for a *Core's* bias, each initial reputation is weighted by the requesting *Core's* latest opinion about a target *Core*. This prioritizes the requesting *Core's* local view over the average opinion of the other *Cores*. Algorithm 5.5 shows the multiplication of a *Core's* reputation by the last reported opinion of the requesting *Core* about the former.

Additional divisions and multiplications are required to minimize the error from integer division, while ensuring that the resulting values remain within the appropriate range.

```
uint index_target = reputations[i].index;
uint length_opinions = adj_matrix[index_requester][index_target].
    opinions.length;
if (length_opinions > 0){
    uint64 latest_opinion = adj_matrix[index_requester][index_target
        ].opinions[length_opinions-1] * MULTIPLIER;
    reputations[i].final_reputation = (latest_opinion * reputations[
        i].final_reputation) / (100 * MULTIPLIER);
}
// the calling node's own reputation is squared to account for the
    reduction of the other nodes reputation
else if (index_target == index_requester){
    reputations[i].final_reputation = (reputations[i].
        final_reputation**2) / (MULTIPLIER/1000);
}
```

Listing 5.5: Weighting Final Reputation of Core by Opinion of Calling Core in Reputation System

In the last step, the final aggregated reputation of each *Core* is returned to the requesting *Core*. The dictionary returned to the requesting *Core* contains additional information about iterative refinement of the individual *Cores* for analysis and debugging purposes. Required for the *Blockchain Aggregator* to perform the weighted aggregation is the minimal dictionary with the structure *neighbour* : *reputation*.

Weighted Aggregation by Reputation

Algorithm 5.6 shows the final step of the *Core's* method `run_aggregate`, where the received reputation values are used to aggregate all buffered models. In an initial step, all reputation values are normalized to sum up to one. The buffered models are then summed up by the normalized reputations of each model's sender. This results in the models being aggregated by the relative weight of their sender's reputation.

```
# initialize empty model
final_model = {layer: torch.zeros_like(param).float() for layer,
    param in local_model.items()}

# cover case where no models were buffered or none of the senders
    registered themselves yet
if sum_reputations > 0:
    for sender in normalized_reputation_values.keys():
        for layer in final_model:
```

```
8                final_model[layer] += current_models[sender][layer].
                     float() * normalized_reputation_values[sender]
9    # otherwise, skip aggregation
10   else:
11       final_model = local_model
```

Listing 5.6: Weighted Aggregation by Reputation in Blockchain Aggregation Algorithm

Security Measures

The *Reputation System* has implemented various mechanisms to increase the resilience against malicious use. The underlying adjacency matrix represents all reported opinion values in a directed graph. Reporting opinion values increases the data available for anomaly and poisoning detection. Therefore, pushing opinion values to the *Reputation System* is less restricted by security measures. However, computing reputation values from the pushed opinion values is critical and creates various vulnerabilities. Security measures focus on the generation of robust reputation values rather than preventing the reporting of invalid opinion values.

The first basic mechanism is the exclusion of opinion values stored in an unconfirmed edge. Two *Cores* are confirmed neighbors if both have pushed an opinion about each other at minimum once. This increases the time and effort required to frequently and maliciously change identity. Moreover, it prevents *Cores* from rating themselves or *Cores* from which they have no evidence of the training quality.

If a *Core* rates a *Core*, which is currently unknown to the *Reputation System*, the unknown *Core* is registered as an unconfirmed member. This adapts the *Reputation System* to the newly discovered node but prevents any values from being taken into account for the calculations of reputation values. This forces malicious *Cores* to register their identity to the *Reputation System* for being included in the aggregation of the honest aggregators.

The second basic mechanism lies in using the median reputation value to detect malicious participants. In the scenario of multiple malicious *Cores* pushing high opinion values about each other, trust in higher reputation values would be misleading. Therefore, the median reputation is used to compute the deviation and reduce the weights of outliers for high and low reputation values.

The third mechanism prevents malicious *Cores* from reporting incorrect opinion values about honest *Cores*. The currently implemented local opinion metric for two *Cores* exchanging models is symmetric. Therefore, both *Cores* are expected to agree on each other's quality of contribution. If two *Cores'* opinions about each other have a high deviation, both are suspected of not reporting the correct opinion values. The current implementation marks a *Core* as malicious if its average difference in the last pushed opinion about its confirmed neighbors is higher than a dynamic threshold.

Chapter 6

Evaluation

In this chapter, the implemented blockchain infrastructure and distributed reputation system are evaluated. The chapter is split into two chapters. The first chapter evaluates the *Blockchain Network's* performance and resource utilization. The second chapter evaluates the *Reputation System's* ability to detect and mitigate poisoning attacks.

6.1 Performance and Resources

Evaluating the computational resources and performance visualizes the trade off between using resources for training or for making FL more secure. Blockchain and FL training have both a high need for computational resources to stay in sync and contribute to the individual P2P network. The following section elaborates on conducted experiments regarding monetary, time and computational resources used by the newly implemented blockchain infrastructure.

6.1.1 Set Up

All experiments measuring the resource utilization were conducted under the same hardware and FedStellar settings. For running FedStellar a virtual machine with 62 GiB of memory and a AMD-EPYC 32 core processor assigned was used. The underlying scenario consists of ten participating DFL nodes arranged in a full mesh topology.

Ten participating *Cores* were set to train a multi layer perceptron model (MLP) on the MNIST dataset with a Non-IID distribution. The training consisted of five rounds with one epoch each and a batch size of 32. The *Blockchain Network* consists of three *Validator Nodes* with a block time of one second. Using less than three *Validator Nodes* drastically reduces the stability of the PoA concensus algorithm, which was the reason for omitting those configurations. Further, the *Blockchain Network* consists of an *Oracle*, a *Non-Validator Node*, and a *Boot Node* each. Other than the number of *Cores*, the chosen settings reflect FedStellar's current default parameters.

6.1.2 Computational Resources

For an estimation of the trad off between training and validation, the CPU time and the average memory utilization were measured during the training process. Since the the VM was not assigned a Graphics Processing Unit (GPU), all node types utilized the same CPU and memory resources without bypassing calculations to the GPU.

Relative CPU and Memory utilization

To compare the computational resource utilization of the *Blockchain Network* identical scenarios with increased numbers of *Validator Nodes* were observed. This monitoring was accomplished through the continuous tracking of the hardware utilization for each Docker container during scenarios, facilitated by a Python script. The CPU times of the *Validators* and the *Cores* were summed up to categorize the utilization per type of node rather than single entities. Further, the legends of the subsequent figures are ordered in descending order of the total amount of resources utilized over all runs shown in the figure.

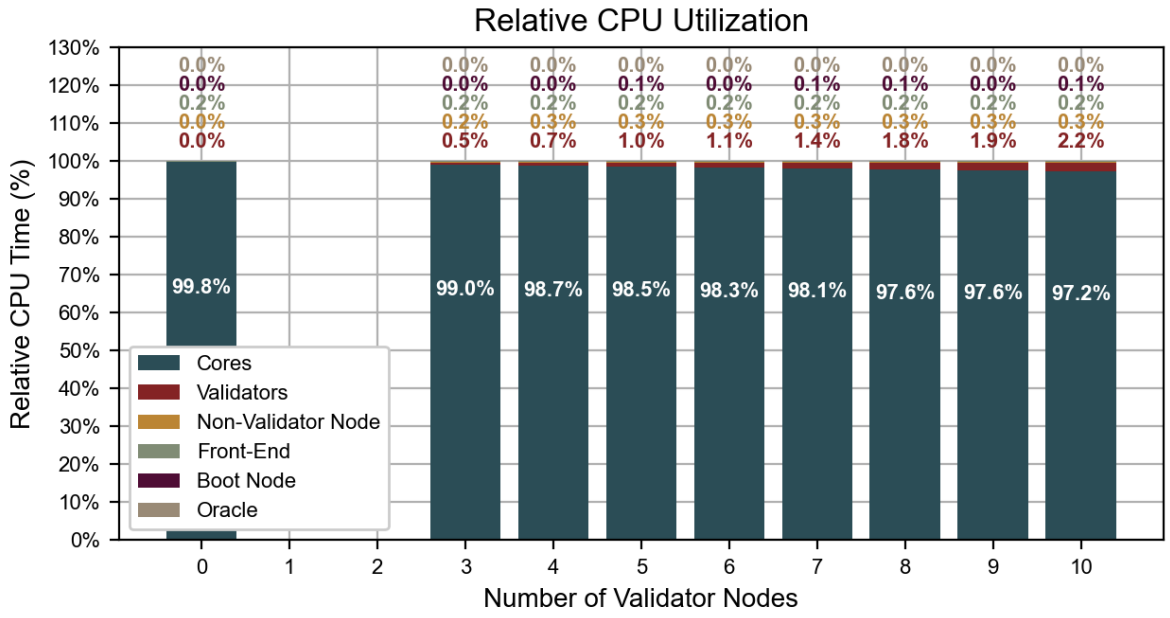

Figure 6.1: Relative CPU Utilization

Figure 6.1 shows the relative amount of CPU-time over the total amount of CPU-time used by each type of node. Similarly, Figure 6.2 visualizes the relative amount of average memory used during a scenario. Both figures visualize the relative amount of the individual resource used for training, general infrastructure or validation.

