Cross-Chain Training of Learning Models via Blockchain Interoperability

A thesis submitted in partial fulfillment of the requirements for the degree of

Masters of Technology (Dual Degree)

in

Computer Science and Engineering

by

Sarthak Chakraborty
(Roll No. 16CS30044)

Under the guidance of
Dr. Sandip Chakraborty

Computer Science and Engineering
INDIAN INSTITUTE OF TECHNOLOGY KHARAGPUR
West Bengal, India
May, 2020

Certificate

*This is to certify that the work contained in this thesis entitled "**Cross-Chain Training of Learning Models via Blockchain Interoperability**" is a bonafide work of **Sarthak Chakraborty (Roll no. 16CS30044)**, carried out in the Department of Computer Science and Engineering, Indian Institute of Technology Kharagpur under my supervision and that it has not been submitted elsewhere for a degree.*

Dr. Sandip Chakraborty
Assistant Professor
Department of Computer Science & Engineering
Indian Institute of Technology Kharagpur,
West Bengal

November, 2020
Kharagpur

Acknowledgements

I would like to thank my guide, Dr. Sandip Chakraborty for his exceptional guidance and support, without which this project would not have been possible. He has always motivated me to explore as much as I can, look into multiple papers and try as many ideas as I can. He has always supported me through whatever problems I faced during the project. I express my deep gratitude to Mr. Bishakh Chandra Ghosh for his guidance during the implementation of the project and for providing his valuable inputs and helping me in the project.

In conclusion, I recognize that this project would not have been possible without the support from the Department of Computer Science and Engineering, IIT Kharagpur. Many thanks to all those who made this project possible.

Sarthak Chakraborty

Abstract

Blockchains have been used increasingly in various domains including education, AI, health-care, supply chain and others. The primary reason for its success is the assured privacy and immutability of records. On the other hand, with an increase in the amount of user data, the paradigm of distributed machine learning has gained importance where a model is trained on local clients instead of training the model with complete data in a central server. Federated Learning has gained traction as the most popular distributed training method that also ensures security guarantees while training model on edge clients and can support training of a model with datasets having non-iid distribution.

Recently, to increase security and maintain privacy, Federated Learning has been coupled with blockchain where individual models are stored as a ledger by the clients and can be used to aggregate and update the global model. Researchers have proposed various methods to tackle this problem of coupling federated learning with blockchain. However, in all of the techniques, only a single blockchain network was used to train the model.

This report presents a method to incorporate interoperability of blockchains to ensure cross-chain training of a learning model. This might be a sought-after feature when multiple organizations under different blockchain networks might opt for a multi-task learning/transfer learning scheme for their models. The principle idea is to train a model in one blockchain and then transfer the state of the model to a foreign blockchain for further use. Thus, we have developed an end-to-end system architecture than can enable cross-chain training of a learning model via transfer of appropriate assets. This involves designing of a federated learning system intertwined with blockchain network, transfer of assets and collective signing of the transactions before transferring and then verification of the model when they are transferred across networks.

Contents

Chapter 1

Introduction

In recent times, Blockchain [1] has been one of the most sought-after innovation due to its wide applicability in various domains. Blockchain has been capturing the attention in industry as well as in academia due to the development of real-world applications [2, 3, 4]. Its applicability has been introduced in fields like internet of things [5], supply chain [6], health-care [7], education [8], financial markets [2], e-Commerce etc. Not only its sky-rocketing demand, but also the performance of blockchain is advancing towards delivering a performance similar to, if not surpassing the centralised systems [9]. However, researches in blockchain is still in a comparatively embryonic state, and the common people are unaware of the concepts behind the working of blockchain. Hence, blockchain systems still has a long way to go before getting adopted into the masses.

On the other hand, the field of Artificial Intelligence and Machine Learning has gained a lot of traction due to its unprecedented accuracy in various tasks, like medical research, autonomous vehicle driving, computer vision, path planning, etc. However, in order to achieve even higher precision and accuracy, huge amounts of data is required. Sometimes, the data is in itself sensitive(eg. medical data, personal information, etc.) and hence the authorities are not open to share the data. This has resulted in the development of distributed deep learning which has been investigated extensively in the recent years.

1.1 Motivation

Distributed deep learning has brought in the idea of privacy-preserving deep learning. Various models have been employed in the past, among which *Federated Learning* [10, 11] is the most popular and a widely adopted system. Federated Learning is a privacy-preserving, distributed machine learning model developed in 2017, where massive amount of decentralized datasets can be trained and complementary knowledge can be transferred

among distributed model trainers. In Federated learning, a coordinating server, also termed as the parameter server learns a global model by aggregating locally computed models uploaded by various clients which are trained on their local data [11]. More details on how a federated learning system performs the model training is discussed in section 2.1.

Despite numerous advantages, FL must guarantee the privacy of data and several methods like differential privacy have been developed to ensure it. While trained models are transferred to the server from the clients or vice versa, they are encrypted using Differential Privacy [12] where noises are added in the gradients that are uploaded, achieving a trade-off between data privacy and training accuracy. However, Hitaj et.al. [13] pointed out that differential privacy failed to protect data privacy and demonstrated that a curious parameter server can learn private data through GAN (Generative Adversarial Network) learning. Various other researches has been conducted to ensure privacy of the data transferred in a federated learning setup. Despite these numerous researches, there are two serious problems that needs to be addressed and have received less attention so far [14]. The first one is that existing work generally considered privacy threats from curious parameter server, neglecting the fact that there exists other security threats from dishonest behaviors in gradient collecting and parameter update that may disrupt the collaborative training process. The second problem is that in existing schemes clients are assumed to have enough local data for training and are willing to cooperate which is not always true in real applications. Several other works have also been conducted to preserve privacy in the domain of federated learning [15].

To maintain data privacy, recent studies [14, 16] has aimed to use blockchain in distributed deep learning, mainly focusing on federated learning. These methods aim to provide data confidentiality and incentive mechanism for parties that are willing to participate in collaborative training. However, they have employed a single blockchain network to base their FL model on. However, with recent developments, several authorities maintaining data privacy for their training model on individual blockchains may want to collaboratively train the model with other organizations. An use case of this might be the case where two medical organizations have their own respective learning model trained and stored under a blockchain network, but wants to refine their model using transfer learning or learn a new model using multi-task learning with reference from another model that was trained under a different blockchain network. The motivation behind such a desire can be to effecitvely use the latent features of data from several sources and use them to learn a model that can exhibit a higher accuracy. Such a scenario effectively boils down to interoperability of blockchain and training of a neural network in two

different chains. In simple words, it is employing a blockchain to train a learning model partially, and then transferring the"state" of the model to the participating blockchain where the model is trained till completion.

1.2 Contribution

There are several roadblocks in achieving the above-mentioned idea of cross-training learning models trained under the hood of two different blockchain networks. Initially, an effective design of assets need to be carried out that can be shared across multiple blockchain networks such that it is verifiable. Also, integrity of the various models needs to be checked. To cope up with these issues, we have proposed a system architecture that aims at training a model across different blockchain network by transferring appropriate assets to make the data verifiable. The primary contributions of this work are:

1. We have developed a federated learning architecture that can train a model in a distributed fashion and update the blockchain ledger after transforming the weights in the form of assets.

2. We have designed a system architecture to enable cross-chain training of a learning model. This will enable us to train a learning model partially under one blockchain network while using the weights to train the same or different model under the complimentary blockchain network.

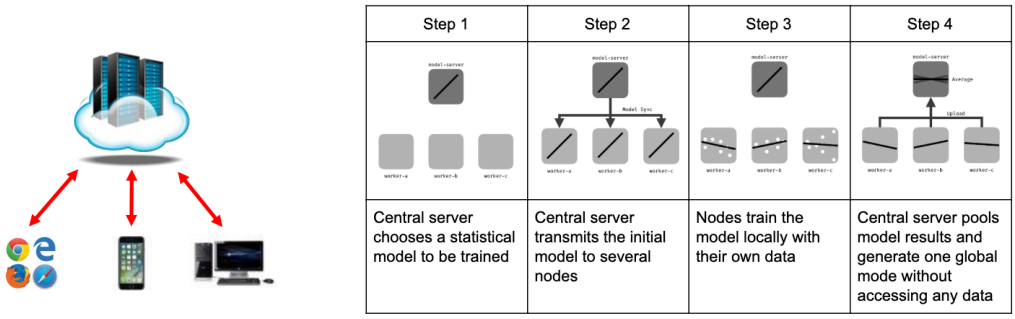
3. We have employed a signing methodology that is easily verifiable and can be used to dictate the authenticity of the transferred assets.

This report in organised as follows:
In Chapter 2, we discuss the background of how a federated learning works and what is a blockchain. In Chapter 3, we discuss a few previous works based on on this domain and how our work is different from them. In Chapter 4, we discuss in detail the design and working of our system architecture, that includes the designing of federated learning system, integrating it under a blockchain network and enabling the transfer of model weights across two blockchain networks. In the next chapter, we discuss the evaluation testbed, consisting of the different datasets, leanring models, hardware setup and metrics used for evaluation. In Chapter 6, we discuss the evaluation results of the experiments and finally, the last chapter we discuss the conclusions and the future work in the field of cross-chain training of learning models.

Chapter 2

Background

2.1 Federated Learning



(a) *(b)*

Figure 2.1: *Schematic Diagram of a Federated Learning system where a parameter server coordinates the training process while various edge devices(web browser, desktops, or mobile devices)(a) can train the model on their local data and respond back to the parameter server with the updated model. Figure (b) summarizes the training process in FL setup*

Federated Learning, first introduced by Google in 2017 [10] is a technique of distributed machine learning where a model is trained on multiple decentralized edge servers on their local data. In such a setup, there exists a central parameter server that coordinates the entire process of model training, while the actual training of the models is performed on the client devices. Generally, the model is trained whenever there is enough resources available at the client side, for example, when the client device is connected to the WiFi, or plugged to the charging point, etc.

A client joins the Federated Learning setup by sending a registration signal to the parameter server and the latter admits the client. The uniqueness of federated learning

system lies in the fact that not all clients need to be online to participate in the training. Also, the local data available to the clients can be of non-iid distribution. Thus, whenever a client is ready to train a model, it triggers the parameter server to send the latest global model along with any hyperparameters, and then the client refines the model on its local data. Thus, each client performs their *local rounds*. After the local training, it sends the updated gradients back to the parameter server. The parameter server on receiving the gradients from multiple clients may decide to either aggregate the weights and update the global model or wait till a threshold number of clients have responded with the updated local weights. This entire process from the server sending the global model to the clients to aggregating it to form the latest version of the global model comprises a single *global round*. The process of federated learning has been demonstrated in Figure 2.1.

The main difference between federated learning and distributed learning is the assumption of the properties of the local datasets [17]. In distributed learning, the datasets are assumed to have iid distribution as it originally aims to distribute and parallelize computation power, where the server distributes the working load to the clients. While federated learning aims at training on datasets having non-iid distirbution and hence heterogeneous. Also, the size of datasets can vary among the clients.

