StateFL: State Channels Powered Federated Learning

Master of Science Thesis in Computer Science

Distributed Systems Group
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology
Van Mourik Broekmanweg 6, 2628 XE Delft, The Netherlands

Ioana Paula Iacoban

May 2024

Author
Ioana Paula Iacoban (I.P.Iacoban-1@student.tudelft.nl)
Thesis Advisor
dr. Zekeriya Erkin (Z.Erkin@tudelft.nl)
Daily Supervisor
prof. dr. Stefanie Roos (S.Roos@tudelft.nl)
Title
StateFL: State Channels Powered Federated Learning
MSc Presentation Date
May 30, 2024

Graduation Committee

dr. Zekeriya Erkin	Delft University of Technology
prof. dr. Stefanie Roos	Delft University of Technology
dr. Jérémie Decouchant	Delft University of Technology

Abstract—Federated Learning (FL) is a decentralized machine learning approach that provides a privacy-friendly way of training models by keeping the datasets of participating parties private. Some challenges FL faces are the lack of incentives to encourage participation in the learning process, as well as preventing potential cyber attacks that tamper with the model. Blockchain is an available solution that provides the means to implement incentives to encourage participation and issue penalties to disincentivize malicious behavior. Hence, recent developments introduced blockchain-enabled FL (BCFL) designs for various applications. However, one obstacle that slows down the widespread adoption of this technology is the high latency of blockchain networks due to its laborious consensus protocols. In this paper, we propose StateFL, a revised BCFL architecture that uses state channels (a blockchain scaling solution) in order to ease the load on the blockchain by reducing the number of on-chain transactions, improving the system's latency, and minimizing transaction fees as a result. State channels are governed by smart contracts and enable two parties to exchange information and assets off-chain unless disputes occur. Either channel party can dispute the state of the channel if suspicious behavior is observed. In that case, the dispute is settled on the blockchain. We evaluate StateFL in a series of experiments to establish latency improvements, identify bottlenecks, and quantify the impact of disputes on channel latency and transaction fees. The findings show that the higher the number of FL rounds, the more StateFL outperforms the baseline BCFL, with the exception of a very low number of rounds. In realistic FL scenarios, the rounds are in the order of hundreds making StateFL a solid contender even if disputes do occasionally occur. The bottleneck of StateFL is the channel setup and closure which require extensive interaction with the blockchain.

1 Introduction

Machine Learning (ML) is an empowering mechanism that drives the automation of a variety of tasks, becoming an omnipresent technology in every branch of the industry [1]. Some of the shortcomings of traditional centralized ML is that training requires enormous amounts of high-quality data and the process does not provide data privacy [2]. To obtain an adequate amount of data to perform ML, sourcing from different locations or parties is likely necessary, while preserving the privacy of data remains an open issue [3]. To mitigate the data privacy issue, cryptographic methods such as Differential Privacy [4], Homomorphic Cryptography [5], and Secure Multi-Party Computation [6] have been integrated with ML. The main challenges that come with using cryptography in ML include compromising model accuracy in favor of privacy, limitations in the selection of models that are compatible with these methods, and potential privacy leakages [2]. Hence, the need for more robust and versatile privacy-preserving ML motivated the development of a new approach to perform ML, Federated Learning (FL), introduced by Google researchers [7].

FL is a technique that enables training ML models while keeping raw user data on the user's device to preserve the privacy of the data. This is achieved by performing the ML training locally and then sharing the resulting model that is then aggregated with the results from other FL participating users into a global model [7]. While this approach preserves the privacy of the raw user data, it also introduces new obstacles. Firstly, FL comes with the cost of local computations and the use of user resources. As a result, participation in the FL training process might still not be attractive to users, even though they would benefit from the improved model. Secondly, FL lacks a defense mechanism that prevents malicious user behavior that aims to disrupt the system. As a result, recent developments have been made that combine FL with blockchain [8] such that participation in the process can be incentivized to not only encourage contributions but also enforce honest use of the system. This is achieved by rewarding participants for contributions or penalizing them for detected malicious behavior using cryptocurrency tokens and game theory [9, 10].

Blockchained FL (BCFL) has been integrated into various systems [11, 12, 13] with a wide range of applications, in order to facilitate privacy of data (with FL) and incentives (with blockchain by means of peer verifications, incentives, and penalties). However, blockchain, especially when deployed with proof-of-work consensus (PoW), can become a performance bottleneck and also requires significant computation and power resources. As a result, the transaction processing times and blockchain throughput are significantly higher (ten transactions per second), when compared to custodial payment systems, such as Visa, which can process thousands of transactions per second [14]. Alternative consensus algorithms, such as proof-of-stake (PoS) [15] and proof-of-federated-learning (PoFL) [16], have been proposed to improve the transaction processing rate and latency. However, the consensus remains a bottleneck that halted the widespread adoption of blockchain [17]. Thus, to overcome this challenge, payment or state channel networks running on top of blockchain as a layer-two protocol, are expected to greatly improve the latency, resource utilization, scalability, and flexibility of these systems [18]. Payment channels and state channels allow two parties to create a temporary private channel to exchange transactions without broadcasting them to the blockchain [17]. Only the final balances of the parties and channel state are broadcast to the

network as a transaction when the channel is closed. Hence, this approach eases the load on the blockchain network and lowers the fees because it heavily reduces the number of transactions that need to be verified.

We propose StateFL, a revised BCFL architecture that employs a layer-two protocol on top of the blockchain to improve the latency of the BCFL system by reducing the interactions with the blockchain. In StateFL, state channels are established between the FL server and each FL client, and information about the model updates and aggregation is exchanged using the channel. Thus, there is no need to publish intermediate updates on the blockchain after each FL round, but only when the FL process is complete and no disputes occur between the server and the clients. Additionally, clients are rewarded with cryptocurrency for their contributions after each round of FL.

The main objective of this study is to determine whether StateFL outperforms BCFL. To achieve this, we designed a series of experiments to address the following research questions (RQs):

- RQ 1: Is the latency of StateFL lower than a baseline BCFL system?
- RQ 2: What bottlenecks exist in the BCFL and StateFL systems?
- RQ 3: What is the impact of disputes on channel latency?
- RQ 4: What is the impact of disputes on transaction fees?

In summary, the contributions of this paper are:

- Proposing and implementing StateFL, a new system design that incorporates state channels into the BCFL architecture to improve system performance and scalability.
- Evaluating StateFL by means of experiments measuring the latency of the FL process and its steps, as well as the time and fees required to set up and settle channels.
- Analysing and comparing the bottlenecks and limitations in StateFL, as well as the equivalent BCFL system.

The rest of the paper is structured as follows: Section 2 introduces the required background and related work, Section 3 explains our system model and architecture, Section 4 details our methodology and experimental setup, Section 5 reports our experiment findings, and Section 6 concludes the paper and presents future work.

2 Background and Related Work

In this section, we introduce the fundamentals of blockchain and federated learning. Afterward, we highlight some of the challenges of federated learning and how blockchain aids in mitigating them in BCFL systems. Further, we explore the limitations of blockchain and identify potential solutions that also benefit BCFL frameworks.

2.1 Introduction to Blockchain

A blockchain is a system composed of a peer-to-peer network where the participants store and maintain a ledger using a consensus algorithm to settle on the current state of the ledger in a decentralized manner [19]. The ledger takes the form of a data structure constructed as a series of blocks linked by a cryptographic hash, each block containing a series of transactions, though not necessarily financial transfers, but also smart contracts [20].

Smart contracts greatly extend the versatility and applications of blockchain technology since various algorithms can be embedded into smart contracts and deployed on the blockchain [21]. As smart contracts allow the autonomous execution of arbitrary code, decentralized programmable financial systems can be implemented [22]. Hence, business models such as equity crowdfunding, peer-to-peer lending, and online insurance can run directly on the blockchain without needing a central authority [23]. This is achieved by translating each party's underlying rules and obligations for each scenario into a digital contract (smart contract) that is then added to a block in the blockchain and later executed and enforced.

There are two main types of blockchain systems: permissionless and permissioned [20]. Permissionless or public blockchains consist of a fully decentralized network where any party is free to join or leave the network and can read or write blocks to the ledger. On the other hand, in permissioned blockchains, there is a central authority that manages the access and the rights of each peer in the network. Permissioned blockchains can be further classified into private and consortium blockchains, where the central authority is composed of a single entity or multiple parties, respectively [8].

To agree on the current state of the ledger, decentralized blockchain systems implement consensus protocols. Some of the key consensus protocols used in blockchains are proof-of-work (PoW), proof-of-stake (PoS), and practical Byzantine fault tolerance (PBFT), as well as variations of them, such as PBFT (dBFT) and the delegated versions of PoW (DPoW), PoS (DPoS), where the participants can vote on which miners get to verify the next block [21]. A widely adopted consensus algorithm used in blockchains is PoW [21], in which nodes (miners) compete to propose blocks on the chain by attempting to solve difficult cryptographic hash puzzles. As a result, PoW requires lots of computations and power to bookkeep the ledger, most of which are wasteful [21]. In PoS, on the other hand, verifiers stake a number of assets on the respective chain as collateral, such that, if tampering is attempted and discovered, the malicious verifier would lose their collateral. When

compared to PoW, PoS significantly improves computational resource use; however, its main flaw is that the verifiers with the higher stakes are favored in the selection process. DPoS and DPoW additionally include a voting mechanism to elect the parties, verifying blocks to make the process more democratic. Delegation can also be used to improve efficiency and resource allocation (bandwidth, power, etc.), as employed by [24] and [25], both being implemented with DPoS. In contrast to PoW and PoS, PBFT reaches consensus more rapidly and effectively, although it requires more than a third of the nodes to be trusted [26].

[8] and [20] presented the key properties of the blockchain technology:

- Decentralization and public verifiability: the blockchain runs as a peer-to-peer network where no central authority is in charge of the system, and the network participants verify the state of the ledger using consensus mechanisms.
- Transparency and traceability: updating the state of the ledger is a transparent process required for public verifiability, while the data on the blockchain is also traceable to its source.
- Privacy: can be achieved; however, there is a trade-off to be made between the degree of transparency and the degree of privacy of a blockchain.
- Anonymity: this is achieved through encrypting private information.
- Integrity and immutability: on-chain data is difficult to alter or tamper with due to public verifiability.

From an architectural standpoint, blockchain can be viewed as a layered architecture with four layers, namely: the hardware layer, layer-zero, layer-one, and layer-two [14]. The hardware layer corresponds to the hardware environment in which the software is running. For security reasons, blockchains may run in a Trusted Execution Environment (TEE). The TEE is a secure processing area that ensures confidentiality and integrity by separating the software running in the TEE from the operating system to avoid modifications from other (privileged) software processes [14]. Layer-zero, also called the network layer, is responsible for the connections between the nodes of the blockchain network that follow the peer-to-peer architecture, and it encompasses the whole network stack as opposed to only the networking protocols. The layer-one is the one responsible for hosting the blockchain and ensuring its integrity by implementing the consensus protocol. Finally, layer-two allows for off-chain functionality running on top of the blockchain and provides the opportunity to more easily scale blockchain systems with channels, commit chains, and protocols for referred delegation. Layer-two relies on layer-one for the integrity of the blockchain and assumes that transactions are added to the ledger within a time-bound [14].

2.2 Introduction to Federated Learning

Federated learning is a technique used to build and train machine learning models in a distributed fashion across multiple devices that contribute to the same ML task, without exchanging training data between them. This method enhances privacy and prevents data leakage [7]. Traditionally, there is a central federator or aggregator that is in charge of orchestrating the process by constantly updating and sharing the new model with the participants, such that there is no need for the end-devices to communicate with each other.

