DISCOFL - DISTRIBUTED INCENTIVE SYSTEM FOR COOPERATIVELY ORCHESTRATED FEDERATED LEARNING

André Gaillard, Giacomo Lombardo

Facultat de Informatica de Barcelona
Universitat Politecnica de Catalunya

ABSTRACT

Federated Learning is a novel machine learning paradigm in which a model is trained among distributed participants on local data. Aggregating the individual models with a central server or using decentralized techniques results in a final model that profits from all the local data of the user without having to share it. In this work, we present an architecture for decentralized Federated Learning that uses blockchain to distribute rewards among the participants. We define the notion of trust in such a system and show how our architecture implements trustworthiness. To support our claims, we deliver a prototype application that allows to simulate the full architecture.

1. INTRODUCTION

While the number of commodity devices increases rapidly, traditional Machine Learning techniques fail to leverage their computational capabilities. Instead, most of the computational power is concentrated in large agglomeration units such as data centers. Furthermore, increasing concerns around data privacy and legislation alike pose additional challenges to traditional approaches. A user might feel uncomfortable having data stored intransparently with big companies learning from it. At the same time, said storage of centralized data might be illegal for confidential data such as medical records.

As a response to these challenges, Federated Learning (FL) has emerged as a new machine learning paradigm. In Federated Learning, the global model is trained locally without sharing the training data but only the model. The individual contributions to a global model are then aggregated. At the end of the process a fully trained, global model is obtained. This new paradigm is growing in popularity and it is already used in a number of scenarios of our everyday life. For instance, FL is particularly suited for the training of vocal assistants such as Apple's Siri [1] and Amazon Alexa [2], where sharing sensible data such as the users' voice might pose several privacy threats.

Since the introduction of FL, a number of proposals and takes on this new paradigm have been published, introducing various possibilities of implementation and different architectures. These can be classified in two macro-categories: *Centralised* and *Decentralised* FL. The main difference between these architectures is the presence or absence of a central server coordinating the FL process. A conceptual overview of these two can be found in Figure 1.

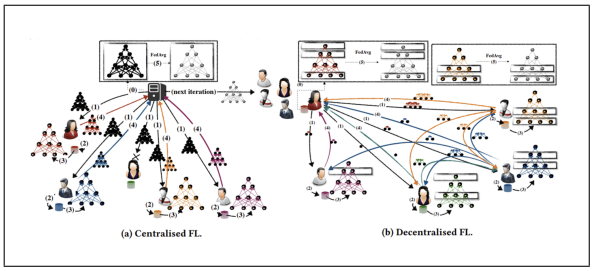

Fig. 1: Overview of Centralised and Decentralised FL architectures.

In centralised FL architectures, the models trained locally by the participants are sent at the end of each round to a central server, which aggregates them and sends back the new model for the new round. This solution heavily facilitates the inclusion of new participants during the process, since they can simply join the FL task by requesting the last global model and be ready to start the training. Additionally, a central server can easily monitor the activity of participants to prevent malicious behaviors that could harm the final result. At the same time, however, the presence of a central server introduces a single point of failure into the training process. Another problem that arises in centralised FL is that the central server must be trusted by every participant. In some scenarios, such as healthcare [3], this cannot be easily obtained.

Contrarily, in decentralised FL the central server is replaced by distributed coordination algorithms and technologies that enable participant to cooperate in training the global model. While decentralised FL solves the problems that can arise in centralised architectures, they come with a number of challenges that need to be addressed. For instance, round coordination is not trivial and fair behaviour must be enforced in a distributed way. Furthermore, a central server

provides computational and storage resources that have to be replaced in some way.

A growing trend in the field of decentralised FL is the inclusion of blockchain technologies to coordinate the FL task as well as to offer incentives for fair behavior from participants. The characteristics of blockchain, i.e consensus, distributedness and immutability, seem particularly suited for FL and a number of proposals on the topic have been presented in recent years [4]. In this paper, we present DISCOFL, a Decentralised FL system with a blockchain-based incentive system. It is a novel architecture that opts to overcome the challenge of enforcing trust in a distributed FL system using blockchain. Not only do we design a novel architecture, we also develop a prototype implementation. After providing a brief overview of related work in the first section, we present the core part of this work, our novel architecture. We then advance to define and analyze our underlying concept of trust and how it is enforced by our architecture. Finally we give a brief overview of the implementation and analyse experimental results to verify the validity of our proposal.