Federated Learning can be of two flavours: synchronous or asynchronous. In synchronous federated learning [10, 18], the parameter server waits till a threshold number of clients responds with the updated gradients. Though it is the most used scheme and the simpler to implement, it suffers from stragglers. If some clients are slow to respond(straggles), the threshold number might not be reached and hence the timely update of the global model will be hampered. To tackle this scenario, asynchronous federated learning [19] was invented where the parameter server has the freedom to update the global model whenever a client responds with a model. If multiple clients responds with the updated model within a short time, the parameter server might choose to update them individually or in a batch. However, this scenario can suffer from stale gradients where a client might respond with a model that was computed with respect to a stale version of global model. Researches have been done to alleviate the effect of stale gradients [20, 21]. However, in this project, we have chosen the synchronous version of federated learning due to its simplicity.

2.2 Blockchain

Blockchain is a decentralized, distributed and public/private ledger that stores transactions and records from various participating parties. It is a peer-to-peer network that

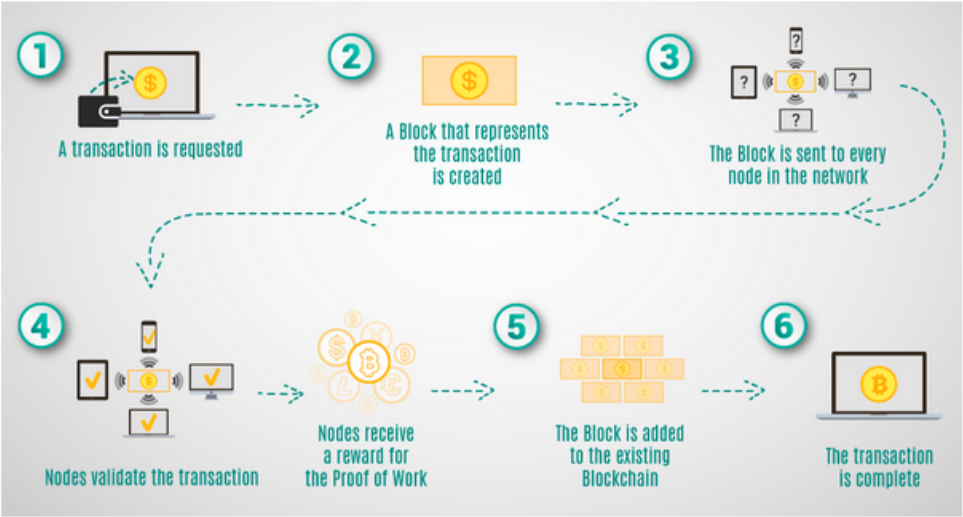


Figure 2.2: *Simplistic operation flow of how a blockchain works. Figure courtesy: Blockchain Architecture Basics: Components, Structure, Benefits & Creation*

does not require any central authority to manage the transactions/communications. In blockchain, few transactions are periodically combined into a single blocks and then chained into the ledger. The principle design of blockchain makes it immutable, that is, a block once added to the ledger cannot be tampered with by the parties participating in it. Thus, blockchain preserves privacy of the data and hence is a useful tool in many applications. The records in a blockchain that are added to the ledger are verified and authenticated by a collaborative opinion of the participating parties. Additionally, in a blockchain, all transactions(entering a block, retrieveing data from a block, deleting a block, etc.) are encrypted and added to the ledger after reaching to a consensus by special nodes called the miner nodes. Each node/peer in a blockchain network holds the entire ledger of transactions thus maintaining secured, synchronized and immutable records. This notion of decentralised security is the principle attraction of blockchain. Use of blockchain also eliminates the possibility of double spending, that is a value can be transferred only once and a transaction can be added to the block only once.

In blockchain, blocks hold batches of valid transactions that are cryptographically hashed and chained into Merkle tree. Each block contains the cryptographic hash of the previous block and this creates the link between consecutive blocks. Blocks are added into the ledger by nodes known as miner nodes. They perform special tasks and hence competes with other miner nodes to win the competition to be eligible to create a new block. Such special tasks, also called consensus algorithms are different for different blockchains and can be called as proof-of-work or proof-of-stake. In Figure 2.2, an overall

operation flow of how a block is added to a blockchain has been described.

Blockchains can be permissioned or permissionless. Permissionless and public blockchains allow any user to create a transaction and add a block to the ledger, for example, Bitcoin and Ethereum. On the other hand, permissioned blockchains like Hyperledger Fabric [22] and Ripple use an access control layer to govern who can access the blockchain and add a block to the ledger. In our work, we have used Hyperledger Fabric, a permissioned open-source blockchain developed by Linux Foundation.

The power of blockchains are enhanced by programs called smart contracts. Smart contracts are essentially codes that are invoked by client applications external to the blockchain network. They are intended to automatically execute, thus managing access and modification to a set of key-value pairs in the state of the transactions. Smart contract can control and execute actions and events in a blockchain based on certain conditions. They were introduced primarily to reduce the need of trusted intermediators and arbitrators.

Chapter 3

Related Works

In this section, we present a brief discussion on the various previous works that have been done on the concept of blockchain, its integration with federated learning as well as interoperability of blockchain networks.

3.1 Blockchain Technology

The concept of blockchain was first introduced and conceptualized by Satoshi Nakamoto in 2008 [3] in the form of bitcoin. It is a cryptocurrency that worked using the principle of a public permissionless blockchain. Since then, blockchain has gained great traction. Due to its decentralized and distributed nature along with its privacy guarantees, several institutions have started using blockchain in varied domains. Bitcoin [2, 3] was the first blockchain technology that was developed which has grown to be a widely-regarded cryptocurrency. There are other public blockchain networks like *Ethereum* [23] and *Litecoin* that enables the use of smart contracts to to automatically execute, control or document legally relevant events and actions according to the terms of a contract or an agreement. For enterprises that use blockchains for storing confidential data needs an access control layer to allow certain actions to be performed only by certain identifiable participants. For such a scenario, permissioned blockchains like *Hyperledger Fabric* [22] and *Burrow* are quite beneficial. A simplified overview of how a blockchain works has been described in section 2.2. Various challenges in research of blockchain has been summarized by Belchior et.al. [24].

3.2 Federated Learning using Blockchain

Federated Learning was first introduced by Google in 2017 [10] and hence has become an important research topic. Due to its promises to maintain privacy of user data, it has been adopted in several industrial applications like GBoard and NVIDIA Clara. Due to the privacy-preserving nature of the federated learning, recent researches has been conducted to incorporate federated learning using blockchain [14, 25, 16]. The main idea is to improve the privacy of the data and the system along with maintaining auditability of the gradients. In simpler words, blockchain stores the gradients/weights of the learned model via FL setup, thus enhancing security of user data along with minimizing the chances of tampering of the model learned. OpenMined [26] is an open source community implementing a federated learning architecture based on smart contracts. DeepChain [14] develops an incentive-based method where the participating nodes are awarded incentives if they can update the model by learning on their local data. The blockchain network stores the gradients and the parameter values after the end of each global iteration and hence extends the chain. Anyone, who wants to use the model for evaluation purposes can use the model by spending a fixed amount of cryptocurrency. FLChain[16], on similar grounds, provides incentives and misbehavior deterrence for collective modelling. It provides a more stable environment by considering each trainer's contribution and reliability.

Several works have been done that incorporates artificial intelligence with blockchain. [27] uses blockchain to securely learn and transfer medical data and model while [28] employs blockchain along with CNN to detect driver's behavior while driving and securely transfer the data to the authority. Goel et.al. [25] provides a novel idea to represent a neural network in a finite blockchain and use it to train the model. A summary of various challenges and research directions are summarized in the survey paper by Salah et.al. [29].

3.3 Cross-Chain Interoperability

Works listed in the above sections tries to integrate a single blockchain with federated learning to enable the training of a model. However, an interesting front of research study is the interoperation of blockchain to transfer assets. Such a methodology would help to enable transfer learning of a model under a blockchain network with the knowledge from another model under a different blockchain network. This makes it essential to enable interoperability of a blockchain. Belchior et. al.[24] provides a survey of various research studies that has been performed on this domain of interoperability of blockchains. Several techniques like trusted relays, side-chains, blockchain of blockchains or hashed time lock

contracts(HTLC) have been explored. Each of them tries to transfer operation flow by allowing clients of a separate network to download the asset and store them in their own local ledger. However, most of these techniques focus on public, permissionless blockchain networks. It is easier to transfer asset within two public blockchains since all clients has the permission to alter the state of the blockchain and hence can download blocks as well. On the other hand, Hyperservice [30] aims to achieve interoperation across heterogeneous blockchains. On a similar front, CollabFed [31] leverages public-private blockchain interoperability by allowing the members of the private consortium to participate in the public blockchain to represent themselves as their own trusted agent. IBM in their studies [32] uses relay service and several additional smart contracts to verify the document transferred among permissioned blockchains. Cryptographic signature mechanisms to digitally sign the documents like Collective Signature [33] ensures the verifiability.

However, we can see that though [32] reaches close to our prime objective in this project, yet it does not try to integrate the interoperability system with learning a model via federated learning. Thus, our objective is different from the above research works in the way we try to optimize the interoperability architecture between two permissioned blockchains to train a learning model collectively after transfer of assets.

Chapter 4

System Description

Transferring of assets across blockchain networks involves a collection of tasks that need to be performed. However, the first requirement is to decide the category of blockchain networks. Since the primary use case of our system involves the learning of models between two or more organizations via the sharing of assets, they need to maintain privacy of their blockchain as well as the transfer methodology. Hence, we have considered the blockchain networks to be permissioned network. Thus, only the clients given the permission to access the ledger will be able to do it. On the other hand, permissionless blockchain network would imply that the learning model can be easily accessed by anyone who has the knowledge of the address of the model, that might result in breach of security.

Our overall system architecture contains several components, involving the learning of the model to the storage of the assets in a blockchain network, to the cross-chain transfer methodology applied. We shall demonstrate each working block of our system in detail.

4.1 Federated Learning System

In our system design, we have employed federated learning to train the learning model. In a nutshell, it involves a parameter server, that coordinates the entire learning process along with various clients that train the model using local data and reply the parameter server with the computed weights. The server then aggregates the received weights according to an aggregation strategy and updates its database with the new aggregated model. After the aggregation, the server then sends the updated model to the clients and the process continues.

However, simple it may look, but it involves several subtleties and nuances. The foremost consideration is about whether to use a synchronous federated learning or an

asynchronous one. Although asynchronous federated learning has several advantages over its counterpart in terms of earlier convergence or straggling clients, synchronous provides greater quality of update in each round and is much more easier and comprehendable. Hence, we have used synchronous version of federated learning in our implementation. Synchronous version progresses in rounds, where at each round, the current model at the server is sent to all the clients(or a subset), and during aggregation, the client models are aggregated only when all the clients(or a subset) replies. Until all the clients reply, the server remains blocked. Thus, a perceivable disadvantage that might come to effect is when some clients are very slow to respond. However, we shall ignore the delay in the clients in our experimentation strategy.