[3] identifies three FL categories according to the structure of the end devices' training sets, with respect to the data samples and features: Horizontal Federated Learning (sample-based FL), Vertical Federated Learning (feature-based FL), and Transfer Learning. Horizontal Federated Learning applies when the participants use training data with the same feature space and varying samples, while in the case of Vertical Federated Learning, the set of training samples is the same across devices, but the feature space differs. Finally, in Transfer Learning, neither the samples nor the features coincide between participants.

The workflow of FL consists of the following steps [3, 8, 21, 27], extendable with an additional one if an incentive mechanism is also available [10]:

1. Initialization and participant selection: a selection of end devices is created by the federator, and an initialized global model is distributed across them. The selection can be random or based on the devices' capabilities and available resources in terms of available battery power and the quality of the network connection.
2. Local training: each participant trains the received model with their local dataset and sends back parameter updates to the federator.
3. Aggregation and model update: the federator aggregates the parameter updates from the participants and updates the global model that is to be redistributed for the next iteration of training. FedAvg [28] is one of the potential aggregation algorithms and works by simply having the federator average the received parameter values from the clients into the new updated model.
4. Reward allocation: based on individual contributions, the participants receive rewards from the federator. Although this step is optional, it encourages contributions and can help prevent malicious behavior from the participants [10].

To be resilient and robust, FL systems account for the clients' performance and resources in the selection process [7]. Selecting clients with a stable network connection, sufficient computational resources, and enough power (or remaining battery) to complete the

training are needed for robustness and timely convergence [10]. Furthermore, evaluating the submissions of each client gives insight into the extent to which they are contributing to the model. Therefore, factoring the contribution into the selection process also makes the FL more efficient. Additionally, if rewards are allocated, the high-performing clients are incentivized to continue to contribute, while enforcing penalties prevents adversarial attacks aiming at poisoning the model.

The effectiveness of the incentive mechanism, in terms of participant selection, is based on game theory concepts [10]. One type of model that suits FL setup quite well is the Shapley Value [29, 30] in a cooperative game [10] where the learning process is abstracted to a game. In this case, the FL clients form a coalition working cooperatively towards training and obtaining a model that would benefit them all. Four main axioms govern the Shapley Value [31]:

1. Efficiency: The total amount of rewards earned by the coalition are redistributed among all participants.
2. Null player: Noncontributing participants receive no rewards.
3. Symmetry: If two players contribute equally to all subsets of a coalition, they are rewarded equally.
4. Additivity: The Shapley values of two different games can be added together to represent a new combined game where reward distribution remains possible.

Thus, The Shapley Value makes the game centered on fairness towards the participants based on the contributions they are making towards training the model (common goal), and it is highly applicable to the FL technology. As a result, it has been integrated into some FL frameworks in synergy with blockchains [9, 10, 32].

2.3 Challenges of Federated Learning and a Blockchain Mitigation

Challenges and Limitations of FL

The main challenges FL faces are datasets that are non-independent and non-identically Distributed (non-IID), high communication costs, and vulnerability to adversarial attacks [33]. Furthermore, in traditional FL, the learning is centered around the federator, making it a single point of failure that takes a toll on the system's resilience.

In the context of centralized ML, the dataset is preprocessed, enhanced, and tailored to the model being trained. Thus, assuming IID data in this case is a sensible assumption [34]. However, in FL, the preprocessing of data is not possible since it is stored at different locations on the edge of the network. Hence, the IID data assumption is no longer valid. Instead, the training data is heterogeneous and non-IID. This introduces a degree of incompatibility between the data and the ML model since lots of models are built on statistics that assume IID data. Furthermore, it makes training and validating the models more difficult due to the differences in data distributions between clients that lead to slow convergence and high bandwidth consumption [35], as well as an overall skewed dataset resulting in accuracy loss [36].

Due to the continuous interaction between the federator and each client, the network traffic can be quite demanding, which can lead to bottlenecks in the system [33]. Next to that, in some scenarios, such as the IoT, the clients have limited capabilities or unstable connections, and that has significant effects on the FL process [37, 38]. Therefore, recent FL research has focused on addressing these types of issues, as well as improving energy use. Zheng et al. [38] tackle the network unreliability problem in highly dynamic and mobile networks. The authors also highlight limitations in offloading tasks to the network edge due to the limited available bandwidth. The proposed solution involves transfer learning as the learning approach, which reduces training and communication costs while increasing convergence time and accuracy. Additionally, Stackleberg games are in place as a mechanism for resource optimization [38]. Another approach to save resources that received recent developments is intelligently putting devices to sleep whenever possible [39]. To address the common wireless communication bottleneck, Mhanna and Assaad [40] propose a modified gradient descent algorithm to ease the load on the communication channel.

While FL strengthens client data privacy and can achieve efficient performance by parallelizing the learning task across multiple devices, FL still faces some challenges in its traditional setup. First of all, the trust and transparency of the federator are not guaranteed [21]. Second of all, the model is also vulnerable to attacks and malicious behavior from the clients. As characterized in [10], attacks on FL can include Target Poisoning Attacks, Untargeted Poisoning Attacks, and Free-Rider Attacks. Targeted Poisoning Attacks consist of submitting a backdoor task in the model without damaging the accuracy, while Untargeted Poisoning Attacks mostly aim to deteriorate model performance and cause a high misclassification rate. Finally, Free-Rider Attacks simply use the model without contributing to the training process, submitting fake updates that do not significantly affect the model's accuracy. To prevent such attacks from occurring, the federator could implement defense mechanisms to audit and remove suspicious participants from the process [10].

Moreover, clients can also suffer as a result of data leakage, which occurs when attackers can reconstruct the dataset to some degree from the published parameter updates, hence revealing information about the

clients [33, 41]. According to [33], mitigating leakage can be achieved through compression, by reducing the information made available, encryption, or perturbation, by adding random noise to sensitive information, using techniques such as differential privacy. Each method comes with advantages and disadvantages: compression reduces the communication cost, but degrades the model; encryption guarantees security, but it is costly in terms of communication and computation; perturbation guarantees the data privacy, but impacts the model accuracy. Differential privacy provides a balanced trade-off between efficiency, accuracy, and data privacy [4]. Therefore, it has been widely integrated into FL frameworks [41].

Blockchain Mitigation

Despite traditional FL's novel privacy-preserving advantages, the method is still facing challenges with regard to trust between parties, malicious behavior, and system reliability [3]. Wan and Hu [8] highlight three open issues in the traditional FL framework. The first problem is that the whole process is dependent on the aggregator, who should be online and available throughout the process, which introduces a single point of failure in the system. Second, the lack of incentives makes it challenging to encourage (honest) participation from the clients. To this end, blockchain technology can eliminate or at least alleviate some of the shortcomings of traditional FL [8]. By combining blockchain with FL in a BCFL framework, the single point of failure can be eliminated, leaving the aggregation to be performed by multiple clients. Additionally, the updates can also be pushed to the blockchain and verified in the mining process to make sure the model will contain valid data, while the participants also obtain incentives for training the model and verifying the updates, encouraging (honest) participation [8]. Blockchain can also enable sophisticated approaches to evaluate participants' performance and protect against poisoning attacks [42, 43].

Wan and Hu also formally classified the BCFL architectures in [8], according to the extent to which the blockchain network is coupled with the FL network. The three proposed categories are fully coupled BCFL (FuC-BCFL), flexibly coupled BCFL (FlC-BCFL), and loosely coupled BCFL (LoC-BCFL). More recent studies also propose hybrid blockchain networks, i.e., combining private and public blockchains to increase trust while reducing computational load [44].

In fully coupled BCFLs, the FL clients and blockchain nodes coincide, i.e., the clients train the FL model, push updates to the blockchain, and also verify them and maintain the blockchain, although not all nodes need to participate in a verification or aggregation process. As soon as the local model updates are verified, they are added to a block and stored on the blockchain. If an incentive mechanism is implemented, rewards are distributed. Thus, without a central federator, the single point of failure scenario is avoided, and communication costs are reduced while privacy leakage is prevented. However, the limited network bandwidth can affect latency, and heavy computational resources are also required to perform the FL and administer the blockchain [8].

Flexibly coupled BCFL, on the other hand, keeps the FL and the blockchain as separate networks. The FL clients train the model with local data and submit updates to the blockchain, while dedicated miners verify them and can also aggregate the model and distribute the incentives accordingly. By keeping the networks independent, there is less traffic pressure than in fully coupled BCFL; however, coordination between networks could be challenging [8].

Similar to flexibly coupled BCFL, loosely coupled BCFL relies on stand-alone blockchain and FL networks where the FL clients send model updates to the miners for verification. However, the aggregation is performed by a central server that retrieves the verified updates from the blockchain, performs the aggregation, and shares the new model with the clients. Additionally, the miners will calculate a reputation score for each FL client based on the updates they submit such that rewards or penalties may be applied [8].

BCFL frameworks have been proposed to be integrated into various IoT systems and telecommunication networks to increase their autonomy and robustness [45]. [46] highlight the potential BCFL has to enable intelligent transport systems to train ML models in a trustless environment to improve the efficiency of transport infrastructure, safety, facilities, and even energy use. Furthermore, the applicability can be extended to training autonomous car models, as indicated by [47]. Additionally, [25] aims to improve the latency and reliability of 5G networks with BCFL.

Recent research on the topic of BCFL has brought various approaches to achieve more practical and flexible BCFL environments to facilitate reliable FL. Liu et al. propose the FedCoin framework [9], a BCFL framework that implements Shapley Values to distribute incentives. The framework separates the FL and blockchain into two networks, as in the LoC-BCFL model. In terms of workflow, an FL model requester proposes a task to the FL network. Then, a centralized FL server coordinates the model training and aggregates the updates from the FL clients. Afterward, the FL server posts a special task to the blockchain, containing the set of all local updates, the aggregation function, and the model loss function, such that the blockchain nodes can compute each client's contribution to the model. The consensus is achieved using a custom protocol, proof of Shapley (PoSap). Finally, the incentive is distributed among the clients, according to the established

Shapley Value (SV). According to the payment scheme defined by FedCoin, at the start of the process, the FL model requester deposits V FedCoins in the FL server in exchange for a trained model. The V coins are divided among the FL clients, FL server, and blockchain miners as follows: TrainPrice (training payments to the FL clients), ComPrice (aggregation computation price to the FL server), and SapPrice (payments to the miner for computing the SV). Moreover, the exact division of V into the TrainPrice, ComPrice, and SapPrice amounts is established by a pre-agreed smart contract. The authors evaluated FedCoin to establish whether the scheme can promote training the model with high-quality data and whether PoSap consensus is computationally feasible. The framework was evaluated through experiments that trained models on the MNIST dataset with FedAvg aggregations using the TensorFlow framework, while the blockchain network was simulated in a Docker environment. Finally, the hypothesis is confirmed, and the authors show that PoSap can reach consensus within an upper bound in terms of computational resources, such that clients are incentivized to use high-quality data in the training process.

Cheng et al. [42] propose a similar framework, PoShapley-BCFL, that aims to achieve fairness and robustness for FL. The authors formalize a custom consensus algorithm, Proof of Shapley-Value, which is conceptually akin to PoW as the miner computes a Monte-Carlo-sampling enabled lightweight SV until agreement is achieved. Furthermore, client selection and aggregation are performed by employing a smart contract. The clients are selected based on their SV, and, in the aggregation process, a weight is assigned to the client's update according to the ratios of their SVs, i.e., a weighted aggregation algorithm. Thus, the lower quality updates are differentiated to better improve the next iteration of the model and protect against attacks [42].

