

Department of Computer Science  
The University of Hong Kong  
Final Year Project

Improvement of Blockchain Consensus Algorithm by Integrating  
Distributed Machine Learning

Sangwon Park (3035556060)  
Chungha Yu (3035553135)  
Supervisor: Dr. Chow, Kam Pui

## **Abstract**

With the rise of the blockchain industry, concerns regarding high energy consumption in Proof of Work (PoW) blockchains have become a hotly debated topic. Various solutions have been suggested to address this issue, but they have often failed to maintain the competitive market nature of PoW, which is one of its key benefits. This project's main objective is to tackle high energy consumption by converting wasted energy into a resource for machine learning training. The project intended to develop a novel algorithm called Proof of Machine Learning (PoML), which operates alongside virtual machines. These servers serve as node providers, facilitating the distributed handling of machine-learning tasks. However, during the development of a novel consensus algorithm, the project soon realized construction of a novel algorithm could not be done within the time scope. Hence, the project decided to pivot and develop a Decentralized Application (dApp) where the motivation remains the same: solve high energy consumption problems by rewarding tokens to computational power.

## **Acknowledgment**

We would like to express our gratitude to our supervisor, Dr. Chow, Kam Pui who gave us valuable insights and guidance. The project could also not have been completed without the help of my partner Chungha Yu and Sangwon Park.

## Table of Contents

Abstract.....	i
Acknowledgment.....	ii
Table of Contents.....	iii
List of Figures.....	v
List of Tables.....	vi
Abbreviations.....	vii
1. Introduction .....	1
1.1 Background.....	1
1.2 Motivation .....	2
1.3 Objective.....	2
1.4 Scope and Deliverables .....	2
1.5 Research Gap and Significance .....	3
1.6 Outline of the Report .....	3
2. Methodology.....	3
2.1 Overview of Methodology.....	3
2.2 Technologies and Platform.....	4
2.3 Back End Development.....	4
2.3.1 API Requests Handler .....	5
2.3.2 Main Node for Machine Learning.....	5
2.3.3 Database Management.....	5
2.4 Blockchain & ERC-20 contract.....	6
2.5 Production Management.....	6
2.6 Front End Development.....	7
2.6.1 MetaMask .....	7
2.6.2 Google Cloud Storage .....	8
2.6.3 Vanilla JavaScript.....	9
2.7 Machine Learning Development .....	10
2.7.1 Federated Averaging.....	11
3. Project Development .....	12
3.1 Back End Development.....	12
3.1.1 Interaction with Blockchain.....	12
3.1.2 API Requests .....	17

3.1.3 Database Schema.....	19
3.2 Front End Development.....	20
3.2.1 Overview of Front End Development .....	20
3.2.2 Authentication Flow .....	20
3.2.3 Node Provider Page.....	22
3.2.4 Customer Page.....	24
3.3 Distributed Machine Learning Development .....	27
3.3.1 Overview of Distributed Machine Learning Development .....	27
3.3.2 Node Providers .....	27
3.3.3 Main Server .....	28
3.3.4 Uploading Model and Notifying Customers .....	29
3.3.5 Summary.....	31
4. Project Schedule and Future Work.....	31
4.1 Project Schedule .....	31
4.2 Future Work.....	32
4.2.1 Overview of Future Work.....	32
4.2.2 Back End.....	32
4.2.3 Security Measurements.....	33
4.2.4 Reward & Payment System.....	35
4.2.5 Message Transfer.....	37
4.2.6 Improved Security .....	37
4.2.7 Cloud Servers .....	37
4.2.8 Distributed Machine Learning.....	38
5. Conclusion .....	39
References .....	vii
Appendices .....	ix

## List of Figures

<b>Figure 1</b> Simple Architecture design of back end server that is deployed on render .....	4
<b>Figure 2</b> Screenshots of Project Management in Notion. ....	7
<b>Figure 3</b> Flow chart of how MetaMask is used to connect the client to the blockchain .....	8
<b>Figure 4</b> Google Cloud Storage Buckets are used to store data .....	9
<b>Figure 5</b> Overall flow of the application.....	10
<b>Figure 6</b> The system architecture and data flow for Federated Averaging.....	11
<b>Figure 7</b> Screenshot of the transferring ERC-20 token in back-end level.....	14
<b>Figure 8</b> Screenshot of the log text with successful training. ....	15
<b>Figure 9</b> Screenshot the log text of malicious activities conducted in nodes from the main server. .....	17
<b>Figure 10</b> Screenshots of the schema layouts for ticket, user, and node providers .....	19
<b>Figure 11</b> Login page.....	20
<b>Figure 12</b> Registration page.....	21
<b>Figure 13</b> Page for node providers to join the platform.....	22
<b>Figure 14</b> An alert shows up if the user tries to register for duplicate usernames.....	22
<b>Figure 15</b> A MetaMask alert instructs how much MATIC is needed to join the platform.....	23
<b>Figure 16</b> An alert tells the node provider that it has been successfully registered.....	23
<b>Figure 17</b> Customer page before data upload .....	24
<b>Figure 18</b> MetaMask notification of a pending transaction.....	25
<b>Figure 19</b> Alert describing that the transaction is successful along with the transaction hash.....	25
<b>Figure 20</b> Code example of using POST request for Google Cloud Storage .....	26
<b>Figure 21</b> Alert that indicates uploading of data has been completed .....	27
<b>Figure 22</b> The system architecture and data flow for Federated Averaging.....	28
<b>Figure 23</b> Code to run multiple iterations of federated machine learning.....	29
<b>Figure 24</b> If machine learning is successful, customers receive an email with links to download the model and weights .....	30
<b>Figure 25</b> If machine learning is not successful, then the entire amount the customer initially paid is refunded .....	30
<b>Figure 26</b> Algorithm of Secure FedAvg .....	34
<b>Figure 27</b> Sample diagram showing the architecture design of TEE. ....	35
<b>Figure 28</b> The system architecture and data flow for Federated Averaging.....	38