4.1.1 Server Components

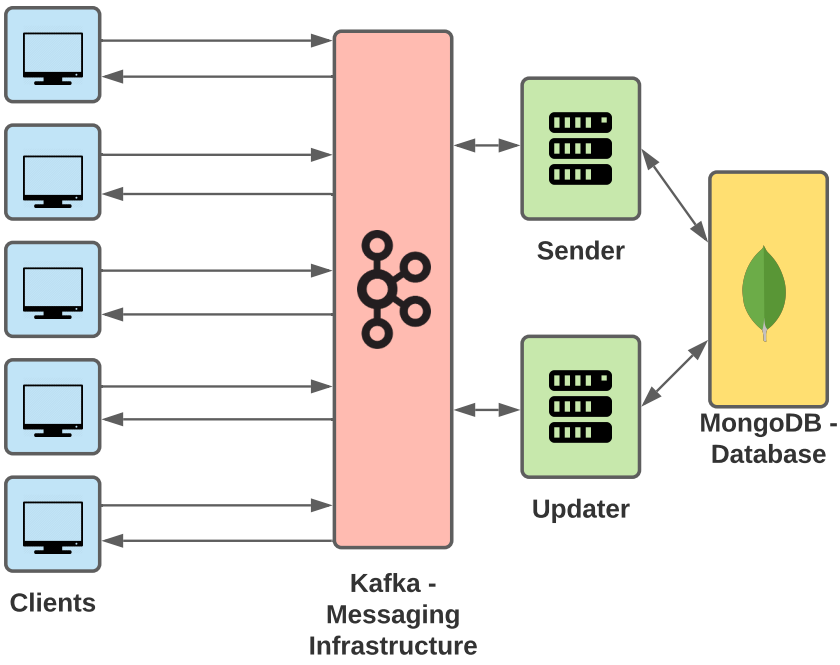


Figure 4.1: *Figure shows a simplified design of our Federated Learning architecture. It has two server components, sender and updater communicating within themselves and with the clients through Kafka, and connected to a MongoDB database.*

Figure 4.1 shows the design architecture of the federated learning system. It involves a **server**, several **clients**, a **messaging infrastructure** and a **database** to store different versions of the models, updates and their metrics. The elementary sub-components of the

server are:

- **Sender:** This component is responsible for selecting a subset of clients at the start of each round, and the updated model aggregated at the server is sent to those set of the clients. There are several theories on what can be the optimal cardinality of the set of clients that can be chosen by the sender. However, we have kept our model simple and fast and will send the model to all the clients.
- **Updater:** Updater is responsible for the aggregation of the received models at the server side. When the models are received by the client, it inserts them in the database and when a threshold number of pending updates are reached, it aggregates them. We have used Federated Averaging [10] as the algorithm for aggregation. After the aggregation, updater updates the global model, changes its version, and stops if a stopping criteria has been reached.

In addition to these components, we also maintain a Coordinator, that initializes the learning model and starts each round.

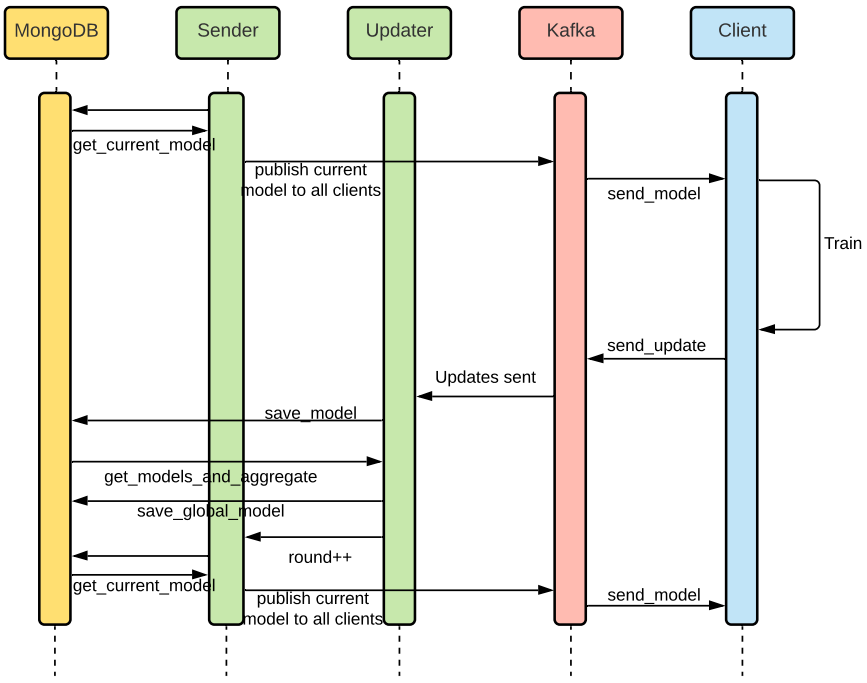


Figure 4.2: *Sequence Diagram showing the operations and Control flow in our Federated Learning architecture.*

For the communication between the clients and the server, our goal is to use some kind of an event streaming platform that can scale up as well when required. Thus, for

this purpose we have used **Kafka** which can be efficiently used in a publisher-subscriber model where the server can publish the updated model and the client can subscribe the messaging from it. A streaming messaging infrastructure is favourable since some straggling clients might not be up-to-date with the current updated model and hence the stream queue that the client has subscribed to might have multiple models. The client can then choose the latest model from the queue and train the local model based on the weights.

The updates from the clients as well as the metrics and the global model are stored in a database on the server side. We have used **MongoDB** as the database which is a popular NoSQL database that stores records as collections(similar to tables in SQL). The sequence of control flow is illustrated in Figure 4.2

Algorithm 1 Federated_Averaging(). The K clients indexed by k; B: local mini-batch size, C: global batch size, E: local epochs, and η: learning rate.

procedure SERVERUPDATE
    Initialize w_0
    for each round $t = 1, 2, \dots$ **do**
        $m \leftarrow \max(C.K, 1)$
        $S_t \leftarrow$ set of m clients
        for k in S_t **do**
            $w_{t+1}^k \leftarrow \text{ClientUpdate}(k, w_t)$
        end for
        $w_{t+1} \leftarrow \Sigma_{k=1}^{K} \frac{n_k}{n} w_{t+1}^k$
    end for
end procedure

procedure CLIENTUPDATE(k, w)
    $\mathcal{B} \leftarrow$ split dataset in client k into batches of size B
    for each local epoch $i \in E$ **do**
        for batch $b \in \mathcal{B}$ **do**
            $w \leftarrow w - \eta \nabla l(w; b)$
        end for
    end for
    Return: w
end procedure

4.1.2 Federated Averaging

To aggregate the updated local models received from the clients, we have used the Federated Averaging algorithm as has been described by McMahan et.al. [10]. Each client replies with the updated local model as well as the number of data points on which the model was trained at the client side. The server then updates its local model weights by an weighted average of the client model weights, with weights being the number of data points at the client. The principle idea of such an aggregation strategy revolves around the fact that more importance should be given to the model of the clients that trained on larger data points. Federated Averaging algorithm has been described in Algorithm 1.

4.2 Blockchain Integration with FL

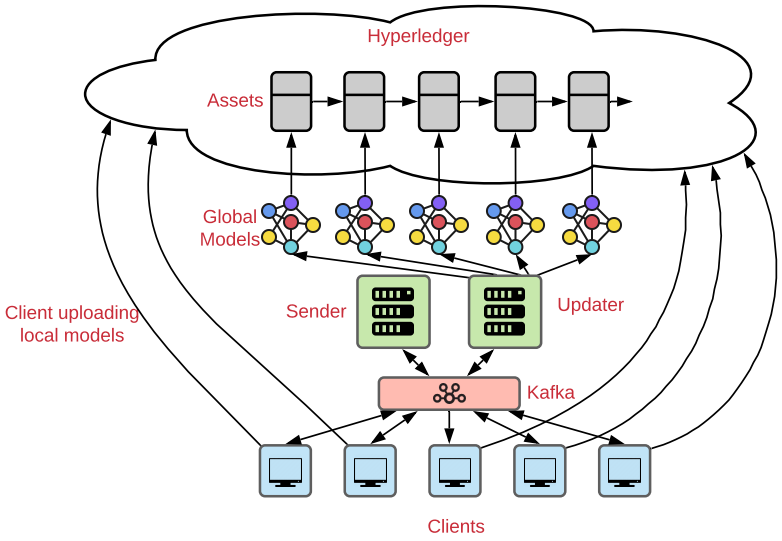


Figure 4.3: *Figure illustrates the working procedure of uploading the learning models in the form of asset from a federated network to the blockchain network.*

Next in our pipeline is the integration of Blockchain within the Federated Learning network. Weng et. al. [14] developed a blockchain-based incentive mechanism and cryptographic primitives for privacy-preserving distributed deep learning. We employ similar techniques where the clients and the parameter server can store the data in a common blockchain. We have used a permissioned blockchain where all the clients and the server have the permission to enter and access the assets stored in it. For our purposes, we have used **Hyperledger Fabric** [22] which is a widely used permissioned blockchain, that

supports the usage of smart contracts and distributed ledger.

Since every client has the permission to access the blockchain network, it alleviates the concern of privacy if the client can enter an asset to the blockchain containing the data about the model weights it learned with some additional parameters. The server can then pull the required assets(probably by round number) and aggregate them using Federated Averaging. Figure 4.3 shows the coalesced architecture that includes blockchain along with the federated learning system. As illustrated in the diagram, along with the federated learning procedure, aggregated global model is uploaded to the blockchain network which is stored in the distributed ledger. Along with the global models, clients can upload their model as well to the blockchain, which will be stored in the same ledger, but have a different ID and an indicator that the asset was formed from the local model of a client.

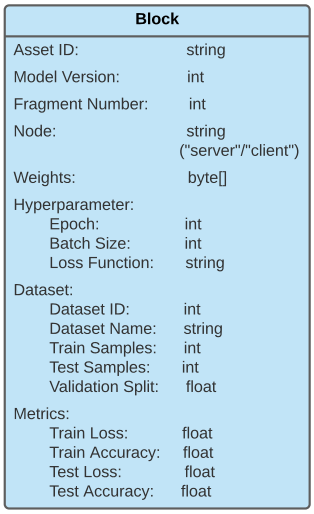


Figure 4.4: *Each block that gets appended to the blockchain contains the information that needs to be shared across multiple blockchains to maintain verifiability.*

The assets that are stored in the blockchain are essentially the training state of the neural network after each round of federated learning. The assets need to contain enough information so that the integrity of the content stored can be verified. Since identical assets are going to be transferred across different blockchain networks, sufficient information are maintained to check the authenticity of the asset. Thus, apart from the model weights/gradients, hyperparameters, dataset characteristics, model architecture and several metrics are stored. Figure 4.4 shows the structure of the asset. Hence each passing round, the chain network is extended by adding the updated model parameters by the client as well as the server.

For different learning models, the size of the asset will be different since the number

of parameters for each model are of varying dimensions. However, hyperledger fabric allows a limited size of asset to be stored in the ledger. Thus, we segment an asset into various fragments of 800 kB each. Thus, for a larger asset, we have greater number of fragments. Since each asset must be stored with a unique ID, thus each fragment will be stored with the ID <asset ID, fragment Number>.