Dias and Meratnia highlight, in [48], the single point of failure issue of FL, as well as the potential blockchain has in mitigating it. Blockchain can eliminate the need for a central aggregator and make the aggregation process more transparent for the FL clients, as parameter updates and aggregations are stored in transactions on the ledger. Hence, the authors propose, implement, and evaluate a BCFL framework, BlockLearning: A Modular Framework for Blockchain-Based Vertical Federated Learning. The framework is designed to be modular and to easily accommodate changes in the FL and blockchain algorithms and protocols. BlockLearning defines three roles within the system, namely, trainers, aggregators, and scorers. Furthermore, the roles can be assigned with flexibility among the participants, and the assignment of multiple roles per node is also possible. Thus, the framework supports all three types of architectures defined in [8]: FuC-BCFL, FlC-BCFL, and LoC-BCFL. Additionally, there is a model owner that deploys a smart contract to initiate the learning process. Then, the participants are selected, and the trainers begin training and sharing parameter updates on the ledger while aggregators compute and share the aggregation results. The scorers give each update a score based on a predefined metric to evaluate individual contributions; however, this step is optional. Lastly, the model owner pushes a transaction on the blockchain that terminates the learning round if consensus on the final state of the model is achieved. Otherwise, the learning round fails. BlockLearning is implemented for the Ethereum blockchain, and the smart contracts are written in Solidity due to their popularity and extensive technical support. Moreover, the FL component is written in Python, as it offers plenty of widely used ML libraries. The framework is evaluated by means of simulations run in a Docker environment. The authors trained a Split-CNN model on the MNIST dataset and evaluated the framework's performance in terms of execution time, transaction cost, transaction latency, model accuracy, convergence, communication, and computation costs. A similar study by Goh et al. [49] proposes and practically develops a BCFL framework where the FL is governed by Ethereum smart contracts. There is no central FL server orchestrating the learning process, but the smart contracts. There are three main roles in the process: the trainers, the evaluators, and the aggregators. The trainers download the initial model and train it with their local data; then, they reference the model update in the smart contract. Next, the evaluators download the submitted model update, evaluate it, and enter a score for each trainer in the smart contract. This is done to potentially exclude low-performing updates or to provide incentives for high-performing updates. Finally, the aggregators update the global model according to the new updates and record a reference to the model in the smart contract. All models and parameter updates are uploaded to the IPFS (InterPlanetary File System) distributed file storage, and they are referenced by their content identifier in the smart contract. Therefore, all parties involved can access the data. Goh et al. [49] mainly focused on verifying the framework for functionality, proposed a performance evaluation for future work, and suggested an off-chain approach as a potential improvement.

2.4 Challenges of Blockchain and an Off-Chain Mitigation

Challenges and Limitations of Blockchain

While the blockchain has the potential to mitigate some of the FL's challenges, it introduces other technical challenges of its own. On the one hand, the Shapley Value in a Cooperative Game can be implemented in FL with the help of blockchain, which can prevent attacks on the FL model and FL clients. On the other hand, the blockchain's security is vulnerable to other types of attacks [50] such as the 51% attack (Sybil attack) [51],

mining pool attack [52], selfish mining [53], forking, and double spending [54] and [55] to name a few. Thus, it is necessary to implement appropriate defenses to have a secure system. Additionally, state-of-the-art research aims to find better existing defenses against these attacks. Chen et al. provide a detailed overview of blockchain attacks and defenses in [56].

Furthermore, blockchain consumes a lot of energy to maintain the ledger, especially when consensus is achieved with PoW [50]. Power consumption has been an obstacle in the adoption of blockchain technology to other fields, but also in scaling the network to a large number. Due to how PoW consensus works, 51% of the network's miners need to agree on the state of the ledger for the mining to continue. Because of this, consensus in large networks takes longer to resolve, making the blockchain slow and unable to scale, and more and more energy is also required. Sharding, side chains, and other consensus protocols such as PoS and PBFT alleviate these issues. However, they are still far behind custodian payment systems in terms of their transaction execution rate [14]. Moreover, the consensus protocol cannot be easily changed after a blockchain network has been deployed and compatibility issues are almost inevitable [14]. As noted previously, FL systems might consist of devices with limited available resources in terms of computation and available battery; hence, considering the power consumption of the blockchain is crucial for the efficiency and usability of the system.

Layer-2 Mitigation

Blockchain scaling solutions have been widely researched, and several options have been proposed and are taking shape and becoming available to developers. However, there is no silver bullet solution to scaling, since different security assumptions are in place between solutions. The essence of the layer-two is to reduce the load put on the blockchain and on the miners, which results in a more efficient use of energy, improved latency, and fewer fees by only putting only a summary of the executed transaction on the main ledger [57]. Scaling at layer-one can be done through sharding (splitting the blockchain into smaller blockchains) and using alternative consensus mechanisms that are more efficient, similar to the Ethereum 2.0 scalability update for Ethereum [57]. At the layer-two, channels, side chains, cross chains, and hybrid solutions can be implemented and ran alongside the main blockchain, as presented in the taxonomy of blockchain scaling methods in [58].

Side chains are independent blockchains that are anchored on the main ledger, allow for parallelizing transaction executions, and reduce the load of the main chain by validating transactions. The main feature of side chains is the ability to transfer assets from the main chain to the side chain by means of a two-way peg, such that more exchanges (transactions) can take place. One of the pitfalls of side chains is that they can run into the same scalability issue as the main chain [57], and it adds more complexity, as well as potentially more attack vectors to the system [14]. Gangwal et al. [58] identify two types of side chains: custodial and non-custodial. In the custodial side chain, funds are directly transferred to the side chain, and it executes its chosen consensus protocol under a set of security and trust assumptions, while non-custodial side chains secure funds on the main chain via smart contracts. Non-custodial side chains can be further distinguished into commit chains and rollups [58].

Commit chains function similarly to channels, specifically payment channel hubs. Furthermore, commit chains are designed to tackle some of the limitations in channels. In contrast to payment channels, an (untrusted) operator manages the transactions between parties and periodically sends a commitment to the main chain. Peers need to be online regularly to check the main chain checkpoint commitments. Moreover, commit chains run continuously, as opposed to the three-state model of channels: establishment, transitions, and dispute or closure [14]. Additionally, the operator is governed by a smart contract such that it cannot act maliciously, while the peers lock funds on the main chain via the smart contract [58].

Rollups also rely on smart contracts to maintain a Merkle Tree (Hash Tree) that bundles multiple transactions together. Transactions are executed in batches and maintained off-chain in the Merkle Tree to significantly improve storage and latency, while only the root of the Merkle Tree is updated on-chain via the smart contract [58]. When it comes to the transaction validation process, two types of rollups can be distinguished: optimistic rollups and zk rollups. Optimistic rollups do not verify transactions unless there is a challenge is requested by submitting a proof to the chain showing that fraudulent transactions exist. Afterward, the contract reverts the transaction batch and the subsequent batches in question. In contrast, zk rollups verify every batch of transactions with zero knowledge (zk) proofs constructed with zk-SNARKs [58].

Cross chains act as bridges between blockchains that allow transferring assets from one chain to another, for instance, from Bitcoin to Ethereum. Hence, the cross-chain mediates inter-blockchain transactions and helps establish mutual trust between users that hold assets on different blockchains [58].

The taxonomy in [58] identifies two hybrid solutions that change the underlying properties of off-chain solutions to improve scalability further. Firstly, bisection protocols aim to address dispute resolution happening on-chain and allow it to be outsourced off-chain in order to reduce the load on the main blockchain further [58]. Secondly, Trusted Execution

Environments (TEE) [59] allow the blockchain software to run on physically isolated hardware from the rest of the operating system, improving security by guaranteeing integrity and confidentiality. Next to that, it adds a layer of trust such that participants are no longer required to lock collateral on the blockchain, and the fees to the miner are ensured [14]. Additionally, TEE can be used with any blockchain system, and it can also parallelize disputes [14].

Channels are bidirectional connections between two peers that allow interactions between the two parties while conforming to a set of rules. Channels can come in the form of payment channels [60] or state channels [61], where the rules of payment channels support the exchange of assets between two parties, while state channels generalize asset exchanges to arbitrary interactions with smart contracts for extended functionality [14]. Both payment channels and state channels have the same lifecycle consisting of three stages: channel establishment, state transition, and closure or dispute (in case the parties disagree on the state of the channel). When establishing a channel, the parties involved lock a certain amount of collateral on the blockchain, which can be then used to transfer funds off-chain over the channel. During state transitions, any of the parties propose a channel update (which can entail an asset exchange), and then the other party signs it with their key to indicate agreement. A dispute is raised and resolved on the layer-one blockchain if a party chooses not to sign. A state transition or state replacement in the channel environment is equivalent to a transaction [58]. Finally, when there is no further use for a channel, the parties may close it, and a transaction is sent to the blockchain such that the ledger is in concordance with the final state of the channel [14].

The main techniques used to replace channel states are Replace-by-Incentive (RbI), Replace-by-Timelock (RbT), Replace-by-Revocation (RbR), and Replace-by-Version (RbV) [14, 58]. In the case of RbI, the sender of a transaction announces a new state while also providing an incentive for the other parties that sign, and the higher the incentive, the more likely it is that the proposed state is accepted [58]. RbT works by assigning each state a time lock that is decremented with every change, and the state with the lowest lock is accepted as the new one [14]. RbR provides the means to revoke a state that has been put on the blockchain; however, for the revocation to take effect, the parties need to agree on the new state within a time frame defined by the main blockchain [58]. When RbV is used, a version number associated with each state is incremented at every change. Hence, the higher the version number, the more recent the state, and the most recent state is agreed upon [14, 58].

Furthermore, Payment Channel Networks (PCN) [62] and State Channel Networks [61] can facilitate payments and smart contracts between parties that are not directly connected, but they both can be reached via an intermediary node. The channel networks increase the connectivity between parties and can be used to lower the overall cost of setting up channels [14]. For brevity, the technical details of channel networks are not elaborated upon since they are not a key element of the research presented in this paper.

To sum up, layer-two is designed to be orthogonal to layer-one, to be compatible with any blockchain [14]. Thus, when scaling becomes a problem in an existing system where changing the fundamental elements of the blockchain is not possible, layer-two scaling is available without significant system changes.

In the context of BCFL, a blockchain scaling solution is expected to significantly improve the system's latency, allowing for a much more efficient training process. Upon comparing the off-chain solutions synthesized in [58], channels are the most fitting since they provide high-speed transaction execution at very low fees, without cumbersome temporal requirements, and offer support for smart contracts, which is highly valuable for future system expandability.

3 StateFL: System Overview

In this section, we define our system model, underlying assumptions, and network structure, followed by the architecture of StateFL.

3.1 System Model

The interconnection structure of the network follows a star shape, with a central federator serving as an orchestrator for the learning process. This design provides efficient routing and coordination among the network participants. Network links utilize TCP for communication, which guarantees several key properties. Firstly, messages sent are ensured to be received without loss. Secondly, received messages are guaranteed to be correct, without any damage. Thirdly, the FIFO property ensures that messages sent along a single connection are received in the order they were sent. Although TCP connections may break, such incidents are detectable. However, whether the other party is synchronized is uncertain, as acknowledgments are not received.

Moreover, finite and bounded delays characterize the network's message transmission. Messages sent along a link, assuming they are not lost, are received within a finite amount of time. Additionally, there exists a known upper bound on the delay experienced by messages, ensuring bounded delays even when messages are not lost.

Synchronous communication is fundamental to the network's operations and StateFL assumes synchrony

in communication rounds. Participants synchronously engage in communication rounds, such as FL rounds. Moreover, state channels in the incentive network require synchronous turn-taking between the channels' actors, enforced within the channel protocol. Synchrony is also an assumed property of PCNs from which our incentive network is theoretically derived.

The network also operates on certain security assumptions. Communication within the network is not encrypted, but chain transactions are signed with the account keys of the parties involved, ensuring authenticity. Participants are expected to behave rationally within the network's protocols. However, privacy and confidentiality are not provided by default. A rational adversary may attempt to manipulate the blockchain; however, such actions are mitigated through signatures.