Figure 6.1 shows that the majority of CPU time is used for training and only a fraction for the *Blockchain Network* and the *Front End*. Similarly, Figure 6.2 highlights that the training process uses the majority of the available memory. Compared to the CPU time,

the relative use of memory for the validation is higher. Moreover, with only up to four *Validator Nodes* the *Front End* uses the second most amount of memory compared to the other node types.

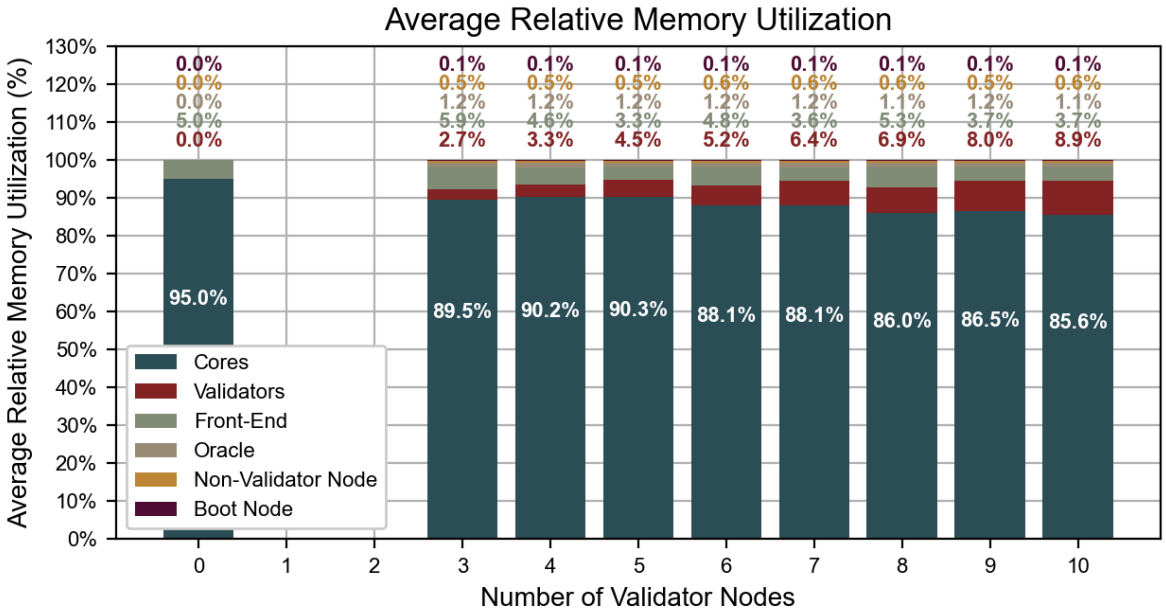

Figure 6.2: Average Relative Memory Utilization

Furthermore, the increase in the relative amount of memory is higher than the increase in CPU time for the *Validator Nodes*. This demonstrates that the memory is a higher trade off with increased *Validator Nodes*, compared to the CPU time. Using the minimal infrastructure of three *Validator Nodes* results in a loss of five percent of memory and less than one percent in CPU time for the validation instead of training.

Absolute CPU and Memory utilization

Excluding the *Cores* and the *Front-End* from Figure 6.1 and 6.2, allows a more focused analysis of the change in computational resources by increased number of *Validator Nodes*. Figure 6.3 shows the absolute CPU-time per type of node of the *Blockchain Network*. Similarly, figure 6.4 shows the absolute average utilized memory per type of node of the *Blockchain Network*.

Figure 6.3 reveals that the *Non-Validator Node's* CPU-time also increases with increased number of *Validator Nodes*, although only slightly. Surprisingly, the *Oracle* uses less CPU-time than the *Boot Node*. This, although the *Oracle* is running a REST server while the *Boot Node's* purpose is the sole exchange of addresses for the *Blockchain Network*. Overall, the increase in CPU-time of the *Validator Nodes* in the observed range is approximately linear.

In contrast to the CPU-time, the *Oracle* uses the second most amount of the memory of all the nodes from the *Blockchain Network*. Its absolute utilization of memory stays

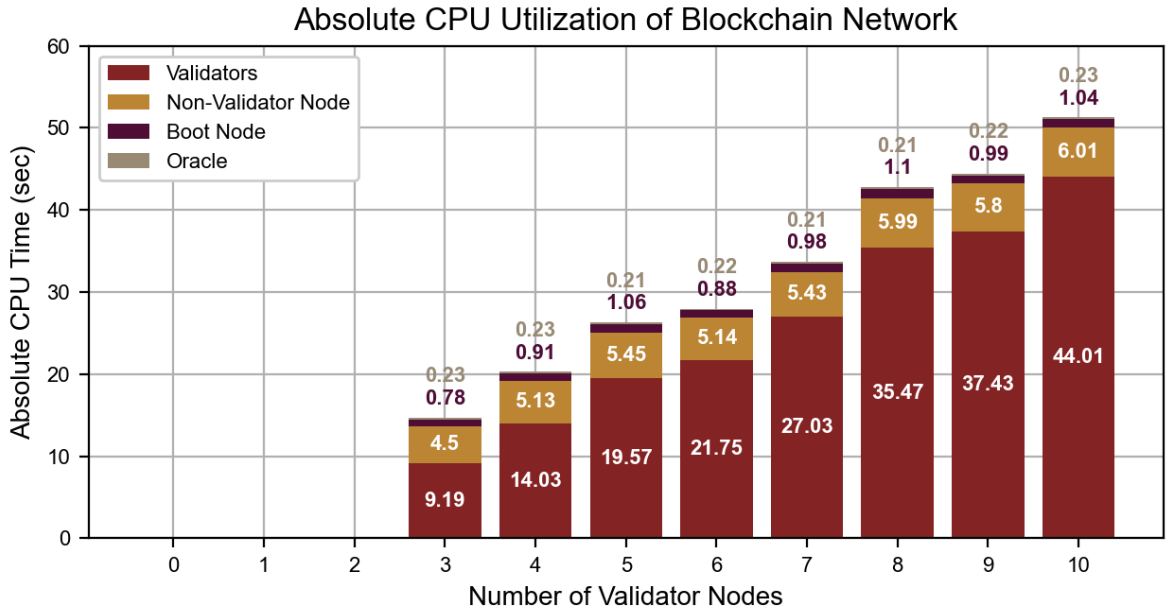

Figure 6.3: Absolute CPU Utilization of Blockchain Network

constant with increased *Validator Nodes*. Since the *Oracle's* purpose is fulfilled at training start, its load does not increase with a higher number of *Validator Nodes* during training. Similarly to the CPU-time the *Non-Validator Node's* demand of memory correlates with the number of *Validator Nodes*, although it is moderately for both resource types. Overall, the increase in total memory for all *Validator Nodes* is again approximately linear in the observed range.

6.1.3 Time and Gas Cost

Blockchain technology is known to increase the security at the cost of validation time and computational overhead. Since the *Reputation System* is deployed on the *Blockchain Network*, latency and computational overhead are passed on to FedStellar. The following experiments focus on the impact on aggregation time and the theoretical financial costs which would arise if the public Ethereum main chain was used instead of the implemented private blockchain.