4.3 Cross-Chain Transfer

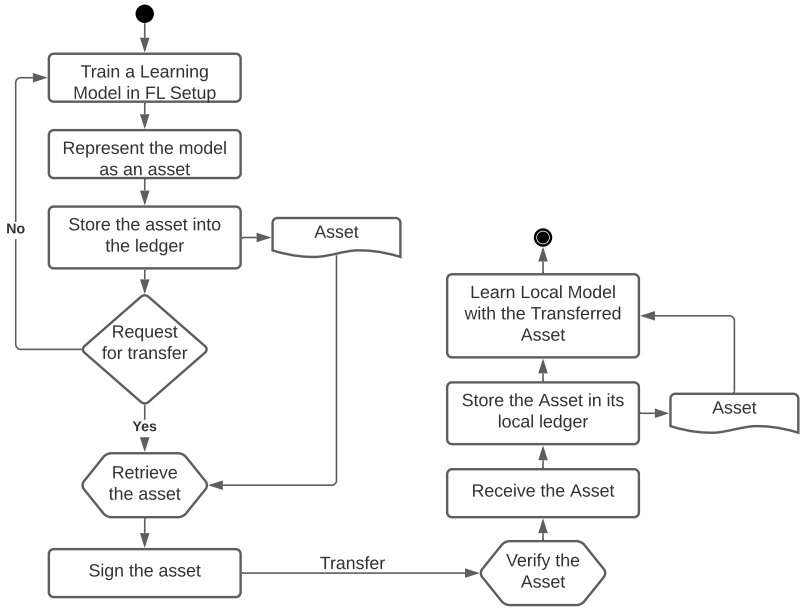


Figure 4.5: *Overall Flow of Operation Control in our System*

From the previous sections, we have an initial architecture of a blockchain network integrated with Federated Learning system that can be used to train a learning model and store the state of the model as an asset in its ledger. However, our objective is to transfer the stored state between multiple blockchain networks. The primary objective of our work in depicted by the flowchart in Figure 4.5. It is easy to achieve this feat in permissionless blockchain since a node outside the same blockchain network can access the ledger and retrieve the state of the learning model. However, this problem gets exceedingly difficult when both the participating blockchain networks are permissioned. In such a scenario, nodes from a different blockchain network (Blockchain B) does not have an access to the ledger of Blockchain A and hence must trust on a node in the network of Blockchain A node to transfer the asset and provide with the information it wants.

Let blockchain $A$and B be two permissioned blockchains, and nodes n_A and n_B be one of the participating nodes in A and B respectively. Node n_B can ask to node n_A to transfer the assets store in Blockchain A, that is, a state of the learning model. Now, it is the responsibility of n_A to transfer the asset to n_B such that n_B can verify the integrity and the authenticity of the asset. Inspired by the design presented by Abebe et. al.[32], we have followed a relay-based asset transfer schema as shown in Figure 4.6.

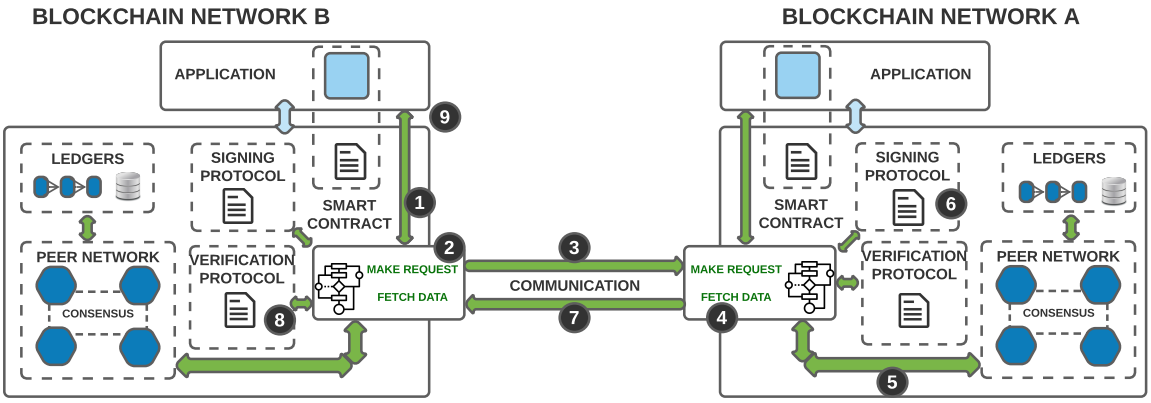


Figure 4.6: *Control Flow for Cross-Chain Interoperation of Assets (denoted by green numbered arrows) between two permissioned Blockchain networks.*

We have used a relay service to facilitate the transfer of assets between the two networks. A separate relay component has been used so that there is no need to modify the network protocols and node implementations. Thus, each of the two blockchain networks have a relay service of their own that is responsible for making a transfer request to the other blockchain, or serve the request of the asking blockchain. Whenever, a transfer of assets is deemed necessary, any client (say, node n_B) can ask the relay service which can then forward the request to the relay service of the destination blockchain. Each relay service can run a smart contract to check the configuration of the foreign network or whether they are trusted, which we exclude however. Similarly, since we are not maintaining any confidential information in the form of an asset in the blockchain, we don't run a validity check on whether the queried asset can be transferred or not. However, such a check is necessary if confidential informations are stored in the asset. After the destination relay service has retrieved the accurate asset, it is then signed by all the clients in that network and then transferred to the source blockchain. It is the responsibility of the source blockchain to verify the authenticity of the incoming data before writing to its own local ledger. The steps involved in the message flow required for the interoperability of blockchain are formulated as below:

1. The client application within blockchain network B requests to its local relay service for the required data by specifying the unique identification of blockchain A and the asset to invoke, that is, whether it wants the latest stored global model or a stale one, along with any arguments. Specifically, the *make_request()* handler of the local relay service is called by the client application.

2. The relay service then performs finds the destination address of the relay service associated with blockchain A.

3. After retrieving the destination address, the relay service of blockchain B serializes the message and communicates to the destination relay via HTTP POST request, along with the parameters sent by the client application. Specifically, it calls the *fetch_data()* handler of the destination relay service.

4. On receiving a POST request on the *fetch_data()* handler, the relay service of blockchain A then deserializes the message and decodes the arguments in the message. Based on the arguments, it will retrieve the specified asset.

5. Next, the relay service retrieves the appropriate asset stored in the distributed ledger via evaluating a transaction. Precisely, the relay service first extracts the ID of the assets stored in the ledger along with the iteration number that the model weight corresponds to. It then finds which asset the relay service of blockchain A requested for and retrieves the asset for only that particular ID. We have established earlier that the asset is stored in the ledger in chunks with corresponding fragment number as well. Thus, as we retrieve each chunk of the specified asset, we will concatenate them to form a single asset, which will then be transferred.

6. The asset is then signed by the peers following the CoSi protocol (Section 4.3.2) so that the asset can be verified for its authenticity at the receiving end. The signature from each peer collectively forms the proof satisfying the verification policy

7. Relay service of blockchain A then serializes the HTTP response and sends it to the relay service of blockchain B, thus transferring the asset.

8. Relay service at blockchain B then invokes the Verification protocol and verifies whether the received data is authentic or not.

9. Finally, the relay service updates the transferred asset to the local ledger and gives a response back to the client application that invoked it earlier..

We have not separated the control flow from the data flow in our architectural design, that is, we will be transferring the assets through the same channel from where the control message(request message) was sent. The main reason is because we are involved with only two blockchain networks communicating with each other. Thus, we experience no delays or congestion with having the same control and data channel. However, with larger scale blockchain networks, where multiple relay service can communicate with each other, it would be beneficial to separate out the two channels to avoid congestion and increase responsiveness.

It is to be noted that if we follow the above-mentioned steps, it is evident that we need to decide upon the amount of information in the form of a single or multiple assets should be transferred so that verification of the content of the transferred data can be carried out. Similarly, some sort of signing the asset should also be performed which will enable the authentication of the network or the clients on the receiving side. Thus, we will discuss these points in details.

4.3.1 Asset Design

For verification of the transferred asset, that implicitly represents a state of the learning model, transferring just the weights shall not suffice. The receiving nodes of the blockchain should also be able to verify whether the weights are truly learned or was there an attack in the middle or untrusted source sent the weights that redistributed the values of the weights. However, in an asset that we store, we not only store the weights, but also include which model it was on, which dataset was used to generate the weights and what are the metrics achieved by employing the same weights. Thus, we claim that the asset corresponding to the global model is enough to verify the integrity of the model. Thus, if we intend to transfer the weights and use it in a transfer learning objective, we shall design the asset that needs to be transferred in the following sub section.

Transfer Learning Objective

Transfer Learning is a branch of machine learning where we use the knowledge gained by some task to perform a related task. As shown in Figure 4.7, Network A has trained on its own specific task using its own input. The parameters and weights of the model are then transferred to Network B, which can then refine the model based on their own input or change a few of the output layers depending on the task they solve. In layman terms, transfer learning helps to give an intelligent and task-specific initialization of the parameters.

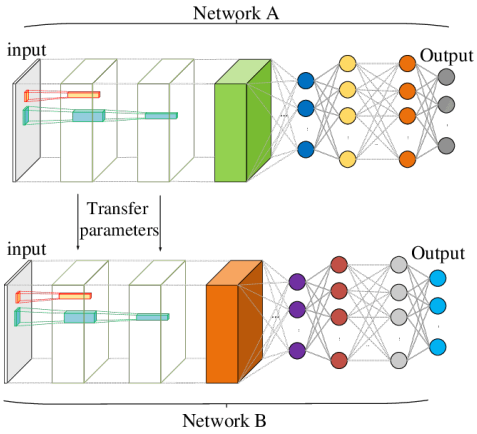


Figure 4.7: *Principle of Transfer Learning*

In our scenario, the first network maintaining Blockchain A performs a Federated Learning task using a learning model. It has its trained weights stored in the blockchain in the form of assets, thus preserving the weights calculated in various rounds. When a transfer of weights in the form of asset is requested, only the final asset storing the weights of the global model in the latest round is sufficient to transfer. The asset contains various auxillary informations including the values of the parameters, loss function, loss value, test accuracy value and other metrics(if needed), and the dataset characteristics used while evaluating the model (as described in Figure 4.4). Verification of the authenticity of the learning model based on the transferred attributes can then be done by the relay service on the receiving end. Thus, it can then use the complete set of transferred parameters or a partial set of parameters as a starting initialization of a new model and can train their own learning model via Blockchain B.

4.3.2 Collective Signing

To reach a consensus in a blockchain network during the transfer of assets, we use the methodology of Collective Signing (CoSi) as presented by Syta et.al. [33]. It is a scalable collective signing procedure that allows the witnesses to collectively sign a transaction within a few seconds.