During the FL process, we assume the parties to behave rationally. Using a Shapley Value reward system can enforce this behavior since the FL clients are rewarded when submitting adequate data and potentially penalized when submitting inadequate data. This Shapley Value scheme can also be extended to include the server side by having nodes verify the server aggregations. This can be achieved using a quorum of nodes recomputing the aggregation and checking it against the submitted one. Depending on the outcome, a penalty or reward can be allocated to the server that submitted the aggregated model. This is viable in a system with multiple FL servers with a pool of clients that also verify aggregations with a custom distributed verification process or consensus to evaluate server aggregations, such that there is no need to trust the FL servers. Therefore, rational behavior is a sufficient security assumption in this context since the Shapley Value scheme prevents attacks that follow non-rational behavior. As this paper is not highly focused on the security aspect of the system, the implementation of the Shapley Value incentive protocol and the custom consensus is proposed as future work. Our experiments focus on evaluating the feasibility and performance of using state channels in the FL context.

3.2 System Architecture

The architecture of StateFL resembles the loosely coupled BCFL formalized in [8] that relies on a separate blockchain network and an FL network where the FL clients send model updates to the miners for verification. According to [8], the advantages of loosely coupled BCFL are that the networks are fully independent, and the dataset is better retained on the clients. The separation also contributes to modularity. This architecture is more likely to be encountered in practice when commercially available public blockchains are chosen. Our system consists of an FL network that implements the traditional FL workflow and a blockchain that runs state channels on top of the ledger, which we call an incentive network. As such, the FL server opens one channel with each FL client before starting the learning process. The FL clients and server are connected to a state channel client where they post updates about the ongoing process, and, at the end of each round, a payout is made to the contributing FL clients. Upon completing the learning process, the channels are closed, and the updated balances are reflected on the blockchain via transactions signed by both the clients and the server. An overview of the system architecture is visualized in Figure 1.

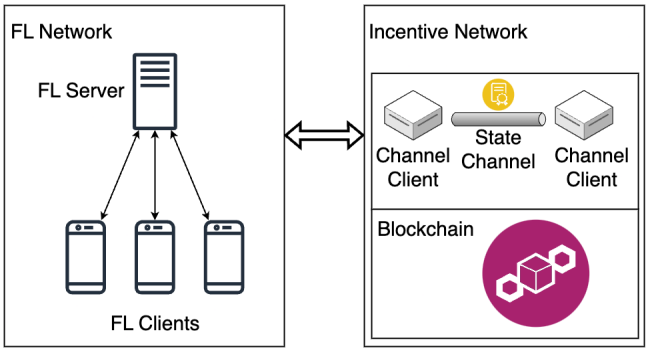

Figure 1: **System Architecture**

The FL algorithm is adjusted to integrate communication between the FL network and the incentive network via WebSocket so that the FL clients and server can interact with the state channels. As such, the round-based FL workflow is reflected in the state channel interaction. To illustrate the state channel-level workflow, we modeled the states that the channel transitions through during the FL process in the finite state machine presented in Figure 2. First of all, the FL server indicated the number of rounds to be had in the FL process, and a reference to the model is also provided. Since ML models are substantial in size, they are stored on the IPFS distributed file system and referenced by a content identifier, which is a cryptographic hash of the content. Second, the FL client uploads the updated model parameters to IPFS and sets the weight field in the state channel with the respective content identifier. Third, the FL server aggregates and evaluates the new model and provides performance metrics in accuracy and loss. Finally, when the round ends, the FL client receives a reward, and the process either continues to the next round or stops if the agreed number of rounds is reached. If the client does not provide a parameter update within a set time interval, it times out to prevent the workflow from stopping before continuing to the next round. The FL clients are paid per round instead of per workflow, to accommodate incidents such as failures that can cause

the client to become unresponsive and time out. Thus, the FL clients are still compensated for their partial contribution to the end model.

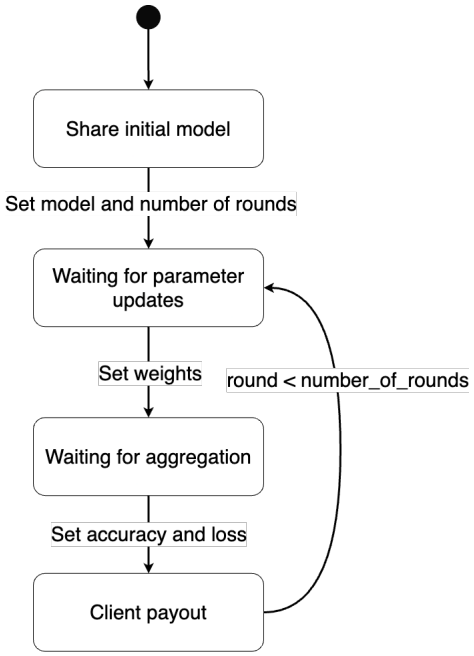

Figure 2: **State Channel FSM**

The state channel stores a reference to the ML model being trained and the client model updates, as well as performance metrics of the newly aggregated model. The performance metrics slots can be used to estimate the client's contributions to the aggregated model that allows for game theory approaches such as cooperative games with Shapley Value.

Implementation-wise, the FL network is implemented with the Flower framework, written in Python, which supports large-scale heterogeneous FL (up to approximately 15M clients) [63]. Furthermore, Flower is ML framework-agnostic and allows custom implementations of aggregation algorithms. In addition, it supports deployment on edge devices and simulations on single-node or multi-node compute clusters [63], thus making it a versatile framework suitable for research experiments and practical applications.

On the incentive network front, the state channel infrastructure that our system is built upon is developed using the Perun framework [64], written in Go. Perun supports fast off-chain payment and state channels that parties can use to transact without interacting with the ledger, unless disagreements require dispute resolution. When disputes occur, the network relies on the blockchain consensus to resolve the dispute. In our case with state channels, the state channel falls back on the smart contract that governs it to make sure the channel can be settled fairly. Hence, our ledger is an Ethereum blockchain running in the Ganache test environment [65] with smart contracts written in the Solidity programming language. The aforementioned execution flow presented in Figure 2 is implemented both at the channel level and the blockchain level in the form of a smart contract to resolve disputes via the consensus mechanism.

4 Methodology and Experimental Setup

This section details our methodology, followed by specifics of the experimental setup.

4.1 Methodology

To answer our research questions, we implemented a customizable FL testbed using the Flower FL Framework [63]. The testbed supports configuring an ML model, providing the number of federated learning rounds and the number of clients that should participate in the learning. Additionally, incentive methods and parameter storage methods can be toggled on or off. As such, to establish the performance of our proposed system, we compare the latency of three system architectures: a traditional FL system with no incentives, a BCFL system with an Ethereum blockchain (baseline), and a BCFL system with state channels (StateFL). The traditional FL system is measured for further reference as it represents the lower bound for latency since it runs FL without additional processes or functionalities such as incentives and storage.

The baseline BCFL system we compare StateFL against follows the same execution flow as StateFL. The difference lies in the fact that StateFL stores the ML model, evaluation metrics of the model, and the IPFS references in the channel state until closure, while BCFL stores commits of all intermediate values in a smart contract transaction. Thus, when the FL begins, the server initializes a smart contract instance with each client, and, as the learning progresses, the values of the contracts' attributes are updated through a transaction. First, the initial model and number of rounds are specified at the creation of each contract. Then, during each round, the server sets the IPFS references of parameter updates received from the clients, as well as the reference of the aggregated model from that round. Finally, at the end of the round, the server transfers a reward in ether cryptocurrency (ETH) to each client that participated in the FL. Both StateFL and the baseline BCFL use the same type of blockchain with the same configuration, a Ganache Ethereum test network. To simplify the complexity of the baseline BCFL system, we assume that, from an attack model perspective, the server is a trusted third party, hence the only party with access to the smart contract. Therefore, the smart contract will be able to store data relevant to the model in the ledger, and the client receives a reward after completing each round, similar to StateFL. The

smart contracts are implemented in Solidity version 0.7.0, and the BCFL accesses them through the Python Web3 library [66].

To quantify system performance and answer RQ 1 and RQ 2, we investigate the latency of the following processes and define the following metrics:

- Training per client = The mean time the clients spend training throughout the entire FL process. The mean is calculated per client.
- Aggregation per round = The mean time with respect to rounds that the server takes to aggregate the parameter updates of the clients.
- Completed FL round = The mean time takes to complete an FL round averaged over all rounds.
- FL process = The FL time from start to finish, i.e., since model initialization until completing the last round.
- Channel opening time = The time the server takes to open a channel with all clients.
- Channel settling time = The time the server takes to settle the channel with all clients.
- End-to-end (E2E) time = The time it takes to complete the FL process and settle transactions on the blockchain (if applicable).

To answer RQ 3 and RQ 4 we compare the latency, transaction fees, and number of on-chain transactions of StateFL channels from and including set up until the closure is concluded. This comparison is conducted under two scenarios: when no disputes occur and when disputes occur with a specified probability.

4.2 Experimental Setup

The experiments were conducted on a computer with an Apple M1 Pro SoC and 32 GB RAM. The evaluation testbed and data analysis software used for the experiments can be found at [67]. The FL state channel application can be found at [68]. A simplified version of the FL state channel application used for testing and development purposes is available at [69]. The more complex version provides command-line interfaces as separate processes to interact with the state channel client, while the simplified version executes prescripted instructions.

RQ 1 and RQ 2

The input parameters of our experiment design for RQ 1 and RQ 2 are the number of FL clients, the number of federated rounds, and the FL system in use. We run our experiments with a varying number of clients {2, 5, 10, 25, 50} and rounds {1, 5, 10} for each of the three systems {FL, BCFL, StateFL} following a full factorial design of experiments for the selected values.

In our experiments, we use the CIFAR10 and the LeNet-5 CNN model [70] implemented in Pytorch [71]. Since this paper is not focused on investigating the ML side of the system, a simple CNN model that trains in a short amount of time is suitable for our purposes. Furthermore, the FL clients receive the same model and have equally sized datasets. The IPFS storage node is available as a Docker container, and the version used is IPFS version 0.7.0. Network latency is simulated by artificially delaying transmissions on each network link by a randomly sampled value from the PlanetLab Network Latency Dataset released in [72].

To mitigate the misleading effects of variations in latency due to randomness, the experiment is repeated for each set of input parameters 10 times, and the mean values of the 10 runs constitute the final results illustrated in the upcoming section. Two-tailed t-tests are performed to check if the difference in latency between systems is statistically significant. Formally, we compute the difference of two means statistical tests with two independent samples of size 10 (StateFL and BCFL). Since the sample size is small, we use a t-distribution and an alpha of 0.05.

RQ 3 and RQ 4

Since the scope of RQ 3 and RQ 4 is centered solely on the channel, we simplify our setup and exclude the FL process, focusing only on the incentive network of the system. Further, to obtain a more accurate measurement we study the latency, fees, and number of transactions on a single link between two parties. We peruse this approach under the assumption that the blockchain does not experience congestion that delays transaction processing or raises transaction fees. Additionally, if the incentive network is employing channels for all blockchain interaction, the no congestion assumption is sound to make since it is highly unlikely to occur in practice.

The input parameters for this experiment are the probability p under which a dispute will occur {0, 0.25, 0.5, 0.75, 1} (for RQ 3 only) and the number of rounds {1, 2, 3, 4, 5, 6, 7, 8, 9, 10} (for both RQ 3 and RQ 4). The number of rounds is the only parameter of the channel that influences its runtime and on-chain fees in case of disputes, while the others are related to the model identifier and accuracy and loss scores, thus, they are excluded from the analysis. Similar to the experiments of RQ 1 and RQ 2, 10 latency measurements are taken, and the mean value accounts for the final result we perform the measurements with no disputes and with 100% disputes. The latency with intermediate dispute probabilities is calculated by computing the mean of 10 randomly sampled data points where p% of them are latencies measured with 100% disputes while the remaining $(1-p)$% are latencies measured without disputes. Averaging latencies using a proportion of measurements with no disputes and 100% disputes mirrors the results of running experiments with intermediate probabilities because the random sampling

statistically represents the expected distribution of dispute occurrences, ensuring equivalent outcomes.

