Execution of a Federated Learning process within a smart contract

Andrew R. Short
Dept. of Industrial Design
and Production Engineering
University of West Attica
Athens, Greece
ashort@uniwa.gr

Helen C. Leligou
Dept. of Industrial Design
and Production Engineering
University of West Attica
Athens, Greece
e.leligkou@uniwa.gr

Efstathios Theocharis
Dept. of Industrial Design
and Production Engineering
University of West Attica
Athens, Greece
stheo@uniwa.gr

2021 IEEE International Conference on Consumer Electronics (ICCE) | 978-1-7281-9766-1/20/$31.00 ©2021 IEEE | DOI: 10.1109/ICCE50685.2021.9427734

***Abstract*—High quality datasets have always been valuable for the creation of Machine Learning (ML) models. It therefore makes sense to provide rewards to users that participate in a Federated Learning (FL) process with such datasets. In this competitive scene, we design a solution that leverages a blockchain network, a smart contract and a model verification algorithm in order to coordinate the training process, record user performance and provide rewards in a transparent manner.**

Keywords—Blockchain, federated learning, smart contract, verification algorithms

I. Introduction

Federated Learning (FL) is a promising technology used for machine learning (ML), allowing for a model to be trained in a decentralized manner. Since participating users only share gradients (in the form of model updates) instead of actual data, this process offers improvements in terms of privacy. In recent years, researchers are proposing ways to couple the FL process together with blockchain technologies in order to offer improvements in different aspects. More specifically, relevant research has shown that such solutions can offer improved auditability [1][2], accountability [3], are able to store model updates in a decentralized manner [4], improve security [5] and can be used to facilitate incentive mechanisms [6].

This paper proposes and specifies the architectural design of such a solution, where the FL process is coordinated by a smart contract running on a private blockchain network. The smart contract is responsible to verify contributions, calculate the global model, and to record users' performance in order to support incentive schemes. To the best of our knowledge, a similar approach (i.e. where a verification algorithm runs within a smart contract in order to verify model updates and in turn provide rewards/incentives) has not been proposed or studied. It is worth mentioning that some terminology used in this paper is HyperLedger Fabric (HLF) specific; however, similar principles may apply to other private blockchain networks. The solution described in this paper aims to offer improvements in the following areas:

A. Security

Running a model update verification algorithm inside a smart contract can protect against model poisoning attacks [7]. In such an attack scenario, an adversary may send rogue model updates in order to affect the global model's performance, alter its behavior or prevent other users from accessing it [8]. In solutions that utilize some sort of incentive scheme, this kind of verification process is required. Otherwise, a malicious user could benefit from such a scheme by sending empty or otherwise worthless contributions, the same time degrading the performance of the global model.

B. Incentive Schemes

In order to enable trust, the smart contract can be used to record each participating user's performance. A distributed ledger of a blockchain network is a perfect medium to store rewards in a transparent way, due to the fact that it is decentralized. For instance, HLF can be configured so that at least a certain percentage of nodes are required to reach consensus. In addition, by executing the verification algorithm inside a smart contract, a higher level of transparency is achieved which is especially important in our setup, as the output of this algorithm is used to calculate incentives.

We envision that the solution will be valuable in situations where a higher level of trust is required, because it can enable competing entities to collaboratively improve and use a shared machine learning while also giving prominence to active users. The placement of the blockchain nodes depends on the use case. For instance, when a small number of competing industries or associations cooperate in a FL process, they may choose to each host a node (i.e. the node is closer to the end user). Another option would require a trusted third party to host the solution.

The following sections describe the algorithm used for verification and reward calculation (section II), the workflow of the training process when this is coordinated by a smart contract (section III), examples of incentive schemes (section IV), specifications of the functions that need to be supported by the smart contract (section V), the specifications for the distributed ledger data structure (section VI) followed by conclusions and future work (section VII).