Aggregation Time

The aggregation process of a *Core* using the *Blockchain Aggregator* requires two transactions to the *Blockchain Network*. The first call is used to write the newest computed opinion values to the DL while the second call requests the latest aggregation weights. Compared to the second call, the first call changes the state of the DL and must therefore await the transaction's successful validation into a permanent block. Depending on the current state of the *Validators* the aggregating *Core* has to wait up to a full block

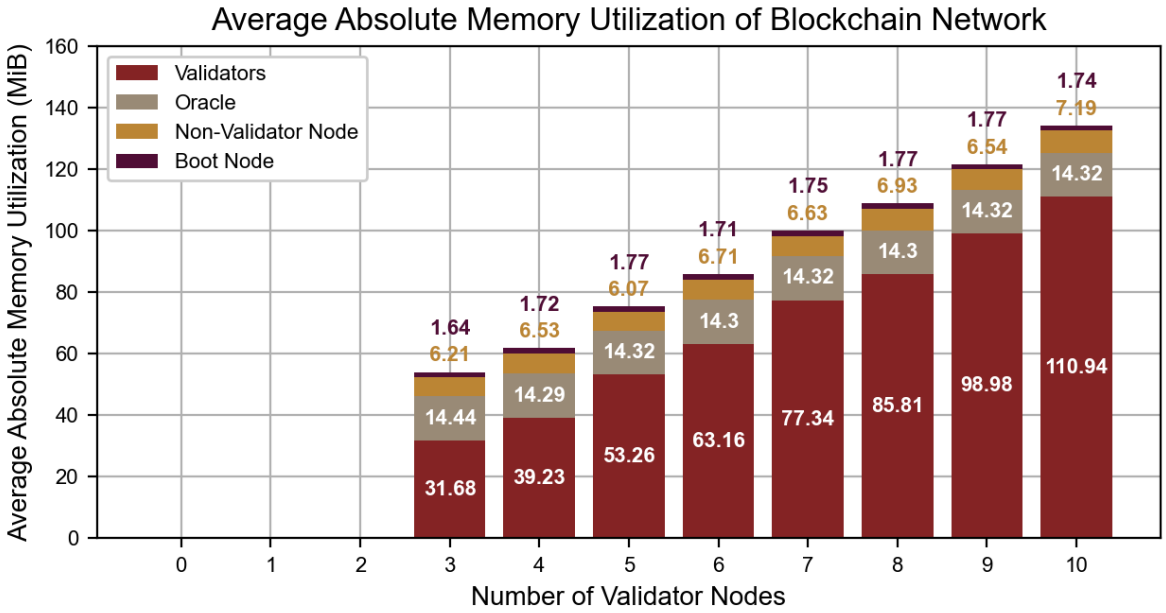

Figure 6.4: Average Absolute Memory Utilization of Blockchain Network

time. This behavior introduces high deviation in the aggregation time and introduces a correlation between aggregation time and block time.

Figure 6.5 shows the average aggregation time over all *Cores* with increased block time. The x-axis in figure 6.5 defines the block time in seconds, imposed by three *Validator Nodes*. The y-axis shows the average aggregation time over multiple runs averaged over all aggregating *Cores*. The set up used for this experiment is equal to the initial setup, with ten *Cores* arranged in a full mesh topology.

A block time of zero forces the *Validator Nodes* to validate and finalize a single block for each transaction. This results in a submitted transaction instantly being validated and minimizes the response time. However, this increases the vulnerability against attacks aiming at maliciously changing the state of the ledger. Figure 6.5 empathizes the implementation's achievable lower bound of .47 seconds. Block times up to five seconds introduce high deviations introduced by mechanisms not entirely clear during the current stage of analysis. Measurements with block times of five seconds and above show less deviation and empathize a trend. The average aggregation time is strictly lower than the block time and in the observed range on average 72 to 86% of the measurement's block time. Because of the limited throughput of the Ethereum main chain, transactions are usually prone to await multiple block times. The realistic aggregation time is therefore arguably higher than the current measurements with the main chain's block time of 12 seconds.

Figure 6.6 shows the average aggregation time by aggregation algorithm using the previous base scenario. FedAvg averages the models without performing any poisoning detection or mitigation. With an average aggregation time of 0.03 seconds, its aggregation time is noticeably lower compared to the other algorithms. TrimMedian performs outlier filtering before averaging the models and is with 0.10 seconds the fastest measured aggregation

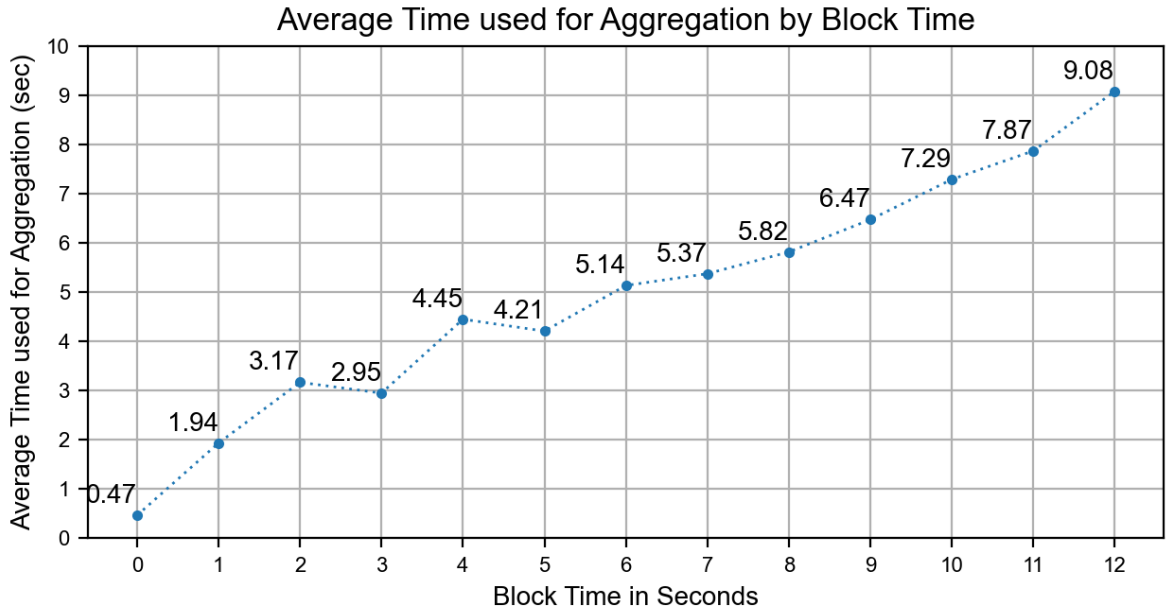

Figure 6.5: Average Time used for Aggregation by Block Time

algorithm performing poisoning mitigation. Krum performs a more sophisticated outlier filtering relying on the geometric median of models. Its aggregation time is three times higher than TrimMedian. The *Blockchain Reputation* algorithm with its *Reputation System* has the highest average aggregation time. With a block time of zero its average aggregation time is 42% higher than the one of Krum. Considering the *Blockchain Reputation's* high computational overhead, the algorithm introduces only low additional latency.