5 Analysis of Results

In this section, we report the results of our experiments and aim to answer the research questions formulated in the previous section. We dedicated an individual subsection for each research question.

5.1 RQ 1: StateFL latency analysis

The latency of StateFL compared to BCFL and traditional FL is shown in the figures in Appendix A. Additionally, Figures 3, 4, and 5 illustrate the cumulative time spent in each step of the process: training, aggregating, and channel opening/closing (for StateFL). Noticeable differences between StateFL and BCFL can be seen in the aggregation time (Figure 10), mean round time (Figure 12), mean FL time (Figure 9), and E2E time (Figure 8). StateFL performs significantly better in aggregation time, round time, and FL time where it is quite close to the FL system. In E2E time, however, BCFL has better results when the number of rounds is low, but StateFL outperforms it when the number of rounds is higher. In contrast, the difference in mean training time (Figure 13) is not as pronounced, as expected, since interaction with the incentive network is minimal and only present on the client side for StateFL. Clients updating weights in the state channel do not create blockchain transactions unless there is a dispute, thus, setting the channel's state is a fast operation.

The most noticeable difference in latency between BCFL and StateFL is in the aggregation time (Figure 10) where BCFL takes from $\approx$ 18 s for 2 clients and 1 round until $\approx$ 9 min for 50 clients and 10 rounds vs $\approx$ 2.5 s and $\approx$ 27 s in StateFL. This difference is reflected further in round time (Figure 12) since the long aggregation time of BCFL causes rounds to take longer from $\approx$ 55 s until $\approx$ 18 min vs $\approx$ 30 s and $\approx$ 6 min in StateFL for the same configurations. The effect is further propagated to FL time (Figure 9) as longer rounds cause the FL to take longer from $\approx$ 38 s to $\approx$ 130 min vs $\approx$ 22 s to $\approx$ 47 min in StateFL. The E2E time, however, displays a different trend because of the channel setup and closure that happens before and after the FL process in StateFL. In Figure 8 we observe that BCFL performs better when the number of rounds is low (< 5), while StateFL performs better when the number of rounds is higher (> 5). When the number of rounds is 5 BCFL and StateFL tend to have similar performance.

We confirm these observations with a two-tailed t-test to determine the statistical significance of the differences in latency between the two systems. An alpha of 0.05 is chosen, meaning that p-values smaller than 0.05 are considered statistically significant. The t-test results are shown in the tables in Appendix A. Table 3 shows some significant differences in training time with StateFL taking longer than BCFL in most cases, however, this does not hold for every configuration (10 clients and 10 rounds, as well as 25 clients and 10 rounds). As there is no apparent trend, this likely occurs due to randomness in process scheduling. Tables 4, 5, and 6 confirm significant differences in aggregation time, FL time, and round time, respectively, attributed to the FL server waiting for the transaction updating the smart contract attributes. Table 7 highlights a significant difference in E2E time, with StateFL being slower when the number of rounds is 1 or the number of clients is below 5. There are no significant latency differences when the number of rounds is 5 and the number of clients is 10 or higher. StateFL significantly outperforms BCFL when the number of rounds is above 5 and the number of clients is 10 or more. This is because StateFL interacts with the blockchain only when the channels are created and closed and not in between rounds, thus, adding more rounds to the FL does not affect the latency of the FL network. The latency of the incentive network in StateFL increases with the number of clients because of the number of channels to open and close, as there is 1 channel per client. In contrast, in BCFL, the server updates the smart contracts and issues coin transfers for each participating client at the end of each round. Hence, more rounds and more clients result in more blockchain interactions and higher latency.

In summary, StateFL generally performs better than BCFL, especially in scenarios with higher numbers of rounds, due to its efficient channel operations that minimize blockchain interactions during the FL process. BCFL, on the other hand, shows better E2E time performance with fewer rounds as StateFL experiences increased latency due to the need to open and close more channels. BCFL's latency increases more noticeably with the number of rounds and clients due to frequent blockchain interactions.

5.2 RQ 2: BCFL and StateFL bottlenecks

In the case of BCFL, the server aggregation process is the main bottleneck. As the number of FL rounds and clients increases, system latency also rises. This is demonstrated when examining the proportions of time spent training and aggregating, FL time, and aggregation time in Figures 4, 9, and 10. The bottleneck occurs because the server must sequentially update a smart contract instance with each client's new information and initiate a coin transfer. Consequently, more clients and rounds lead to more time spent on aggregation. When this process is inherently slow, the latency significantly compounds over time. This effect is clearly shown when comparing Figures 3 and 4, where BCFL's aggregation time is significantly higher than that of FL.

In contrast, for StateFL, the number of rounds has little impact on the system's latency resulting from the addition of incentives. When comparing Figures 9,

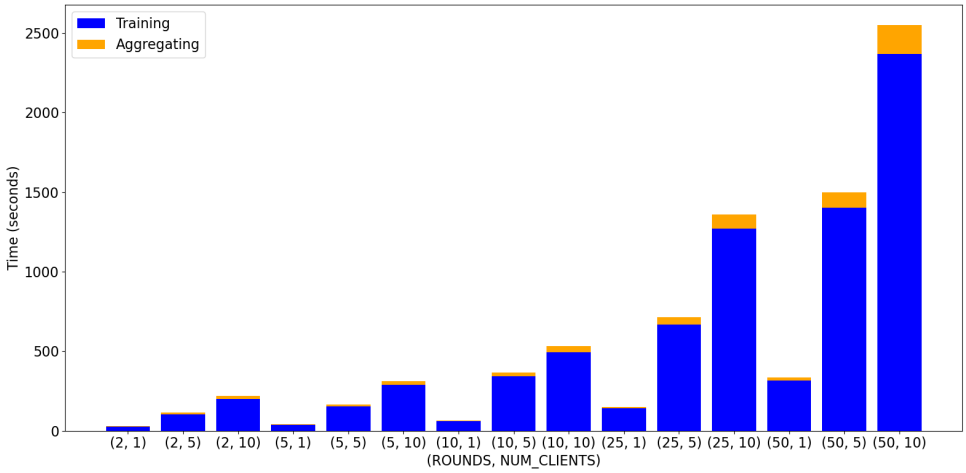

Figure 3: **Proportion of time spent training and aggregating in FL**

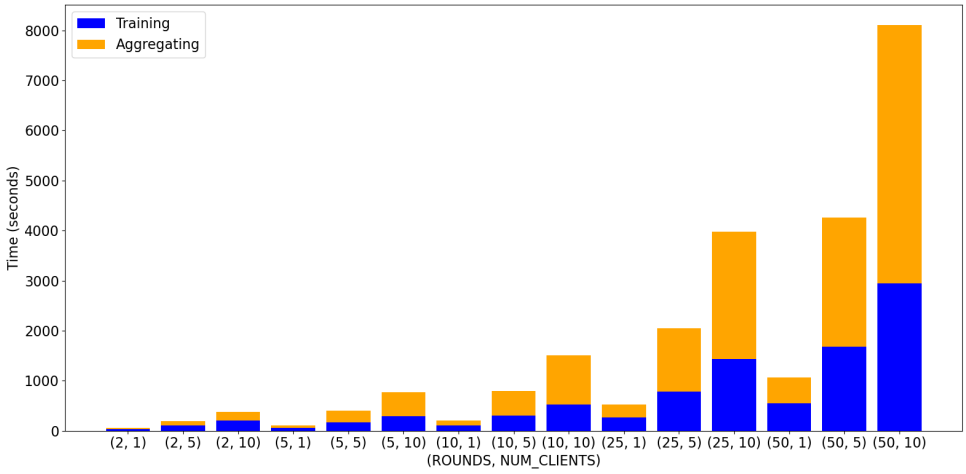

Figure 4: **Proportion of time spent training and aggregating in BCFL**

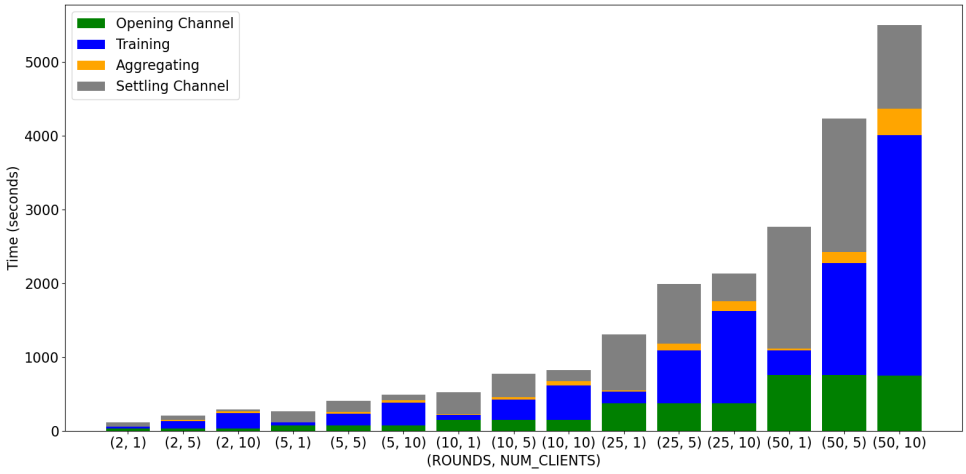

Figure 5: **Proportion of time spent opening/settling channels, training, and aggregating in StateFL**

10, 12, and 13 between StateFL and the FL system, the differences in latency are not substantial. However, the E2E time, shown in Figure 8, reveals more pronounced latency differences due to the additional channel setup and closure times. The main bottleneck in StateFL is the process of opening and settling channels, which increases with the number of clients since this determines the number of channels in the system. This effect is visualized in Figure 5, where data points with 1 and 5 rounds show more time spent on setting up and closing channels than on training and aggregating. Therefore, this bottleneck is primarily influenced by the number of clients. However, it is alleviated when the number of rounds is high (10 or more), as the benefits of more rounds outweigh the setup costs.

5.3 RQ 3: disputes' impact on latency

Our experiments recorded a channel opening time of approximately 5 seconds and a closure time of approximately 15 seconds. If no disputes occur, the channel can transition between states in the order of milliseconds. However, when disputes occur, the number of on-chain transactions that need to be processed increases, which adds more delays. In case no disputes occur, 8 transactions are recorded on-chain to deploy the smart contracts, open, and settle the channel. One aspect worth noting is that there is an initial one-time cost of approximately 14 seconds to deploy the state channel smart contracts on the blockchain before any channels can be opened.

Table 1 presents the number of FL rounds the channel recorded, channel latency when transitioning states until closure, and the number of on-chain transactions (TX) when disputes occur in every FL round. We can observe that with each added FL round, 2 on-chain transactions are added to the total count, and that results in approximately 10 extra seconds in latency. The transactions correspond to a dispute from the client and a dispute from the server when setting the model weights and aggregating. This indicates a linear increase in latency with the number of rounds. As a result, the latency values for higher numbers of rounds can be predicted using linear regression. One detail to note is that there is an increase of 3 transactions between the no disputes configuration and one disputed round. The extra transaction accounts for registering the channel dispute on the blockchain as a separate transaction, which contributes to additional dispute setup latency when initialing the first dispute on a channel. This initial transaction is made to register the channel state on the blockchain and serve as a starting state for the on-chain verification process.