In our model of the system, we use the Collective Signing procedure whenever there is a need for the transfer of assets. This would ensure that the receiving end can perform verification of the received data and confirm the authenticity. The relay service is responsible to initiate the Cosigning procedure. The relay service defines the number of witness cosigners and defines the document that need to be signed, which in our system modelling is the assets that are transferred. Witness cosigners in the figure are the nodes(clients)

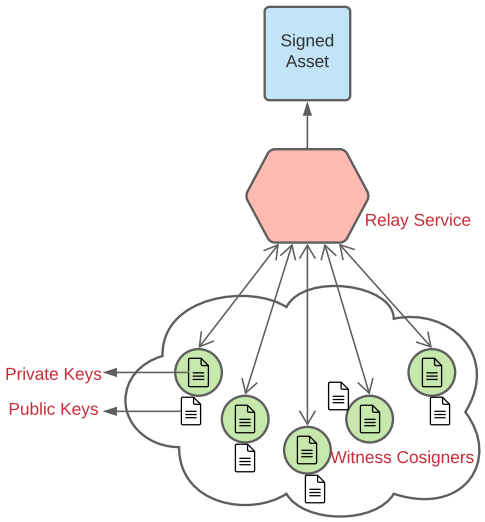


Figure 4.8: *Collective Signing Architecture.*

that participate in collectively signing the records. Witnesses also checks the syntactic and semantic correctness of the documents that the authority requests them to sign. This is needed to prevent any malicious intentions from any requesting node to rewrite history. Numerous digital signature schemes that support efficient public key and signature aggregation can be employed with CoSi. Some of the common examples are Schnorr signatures[34], multisignatures[35] and Boneh-Lynn-Shacham (BLS) [36] cryptosystem.

In our CoSi protocol, we have used BLS signatures where the key is generated for all the witness cosigners, where a signed entity is shared with the receiving relay service. On receiving the transferred assets along with the signed entity, the relay service will then verify the authenticity of the signed entity using the secret keys and then accept/reject the received weights according to the verification outcome. After the verification is successful, the relay service can then either enter the received asset directly to its blockchain or a node can be given the responsibility to perform the task.

However, there is an implicit assumption that the identity and hence the secret keys need to be shared beforehand among the two blockchain networks. This would facilitate the verification of the signed entity at either ends. In our system modelling, we have taken the assumption that the keys generated for BLS signature is shared beforehand. However, Ghosh et.al. [37] in their recent work has demonstrated a method where the identity plane is kept separate from the data plane. The identity and the configuration files are shared through a trusted channel beforehand in the identity plane, which is run on top of the data plane that is responsible for the transfer of assets between two blockchain networks.

Chapter 5

Evaluation Testbed

This chapter will abridge the evaluation of our end-to-end system. The evaluation strategy has three components, namely, evaluating the accuracy of the learning model trained using federated learning, evaluation of the overheads for entering the state of the learning model in the form of an asset into the blockchain, and finally evaluating the overheads for a cross-chain transfer of assets. First, we shall give a brief description of the hardware system employed, followedby the description of the datasets on which we have evaluated our learning model, and finally discuss about the metrics.

5.1 Hardware Setup

We have used AWS EC2 t2.2xlarge instances for running our federated learning system and training our learning model. We have used several such instances acting as a client while one instance being hosted as a FL server. Hyperledger Fabric network is run on the same instance, that is, the EC2 instance hosting the FL server houses the fabric peers and the orderers.

AWS EC2 t2.2xlarge instances have octacore CPU with 32 GB memory running on 3GHz Intel Xeon processors. We have used HTTP request-response protocol to run a golang application developed for entering the asset in the blockchain, and to make request to the relay service to initiate the transfer of assets.

5.2 Dataset

For the Federated Learning via blockchain A, we have used **Cifar-10** dataset. The dataset has 50k training images and 10k test images. The image sizes are 32×32 and are divided

into 10 classes, each class having equal number of training as well as testing images. The datasets were cleaned and then sampled into n equal sets, where n is the number of clients involved in the federated learning model.

The complete training dataset was identically and independently distributed among the clients, so that each client had equal number of images from each class. This implies that the dataset assigned to each client had almost identical distribution, leading to a much smoother convergence. It is to be noted that Federated Learning can handle non-iid distribution of dataset as well. The client then used their respective datasets for training with a train-test split of 0.1. Thus, for each client, we have maintained a training set and a testing/validation set.

For the final part of evaluation where the effectiveness of the transferred weights is verified, we perform a transfer learning task of image classification with varying models. We use **Cifar-100** dataset. Similar to Cifar-10, this dataset has 50k training images and 10k testing image, where each image is of 32×32 size, and labelled with a coarse category and a fine subcategory. There are 20 coarse categories and 100 fine subcategories, however, we use the coarse categories as our labels while evaluating. For the transfer learning task, we have performed a central learning in place of federated learning since this was not our principal goal and was performed only to verify the effectiveness of the transferred weights.

5.3 Learning Models

For the federated learning task, we have evaluated our dataset on two models. We give their description below.

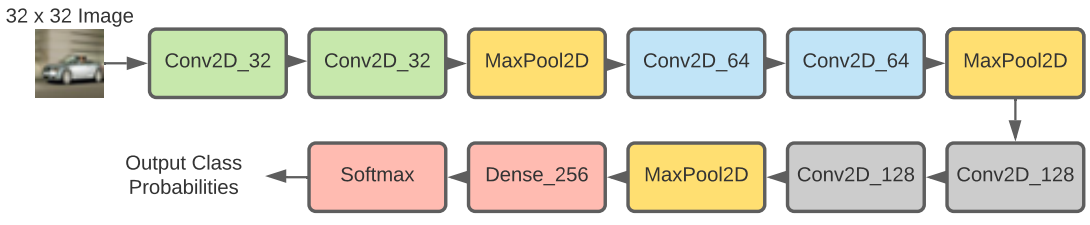


Figure 5.1: *Architectural Overview of the Simple CNN Model*

5.3.1 Simple CNN

Simple CNN is a 6-layered CNN model followed by a feed-forward dense hidden unit and an output layer. The structure of the CNN model is Figure 5.1. Each convolution layer has a 3×3 kernel with 'same' padding. Number of convolution filters increases with gradually, with occasional Maxpooling layer and a dropout. All the layers have the same 'Relu' activation function. The total number of parameters for SimpleCNN is $814,122$.

5.3.2 Compressed VGG

This is a shortened and compressed version of the VGG11 model. The VGG module has a total of 7 "convolution" layers with occasional Maxpooling, having a total of 171,682 parameters. However, an interesting feature in our model is that we have replaced the "convolution" layers with **Fire** module, as illustrated in Figure 5.2. Inspired by the architecture of SqueezeNet [38], we have designed our VGG with Fire module so that it has fewer parameters compared to the original VGG. This would facilitate the storage of the asset in the ledger with significant ease, as well as would take less time to transfer. However, care was taken that it did not result in accuracy being decreased by a significant amount.

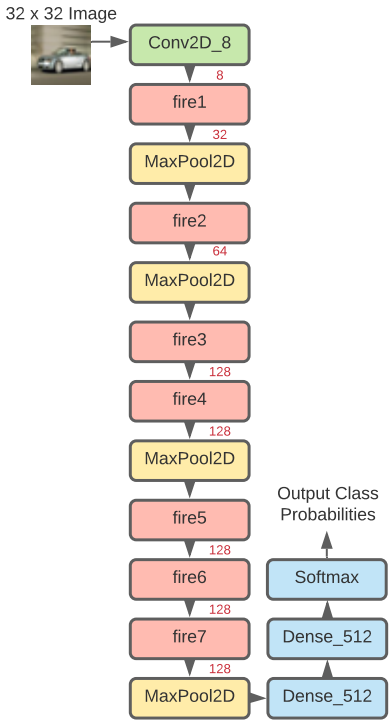


Figure 5.2: *Micro-architecture of the compressed VGG model, involving Fire modules*

The overall structure of Compressed VGG is described in Figure 5.2. In place of a convolution filter that takes a $h_{input} \times w_{input} \times c_{input}$ dimension input and produces a $h_{output} \times w_{output} \times c_{output}$ dimension output, we have used Fire module that performs similar operations, but using fewer parameters. The main idea of Fire module is to replace a 3×3 filter with several 1×1 filters, which is called the *squeeze layer*. The output from the *squeeze layer* is then passed to the *expand layer*, which contains 1×1 as well as 3×3 filters, as illustrated in Figure 5.3. The output from the filters in expand layer is then

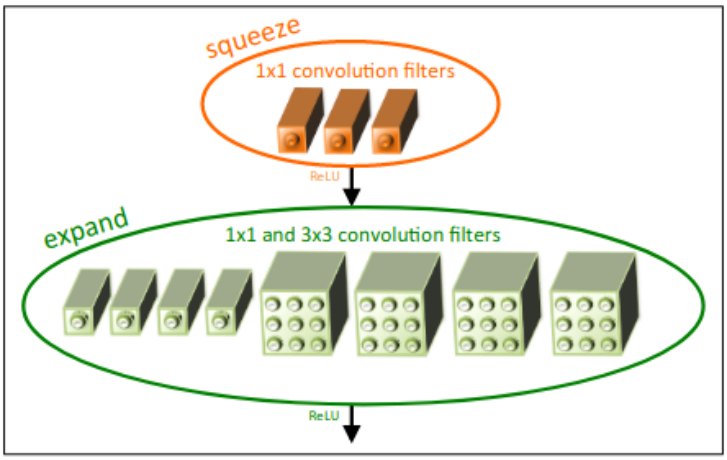


Figure 5.3: *Micro-architectural view of the Fire module, showing the organization of* 1×1 *and* 3×3 *convolution filters. Figure credits: [38]*

concatenated together to form the output. This strategy downsamples the network and reduces the number of input channels to the 3×3 convolution filters.

For evaluating the transfer overheads, along with the above two models, we have computed the statistics on MobileNet-v2 as well as ResNet-18. These two models are closer to the practical models with larger number of parameters and hence would give an understanding of the expenses of transferring assets in a practical scenario involving several organization.

5.4 Metrics

5.4.1 Model Performance Metrics

For evaluating the performance of the learning models, we have primarily used *Accuracy* and *Loss value*, the two most common metric. In the federated learning task, we have computed accuracy metrics and loss values of varying types depending on the execution point where we have checkpointed them. These are:

- **Pre-Test:** Evaluation of Global Model on the global test dataset
- **Pre-Val:** Evaluation of Global Model on each client test dataset and averaged over clients
- **Post-Test:** Evaluation of Client Models on the global test dataset
- **Post-Val:** Evaluation of Client Models on their own local test dataset and averaged over clients

For the transfer learning task, we compute the usual training loss and accuracy along with validation loss and accuracy, which we will present in our evaluation section.

5.4.2 Transfer Overhead Metrics

In order to enable the transfer of assets, which in our case are the weights, from one blockchain network to the other, we would first need to enter the assets into the blockchain network from the federated learning setup. This would require an overhead time for asset creation and asset entering sub-tasks. This would differ for various models since each model weights are of varying sizes and hence would require varying time for entering them into the blockchain network.

For the transfer of assets, the overheads are to retrieve the relevant asset that needs to be transferred from the distributed ledger, along with the signing overhead. On the receiving side, the relay service performs the verification which is included in the transfer overhead as well, which we report.

5.5 Other Hyperparameters

Since we are using Synchronous federated learning, we will be having a versioning scheme, where the version of the global model is updated after every global round. Each global round includes 2 local epochs, that is, the clients will run the model on their local dataset for 2 epochs.

For the model training, we have used Sparse Categorical Cross-Entropy as the loss and Adam Optimizer with a learning rate of 0.001.