2. RELATED WORK

Among the various proposals of blockchain-based FL, we center our attention on three contributions which inspired our work.

The authors of [5] propose a FL architecture fully built on top of the Ethereum blockchain, proposing to store the global model in the blockchain and a bidding system to allow participants to update it. The main challenge with this approach is the limited storage space of the smart contracts, requiring sophisticated chunking algorithms to store the model on the blockchain.

In [6], on the other hand, the authors propose a FL architecture combining blockchain to incentivize participants' good behavior as well as IPFS [7] to host the models. Furthermore, they focus on providing a contribution scoring procedure and other features such as differential privacy in order to enhance the robustness and the trustworthiness of the system.

In contrast to a contribution scoring procedures, the authors of [8] propose a blockchain-based incentive mechanism to enable trustworthy FL by introducing a competition system between the task participants.

Our proposal is based on the innovations proposed in these three papers and combines features of these systems with the goal of ensuring trust among the participants in the system.

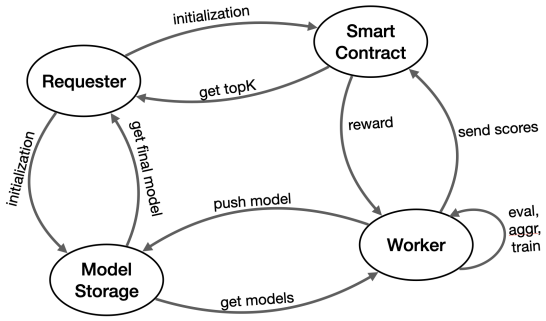

Fig. 2: Overview of DiscoFL components and their interactions.

3. ARCHITECTURE

In this section we provide a complete overview of the DISCOFL architecture. Our architecture consists of *actors* that provide the computational power of their commodity devices. To coordinate the actors, the architecture contains so-called *components*. We proceed by defining these underlying parts of the architecture. Given the key parts of the architecture, we then describe the interaction between each of them.

3.1. Components

In order to coordinate actors and to store the DISCOFL task information, the DISCOFL architecture is based on two main components:

Smart Contract. The smart contract contains the necessary information to coordinate the FL task and distributes the rewards among its participants. The SC also stores information about the participants and the model evaluations sent by them at the end of each round.

Model Storage. The model storage contains the models trained by the participants at the end of each round. This component can be implemented in a variety of ways, ranging from a centralised model storage to a fully decentralised one based on IPFS or even on blockchain.

3.2. Actors

At the base of every DISCOFL task, two classes of actors are to be distinguished:

Requester. The requester initiates a DISCOFL task by deploying the smart contract and pushing the initial model to the model storage. It also specifies additional parameters such as the number of rounds and the reward to be distributed among the participants. Since DISCOFL is completely model agnostic, it allows the requester to push an arbitrary model for an arbitrary task to the model storage.

Workers. Workers take part in the FL task created by the requester. They train the model through a round-based

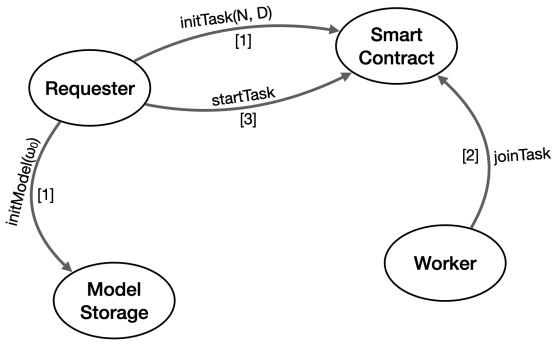

Fig. 3: Interactions during the starting phase of the task.

system on their own data and gain rewards according to their performance.

3.3. FL Task Structure

To better understand the interactions between the various actors and components of DISCOFL depicted in Figure 2, we split the DISCOFL task in three main phases: Start, Training and End. For each of them, we provide a detailed description of the various actions taking place.

3.3.1. Start phase

To begin a DISCOFL task, the requester deploys the smart contract and initializes it by specifying two main parameters: i) the number of rounds N of the FL task training phase and ii) the total amount D to be distributed to workers as a reward. Additionally, the requester also pushes the initial model to the model storage w_0 which will be retrieved by the workers at the beginning of the training phase and acts as a basis for the structure of the trained models. w_0 does not only contain the initial parameters, but also the structure of the model (i.e the number of layers and so on). Once the task has been initialized, workers can join it by a simple interaction with the smart contract, which stores their address and notifies the requester. As soon as the task has gathered enough workers or a certain amount of time has passed, the requester triggers the training phase through the smart contract. An overview of this phase can be found in Figure 3.