In Figure 6 we observe two graphs illustrating the linear regression functions of latency with respect to the number of rounds and the frequency of disputes. The first graph plots the channel state transitions, while the second also includes channel opening and closure. We

Table 1: **100% Disputes Latency and TX Count**

Rounds	Latency (s)	TX Count
1	25.82441637	11
2	35.17835116	13
3	45.25600481	15
4	55.38022993	17
5	65.46270877	19
6	75.56843389	21
7	85.60973508	23
8	95.73326700	25
9	105.82886574	27
10	116.04423808	29

plot distinct functions in the cases when disputes do not occur at all, they occur in 25%, 50%, 75%, and 100% of rounds. Additionally, we also plot the latency of the blockchain and smart contract from the BCFL system for comparison. As expected, the channel with no disputes' function increases insignificantly compared to the others, due to the difference in magnitude, while the channel with 100% disputes function grows the fastest out of the channel functions. The blockchain from the BCFL presents the highest slope but performs better than the channels with disputes when the number of rounds is low (≤ 6). The main reason for the steeper increase is because in the BCFL system transactions are made to set the model weights, aggregation results, and coin transfer reward to the client after each round. In contrast, in StateFL the client funds are withdrawn from the channel to the blockchain only when the channel is settled, as opposed to after each round. In realistic FL scenarios, the number of rounds is in the order of hundreds, thus, channels provide a substantial latency improvement, even if disputes do occasionally occur.

5.4 RQ 4: disputes' impact on TX fees

Table 2 presents the number of FL rounds the channel recorded, the total gas usage of the blockchain, and the number of on-chain transactions when disputes occur in every FL round. With every transaction, we observe an increase of approximately 0.0010 Gwei in the total gas usage of the network, where 1 Gwei is one-billionth of 1 ETH. Since each disputed round increases the total transaction count by 2 transactions, we identify a linear trend in gas usage with respect to the number of disputed rounds in the channel. This trend is visualized in Figure 7 where the network gas usage is plotted with respect to the number of rounds.

When no disputes occur, the network registers a gas usage of 0.005385 Gwei for 8 transactions and stays the same regardless of the number of rounds, since more rounds do not generate more transactions, unless disputes are initiated. The initial 3 transactions, amounting to 0.005019 Gwei in fees, correspond to the one-time smart contract deployment required before

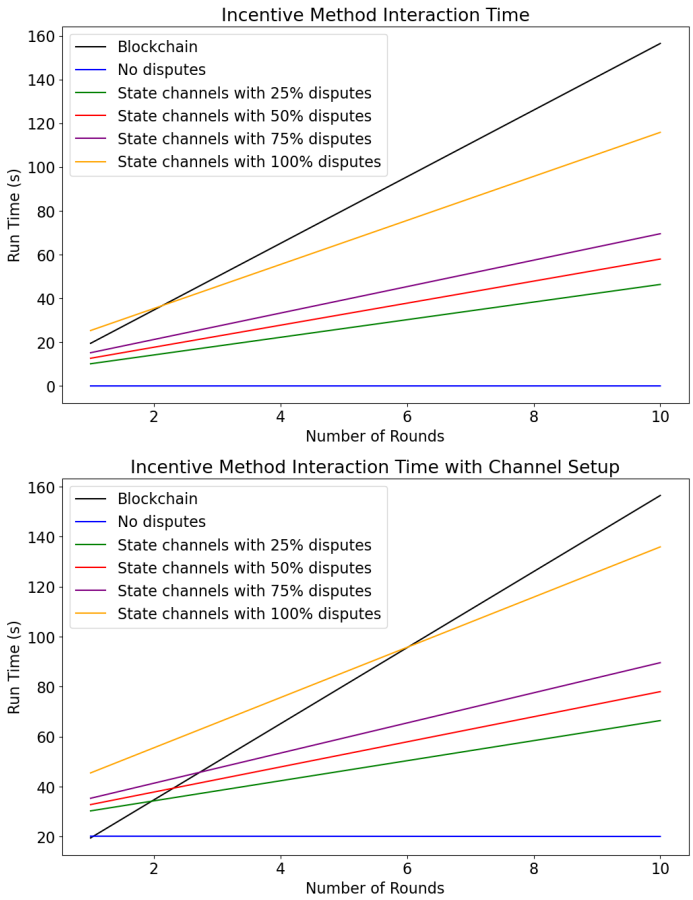

Figure 6: **Linear regression functions of the channel latency according to the number of FL rounds and dispute probability**

opening channels. This highlights the financial benefit of using channels over on-chain operations for FL as they reduce the number of transactions on the network, but also the load, which results in lower fees and fewer transactions.

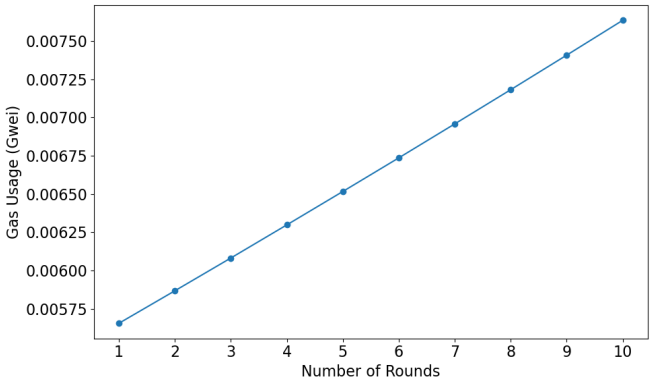

Figure 7: **Total blockchain gas usage with disputes**

Table 2: **On-chain Gas Usage and TX Count**

Rounds	Total Gas Usage (Gwei)	TX Count
1	0.005655	11
2	0.005868	13
3	0.006082	15
4	0.006298	17
5	0.006516	19
6	0.006736	21
7	0.006958	23
8	0.007182	25
9	0.007407	27
10	0.007635	29

6 Conclusions and Future Work

In this paper, we propose StateFL, a revised BCFL architecture to improve the performance of existing frameworks in terms of latency and transaction fees. This is achieved by leveraging state channels to minimize blockchain interactions. We evaluate StateFL in a series of experiments to establish whether it has a lower latency than a baseline BCFL system. Furthermore, we identify existing bottlenecks in StateFL and BCFL and we analyze the impact of disputes on channel latency, as well as transaction fees.

StateFL outperforms BCFL in scenarios involving a high number of FL rounds (generally > 5) because of its efficient off-chain interactions. BCFL experiences a bottleneck on the FL server side due to the frequent blockchain interactions required for smart contract updates and coin transfers that happen during the aggregation stage at the end of every FL round. Thus, in the case of BCFL, the long aggregation leads to longer rounds and subsequently longer FL time, and the latency increases with a higher number of clients and rounds. StateFL, however, experiences a bottleneck during channel setup and closure which increases with the number of clients, irrespective of the number of rounds. The performance gains from reduced blockchain interactions become increasingly significant as the number of rounds rises. Thus, in typical FL scenarios with hundreds of rounds, the benefits of using state channels outweigh the cost of opening and closing channels and even the cost of occasional disputes.

In our investigation of the impact of disputes on latency and gas usage, we found that both metrics increase linearly with the number of FL rounds. Each disputed round adds 2 on-chain transactions with a transaction fee of approximately 0.0010 Gwei. The fact that both parties involved in the channel need to contribute to the transaction fees desensitizes the misuse of this mechanism by rational participants. StateFL has the potential to reduce the load on the blockchain by offloading transactions to state channels which results in a much lower number of transactions and, as a result,

gas usage. However, one downside of channels in the FL context is that client rewards are accessible only after closing the channel.

To further improve StateFL, we propose implementing additional security measures to protect against potential attacks to ensure the robustness of the StateFL framework. Since we focused our attention on StateFL from a systems performance perspective, ironing out the security-related details fell outside of the scope of this paper. On the FL network front, improvements can be made by adding mechanisms to prevent data leakages, such as differential privacy [4]. This addition would make it difficult to infer information about the dataset from model updates. Moreover, model poisoning attacks, free-riding, and other malicious behavior can be prevented using Shapley Value-based smart contracts distributing rewards or penalties among clients based on their input [10]. Furthermore, designing a mechanism verifying the correctness of the server's aggregation process would also enhance trust and reliability in the FL system. This refinement would provide the means to hold accountability for both the server and the clients. Lastly, to obtain more accurate measurements of StateFL's latency we propose experimenting on a distributed network. Thus, delays due to network latency would be captured more accurately.

Acknowledgements

Delving into the realm of Computer Science and embarking on this thesis has been quite a journey, one I could not have navigated alone. My gratitude goes out to those who stood by me, offering their support to keep me on course.

First and foremost, I would like to thank Stefanie for her invaluable guidance and patience, steering me to the final chapter of this journey. Our weekly meetings were not just productive but also enjoyable, and I'm truly grateful for that. Special appreciation also goes to Zeki Erkin for his dedicated attention to this thesis project, ensuring a seamless progression. I am also thankful to Jérémie Decouchant for taking part in the thesis committee and to Andrei for his meticulous proofreading.

Thank you, Timon, for the fun lunch breaks we took as we worked on our projects and for your countless encouragements and support, especially when things got challenging. The fishbowl office days were good times!

I would also like to thank Andrei, Mălina, Nico, Radu, and Silviu for the many wonderful memories we crafted during our study abroad adventure. With you, I felt more at home in a new country that we explored together. It was truly fantastic, and I had a blast!

Finally, massive thanks to my family, especially my parents, for giving me the opportunity to study in the Netherlands. Your support allowed me to discover a new culture and connect with outstanding individuals from across the globe. I definitely learned a lot from it. Thank you for everything!

References

[1] Michael I. Jordan and Tom M. Mitchell. "Machine learning: Trends, perspectives, and prospects". In: *Science* 349 (2015).

[2] Youyang Qu et al. "Blockchain-enabled Federated Learning: A Survey". In: *ACM Comput. Surv.* 55 (2022).

[3] Qiang Yang et al. "Federated Machine Learning: Concept and Applications". In: *CoRR* abs/1902.04885 (2019).

[4] Zhanglong Ji, Zachary C Lipton, and Charles Elkan. "Differential privacy and machine learning: a survey and review". In: *arXiv preprint arXiv:1412.7584* (2014).

[5] Zhigang Chen et al. "Bibliometrics of machine learning research using homomorphic encryption". In: *Mathematics* 9 (2021).

[6] Ian Zhou et al. "Secure Multi-Party Computation for Machine Learning: A Survey". In: *IEEE Access* (2024).

[7] Jakub Konečný et al. "Federated Learning: Strategies for Improving Communication Efficiency". In: *NIPS Workshop on Private Multi-Party Machine Learning*. 2016.

[8] Zhilin Wang and Qin Hu. "Blockchain-based Federated Learning: A Comprehensive Survey". In: *CoRR* abs/2110.02182 (2021).

[9] Yuan Liu et al. "Fedcoin: A peer-to-peer payment system for federated learning". In: *Federated learning: privacy and incentive*. Springer, 2020.

[10] Jiyue Huang et al. *An Exploratory Analysis on Users' Contributions in Federated Learning*. 2020.

[11] Yang Zhao et al. *Privacy-Preserving Blockchain-Based Federated Learning for IoT Devices*. 2021. arXiv: 1906.10893 [cs.CR].

[12] Hyesung Kim et al. "Blockchained On-Device Federated Learning". In: *IEEE Communications Letters* 24 (2020).

[13] Jiasi Weng et al. "DeepChain: Auditable and Privacy-Preserving Deep Learning with Blockchain-Based Incentive". In: *IEEE Transactions on Dependable and Secure Computing* 18 (2021).

[14] Lewis Gudgeon et al. "Sok: Layer-two blockchain protocols". In: *Financial Cryptography and Data Security: 24th International Conference, FC 2020, Kota Kinabalu, Malaysia, February 10–14, 2020 Revised Selected Papers 24*. Springer. 2020.