II. Verification of Contributions

The calculation of the next global model version in a FL process typically involves averaging the model weights received from all contributions while also taking into account the number of samples that each client possesses [9]. However, since training in a FL process is performed at the edge, data is not transmitted and is not collected at a central location. It is therefore challenging to assess the honesty of a user's statement regarding the amount of data used during the training process. An attacker may take advantage of this inability, falsely report

This work was supported by the University of West Attica

that he used large data samples, in order to maximize his influence on the global model. It has been demonstrated that a known-good verification data set (kept private from end users) along with a verification algorithm can be used to evaluate individual model contributions, without relying on data sample size numbers, thus providing a level of protection from the aforementioned type of attack [7].

Our proposed solution also uses the verification algorithm in the decision-making process, depending on the incentive scheme used. In the simplest form, we can offer rewards to users, whose contributions were able to pass the verification tests.

III. Overview of the Learning Process (Workflow)

Each training round comprises of three main processes. During the first process, participating users request and receive the weights of the current model version. During this step, the smart contract is invoked, which in turn queries the ledger for the relevant data. It should be noted that the ledger is not altered at this stage.

After downloading the latest model version, the end-user is able to use it, as well as to perform local training. The resulting gradients (in the form of model weights) are then sent back to the blockchain network. During this step, a function of the smart contract is invoked, which is responsible for the following tasks: a) receives the gradients as an input from the end-user, b) verifies the quality of the update by executing a model update verification algorithm, c) depending on the results of the verification algorithm, either discard or save user contributions on the ledger.

The last process in the FL workflow is to perform federated aggregation e.g. by averaging all meaningful contributions stored in the ledger and use the result to create the next model version. This process should be triggered to execute at pre-defined intervals depending on the specific use-case e.g. when a certain contribution goal has been reached or when a certain time has elapsed. If the training round is very short, there might not be enough contributions to produce a model with significant improvements. On the other hand, if the training round is very long, users will be delayed in downloading a better-performing version of the model and as a consequence the learning performance will decrease. This last process concludes the training round.

IV. Possibilities for Rewards

After each training round is complete, it is possible to record the performance of each user (in terms of useful contributions) in the ledger. During the training process, user contributions are evaluated against a verification algorithm. The output of this verification function can be used as a metric of users' performance. In the simplest form, the distributed ledger will store increments, that represent the number of successful contributions (i.e. model updates that were deemed useful by the verification algorithm). These scores can be perceived the same way as cryptocurrency tokens, in the sense that they are stored on a secure medium, and can be exchanged for rewards. In a commercial setting, the tokens can be exchanged with money in order to compensate the user for his effort in performing local training and for supplying the FL process with usefull model updates. In other use cases, tokens could be exchanged for additional services e.g. access to download and use supplementary Machine Learning (ML) models. Alternatively, the lack of tokens could be perceived as an indication of misbehaving or unproductive users and therefore potentially be used to prohibit them from further participating in the federated learning process.

V. Smart Contract Functionality

This section discusses the functionality that should be implemented in the smart contract as a minimum, in order to support the workflow discussed in the previous section.

The first function is responsible for enabling participating users to acquire the latest model version, during the first phase of the training round. The function is defined as GetLatestModelParameters(), queries the ledger and returns a multidimensional table of weights corresponding to the latest model version. This function is invoked once by each participating user. However, since only a simple query is performed against the ledger, we expect that the computational requirements will be low.

Figure 1 below shows interactions between users, smart contract and the distributed ledger during the early stages of the training round (download of model weights).