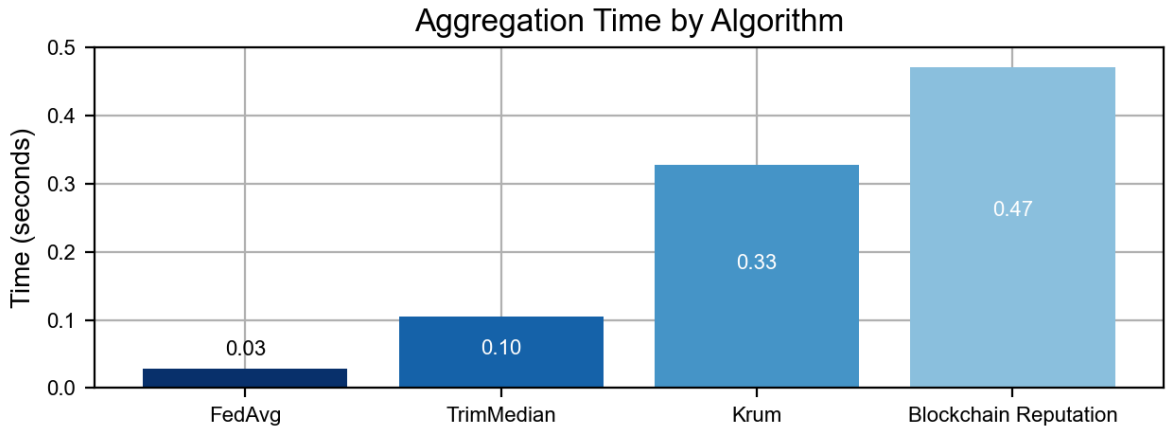

Figure 6.6: Average Aggregation Time by Algorithm

Gas Costs

In Ethereum every transaction is deterministically estimated by the computational resources it requires to change the state of the DL. This estimation is quantified in gas. The cost of a transaction is the product of gas and the gas price (ETH/gas). The gas price is set by the transacting account and can vary for each individual transaction. Transactions with a higher gas price cost the transacting account more but are more likely to get validated faster. This, since each validator can freely decide what minimal compensation it requires for its computational resources to be profitable. Further, validators prefer validating transactions with a high gas price to earn more for providing computational resources. Transactions with a low gas price therefore are valid but might never get validated.

The newly introduced *Blockchain Network* is a permissioned private blockchain using PoA for reaching consensus. Gas costs still exist as a theoretical concept of paying for transactions. However, compared to the Ethereum main chain, there does not exist a market between Fiat currencies and the native currency of the private blockchain. Further, validation is not a competitive process. Therefore, the gas price can be arbitrarily low if the *Validator Nodes* are configured to accept such low priced transactions. Nevertheless, the gas price for transactions can still be set to a realistic gas price of the Ethereum main chain. This results in an estimation of the costs in ETH if the *Reputation System* would be deployed on the public Ethereum main chain. To estimate the cost in a Fiat currency, the cost in Ether are be multiplied by a market exchange rate of Fiat/Ether. For the estimation, the average gas price and exchange rate on the randomly selected date of April 8th, 2024, were utilized. On that day, the average gas price was 27.3 Gwei per gas unit. Concurrently, the average exchange rate for Ethereum (ETH) to US Dollar (USD) stood at 3694.56 USD per ETH.

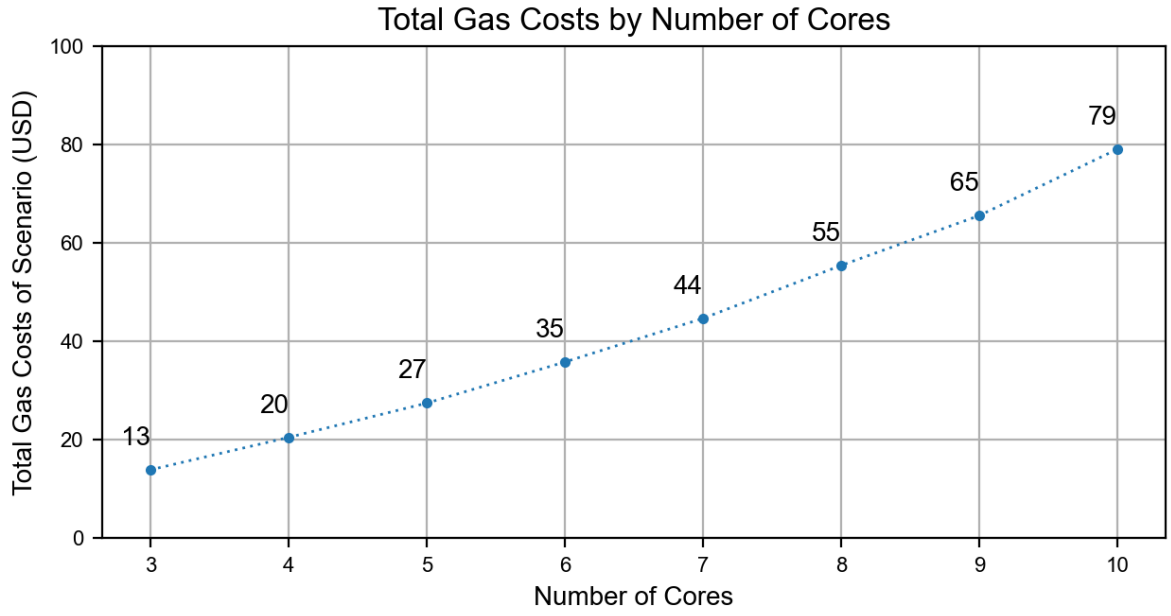

Figure 6.7: Total Gas Costs By Number of Cores

Figure 6.7 shows the total gas costs by number of *Cores*. The x-axis shows the number of *Cores* using the *Reputation System* for aggregation. The y-axis shows the total costs of all individual transactions from the *Cores* and the *Oracle*. This includes the deployment costs, the provision of funds to the *Cores* and the individual transactions of storing reputation values on the *Reputation System*.

The *Reputation System* allows the individual *Cores* to store an opinion about every other *Core*. This quadratic relationship is reflected in the growing rate of the total gas costs. Further, the absolute costs are high, given that each *Core* stores in each of the five rounds a single integer for each other *Core*. However, Ethereum's gas cost and market exchange rate are highly volatile and sensitive to changes in the protocol and the chain's current utilization.

6.2 Reputation System

The main purpose of the *Blockchain Network* and its *Reputation System* is the secure aggregation of models in DFL scenarios. In the following, the *Reputation System's* ability to detect and reduce the impact of model poisoning is evaluated. Further, one of the *Reputation System's* basic feature for detecting and preventing noise attacks on the system itself is tested.

6.2.1 Defense against Model Poisoning

The newly introduced *Reputation System* has the purpose of providing aggregation weights according to the reputation of the individual models. While any local trust metric can be used to report an opinion about models to aggregate, a combination of Cosine similarity and Euclidean distance is currently set for the scenario.

FedStellar's existing *NoiseInjectionAttack* was used to simulate a poisoning attack on a DFL scenario. Its current implementation sets malicious nodes to aggregate all received models with the benign aggregation algorithm before poisoning the newly aggregated model. After the local poisoning, the malicious nodes proceed training the model honestly.

The model poisoning is achieved by Gaussian-distributed additive noise. A random Gaussian distribution with the same dimensions as the model is generated and scaled by a static factor before being summed with the model to poison. Gaussian distribution was generated to have a mean of zero and a variance of one. The normal distribution was scaled up by a factor of ten which preserves the mean but increases the variance to 100.

For the experiment ten participating *Cores* were set to train a multi layer perceptron model (MLP) on the MNIST dataset with a Non-IID and an IID scenario. For both data distributions the default settings of FedStellar were used. The distribution for the IID scenario are implemented using a Dirichlet distribution with an alpha of one half. The training consisted of ten rounds with one epoch each and a batch size of 32. The performance of the aggregation algorithms Krum and FedAVG were compared to the performance of the *Reputation System*.

Non-IID Scenario

Figure 6.8 shows the change in accuracy with the individual aggregation algorithms by increased number of malicious *Cores* for a Non-IID scenario. The *Blockchain Reputation's* accuracy remained stable and high with increased percentage of malicious *Cores*. FedStellar's implementation of the Krum aggregation algorithm shows a similar resilience against model poisoning but is outperformed by the *Blockchain Aggregator*. FedStellar's implementation of the FedAvg aggregation algorithm does not perform any poisoning defense. Therefore, its accuracy rapidly declines with noise injecting *Cores*. Furthermore, the accuracy of the final model with the *Blockchain Aggregator* outperformed both Krum and FedAvg without active poisoning.