[15] Fahad Saleh. "Blockchain without waste: Proof-of-stake". In: *The Review of financial studies* 34 (2021).

[16] Xidi Qu et al. "Proof of Federated Learning: A Novel Energy-Recycling Consensus Algorithm". In: *IEEE Transactions on Parallel and Distributed Systems* 32 (2021).

[17] Lydia D Negka and Georgios P Spathoulas. "Blockchain state channels: A state of the art". In: *IEEE Access* 9 (2021).

[18] Joseph Poon and Thaddeus Dryja. *The Bitcoin Lightning Network: Scalable off-chain instant payments*. Tech. rep. Technical Report (Draft). 2016. URL: https : / / lightning . network / lightning - network-paper.pdf.

[19] Satoshi Nakamoto. "Bitcoin: A Peer-to-Peer Electronic Cash System". In: *Cryptography Mailing list at https://metzdowd.com* (Mar. 2009).

[20] Karl Wüst and Arthur Gervais. "Do you Need a Blockchain?" In: *2018 Crypto Valley Conference on Blockchain Technology (CVCBT)*. 2018.

[21] Dun Li et al. "Blockchain for Federated Learning toward Secure Distributed Machine Learning Systems: A Systemic Survey". In: *Soft Comput.* 26.9 (2022).

[22] Vitalik Buterin. *Ethereum: A Next-Generation Smart Contract and Decentralized Application Platform*. 2013. URL: https://ethereum.org/en/whitepaper/.

[23] Shuai Wang et al. "An Overview of Smart Contract: Architecture, Applications, and Future Trends". In: *2018 IEEE Intelligent Vehicles Symposium (IV)*. 2018.

[24] Yunlong Lu et al. "Low-Latency Federated Learning and Blockchain for Edge Association in Digital Twin Empowered 6G Networks". In: *IEEE Transactions on Industrial Informatics* 17.7 (2021).

[25] Yunlong Lu et al. "Blockchain and Federated Learning for 5G Beyond". In: *IEEE Network* 35.1 (2021).

[26] Miguel Castro, Barbara Liskov, et al. "Practical byzantine fault tolerance". In: *OsDI*. Vol. 99. 1999. 1999.

[27] Mansoor Ali, Hadis Karimipour, and Muhammad Tariq. "Integration of Blockchain and Federated Learning for Internet of Things: Recent Advances and Future Challenges". In: *Computers Security* 108 (June 2021).

[28] Liam Collins et al. *FedAvg with Fine Tuning: Local Updates Lead to Representation Learning*. 2022.

[29] Alvin E Roth. *The Shapley value: essays in honor of Lloyd S. Shapley*. Cambridge University Press, 1988.

[30] Liangqi Yuan et al. *Communication-Efficient Multimodal Federated Learning: Joint Modality and Client Selection*. 2024. arXiv: 2401.16685 [cs.LG].

[31] Daniel Fryer, Inga Strümke, and Hien Nguyen. "Shapley values for feature selection: The good, the bad, and the axioms". In: *Ieee Access* (2021).

[32] Guan Wang, Charlie Xiaoqian Dang, and Ziye Zhou. "Measure contribution of participants in federated learning". In: *2019 IEEE international conference on big data (Big Data)*. IEEE. 2019.

[33] Virat Shejwalkar and Amir Houmansadr. "Manipulating the byzantine: Optimizing model poisoning attacks and defenses for federated learning". In: *NDSS*. 2021.

[34] Venkataraman Natarajan Iyer. "A review on different techniques used to combat the non-IID and heterogeneous nature of data in FL". In: *arXiv preprint arXiv:2401.00809* (2024).

[35] Fernanda C Orlandi et al. "Entropy to mitigate non-IID data problem on Federated Learning for the Edge Intelligence environment". In: *IEEE Access* (2023).

[36] Kevin Hsieh et al. "The non-iid data quagmire of decentralized machine learning". In: *International Conference on Machine Learning*. PMLR. 2020.

[37] Jason Posner et al. "Federated Learning in Vehicular Networks: Opportunities and Solutions". In: *IEEE Network* 35.2 (2021).

[38] Guhan Zheng et al. "Mobility-Aware Split-Federated With Transfer Learning for Vehicular Semantic Communication Networks". In: *IEEE Internet of Things Journal* (2024).

[39] Tianzhu Pan, Xuanli Wu, and Xuesong Li. "Dynamic Multi-Sleeping Control with Diverse Quality-of-Service Requirements in Sixth-Generation Networks Using Federated Learning". In: *Electronics* 13.3 (2024).

[40] Elissa Mhanna and Mohamad Assaad. *Rendering Wireless Environments Useful for Gradient Estimators: A Zero-Order Stochastic Federated Learning Method*. 2024. arXiv: 2401.17460 [cs.LG].

[41] Jonas Geiping et al. *Inverting Gradients – How easy is it to break privacy in federated learning?* 2020.

[42] Ziwen Cheng et al. "PoShapley-BCFL: A Fair and Robust Decentralized Federated Learning Based on Blockchain and the Proof of Shapley-Value". In: *International Conference on Neural Information Processing*. Springer. 2023.

[43] Rashmi Thennakoon et al. "Decentralized Defense: Leveraging Blockchain against Poisoning Attacks in Federated Learning Systems". In: 2024.

[44] Minghao Wang et al. "Public and Private Blockchain Infusion: A Novel Approach to Federated Learning". In: *IEEE Internet of Things Journal* (2024).

[45] Yinan Liang and Shuo Xu. "Blockchain-Based Federated Learning for IoT and Industrial IoT: A Review". In: *IEEE Access* 8 (2020).

[46] Xiaoding Wang et al. "Heterogeneous Blockchain and AI-Driven Hierarchical Trust Evaluation for 5G-Enabled Intelligent Transportation Systems". In: *IEEE Transactions on Intelligent Transportation Systems* (2021).

[47] Shiva Raj Pokhrel and Jinho Choi. "Federated Learning With Blockchain for Autonomous Vehicles: Analysis and Design Challenges". In: *IEEE Transactions on Communications* 68.8 (2020).

[48] Henrique Dias and Nirvana Meratnia. "Block-Learning: A Modular Framework for Blockchain-Based Vertical Federated Learning". In: *International Conference on Ubiquitous Security*. Springer. 2022.

[49] Eunsu Goh et al. "Blockchain-Enabled Federated Learning: A Reference Architecture Design, Implementation, and Verification". In: *IEEE Access* (2023).

[50] Neil Efren Villanueva. "Blockchain Technology Application: Challenges, Limitations and Issues". In: *Journal of Computational Innovations and Engineering Applications Vol* 5.2 (2021).

[51] Fredy Andres Aponte-Novoa et al. "The 51% attack on blockchains: A mining behavior study". In: *IEEE Access* 9 (2021).

[52] Seonggeun Kim and Sang-Geun Hahn. "Mining pool manipulation in blockchain network over evolutionary block withholding attack". In: *IEEE Access* 7 (2019).

[53] Ittay Eyal and Emin Gün Sirer. "Majority is not enough: Bitcoin mining is vulnerable". In: *Communications of the ACM* 61 (2018).

[54] Usman Chohan. "The Double Spending Problem and Cryptocurrencies". In: *SSRN Electronic Journal* (Jan. 2017).

[55] Tong Cao, Jérémie Decouchant, and Jiangshan Yu. "Leveraging the Verifier's Dilemma to Double Spend in Bitcoin". In: *arXiv preprint arXiv:2210.14072* (2022).

[56] Yourong Chen et al. "A survey on blockchain systems: Attacks, defenses, and privacy preservation". In: *High-Confidence Computing* 2 (2022).

[57] Cosimo Sguanci, Roberto Spatafora, and Andrea Mario Vergani. "Layer 2 blockchain scaling: A survey". In: *arXiv preprint arXiv:2107.10881* (2021).

[58] Ankit Gangwal, Haripriya Ravali Gangavalli, and Apoorva Thirupathi. "A survey of Layer-two blockchain protocols". In: *Journal of Network and Computer Applications* 209 (2023).

[59] Mohamed Sabt, Mohammed Achemlal, and Abdelmadjid Bouabdallah. "Trusted execution environment: what it is, and what it is not". In: *2015 IEEE Trustcom/BigDataSE/Ispa*. Vol. 1. IEEE. 2015.

[60] Matthew Green and Ian Miers. "Bolt: Anonymous payment channels for decentralized currencies". In: *Proceedings of the 2017 ACM SIGSAC conference on computer and communications security*. 2017.

[61] Stefan Dziembowski, Sebastian Faust, and Kristina Hostáková. "General state channel networks". In: *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. 2018.

[62] Nikolaos Papadis and Leandros Tassiulas. "Blockchain-Based Payment Channel Networks: Challenges and Recent Advances". In: *IEEE Access* 8 (2020).

[63] Daniel J. Beutel et al. *Flower: A Friendly Federated Learning Research Framework*. 2022. arXiv: 2007.14390 [cs.LG].

[64] PolyCrypt GmbH. *Perun Non-technical Summary*. https://perun.network/wp-content/uploads/Perun2.0.pdf. 2018.

[65] Garima Mathur. *GANACHE: A Robust Framework for Efficient and Secure Storage of Data on Private Ethereum Blockchains*. Oct. 2023.

[66] Wei-Meng Lee. "Developing Web3 dapps using Python". In: Apr. 2023.

[67] I.P. Iacoban. *fl-testbed*. URL: https://github.com/iiacoban42/fl-testbet.

[68] I.P. Iacoban. *FL-with-state-channels*. URL: https://github.com/iiacoban42/FL-with-state-channels.

[69] I.P. Iacoban. *fl-state-channel-app*. URL: https://github.com/iiacoban42/fl-state-channel-app.

[70] Y. Lecun et al. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11 (1998).

[71] Adam Paszke et al. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *CoRR* abs/1912.01703 (2019).

[72] Rui Zhu et al. "Network Latency Estimation for Personal Devices: A Matrix Completion Approach". In: *IEEE/ACM Transactions on Networking* 25 (2017).

A APPENDIX

A.1 T-test Tables

Table 3: **T-test on Mean Training Time**

(#C, #R)	p-value	t-value	significant
(2, 1)	0.962	0.048	False
(2, 5)	0.000	-5.035	True
(2, 10)	0.000	-7.848	True
(5, 1)	0.026	-2.425	True
(5, 5)	0.000	-10.740	True
(5, 10)	0.000	-18.114	True
(10, 1)	0.037	-2.251	True
(10, 5)	0.134	-1.511	False
(10, 10)	0.000	4.980	True
(25, 1)	0.043	-2.178	True
(25, 5)	0.686	-0.405	False
(25, 10)	0.000	6.896	True
(50, 1)	0.002	-3.571	True
(50, 5)	0.143	-1.477	False
(50, 10)	0.120	-1.560	False

Table 4: **T-test on Mean Aggregation Time**

(#C, #R)	p-value	t-value	significant
(2, 1)	0.0	86.751	True
(2, 5)	0.0	81.857	True
(2, 10)	0.0	69.009	True
(5, 1)	0.0	100.240	True
(5, 5)	0.0	140.923	True
(5, 10)	0.0	124.915	True
(10, 1)	0.0	136.335	True
(10, 5)	0.0	193.528	True
(10, 10)	0.0	230.454	True
(25, 1)	0.0	188.802	True
(25, 5)	0.0	95.378	True
(25, 10)	0.0	206.349	True
(50, 1)	0.0	422.501	True
(50, 5)	0.0	348.368	True
(50, 10)	0.0	44.334	True