Chapter 6

Evaluation

This chapter discusses the results of the evaluation carried out over the duration of the project. The chapter is organised as follows:

1. Evaluation of the learning models trained via synchronous Federated Learning system.
2. Analysis of the overheads incurred for storing the model weights in the form of assets transferring them.
3. Evaluation of the effectiveness of the transfer of weights.

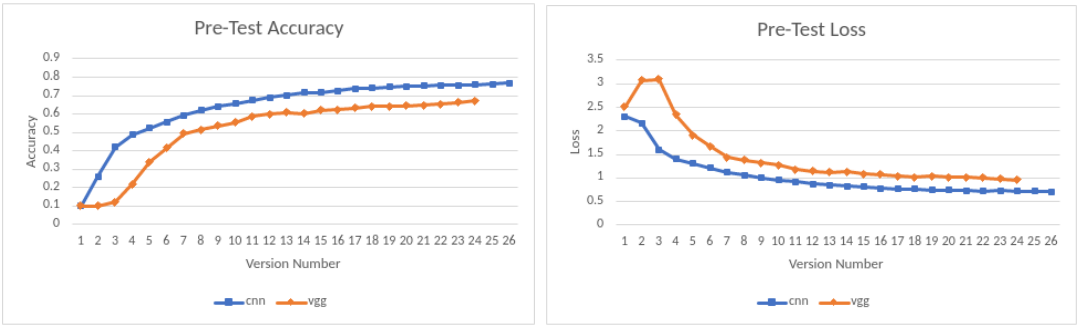
6.1 Performance of Federated Learning System

The first set of experiments were performed to judge the performance of the learning model that was trained via the designed Federated Learning. Such a evaluation would serve two-fold purpose, first is the validity of the Federated Learning system, and second is to have an idea as to which of the models described in the previous section would be a good candidate for the transfer learning, and would be beneficial in terms of transferring the weights. We have evaluated the learning models on the cifar-10 dataset through a server-client system having various clients.

6.1.1 Small Scale - 10 Clients

In these set of experiments, we will demonstrate the results where 10 clients were used for training in the Federated Learning setup. For the *SimpleCNN* model, we have used a batch size of 32, while for the *Compressed VGG* model, we have employed a batch size of 16.

We first present the plots of accuracy and loss for both the models against the version numbers. We will report the Pre-Test metrics in Figure 6.1.

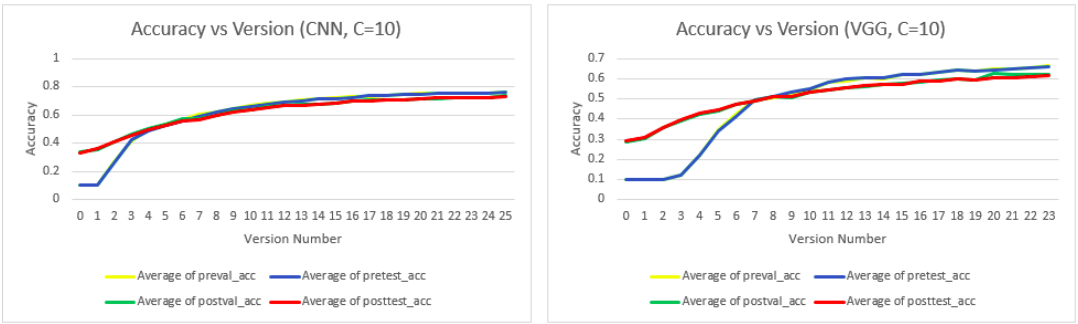


(a) *(b)*

Figure 6.1: *Comparing the variation of pre-test accuracy and loss value of SimpleCNN and CompressedVGG against version numbers. 10 clients were used to train the model in an FL setup.*

We see in the above diagram that SimpleCNN has greater accuracy and lower loss value than the CompressedVGG model, which is explainable since SimpleCNN has more number of parameters than CompressedVGG and hence can capture more complex relationships in the dataset. It is to be noted that the version number is incremented after the clients have replied with their corresponding updates and they are aggregated by the Updater. Thus, within each version update, each clients have run 2 epochs of training. We shall discuss the time it takes to update from one version to the other and compare with the large scale experiments in Sub-section 6.1.3.

In Figure 6.2, we report all the four metrics outlined in Section 5.4.1 for accuracy for both the models. We see that in both the plots, Pre-Test accuracy is higher than



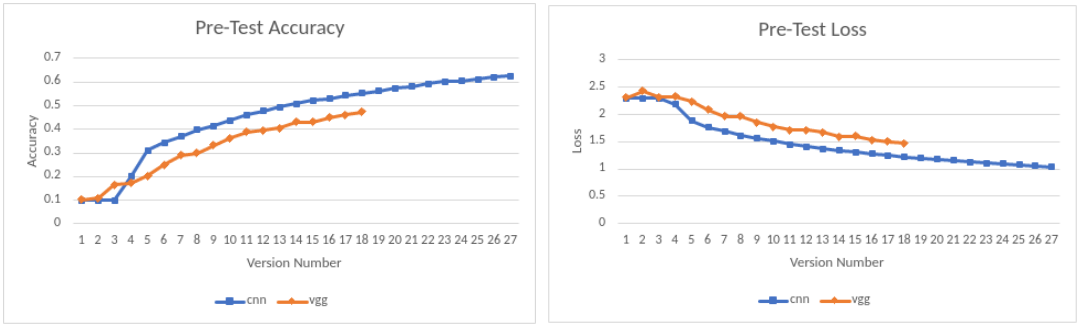
(a) Learning Model: SimpleCNN *(b) Learning Model: Compressed VGG*

Figure 6.2: *Accuracy metrics for both models against version numbers. 10 clients trained the model in an FL setup.*

Post-Test accuracy, that is, the accuracy of the aggregated global models surpasses that of the trained local models, which means that aggregation of the weights indeed results in a better learning of the model. Aggregation also provides a regularizing effect, since no client model gets overfitted on their local dataset.

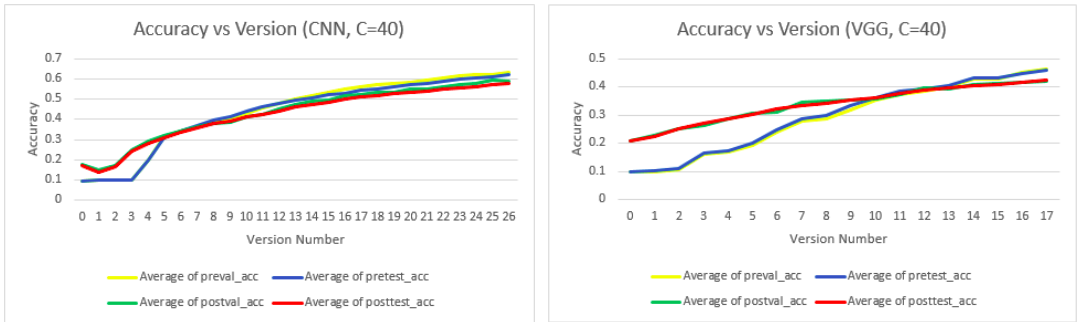
6.1.2 Large Scale - 40 Clients

In this set of experiments(Figure 6.3), we demonstrate similar results as above where 40 clients were used for training. Batch sizes and the evaluation metric are similar as on before.



(a) *(b)*

Figure 6.3: *Comparing the variation of pre-test accuracy and loss value of SimpleCNN and CompressedVGG against version numbers. 40 clients were used to train the model in an FL setup.*



(a) *Learning Model: SimpleCNN* ***(b)*** *Learning Model: Compressed VGG*

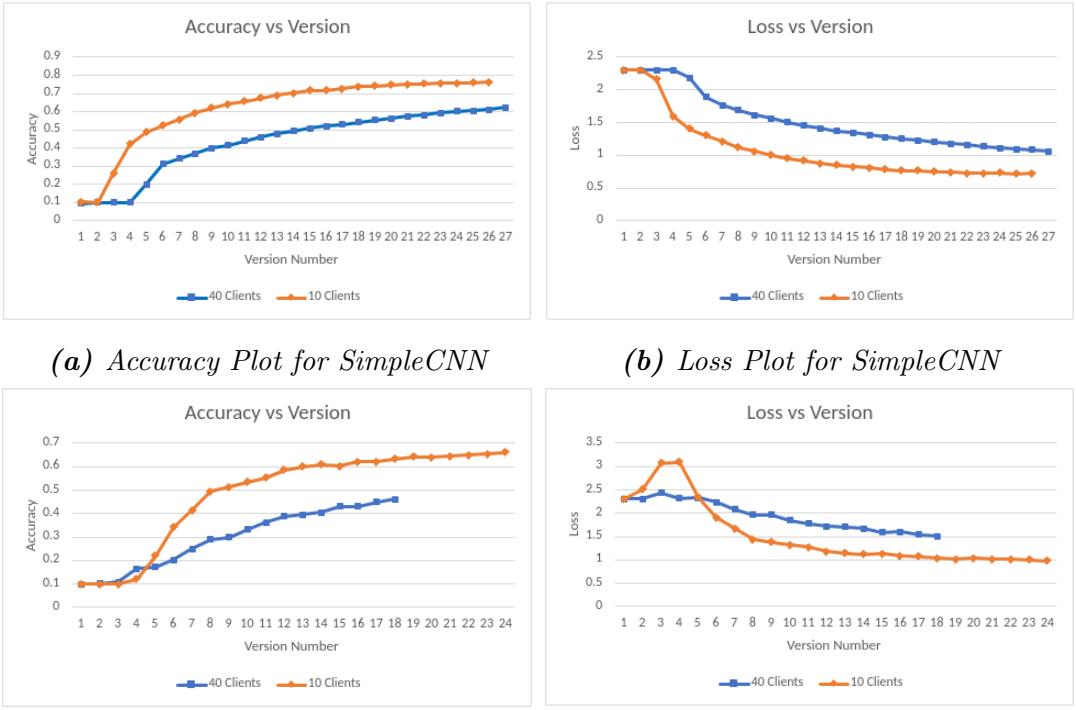
Figure 6.4: *Accuracy metrics for both models against version numbers. 40 clients trained the model in an FL setup.*

Similar to the previous result, we see that SimpleCNN beats CompressedVGG in terms of both accuracy and loss. The explanation remains similar as of above. However,

we will notice that the value of accuracy reached is lower than the value reached in the previous experiment. We shall elicit on this in the next sub-section. In Figure 6.4, we report the various accuracy metrics for both the learning models. We again see that Pre-Test metric surpasses the Post-Test metric, which again proves that aggregation creates a regularizing effect, thus increasing the efficacy of the model.

6.1.3 Comparison

We have seen how the two models behave when they are experimented with a set number of clients in the above two results. However, in this section, we shall elicit how the two results compare against each other. The comparison can be based on two varying reasons, namely, whether the accuracy increases by training on a larger set of clients, and whether the model gets trained faster with increasing the number of clients.