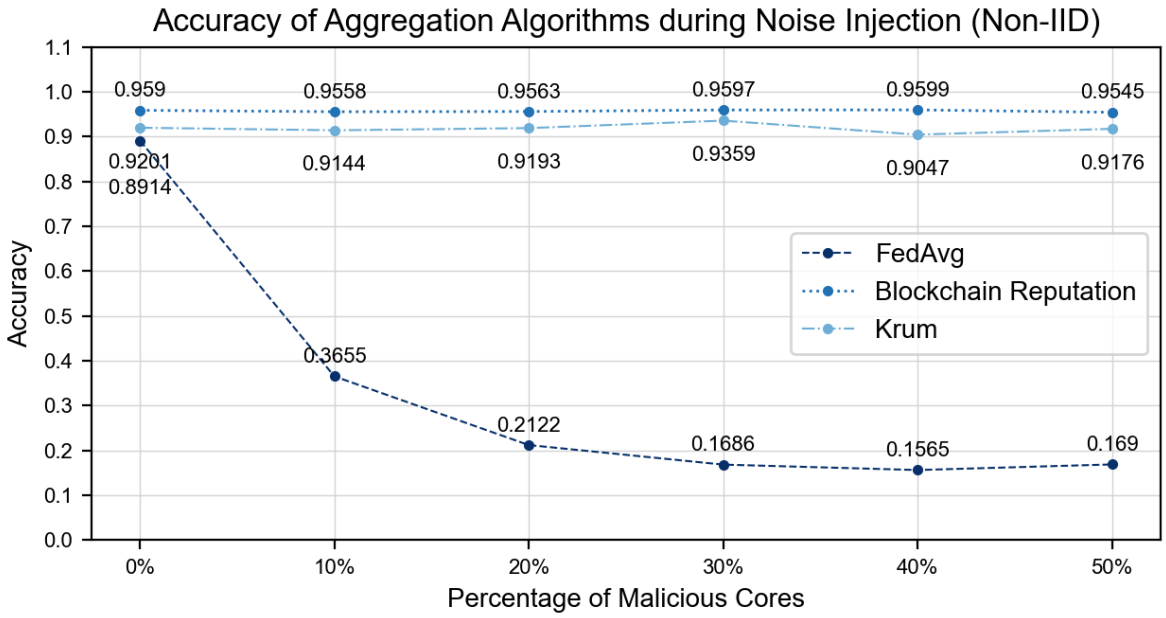

Figure 6.8: Accuracy of Aggregation Algorithms during Noise Injection (Non-IID)

IID Scenario

Figure 6.9 shows the change in accuracy with the individual aggregation algorithms by increased number of malicious *Cores* for an IID scenario. The *Blockchain Reputation's* accuracy remained stable and high with increased percentage of malicious *Cores*. Similarly to the Non-IID scenario in Figure 6.8, FedStellar's implementation of the Krum aggregation algorithm achieves a similarly high accuracy. Overall, the accuracy of the final model with the *Blockchain Aggregator* outperformed both Krum and FedAvg without active poisoning in both Non-IID and IID scenarios.

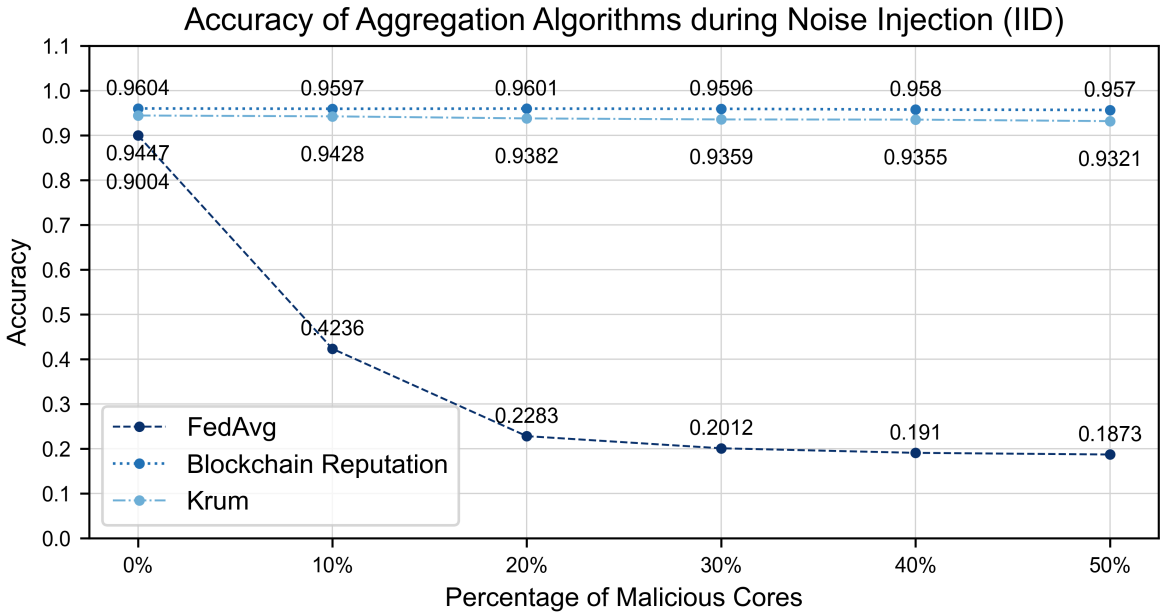

Figure 6.9: Accuracy of Aggregation Algorithms during Noise Injection (IID)

Separation of Consistent Groups

Figure 6.10 shows a heatmap of the average reported opinion by *Core* about each other *Core* in the previously used Non-IID scenario with four model poisoning *Cores* over ten rounds. The axes represent the malicious and benign *Cores*, which are arranged in alphabetical order according to their identification index. Since the local opinion is computed using similarity metrics, the values are also an indicator for similarities of the *Core's* models.

The heatmap shows two rather consistent groups reporting high opinion values for *Cores* of their group while reporting low opinion values about the other group's *Cores*. This indicates that models actively poisoned by malicious *Cores* are more similar to each other than to the benign *Cores'* models and vice versa. Surprisingly, the reported opinions of the malicious *Cores* about each other are considerably higher than the opinions of the benign *Cores* about each other.

Two *Cores*, "Benign 1" and "Malicious 4", deviate from their individual group. Both are evaluated as semi-honest by their individual group as well as from the other group. This shows a partial success of *Core* "Malicious 4" in maintaining a constant rate of poisoning while being partially accepted by the benign *Cores* for aggregation. At the same time *Core* "Benign 1" was successfully poisoned by deviating from its group and sharing similarities with actively poisoned *Cores'* models.

The *Reputation System* computes an subjective reputation for each individual *Core*. In scenarios with active poisoning this creates encapsulated groups which highly agree on their quality of contribution. The groups mainly aggregate with other *Cores* of their group while decreasing the aggregation weights for *Cores* of the other group. This reduces the

necessity of classifying the individual groups to be benign or malicious in the *Reputation System*.

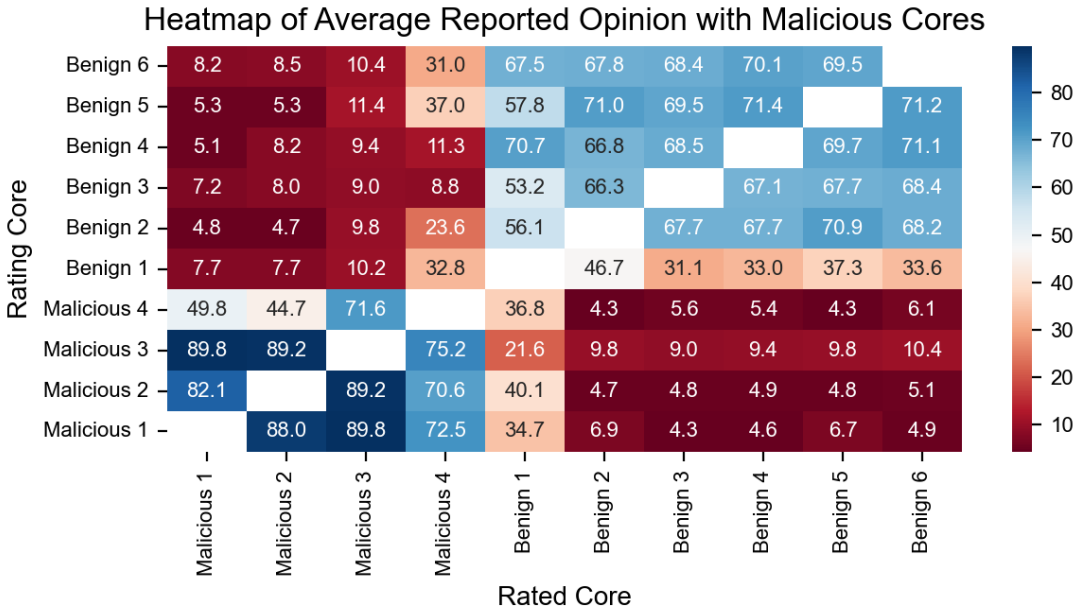

Figure 6.10: Average Reported Opinion with Malicious Cores

6.2.2 Defense against Reputation Poisoning

The *Reputation System* introduces new vulnerabilities by depending on the honest participation of the *Cores*. High deviations in the reputation values reduce the *Reputation System's* ability to detect and mitigate model or data poisoning attacks. Consequently, multiple malicious attacker flooding the *Reputation System* with randomly generated opinion values could make the *Reputation System* unusable.