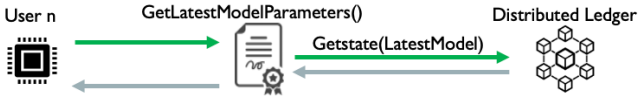

Fig. 1. Interactions between users, smart contract and the distributed ledger – Request of current model version (model weights)

The second smart contract function SubmitModelWeights(weights) is required to facilitate the functions of the second step of the training round (i.e. when a training round is active). When executed, it will receive model updates from the user as an input (in the form of weights), it will then run the verification algorithm, and finally store the contributions to the ledger. We define "weights" in the function above as a multidimensional table of the model weights, corresponding to an individual contribution. This function is also invoked by the user, and can optionally return the contribution assessment result which is derived by the verification algorithm. Two approaches are considered for storing contributions to the ledger. The first approach involves only storing successful contributions i.e. those that improve the accuracy of the model. Another option is to store all contributions along with a metric (derived by the verification algorithm). Depending on the specific use case, we may choose the first approach because it offers performance benefits (since the ledger is being written less often), or the second method which offers higher auditability. In both of these cases, it is anticipated that a moderate amount of computational resources will be required, since the function is called by each participating user, calls external CPU-intensive ML verification libraries, and is expected to affect the state of the ledger.

Figure 2 below shows interactions between users, smart contract and the distributed ledger during the submission of model updates from the users.

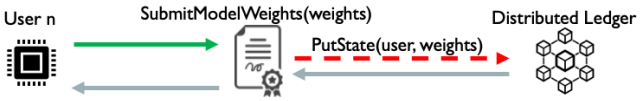

Fig. 2. Interactions between users, smart contract and the distributed ledger – Submission of model weights (user contributions)

The last function CalculateNextModelVersion() is required in order to create the next model version and by doing so, end the current training round. Unlike the previous two functions, it is not invoked by an end user. For this reason, an external trigger should be created as discussed in the previous section. The next model version is created by performing federated averaging among contributions saved in the ledger. The resulted multidimensional matrix is stored in the ledger in order for it to be used during the next training round. Although this function writes to the ledger, it is called only once every training round. Along with the fact that it uses simple mathematical matrix calculations, it is not anticipated to require many computational resources.

Figure 3 below illustrates the interaction between the smart contract and the distributed ledger during the creation of the next model version.

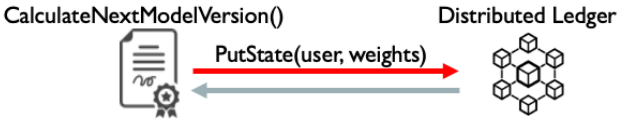

Fig. 3. Interactions between smart contract and the distributed ledger for the creation of the next model version (end of training round)

The table below summarizes the characteristics of the smart contract functions, which are used to predict resources requirements. TR denotes the Training Round.

TABLE I. RESOURCES REQUIREMENTS FOR FUNCTIONS

Function	Attributes		
	Execution Frequency	*Alters the ledger*	*Anticipation of resources required*
GetLatestModelParameters	once per user per TR[a]	NO	LOW
SubmitModelWeights	once per user per TR[a]	YES	HIGH
CalculateNextModelVersion	once per TR[a]	YES	LOW

[a.] Training Round

VI. DISTRIBUTED LEDGER OPERATIONS AND DATA STRUCTURE

The proposed architectural diagram is based on the concept of key-value pair databases. Such type is used as a state database by HyperLedger Fabric (HLF). Smart contract operations are used to read or write to the current state of the database. Each ledger transaction will either create, update or delete (and therefore modify) key-value pairs in the ledger. However, since HLF belongs to the blockchain family, an immutable sequence of blocks contains all transactions that have affected the current state of the database. The copy of the ledger is propagated to all network members.

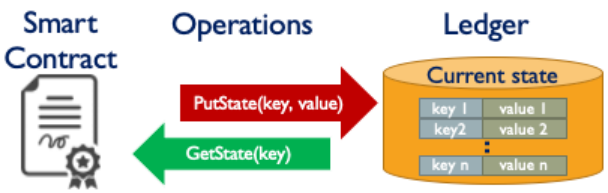

Fig. 4. Operations that access/modify the distributed ledger

The table below summarizes the key-value pairs that will be required as a minimum for the operation of the federated learning process as well as for recording rewards. The key "identity" below refers to array values. For example, we will provision one key-value pair of *ModelUpdate* and *Reward* variables for each participating user.