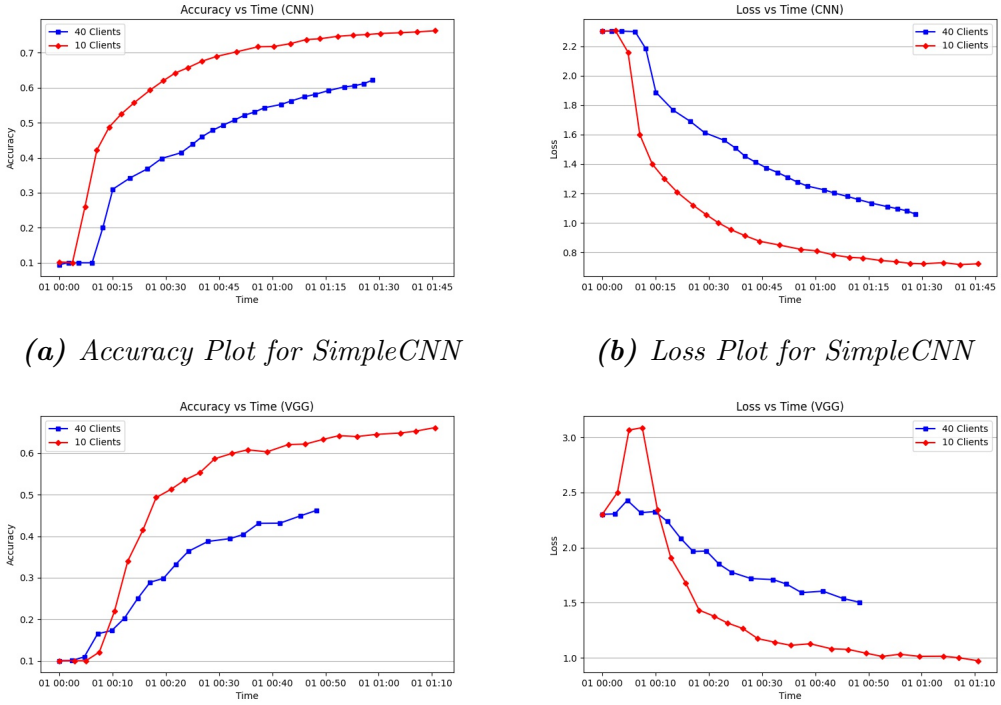
(a) Accuracy Plot for SimpleCNN *(b) Loss Plot for SimpleCNN*

(c) Accuracy Plot for CompressedVGG *(d) Loss Plot for CompressedVGG*

Figure 6.5: *Figure shows the comparison of Federated Learning setup for different number of clients.*

In Figure 6.5, we plot the value of pre-test accuracy and loss against the version numbers for FL setup with 10 and 40 clients. We see that with increase in number of clients, the value of accuracy drops at a specific version, which is true for both the models. This is because with an increase in the number of clients, each client has a lesser

amount of local cifar-10 dataset. Hence, the local model trained on the dataset overfits, this lowering the efficacy. With 10 clients only, each client has a total of 5000 images compared to having 1250 images only when 40 clients were trained.



(a) Accuracy Plot for SimpleCNN

(b) Loss Plot for SimpleCNN

(c) Accuracy Plot for CompressedVGG

(d) Loss Plot for CompressedVGG

Figure 6.6: *Figure shows the comparison of Federated Learning setup for different number of clients against time, that is, how long does it take to reach a certain accuracy when the nunmber of clients are changed*

Table 6.1: Average global round time for federated learning of various models when different number of clients were used for training

Learning Model	**10 Clients**	**40 Clients**
CompressedVGG	3 min 4 sec	2 min 50 sec
SimpleCNN	4 min 39 sec	3 min 23 sec

In Figure 6.6, we plot the accuracy and loss of the two learning models against time. We see that to reach the same accuracy, learning with 10 clients makes it faster than learning with 40 clients. This is primarily because of the same reason as stated above, that is, with increasing clients, each client gets a lower amount of dataset for training the local model. Hence, the quality of the model trained gets compromised. However, from

table 6.1, we see that the average global round duration for both the models is lower when 40 clients are used to train. This, similarly, can be reasoned that with lower amount of dataset, it will be fast to complete an epoch and hence takes an overall lower average global round time.

6.2 Analysis of Transfer Overheads

From the previous section, we found out that the learning model was trained via the Federated Learning setup. We also deduced the better of the two networks that we had experimented with. Next in our pipeline, we need to set our hyperledger network and enter the weights that were calculated into the ledger via the form of assets, and then retrieve them from the ledger when a transfer is requested. Apart from the models that we have evaluated in the previous section, we shall also be testing the overheads on various other models, that are used more in practical scenarios.

6.2.1 Entering Asset

Table 6.2 shows the overhead required to enter the weights in the form of assets into the ledger. The values that we have recorded are averaged across multiple runs.

Table 6.2: Table showing the overhead for entering the asset into Hyperledger

Learning Model	**# parameters**	**Asset Size (MB)**	**# chunks**	**Asset Entry Time (sec)**
Compressed VGG	171,682	4.0	5	1.148
Simple CNN	814,122	19.5	24	5.527
MobileNet-v2	3,239,114	67.3	84	17.869
ResNet-18	11,192,019	232.1	290	63.079

We see that with an increase in the model size, that is, with an increase in the number of parameters in the model, the asset size and the asset entry time scales up linearly as well, which was expected. However, it is to be noted that even for an asset of size 232 MB (for a ResNet-18 model), the time taken by the FL server to update it in the blockchain is around 1 minute. Hence, it shows that the asset will be entered into the ledger before the global model is updated again, thus avoiding a contention in the updating channel.

6.2.2 Transferring Asset

Whenever a request for the transfer of assets arrives at a relay service, it must first retrieve the requested asset from the ledger and defragment it, act as an authority to ordrer the peers to digitally sign it using BLS signatures, and finally reply with the asset. On the receiving end, the relay service must verify the signature first on receiving the asset before committing the transferred asset to its local ledger. In table 6.3, we illustrate the timing overhead incurred at each step. Retrieval Time refers to the amount of time in seconds required to execute Step 5 defined in Section 4.3.

Table 6.3: Table showing the overhead for transferring the requested asset across two blockchain networks

Learning Model	**Asset Size (MB)**	**Retrieval Time (sec)**	**CoSi Time (sec)**	**Verification Time (sec)**
Compressed VGG	4.0	0.404	0.847	0.871
SimpleCNN	19.5	2.261	5.241	6.017
MobileNet-v2	67.3	6.584	18.586	19.844
ResNet-18	232.1	22.051	110.460	154.705

As can be inferred from the table, transfer time of the assets steadily increases with increasing model complexity. However, unlike the asset entering overhead, the transfer overhead scales slightly faster than linear, as is evident from the increase in CoSi Time and Verification Time for ResNet-18. In the experiment above, we have transferred the asset via HTTP POST request-response mechanism. We have assumed a constant sufficient bandwidth and hence have not reported the statistics for the same.

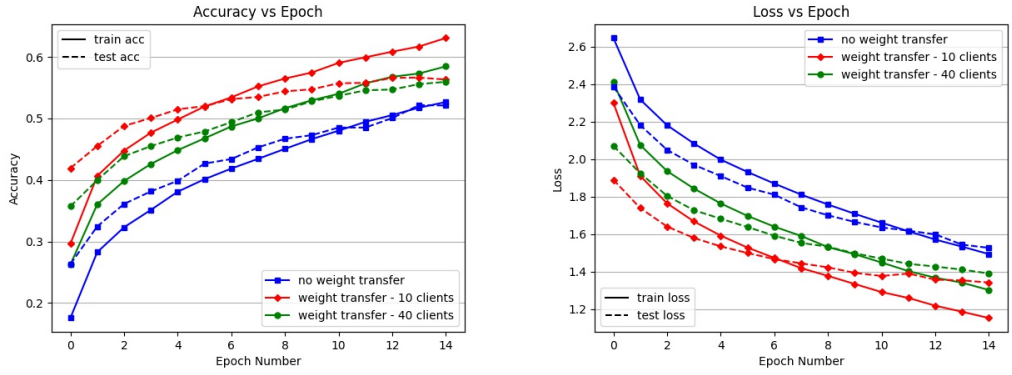
As we have already mentioned, with an increase in the number of transfer requests from various blockchains, a single relay service and a single channel for both control and data flow can become the bottleneck. We see from table 6.3 that ResNet-18 takes a total time of around 4 min 47 sec apart from the transfer time. Out of this, more than 2 minutes of time is taken for retrieving and sigining the asset before the transfer. Hence, if multiple relay service of some blockchain networks start requesting for the asset, the source blockchain can become a bottleneck while trying to attend all the queries. In such a scenario, a relay service can spawn multiple process where each process is responsible to handle a request for one such transfer request. Additionally, maintaining the same control and data channel for all the connecting blockchains shall not be optimal from a scalability point of view. We can separate out the control flow from the data flow, or in

other words, use an FTP channel for transferring the assets. The transfer of assets can take place via a different channel from which the request arrived.

6.3 Evaluating Models via Transfer Learning

Finally, after transferring of the assets from one blockchain network to the other and verifying the authenticity of the transferred asset, we can use the weights to train a learning model on possibly, a different dataset. We will experiment the efficacy of transferred weights via the task of transfer learning. As already point out in a previous section, that we will be training a model on Cifar-100 dataset with data labels as the coarse label.

We will first compute the effect of using the transferred weights as an initialization of the model, which we have taken to be SimpleCNN. We shall also use the baseline where SimpleCNN was trained from scratch.



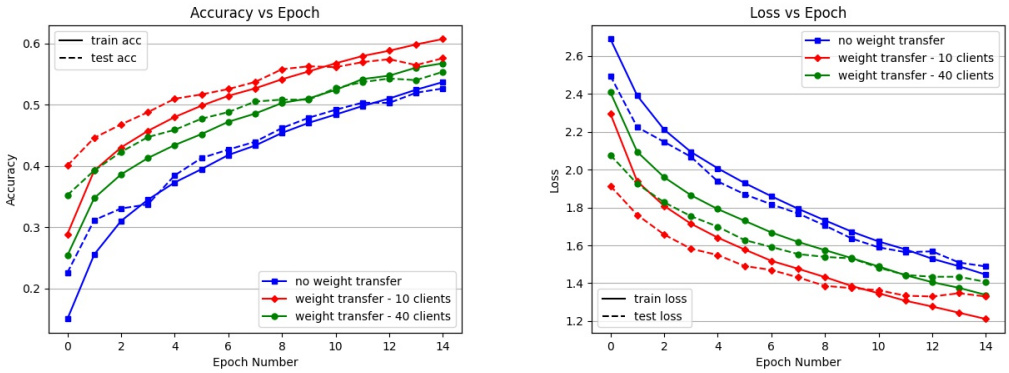
(a) *(b)*

Figure 6.7: *Accuracy and Loss(Training and Testing) of SimpleCNN on cifar-100 dataset trained from scratch as well as initializing the model with the transferred weights.*

From Figure 6.7, we see that the results of training and testing accuracy of SimpleCNN on cifar-100 dataset resonates with the performance of SimpleCNN on cifar-10 dataset. As we have seen in sub section 6.1, SimpleCNN performed better when trained in an FL setup with 10 clients than in a setup with 40 clients. Hence, the quality of weights transferred remains superior as well, also establishing the correctness of the weights transferred. Nonetheless, model learnt with the transferred weights performed better than the model learning from scratch, which confirms the efficacy of the learning model.