For the experiment ten participating *Cores* were set to train a multi layer perceptron model (MLP) on the MNIST dataset with a Non-IID distribution. The training consisted of ten rounds with one epoch each and a batch size of 32. Figure 6.11 shows eight benign *Cores* reporting correct opinion values while two malicious *Cores* start reporting randomly generated values beginning in round five.

Figure 6.11 shows the average computed reputation for both the reputation poisoning *Cores* in red and the benign *Cores* in blue. The box plots visualize the distribution of the computed reputation for all *Cores* in each round. Rounds not affected by reputation poisoning show a stable and uniform average reputation for all *Cores*. As of round five, the randomly generated opinion values start disturbing the reputation of all participating *Cores*. The *Reputation System* was able to automatically recognize the anomaly, reducing the reputation of the malicious *Cores* to zero. This results in the malicious *Cores's* models being excluded from all further aggregations performed by honest *Cores*.

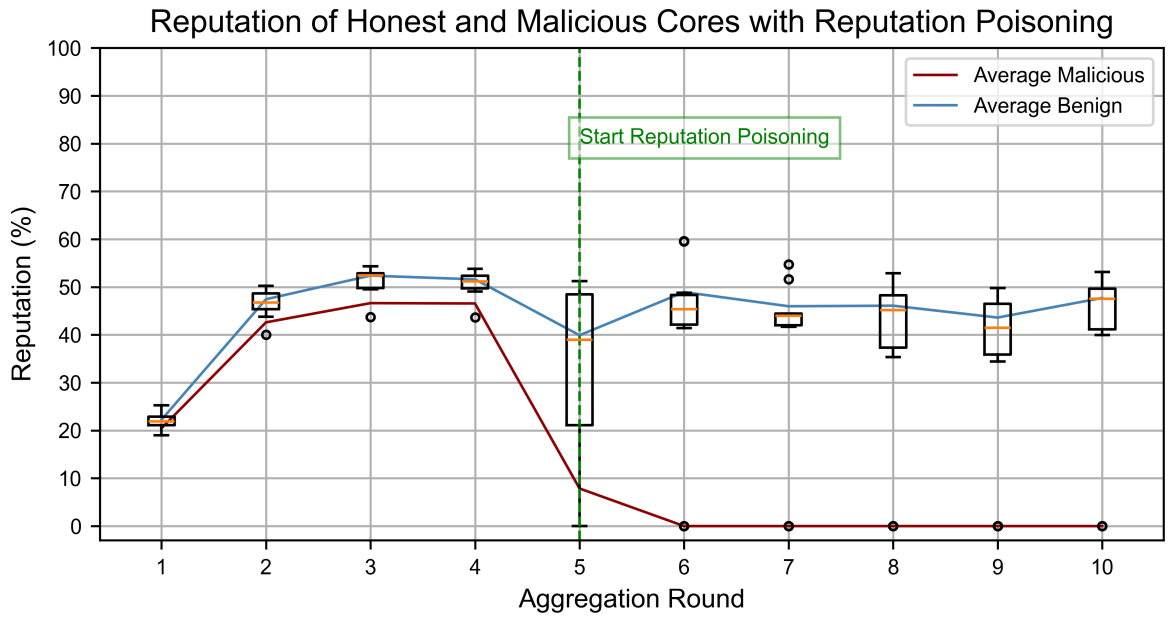

Figure 6.11: Reputation by Participant with Reputation Poisoning

Chapter 7

Summary and Conclusions

The following chapter is split into two sections: Summary and Conclusion. In the summary the insights from the evaluation are summed up, while their limitations and implications for future work are outlined. In the conclusion, the work and its objective are discussed.

7.1 Summary and Future Work

The following section summarizes the results of the conducted experiments and proposes future work. The subsection summarizing the experiments follows the structure of the evaluation having two subsections: Performance and Resources, and the reputation systems efficacy. The summary is followed by acknowledged limitations of the evaluation and its results.

7.1.1 Performance and Resources

The experiments have shown that the *Blockchain Network* is substantially less demanding in CPU time and memory compared to the training *Cores*. Whereas only a fraction of the available CPU time is used for the *Blockchain Network*, it is more demanding in memory. The *Validators* utilize the majority of the *Blockchain Network's* memory and CPU time, while the other components such as the *Boot Node*, the *Oracle* and the *Non-Validators* use only minor parts of the available resources. The increase in demand of CPU time and memory with additional *Validators* is comparable. However, the absolute demand in memory is substantially larger than the demand in CPU time. Overall, the resource utilization of the *Oracle* remain at a constant level while the CPU time of the *Non-Validator Node* and *Boot Node* correlated positively with the utilized CPU time of the *Validators*. In summary, memory is the limiting factor for scaling the *Blockchain Network* while the increase in *Validators* simultaneously increases the demand in resources for other types of nodes.

Furthermore, the experiments have shown a positive correlation of the aggregation time with the block time of the *Blockchain Network*. However, the average aggregation time converged to a strictly lower time than the underlying block time. Moreover, the aggregation time of the *Blockchain Aggregator* is substantially slower than FedStellar's existing algorithms, using Ethereum's original block time of 12 seconds. Setting a block time of zero seconds for the *Blockchain Network* still ranks the *Blockchain Aggregator* last in terms of speed. Considering the high computational overhead of the *Reputation System*, the *Blockchain Aggregator* is with an increased aggregation time of 42% noticeably slower than Krum, but still achieves a reasonable performance. Measurements of the gas costs have shown a non-linear increase in total gas costs with an increased number of *Cores*. Therefore, scenario with a high number of participating *Cores* would result in considerably high operational costs, given a randomly selected average daily market price of Ether to USD.

In summary, the underlying blockchain infrastructure introduces high latency during aggregation due to the block time. However, using the minimal possible block time results in comparably low latency for the additional gain in security. On the other hand, the increase in financial costs are an additional limiting factor for scenarios involving a larger number of *Cores* if Ethereum's main chain was used.

7.1.2 Reputation System

The *Blockchain Aggregator* making use of the *Reputation System* achieved a stable and high accuracy during active model poisoning attacks. Even with 50% of the *Cores* actively poisoning, the loss in accuracy is minimal. Furthermore, the measured accuracies of the evaluated Non-IID and the IID scenarios differ only marginally. Compared to the Krum aggregation algorithm, the *Blockchain Aggregator* achieved a consistently higher and more stable accuracy over all experiments. In summary, the *Blockchain Aggregator's* defense against poisoning attacks has shown to be effective in the evaluated scenarios.

The *Reputation System's* defense mechanisms for detecting asymmetric opinions resulted in the exclusion of the malicious *Cores* from the aggregation with the remaining *Cores*. However, the created noise remained in the *Reputation System* and destabilized the computation of the reputation by increasing the variation in opinions. In summary, the *Reputation System's* mechanism have shown to be capable of detecting and mitigating malicious behavior. On the other hand, during an attack the detected noise is not removed from the system and keeps affecting its reputation algorithms.

7.1.3 Limitations

While the evaluations provided insights into the performance and effectiveness of the implemented system, several limitations should be acknowledged.