3.3.2. Training phase

The goal of the training phase is to obtain a fully trained final model which will then be retrieved by the requester at the end of the DISCOFL task for the use of downstream tasks. During this phase, workers train the model for the number of rounds specified by the requester during the starting phase on their local data. The whole process is illustrated in Figure 4

Each round starts with the workers retrieving the trained models of all other worker of the previous round from the model storage to evaluate it on the local data. This allows the worker to assign a score the models of all other workers. This score is then pushed to the smart contract. After the evaluation, each worker aggregates the retrieved model parameters (including its own) and trains on it. After training, workers push the newly obtained model to the model storage.

In the meantime, the smart contract aggregates the submitted scores in order to obtain the Top K (K being a tuneable parameter) of the best performing workers in the previous round. The Top K is then used to distribute rewards to the workers according to their performance. This process repeats for N rounds.

We want to note here that during the training phase the requester is not involved in any interaction. Given the limited computational capabilities of the smart contract, however, some operations such as score aggregation and the calculation of the previous round's Top K can be offloaded from the smart contract to the requester machine. In this way, not only we do not waste the computational resources but we could also opt for more complicated computations than simple Top K. Furthermore, we save on transaction costs.

3.3.3. Ending phase

Once the training phase has concluded, the requester can retrieve the final global model from the model storage and close the task by simply calling a function of the smart contract as depicted in Figure 5.

4. TRUST

Once defined the architecture of DISCOFL, we analyze why such a system is trustworthy and ensures good behavior of its participants. In this section, we are going to provide our definition of trust and explain how such a definition applies to DISCOFL.

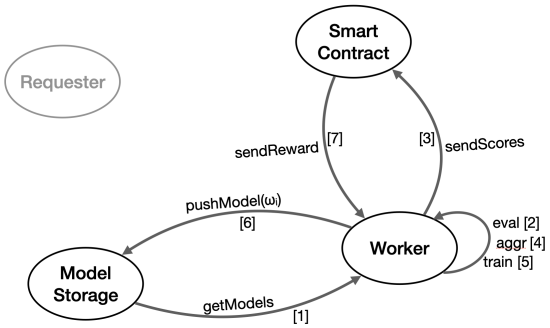

Fig. 4: Interactions during the training phase of the task.

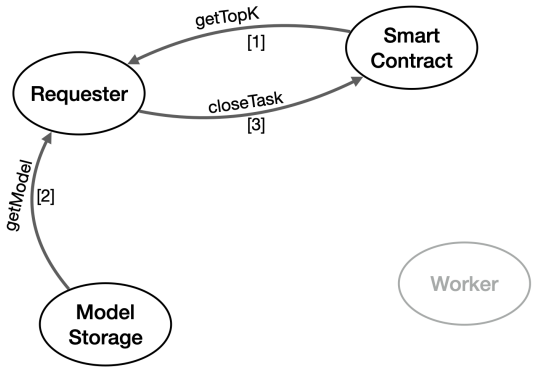

Fig. 5: Interactions during the ending phase of the task.

4.1. Trust Definition

To understand if a system can be considered trustworthy it is necessary to define what trust is in the system's context. We give the following as the definition of a trustworthy system:

Definition 1 *A decentralised FL system is* ***trustworthy*** *if and only if it ensures fair behavior of its participants and rewards them according to their contribution to the task.*

As per this definition, the trustworthiness of the DISCOFL system can be grouped into two main aspects: i) fair behavior of participants and ii) contribution-based reward distribution. Concerning the fair behavior of participants, the systems needs to be able to identify malicious participants and avoid that their behavior affects the overall result of the task. On the other hand, participants need to be incentivized to contribute to the FL task in a fair and effective manner to obtain the best final result. In the following sections we will analyze how these two aspects are implemented in DISCOFL.

4.2. Fair behavior

According to Definition 1, a trustworthy system needs to ensure that participants behave fairly. This means identifying malicious users and limit their impact on the final result of the FL task. As mentioned before, in decentralised FL architectures the absence of a central server poses a non-trivial challenge to this requirement. In DISCOFL, we focus on preventing two types of malicious activity: i) poisoning the global model with poor quality submissions (i.e training on garbage data sets or training on data from a different distribution than intended) and ii) alter the reward system by submitting corrupted evaluations.