We have also tested the transfer learning task on a different model but a structure inspired by SimpleCNN. This was done to ensure that we can use the weights as an initialization for a few parts of the learning model. Effectively, we added 3 convolution

layers of 256 filters each on the top of SimpleCNN followed by a max-pooling layer. Each of the convolution layers had a 3×3 kernel. Finally, a dense layer of 512 units was added which was followed by the output layer. Let us call this model TransferCNN for discussion in the next few paragraphs.



(a) ***(b)***

Figure 6.8: *Accuracy and Loss(Training and Testing) of TransferCNN on cifar-100 dataset trained from scratch as well as initializing the model with the transferred weights.*

On a similar note, we observe in Figure 6.8 that the model having initialization with the weights transferred from the FL setup with 10 clients worked better than the rest. This, the aforementioned experiments cement the validity of the weights transferred and tries to explain the results.

Chapter 7

Conclusion and Future Work

7.1 Conclusion

In summary, we have developed an architecture that can enable the cross-chain training of a neural network by exploiting the idea of interoperability between two permissioned blockchain. From the discussions and extensive experiments performed above, several conclusions have surfaced which are as follows:

1. We have developed a scalable federated learning system that can be deployed on multiple machine to train a learning model with varied number of clients.

2. We have developed an effective strategy to store the model weights/gradients in a blockchain network that makes it secured and can be easily transferred to a requesting client under a different blockchain network.

3. We have intelligently designed the structure of the assets that need to be transferred to verify the integrity of the model as well as perform transfer learning or similar tasks.

4. we have used relay service which acts as a mediator for the transfer of assets. A request for an asset is generated at the relay service in the requesting end and it is forwarded to the relay service at the providing end. This helps to facilitate the transfer of assets. We have used the collective signing methodology to sign and verify the authenticity of the asset transferred.

5. Finally, we have experimented about the correctness of the weight transferred by learning a model via transfer learning with the weights transferred as an intialization.

7.2 Future Works

In the system of interoperation developed, a single relay service can become a potential bottleneck when the number of requests increases. Thus, as a next step, we would try to allocate a single process or a thread to handle the transfer of request to a single requester. That would solve the issue where a single process blocks the concurrent interoperability requests.

We have used a single control and data channel to transfer the assets as well as communicate with other relay services. However, a potential improvement is to segregate the channel into a control channel and a separate data channel, which helps in maintaining better concurrency for large scale experiments.

We have designed the asset to be transferred keeping in mind of the task of transfer learning. However, we can use our system for Multi-Task Learning as well. A crude proposition for entertaining such a system is to transfer not only the final asset containing the latest global model, but also subsequent updates to the model. This would be congruent to the idea where two learning heads can utilize the same shared weights to train. This would be beneficial if two different FL setup under varying permissioned blockchain network wants to collaborate among themselves to train a multi-task learning model.

Finally, we also plan to further validate our work large scale experiments pertaining to higher-end models on larger datasets.

References

[1] Michael Crosby, Pradan Pattanayak, Sanjeev Verma, Vignesh Kalyanaraman, et al. Blockchain technology: Beyond bitcoin. *Applied Innovation*, 2(6-10):71, 2016.

[2] Aviv Zohar. Bitcoin: under the hood. *Communications of the ACM*, 58(9):104–113, 2015.

[3] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. Technical report, Manubot, 2019.

[4] Anwaar Ali, Siddique Latif, Junaid Qadir, Salil Kanhere, Jatinder Singh, Jon Crowcroft, et al. Blockchain and the future of the internet: A comprehensive review. *arXiv preprint arXiv:1904.00733*, 2019.

[5] Tiago M Fernández-Caramés and Paula Fraga-Lamas. A review on the use of blockchain for the internet of things. *Ieee Access*, 6:32979–33001, 2018.

[6] Saveen A Abeyratne and Radmehr P Monfared. Blockchain ready manufacturing supply chain using distributed ledger. *International Journal of Research in Engineering and Technology*, 5(9):1–10, 2016.

[7] Matthias Mettler. Blockchain technology in healthcare: The revolution starts here. In *2016 IEEE 18th international conference on e-health networking, applications and services (Healthcom)*, pages 1–3. IEEE, 2016.

[8] Muhamed Turkanović, Marko Hölbl, Kristjan Košič, Marjan Heričko, and Aida Kamišalić. Eductx: A blockchain-based higher education credit platform. *IEEE access*, 6:5112–5127, 2018.

[9] Christian Gorenflo, Stephen Lee, Lukasz Golab, and Srinivasan Keshav. Fastfabric: Scaling hyperledger fabric to 20,000 transactions per second. In *2019 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, pages 455–463. IEEE, 2019.

[10] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Artificial Intelligence and Statistics*, pages 1273–1282. PMLR, 2017.

[11] Keith Bonawitz, Hubert Eichner, Wolfgang Grieskamp, Dzmitry Huba, Alex Ingerman, Vladimir Ivanov, Chloe Kiddon, Jakub Konečnỳ, Stefano Mazzocchi, H Brendan McMahan, et al. Towards federated learning at scale: System design. *arXiv preprint arXiv:1902.01046*, 2019.

[12] Reza Shokri and Vitaly Shmatikov. Privacy-preserving deep learning. In *Proceedings of the 22nd ACM SIGSAC conference on computer and communications security*, pages 1310–1321, 2015.

[13] Briland Hitaj, Giuseppe Ateniese, and Fernando Perez-Cruz. Deep models under the gan: information leakage from collaborative deep learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 603–618, 2017.

[14] Jiasi Weng, Jian Weng, Jilian Zhang, Ming Li, Yue Zhang, and Weiqi Luo. Deepchain: Auditable and privacy-preserving deep learning with blockchain-based incentive. *IEEE Transactions on Dependable and Secure Computing*, 2019.

[15] Xiaofeng Lu, Yuying Liao, Pietro Lio, and Pan Hui. Privacy-preserving asynchronous federated learning mechanism for edge network computing. *IEEE Access*, 8:48970–48981, 2020.

[16] Xianglin Bao, Cheng Su, Yan Xiong, Wenchao Huang, and Yifei Hu. Flchain: A blockchain for auditable federated learning with trust and incentive. In *2019 5th International Conference on Big Data Computing and Communications (BIGCOM)*, pages 151–159. IEEE, 2019.

[17] Qiang Yang, Yang Liu, Tianjian Chen, and Yongxin Tong. Federated machine learning: Concept and applications. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 10(2):1–19, 2019.

[18] Jakub Konečnỳ, H Brendan McMahan, Felix X Yu, Peter Richtárik, Ananda Theertha Suresh, and Dave Bacon. Federated learning: Strategies for improving communication efficiency. *arXiv preprint arXiv:1610.05492*, 2016.

[19] Cong Xie, Sanmi Koyejo, and Indranil Gupta. Asynchronous federated optimization. *arXiv preprint arXiv:1903.03934*, 2019.

[20] Ming Chen, Bingcheng Mao, and Tianyi Ma. Efficient and robust asynchronous federated learning with stragglers. In *Submitted to International Conference on Learning Representations*, 2019.

[21] Shuxin Zheng, Qi Meng, Taifeng Wang, Wei Chen, Nenghai Yu, Zhi-Ming Ma, and Tie-Yan Liu. Asynchronous stochastic gradient descent with delay compensation. In *International Conference on Machine Learning*, pages 4120–4129. PMLR, 2017.

[22] Hyperledger fabric. https://www.hyperledger.org/use/fabric.

[23] Ethereum whitepaper. https://ethereum.org/en/whitepaper/.

[24] Rafael Belchior, André Vasconcelos, Sérgio Guerreiro, and Miguel Correia. A survey on blockchain interoperability: Past, present, and future trends. *arXiv preprint arXiv:2005.14282*, 2020.

[25] Akhil Goel, Akshay Agarwal, Mayank Vatsa, Richa Singh, and Nalini Ratha. Deepring: Protecting deep neural network with blockchain. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 0–0, 2019.

[26] Openmined - building safe artificial intelligence. https://openmined.org/.

[27] Seong-Kyu Kim and Jun-Ho Huh. Artificial neural network blockchain techniques for healthcare system: Focusing on the personal health records. *Electronics*, 9(5):763, 2020.

[28] Muhammad Z Khan, Muhammad UG Khan, Omer Irshad, and Razi Iqbal. Deep learning and blockchain fusion for detecting driver's behavior in smart vehicles. *Internet Technology Letters*, page e119, 2019.

[29] Khaled Salah, M Habib Ur Rehman, Nishara Nizamuddin, and Ala Al-Fuqaha. Blockchain for ai: Review and open research challenges. *IEEE Access*, 7:10127–10149, 2019.

[30] Zhuotao Liu, Yangxi Xiang, Jian Shi, Peng Gao, Haoyu Wang, Xusheng Xiao, Bihan Wen, and Yih-Chun Hu. Hyperservice: Interoperability and programmability across heterogeneous blockchains. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pages 549–566, 2019.

[31] Bishakh Chandra Ghosh, Tanay Bhartia, Sourav Kanti Addya, and Sandip Chakraborty. Leveraging public-private blockchain interoperability for closed consortium interfacing. *arXiv preprint arXiv:2104.09801*, 2021.

[32] Ermyas Abebe, Dushyant Behl, Chander Govindarajan, Yining Hu, Dileban Karunamoorthy, Petr Novotny, Vinayaka Pandit, Venkatraman Ramakrishna, and Christian Vecchiola. Enabling enterprise blockchain interoperability with trusted data transfer (industry track). In *Proceedings of the 20th International Middleware Conference Industrial Track*, pages 29–35, 2019.

[33] Ewa Syta, Iulia Tamas, Dylan Visher, David Isaac Wolinsky, Philipp Jovanovic, Linus Gasser, Nicolas Gailly, Ismail Khoffi, and Bryan Ford. Keeping authorities" honest or bust" with decentralized witness cosigning. In *2016 IEEE Symposium on Security and Privacy (SP)*, pages 526–545. Ieee, 2016.

[34] Claus-Peter Schnorr. Efficient identification and signatures for smart cards. In *Conference on the Theory and Application of Cryptology*, pages 239–252. Springer, 1989.

[35] Mihir Bellare and Gregory Neven. Multi-signatures in the plain public-key model and a general forking lemma. In *Proceedings of the 13th ACM conference on Computer and communications security*, pages 390–399, 2006.

[36] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the weil pairing. In *International conference on the theory and application of cryptology and information security*, pages 514–532. Springer, 2001.

[37] Bishakh Chandra Ghosh, Venkatraman Ramakrishna, Chander Govindarajan, Dushyant Behl, Dileban Karunamoorthy, Ermyas Abebe, and Sandip Chakraborty. Decentralized cross-network identity management for blockchain interoperation. *arXiv preprint arXiv:2104.03277*, 2021.

[38] Forrest N Iandola, Song Han, Matthew W Moskewicz, Khalid Ashraf, William J Dally, and Kurt Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size. *arXiv preprint arXiv:1602.07360*, 2016.