Key	*Value*
ModelWeights	Multidimensional matrix containing weights for current model version
ModelUpdate(identity)	Multidimensional matrix containing weights (one for each [identity])
Reward(identity)	Value proportional to quantity/quality of user contribution (one for each [identity])

VII. CONCLUSIONS AND FUTURE WORK

The solution presented in this paper describes how a private blockchain network can be used in a unique way in order to support a FL process. It improves trust and security by running a verification algorithm within a smart contract which is executed amongst multiple nodes. The network can be configured to require consensus among a certain percentage of nodes.

In the future we intend to implement the solution using the following tools and technologies: a) HyperLedger Fabric, b) an algorithm to verify updates and provide rewards [7] and c) Tensorflow (opensource machine learning tools). Areas of interest include i) the feasibility of running machine learning libraries within a smart contract, ii) implications in terms of consensus when the results of the verification algorithm are not exactly the same between nodes (e.g. rounding errors), iii) performance overhead imposed by the blockchain network, iv) impact of the different design choice on performance (e.g. logging all contributions to the ledger vs logging only useful contributions, v) specification of hardware (computational, memory) requirements of the solution.

ACKNOWLEDGMENT

All authors would like to thank the University of West Attica for the financial support provided to them to undertake this research project.

REFERENCES

[1] J. Passerat-Palmbach, T. Farnan, R. Miller, M. S. Gross, H. L. Flannery, and B. Gleim, "A blockchain-orchestrated Federated Learning architecture for healthcare consortia," Oct. 2019, arXiv:1910.12603

[2] J. Weng, J. Weng, J. Zhang, M. Li, Y. Zhang, and W. Luo, "DeepChain: Auditable and Privacy-Preserving Deep Learning with Blockchain-based Incentive," *IEEE Trans. Dependable Secur. Comput.*, vol. PP, no. 8, pp. 1–1, 2019, doi: 10.1109/tdsc.2019.2952332.

[3] D. Preuveneers, V. Rimmer, I. Tsingenopoulos, J. Spooren, W. Joosen, and E. Ilie-Zudor, "Chained Anomaly Detection Models for Federated Learning: An Intrusion Detection Case Study," *Appl. Sci.*, vol. 8, no. 12, p. 2663, Dec. 2018, doi: 10.3390/app8122663.

[4] H. Kim, J. Park, M. Bennis, and S.-L. Kim, "Blockchained On-Device Federated Learning," *IEEE Commun. Lett.*, vol. PP, no. c, pp. 1–1, 2019, doi: 10.1109/LCOMM.2019.2921755.

[5] U. Majeed and C. S. Hong, "FLchain: Federated Learning via MEC-enabled Blockchain Network," in *2019 20th Asia-Pacific Network Operations and Management Symposium (APNOMS)*, 2019, no. September, pp. 1–4, doi: 10.23919/APNOMS.2019.8892848.

[6] J. Kang, Z. Xiong, and D. Niyato, "Incentive Mechanism for Reliable Federated Learning : A Joint Optimization Approach to Combining Reputation and Contract Theory," *IEEE Internet Things J.*, vol. PP, no. c, p. 1, 2019, doi: 10.1109/JIOT.2019.2940820.

[7] A. R. Short, H. C. Leligou, M. Papoutsidakis and E. Theocharis, "Using Blockchain Technologies to Improve Security in Federated Learning Systems", 2020 IEEE 44th Annual Computers, Software, and Applications Conference (COMPSAC), doi: 10.1109/COMPSAC48688.2020.00-96

[8] E. Bagdasaryan, A. Veit, Y. Hua, D. Estrin, and V. Shmatikov, "How To Backdoor Federated Learning," Jul. 2018. arXiv:1807.00459

[9] F. Sattler, S. Wiedemann, K. R. Muller, and W. Samek, "Robust and Communication-Efficient Federated Learning from Non-i.i.d. Data," *IEEE Trans. Neural Networks Learn. Syst.*, vol. 31, no. 9, pp. 3400–3413, 2020, doi: 10.1109/TNNLS.2019.2944481.