Table 5: **T-test on FL Time**

(#C, #R)	p-value	t-value	significant
(2, 1)	0.0	85.780	True
(2, 5)	0.0	52.103	True
(2, 10)	0.0	32.739	True
(5, 1)	0.0	45.575	True
(5, 5)	0.0	67.052	True
(5, 10)	0.0	69.308	True
(10, 1)	0.0	30.854	True
(10, 5)	0.0	40.799	True
(10, 10)	0.0	66.774	True
(25, 1)	0.0	62.492	True
(25, 5)	0.0	59.192	True
(25, 10)	0.0	59.991	True
(50, 1)	0.0	63.229	True
(50, 5)	0.0	52.662	True
(50, 10)	0.0	48.032	True

Table 6: **T-test on Round Time**

(#C, #R)	p-value	t-value	significant
(2, 1)	0.0	150.310	True
(2, 5)	0.0	33.725	True
(2, 10)	0.0	36.340	True
(5, 1)	0.0	52.739	True
(5, 5)	0.0	64.607	True
(5, 10)	0.0	73.316	True
(10, 1)	0.0	41.008	True
(10, 5)	0.0	14.928	True
(10, 10)	0.0	44.743	True
(25, 1)	0.0	96.189	True
(25, 5)	0.0	33.056	True
(25, 10)	0.0	50.761	True
(50, 1)	0.0	92.001	True
(50, 5)	0.0	38.515	True
(50, 10)	0.0	7.094	True

Table 7: **T-test on E2E Time**

(#C, #R)	p-value	t-value	significant
(2, 1)	0.000	-338.774	True
(2, 5)	0.000	-12.443	True
(2, 10)	0.000	18.621	True
(5, 1)	0.000	-169.117	True
(5, 5)	0.000	-4.972	True
(5, 10)	0.000	45.276	True
(10, 1)	0.000	-118.161	True
(10, 5)	0.926	-0.094	False
(10, 10)	0.000	46.849	True
(25, 1)	0.000	-65.714	True
(25, 5)	0.915	-0.109	False
(25, 10)	0.000	42.313	True
(50, 1)	0.000	-22.075	True
(50, 5)	0.582	-0.561	False
(50, 10)	0.000	7.201	True

A.2 System Latency Plots

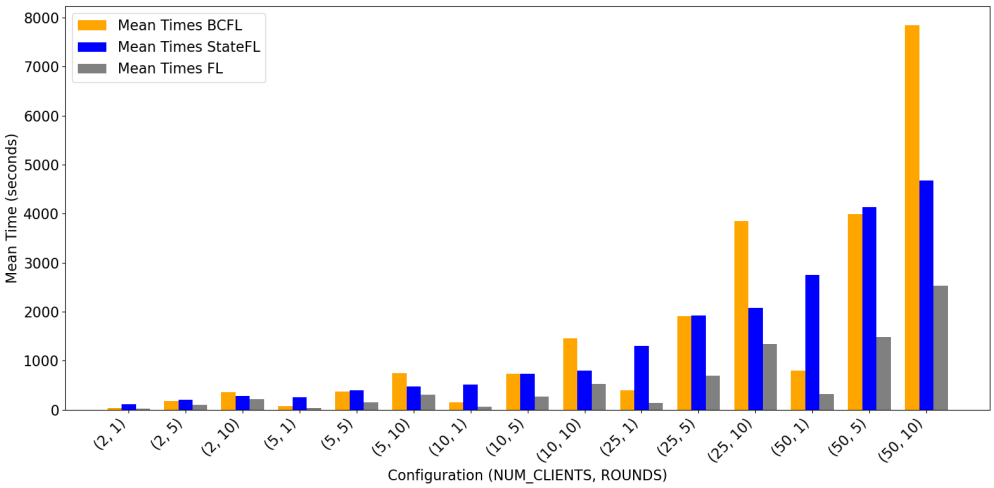

Figure 8: **End-to-End experiment time**

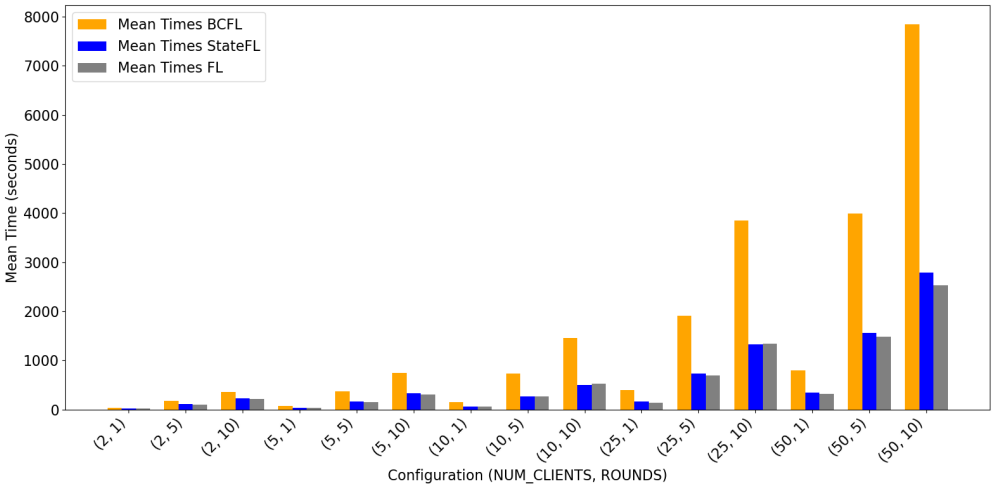

Figure 9: **Federated Learning time**

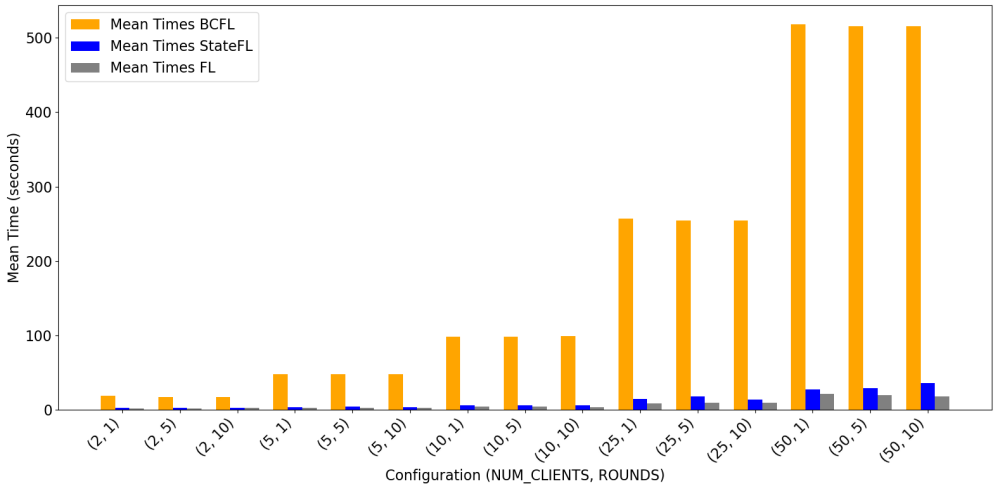

Figure 10: **Mean aggregation time per round**

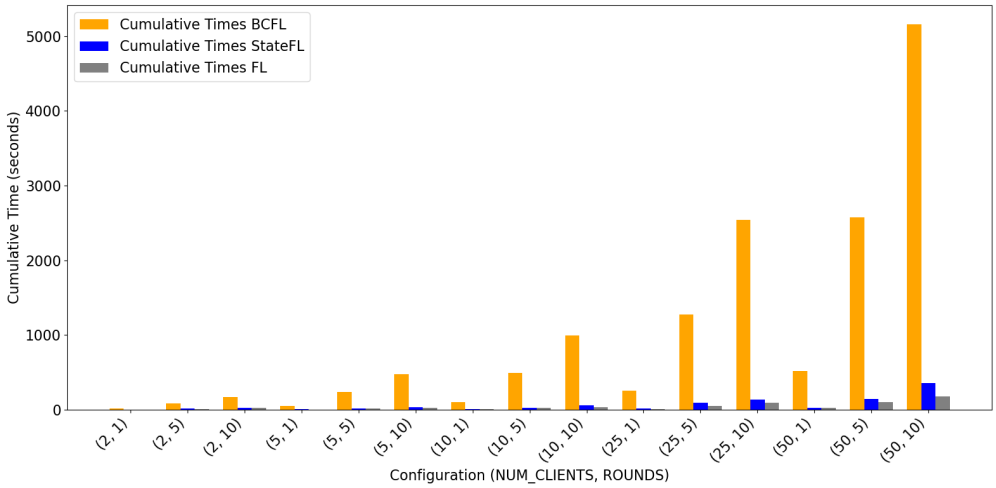

Figure 11: **Cumulative aggregation time (all rounds)**

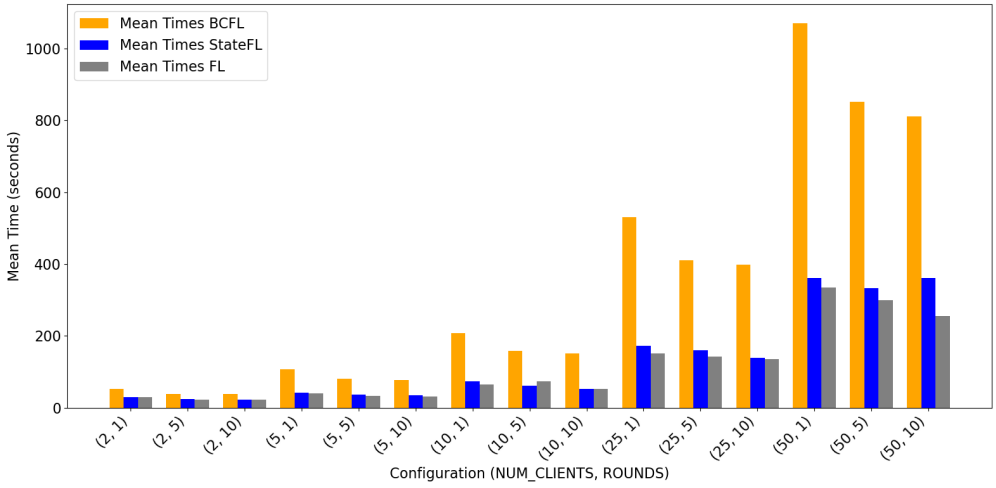

Figure 12: **Mean round time**

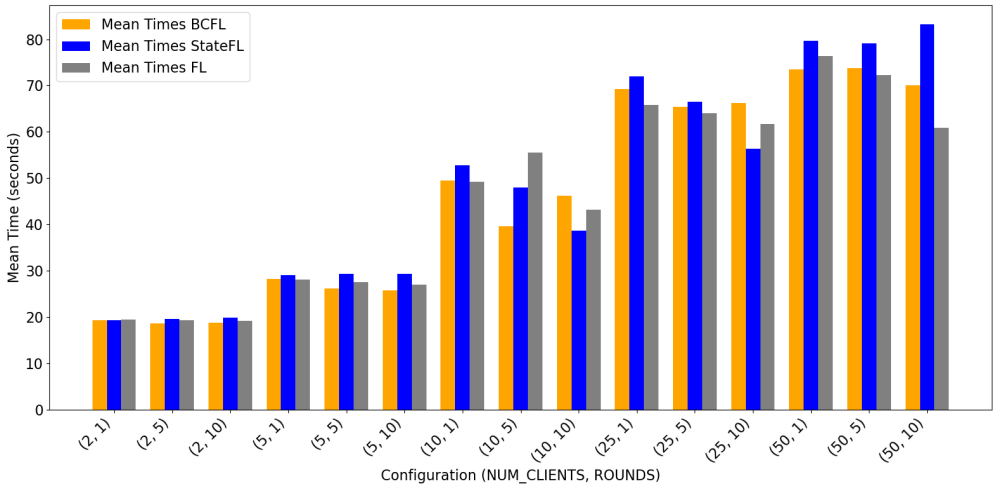

Figure 13: **Mean training time per client**