We prevent the poisoning of the model by introducing a peer-to-peer evaluation system. In each training round, every worker evaluates all the other workers' models submitted in the previous round and assigns a score to each of them. This score is assigned in accordance with the model's performance on the worker's local dataset which is split into validation and training set. If a model performs poorly on the validation set, a low score is assigned to it. In this way, if a worker submits a low-quality model, such model will receive low scores resulting in a low aggregated score. A low scoring model will then be less influential in the model aggregation phase, preventing model poisoning.

A big part of DISCOFL's trustworthiness relies on the evaluation mechanism. Such mechanism can also be the target of malicious behavior. For instance, a worker may intentionally assign low scores to the other workers' models in order to privilege its model in the score aggregation phase. Such behaviour is prevented by implementing the BlockFLow [6] *contribution scoring procedure*. This algorithm prevents malicious workers affecting the final result by assigning low overall scores to both i) workers who submit a poor-quality model and ii) workers who assign scores that differ significantly from the median of the scores given by the other workers for this model.

Another malicious behavior that could affect the scoring procedure is to privilege another worker's model in the evaluation phase. For instance, a group of workers could intentionally assign to each other higher scores and privilege each other to obtain higher aggregated scores hence higher rewards. Such behaviors are prevented by anonymizing the workers' models pushed to the model storage. In practice, at every round the smart contracts maps to every worker address a random ID which is then associated to the worker's model. In this way, during evaluation phase it is not possible to retrieve the worker's identity hence one cannot assign higher (or lower) scores to a given worker on purpose. Once the scores are submitted, the smart contract then maps them to the worker's real address in order to distribute the rewards.

4.3. Reward Distribution

To incentivize participation and fair behaviour, the system also needs to reward participants according to their contribution. In other words, participants must be incentivized to behave correctly and contribute as much as possible to the completion of the task.

To do so, DISCOFL implements an incentive system based on the distribution of rewards in the form of (crypto-)currency. As explained in section 3.3.1, when initializing a task the requester deposits an arbitrary amount of currency into the smart contract which will then be distributed to the workers during the training phase. The rewards are distributed depending on the scores assigned to each worker from its peer workers. Once these scores are aggregated, it is possible to retrieve the Top K of the workers who submitted the best performing models. The task of computing the Top K can be implemented either directly on the smart contract or on the requester, depending on the complexity

of the task. Once the round Top K is defined, it is used to distribute the ratio D/N of the currency present in the smart contract among the workers, where D is the initial deposit and N is the number of tasks.

The current reward distribution algorithm assigns each worker $\frac{1}{2^i}\%$ of the total amount to be distributed in each round, where i is the position of the worker in the round top k. The only exception to this rule applies to the second-to-last and third-to-last workers who receive respectively $(\frac{1}{2^{N-2}} + \frac{1}{2^{N-2}N})\%$ and $(\frac{1}{2^{N-2}} - \frac{1}{2^{N-2}N})\%$ of the total amount, where N is the total number of workers participating to the round. The worst performing worker receives no reward at all. The reason for that is to prevent workers from being idle during the training while still receiving rewards. Applying this algorithm to a Top K of size $N = 5$ would result in the following reward distribution among the workers: $1\text{st} \leftarrow 50\%, 2nd \leftarrow 25\%, 3rd \leftarrow 15\%, 4th \leftarrow 10\%, 5th \leftarrow 0\%$.

5. IMPLEMENTATION

In this section, we briefly describe the most important parts of our implementation[1]. In particular, we will look at the implementation of the machine learning and the blockchain separately. The workers and the requester are omitted, since their implementation consists mostly of function calls.

It is important to note here that due to a lack of infrastructure, the whole prototype is implemented to simulate the distributed nature. In particular, all the workers as well as the requester run on the same machine after each other. However, the overall order of the training procedure as detailed in section 3 persists. For the model storage, we currently use the local file system. Conceptually, porting this implementation to a distributed system is easy, with a few implementation details such as timing.

5.1. Machine Learning

The prototype is implemented by using Pytorch [9]. Its scripting system allows trainable models to be loaded from a file without knowing the actual class. Consequently, our implementation is completely model agnostic. This means that the requester can distribute any initial model it may please, be it convolutional neural networks, graph neural networks and so on.