- **Resource Constraints**: The experiments were conducted on a single VM. This environment may not represent the performance and resource utilization in a highly distributed and heterogeneous DFL scenario.

- **Dataset & Model Scope**: The evaluation of the poisoning attack was performed using the MNIST dataset on a MLP model which may not reflect more complex models and datasets.

- **Network Topology**: All experiments were conducted using a full mesh topology to increase consistency and get an upper bound estimation for resource usage. Different topologies might affect the performance of the *Blockchain Network* or the *Reputation System*.

- **Poisoning Attack**: For evaluating the efficacy of the *Reputation System* against poisoning attacks, only model poisoning using Gaussian-distributed additive noise was evaluated. To evaluate the overall performance of the *Reputation System*, more experiments using various model poisoning and data poisoning attacks are required.

- **Cost Analysis**: The theoretical cost analysis is based on a randomly selected daily average of the exchange rate of USD to ETH. The Ethereum blockchain and the market price for ETH are both volatile and subject to constant change. For a more comprehensive understanding, further detailed analysis is necessary.

7.1.4 Future Work

The conducted experiments and their results provide new research opportunities. Various dimensions of the current implementation can be improved or leveraged to further explore the use of DL technology within DFL.

The current implementation of the *Reputation System* relies on a trust metric computed by the individual *Cores*. The trust metric is transformed into a local opinion value and reported to the *Reputation System*. Therefore, the performance of the *Reputation System's* reputation algorithm is tightly coupled to the performance of the trust metric. Further research could improve the *Reputation System's* performance by identifying highly capable metrics to sense attacks.

Moreover, the admission for a *Core* to participate in the DFL scenario is currently not restricted. The *Oracle's* role of providing the *Cores* with funds and the address of the *Reputation System* could be extended to implement a strict admission protocol. This would increase the security of the federation and the *Reputation System*.

Conducted experiments have revealed high operational costs for the *Reputation System*. The gas costs of the *Reputation System* could further be reduced by minimizing the storage complexity of the implemented algorithms. This could improve the *Reputation System's* scalability by reducing the operational costs and the latency introduced by the computations.

The current infrastructure of the *Blockchain Network* is solely used for the *Reputation System*. The *Oracle* could be utilized to deploy further chain code with various purposes. Implementing a voting based Krum algorithm or aggregating the models directly on chain could further improve FedStellar's resilience against attacks.

7.2 Conclusions

This work's primary goal was to design and implement a distributed ledger-based reputation system for robust aggregation within DFL. The core feature of the system is the computation of reputation based weights for robust weighted aggregation. To increase the system's flexibility the computation of the local trust metric and the reputation algorithms are decoupled and interchangeable. Additionally, the system has various mechanisms implemented to reduce its vulnerability to attacks targeting the integrity of the reputation algorithms.

Experiments have shown that adjusting the aggregation weights according to the combined trust metrics of the individuals is effective for mitigating the evaluated poisoning attack. The loss in accuracy for both default Non-IID and IID settings of FedStellar were minimal. The reputation system in combination with weighted aggregation demonstrated superior performance compared to the Krum algorithm, both in the presence and absence of active poisoning.

Experiments regarding the computational resources revealed memory to be the most constraining resource when balancing the utilization of available computational capacity for training purposes versus enhancing the security of the DL. Moreover, the inherited properties of the underlying blockchain were identified to affect the performance of the aggregation process. The block time of the underlying blockchain introduces high latency to the aggregation process and positively correlates with the resulting aggregation time. Setting up the blockchain with the minimal possible block time results in the newly proposed algorithm being 42% slower than Krum. This is comparably fast considering the computational overhead compared to the locally and offline computed Krum algorithm. However, the financial costs with non-linear growth being generated by deploying the reputation system on the Ethereum main chain are eminent and reduce the system's scalability.

In conclusion, the proposed system has proven effective in mitigating the evaluated attacks by combining the participants' local views. Beyond the inherited benefits of distributed ledgers, the system inherits certain weaknesses, such as memory requirements, increased latency due to the block time and its financial costs associated with gas fees.

Bibliography

[1] J. Kang, Z. Xiong, D. Niyato, Y. Zou, Y. Zhang, and M. Guizani, "Reliable federated learning for mobile networks", *IEEE Wireless Communications*, vol. 27, no. 2, pp. 72–80, 2020. DOI: 10.1109/MWC.001.1900119.

[2] S. AbdulRahman, H. Tout, H. Ould-Slimane, A. Mourad, C. Talhi, and M. Guizani, "A survey on federated learning: The journey from centralized to distributed on-site learning and beyond", *IEEE Internet of Things Journal*, vol. 8, no. 7, pp. 5476–5497, 2020.

[3] E. T. M. Beltrán, M. Q. Pérez, P. M. S. Sánchez, *et al.*, "Decentralized federated learning: Fundamentals, state of the art, frameworks, trends, and challenges", *IEEE Communications Surveys & Tutorials*, 2023.

[4] V. Shejwalkar, A. Houmansadr, P. Kairouz, and D. Ramage, "Back to the drawing board: A critical evaluation of poisoning attacks on production federated learning", in *2022 IEEE Symposium on Security and Privacy (SP)*, 2022, pp. 1354–1371. DOI: 10.1109/SP46214.2022.9833647.

[5] A. K. Nair, E. D. Raj, and J. Sahoo, "A robust analysis of adversarial attacks on federated learning environments", *Computer Standards & Interfaces*, vol. 86, p. 103 723, 2023, ISSN: 0920-5489. DOI: https://doi.org/10.1016/j.csi.2023.103723. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0920548923000041.

[6] T. Li, A. K. Sahu, A. Talwalkar, and V. Smith, "Federated learning: Challenges, methods, and future directions", *IEEE signal processing magazine*, vol. 37, no. 3, pp. 50–60, 2020.

[7] A. Gholami, N. Torkzaban, and J. S. Baras, "Trusted decentralized federated learning", in *2022 IEEE 19th Annual Consumer Communications & Networking Conference (CCNC)*, 2022, pp. 1–6. DOI: 10.1109/CCNC49033.2022.9700624.

[8] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data", in *Artificial intelligence and statistics*, PMLR, 2017, pp. 1273–1282.

[9] D. I. Dimitrov, M. Balunović, N. Konstantinov, and M. Vechev, *Data leakage in federated averaging*, 2022. arXiv: 2206.12395 [cs.LG].

[10] J. Zhu, J. Cao, D. Saxena, S. Jiang, and H. Ferradi, "Blockchain-empowered federated learning: Challenges, solutions, and future directions", *ACM Comput. Surv.*, vol. 55, no. 11, Feb. 2023, ISSN: 0360-0300. DOI: 10.1145/3570953. [Online]. Available: https://doi.org/10.1145/3570953.

[11] E. Tomás Martínez Beltrán, Á. L. Perales Gómez, C. Feng, *et al.*, "Fedstellar: A platform for decentralized federated learning", *arXiv e-prints*, arXiv–2306, 2023.

[12] B. Anthony Jr, "Deployment of distributed ledger and decentralized technology for transition to smart industries", *Environment Systems and Decisions*, vol. 43, no. 2, pp. 298–319, 2023.

[13] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system", May 2009. [Online]. Available: http://www.bitcoin.org/bitcoin.pdf.

[14] L. Lamport, R. Shostak, and M. Pease, "The byzantine generals problem", in *Concurrency: The Works of Leslie Lamport*. New York, NY, USA: Association for Computing Machinery, 2019, pp. 203–226, ISBN: 9781450372701. [Online]. Available: https://doi.org/10.1145/3335772.3335936.

[15] S. Ølnes, J. Ubacht, and M. Janssen, "Blockchain in government: Benefits and implications of distributed ledger technology for information sharing", *Government Information Quarterly*, vol. 34, no. 3, pp. 355–364, 2017, ISSN: 0740-624X. DOI: https://doi.org/10.1016/j.giq.2017.09.007. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0740624X17303155.

[16] T. Min and W. Cai, "Portrait of decentralized application users: An overview based on large-scale ethereum data", *CCF Transactions on Pervasive Computing and Interaction*, vol. 4, no. 2, pp. 124–141, 2022.