Since the training happens locally without chunking of the model, the actual implementation of the machine learning part does not differ from standard approaches using Pytorch. Referring to validate-train splits, we implement a random split of the data, using 80% of the dataset for training and 20% for validating.

[1]The source code is available on request at `https://github.com/andregail/DS-FinalProject`

In terms of aggregation scheme, we use a simple weighted average according to the scores given to the other workers in the evaluation phase.

5.2. Smart Contract

Once the Machine Learning process has been implemented, it is still necessary to coordinate the various actors taking part to the task in order to obtain the final result. As stated before, in DISCOFL this is performed through a Smart Contract, whose main functionalities are i) round coordination, ii) score aggregation and iii) reward distribution. The Smart Contract is developed in Solidity and deployed on the Ethereum blockchain by the task requester. In our first prototype, given the absence of a network infrastructure, the SC is deployed locally on a Ganache test-net and the application interacts with it via the *web3.py* library.

One crucial aspect that we need to take into account when developing and working with smart contracts is that they have very limited space and computational power. Due to this reason it is not trivial to implement even quite simple algorithms like calculating a Top K, let alone score aggregations or storing a ML model. To circumvent these limitations, we can move the tasks requiring more computational resources from the SC to the machines of the requester and the workers. For instance, as already discussed in Section 3.3.2, during the training phase the requester machine sits idle resulting in a waste of computing resources. If we implement the contribution scoring procedure on the requester machine, we can first of all make an efficient use of these resources and furthermore we can obtain better performances by implementing such algorithm in a Python script rather than in Solidity.

Generally speaking, actors interact with the smart contract to store/retrieve information regarding the FL task in progress. Workers interact with it by retrieving general information such as round number and model URI, pushing their scores at the end of the evaluation phase and receiving rewards at the end of each round. Requesters, in addition to deploying the smart contract, interact with it by initializing the task, retrieving the scores pushed by the workers and pushing the round top k.

6. EXPERIMENTAL RESULTS

To validate our proposal we conduct two experiments with the implemented prototype. The goal of these experiments is to verify whether the model is trained correctly through FL and if the system is resistant to malicious users. We first describe the experiment setup and then analyze the results.

6.1. Experiment Setup

We use our sequential prototype to simulate the distributed behaviour. We opt to solve the problem of classifying the

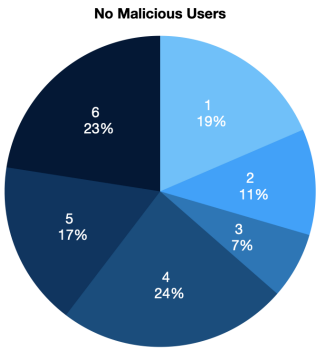

Fig. 6: Reward distribution graph in the experiment without malicious users.

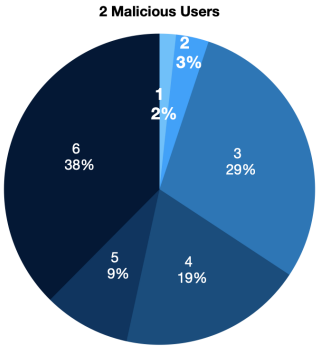

Fig. 7: Reward distribution graph in the experiment with two malicious users (users 1 and 2).

MNIST dataset [10]. To do so, we choose a standard convolutional neural network. With a single worker training on the whole dataset, it scores 97% in accuracy on the test set. We test our approach to distributed machine learning with weighted averaging with up to clients and have not observed significant decrease in performance. The mathematical analysis of our approach and more thorough testing of our aggregation scheme is out of scope of this work.

Given W workers, we randomly split the MNIST test and training dataset into W splits. Each worker gets assigned its own test and training split to conduct the local training on. For our experiments, we choose $W = 6$ and we train the model in 5 rounds. We execute two experiments. The first one has only benevolent users, i.e all users train on their split of the MNIST data set. The second one has two malicious users, meaning that they train on randomly generated data.

The initial deposit is set at 10 ETH, which will be distributed among the W workers. Such a large amount of currency helps to better track the reward distribution and have the final results less affected by the presence of transaction fees.

6.2. Results

The results are shown in Figure 6 and Figure 7. We can see in figure 6 that without malicious workers, the rewards are distributed much more evenly. However, due to our reward distribution function as well as the underlying randomness (i.e some splits might be suited better than others), a uniform distribution is not achieved. In figure 7 on the other hand, it can instantly be read off which users are malicious. Alternating between the last and the second-to-last place, they are still able to get some of the rewards, albeit almost negligible. As the number of workers grow, we expect the discrepancy of rewards earned between benevolent and malicious workers to grow. In terms of performance, the best models of both experiments ended up at around 92% accuracy. This means that even with malicious users, the procedure stays robust in terms of learning.

In summary, the results of the experiments support our claim that incentive system is indeed working as expected. Given the computational power one has to invest together with the transaction costs, the rewards reaped for the malicious users do probably not pay off. At the same time, the top-performers earn a large amount of the distributed share showing that honestly contributing to the training process pays off.

7. FUTURE WORK

The experimental results suggest that the implemented DISCOFL proof-of-concept has the potential to be extended to work on a larger scale, ultimately becoming a FL platform with real-world use cases. To achieve this goal, there are a number of features that haven't been implemented yet or that simply need further improvements. In this section we are going to briefly describe the main aspects that should be addressed in future work.

Network Implementation. The implemented prototype works locally simulating a network of users by sequentially employing workers in each round. In a real scenario, workers participate together and the actors communicate through a network. This poses challenges as the implementation of synchronization mechanisms that still need to be implemented.

Model Storage Decentralization. At the actual state, models are stored in a centralized data structure introducing a single point of failure in the architecture. In future implementations this can be replaced with a decentralised storage such as IPFS.

Cryptography and Anonymity. The current prototype does not yet implement the model anonymization system discussed in Section 4.2. Furthermore, it does not guarantee that the model are kept secret and external users can read them if they manage to access the model storage. In future implementations models will be anonymized and encrypted.

Model Aggregation Algorithm. While the standard weighted average aggregation mechanism we deploy exhibits satisfying performance, it is far from optimal. Consequently, we could implement a set of more sophisticated aggregation algorithms that are fine-tuned depending of which model is used, how many workers there are, and so on. Furthermore, the aggregation requires for careful testing and experimenting.

Reward Distribution Algorithm. The current reward distribution algorithm does not scale well with the number of participants. In real scenarios we can expect thousands or more workers participating: with the actual reward distribution one worker would always receive 50% of the round total rewards and only one worker receives no reward at all. Additionally, workers should be rewarded more staying for more rounds to prevent workers to train only once, reap the fruits and then leave the training process.

8. CONCLUSION

In this work, we designed a novel architecture for distributed Federated Learning based on reward-distribution through a blockchain. We defined the notion of trust in such a system and showed how our architecture fulfills this notion theoretically. To experimentally verify our claims, we implemented a simplified proof-of-concept application and conducted a set of experiments. We conclude from our experiments that our architecture fulfills the expected properties. In particular, we found that an attempt to poison the model does not pay off, neither in reward collection nor in hampering the overall training process. The implementation is completely model agnostic and can be used for any model on any dataset. While the prototype exhibits promising results, we acknowledge that it still suffers from limitations especially in regard to its sequential implementation. However, we are confident that we have laid the foundation for new distributed federated learning applications, advocating the shift from centralized to decentralized federated learning.

9. REFERENCES

[1] Karen Hai, "How apple personalizes siri without hoovering up your data," December 2019.

[2] Christophe Dupuy, Jwala Dhamala, and Rahul Gupta, "Advances in trustworthy machine learning at alexa ai," April 2022.

[3] Abhijit Guha Roy, Shayan Siddiqui, Sebastian Pölsterl, Nassir Navab, and Christian Wachinger, "Braintorrent: A peer-to-peer environment for decentralized federated learning," 2019.

[4] Zhilin Wang and Qin Hu, "Blockchain-based federated learning: A comprehensive survey," 2021.

[5] Paritosh Ramanan and Kiyoshi Nakayama, "Baffle : Blockchain based aggregator free federated learning," 2019.

[6] Vaikkunth Mugunthan, Ravi Rahman, and Lalana Kagal, "Blockflow: An accountable and privacy-preserving solution for federated learning," 2020.

[7] Juan Benet, "Ipfs - content addressed, versioned, p2p file system," 2014.

[8] Kentaroh Toyoda and Allan N. Zhang, "Mechanism design for an incentive-aware blockchain-enabled federated learning platform," in *2019 IEEE International Conference on Big Data (Big Data)*, 2019, pp. 395–403.

[9] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds., pp. 8024–8035. Curran Associates, Inc., 2019.

[10] Yann LeCun and Corinna Cortes, "MNIST handwritten digit database," 2010.