[17] Z. Zheng, S. Xie, H.-N. Dai, *et al.*, "An overview on smart contracts: Challenges, advances and platforms", *Future Generation Computer Systems*, vol. 105, pp. 475–491, 2020.

[18] S. Bouraga, "A taxonomy of blockchain consensus protocols: A survey and classification framework", *Expert Systems with Applications*, vol. 168, p. 114 384, 2021, ISSN: 0957-4174. DOI: https://doi.org/10.1016/j.eswa.2020.114384. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0957417420310587.

[19] W. Cai, Z. Wang, J. B. Ernst, Z. Hong, C. Feng, and V. C. Leung, "Decentralized applications: The blockchain-empowered software system", *IEEE access*, vol. 6, pp. 53 019–53 033, 2018.

[20] Y. Chen, J. Li, F. Wang, *et al.*, "Ds2pm: A data sharing privacy protection model based on blockchain and federated learning", *IEEE Internet of Things Journal*, 2021.

[21] I. M. Coelho, V. N. Coelho, R. P. Araujo, W. Yong Qiang, and B. D. Rhodes, "Challenges of pbft-inspired consensus for blockchain and enhancements over neo dbft", *Future Internet*, vol. 12, no. 8, 2020, ISSN: 1999-5903. DOI: 10.3390/fi12080129. [Online]. Available: https://www.mdpi.com/1999-5903/12/8/129.

[22] F. Leal, A. E. Chis, and H. González–Vélez, "Performance evaluation of private ethereum networks", *SN Computer Science*, vol. 1, pp. 1–17, 2020.

[23] S. Rouhani and R. Deters, "Performance analysis of ethereum transactions in private blockchain", in *2017 8th IEEE international conference on software engineering and service science (ICSESS)*, IEEE, 2017, pp. 70–74.

[24] B. Tyma, R. Dhillon, P. Sivabalan, and B. Wieder, "Understanding accountability in blockchain systems", *Accounting, Auditing & Accountability Journal*, vol. 35, no. 7, pp. 1625–1655, 2022.

[25] E. Androulaki, A. Barger, V. Bortnikov, *et al.*, "Hyperledger fabric: A distributed operating system for permissioned blockchains", in *Proceedings of the Thirteenth EuroSys Conference*, ser. EuroSys '18, Porto, Portugal: Association for Computing Machinery, 2018, ISBN: 9781450355841. DOI: 10.1145/3190508.3190538. [Online]. Available: https://doi.org/10.1145/3190508.3190538.

[26] A. S. Almasoud, F. K. Hussain, and O. K. Hussain, "Smart contracts for blockchain-based reputation systems: A systematic literature review", *Journal of Network and Computer Applications*, vol. 170, p. 102 814, 2020.

[27] Z. Zheng, S. Xie, H.-N. Dai, *et al.*, "An overview on smart contracts: Challenges, advances and platforms", *Future Generation Computer Systems*, vol. 105, pp. 475–491, 2020.

[28] V. Buterin *et al.*, "A next-generation smart contract and decentralized application platform", *white paper*, vol. 3, no. 37, pp. 2–1, 2014.

[29] I. Rõžakov, "A modest comparison of blockchain consensus algorithms", B.S. thesis, University of Twente, 2019.

[30] E. Team, *Exonum Documentation — exonum.com*, https://exonum.com/doc/version/0.12/, [Accessed 23-02-2024].

[31] K. Hoffman, D. Zage, and C. Nita-Rotaru, "A survey of attack and defense techniques for reputation systems", *ACM Comput. Surv.*, vol. 42, no. 1, Dec. 2009, ISSN: 0360-0300. DOI: 10.1145/1592451.1592452. [Online]. Available: https://doi.org/10.1145/1592451.1592452.

[32] S. Marti and H. Garcia-Molina, "Taxonomy of trust: Categorizing p2p reputation systems", *Computer Networks*, vol. 50, no. 4, pp. 472–484, 2006.

[33] F. G. Mármol and G. M. Pérez, "Security threats scenarios in trust and reputation models for distributed systems", *computers & security*, vol. 28, no. 7, pp. 545–556, 2009.

[34] H. Kasyap and S. Tripathy, "Privacy-preserving and byzantine-robust federated learning framework using permissioned blockchain", *Expert Systems with Applications*, vol. 238, p. 122 210, 2024, ISSN: 0957-4174. DOI: https://doi.org/10.1016/j.eswa.2023.122210. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0957417423027124.

[35] J. Castillo, P. Rieger, H. Fereidooni, Q. Chen, and A. Sadeghi, "Fledge: Ledger-based federated learning resilient to inference and backdoor attacks", in *Proceedings of the 39th Annual Computer Security Applications Conference*, 2023, pp. 647–661.

[36] H. Kasyap, A. Manna, and S. Tripathy, "An efficient blockchain assisted reputation aware decentralized federated learning framework", *IEEE Transactions on Network and Service Management*, 2022.

[37] Y. Qu, S. R. Pokhrel, S. Garg, L. Gao, and Y. Xiang, "A blockchained federated learning framework for cognitive computing in industry 4.0 networks", *IEEE Transactions on Industrial Informatics*, vol. 17, no. 4, pp. 2964–2973, 2020.

[38] Y. Zhao, J. Zhao, L. Jiang, R. Tan, and D. Niyato, "Mobile edge computing, blockchain and reputation-based crowdsourcing iot federated learning: A secure, decentralized and privacy-preserving system", *arXiv preprint arXiv:1906.10893*, pp. 2327–4662, 2019.

[39] U. Majeed and C. S. Hong, "Flchain: Federated learning via mec-enabled blockchain network. in 2019 20th asia-pacific network operations and management symposium (apnoms)", *IEEE, 1ś4*, 2019.

[40] D. Cao, S. Chang, Z. Lin, G. Liu, and D. Sun, "Understanding distributed poisoning attack in federated learning", in *2019 IEEE 25th International Conference on Parallel and Distributed Systems (ICPADS)*, IEEE, 2019, pp. 233–239.

[41] D. Unal, M. Hammoudeh, M. A. Khan, A. Abuarqoub, G. Epiphaniou, and R. Hamila, "Integration of federated machine learning and blockchain for the provision of secure big data analytics for internet of things", *Computers & Security*, vol. 109, p. 102 393, 2021, ISSN: 0167-4048. DOI: https://doi.org/10.1016/j.cose.2021.102393. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167404821002170.

[42] R. Xu and Y. Chen, "μDfl: A secure microchained decentralized federated learning fabric atop iot networks", *IEEE Transactions on Network and Service Management*, vol. 19, no. 3, pp. 2677–2688, 2022.

[43] Y. Abuzied, M. Ghanem, F. Dawoud, *et al.*, "A privacy-preserving federated learning framework for blockchain networks", 2023.

[44] go-ethereum Authors. "Go-ethereum". (), [Online]. Available: https://geth.ethereum.org/ (visited on 06/19/2024).

[45] flask Authors. "Flask". (), [Online]. Available: https://flask.palletsprojects.com/en/3.0.x/ (visited on 06/19/2024).

Abbreviations

ML	Machine Learning
SGD	Stochastic Gradient Descent
FL	Federated Learning
CFL	Centralized Federated Learning
DFL	Distributed Federated Learning
DLT	Distributed Ledger Technology
BGP	Byzantine Generals' Problem
P2P	Peer-to-Peer
BTC	Bitcoin
PoW	Proof-of-Work
PoS	Poof-of-Stake
PoA	Proof-of-Authority
DPoS	Delegated-proof-of-Stake
DoS	Denial-of-Service
MAE	Mean Absolute Error
IPFS	Inter Planetary File System
ETH	Ether
TTP	Trusted-Third-Party
DC	Decoupled
CD	Coupled
SC	Semi-Coupled
Blockchain	BC
Arch.	Architecture
UI	User-Interface
API	Application Programming Interface
RPC	Remote Procedure Call
HTML	Hypertext Markup Language
CSS	Cascade Style Sheet
Geth	Go Ethereum
UML	Unified Modeling Language
OOP	Object Oriented Programming
CPU	Central Processing Unit
GPU	Graphics Processing Unit

List of Figures

List of Tables