## List of Tables

<b>Table 1</b> Project Time Schedule .....	32
<b>Table 2</b> Outlines the current payment and reward schema for DML platform.....	36
<b>Table 3</b> Outlines the improved scheme for the payment and reward system in the future of the DML platform.....	36

## **Abbreviations**

**PoW** - Proof of Work

**PoS** - Proof of Stake

**PoML** - Proof of Machine Learning

**P2P** - Peer-to-Peer

**SGD** - Stochastic Gradient Descent

**GCP** - Google Cloud Platform

**HTTP** - Hypertext Transfer Protocol

**HTTPS** - Hypertext Transfer Protocol Secure

**FTP** - File Transfer Protocol

**TCP** - Transmission Control Protocol

**UDP** - User Datagram Protocol

**API** - Application Programming Interface

**MNIST** - Modified National Institute of Standards and Technology database



## 1. Introduction

Section 1.1 explains the background of the topic and the problem of the current platform. Section 1.2 depicts the motivation of the project. Section 1.3 illustrates the objectives of the project. Section 1.4 describes the scope and deliverables, and Section 1.5 outlines the research gap and significance followed by Section 1.6 with the outline of the report.

### 1.1 Background

Since the introduction of Bitcoin by Satoshi Nakamoto in 2008, blockchain has evolved to be applicable not only in finance, government, and education but also as pillars of a new web 3.0 protocol. As blockchain networks do not have a central authority but instead adopt a shared database method as a peer-to-peer network, it increases trust, security, transparency, and traceability of data shared across a network; hence, favored by industries such as financial services, government, and even insurance. [1,2]

However, there is one major downside of the whole system. High computational power results in high energy consumption as the blockchain tries to establish a competitive market. According to the University of Cambridge Electricity consumption index, it is estimated that blockchain consumes electricity at an annualized rate of 127 terawatt-hours, which contributes 0.3% of global annual carbon emission [3,4]. The reason for such high consumption of energy is due to the nature of Bitcoin's Proof of Work (PoW) consensus algorithm.

With the rise of AI and machine learning, having access to powerful computing power is important for efficiency and speed. While machine learning that uses a centralized server has been the conventional approach, experiments conducted show that a distributed approach to machine learning accelerates the training of modest to large-sized models [5]. Popular models such as GPT and BERT are large machine-learning models. Therefore, an algorithm that utilizes the high computational power for machine learning can potentially solve both the downsides of blockchain and increase machine learning access in society.

## 1.2 Motivation

There is a clear need to explore alternative consensus algorithms that can achieve decentralization in an energy-efficient manner. PoW possesses an advantage of formulating a competitive market but in return it wastes energy. If the project can leverage computing power for training machine learning datasets, then such energy is converted to be used in useful work [6]. For the given time scope the project seeks to implement a new web platform where a user who wants to train machine learning datasets can utilize the platform after paying ERC-20 tokens and node providers stake computational power to earn tokens. In the future, the project would be expanded to implement an actual layer 1 blockchain with a novel consensus algorithm.

## 1.3 Objective

The objective of the project shifted from creating a layer 1 blockchain network with a novel consensus algorithm known as Proof of Machine Learning (PoML) to developing dApp for distributed machine learning. The application tackles the problem of high energy consumption by giving token incentives to whomever staking computational power to the project's machine learning platform and providing a secure and transparent payment system through transactions via ERC-20 token in the PoS chain. The project aims to achieve this by completing 3 main objectives listed below:

Objective 1: Develop a web platform through the integration of front-end and back-end

Objective 2: Manage transactions on blockchain for a transparent payment system

Objective 3: Provide a distributed training environment by leveraging federated averaging

Further details will be presented throughout the report.

## 1.4 Scope and Deliverables

The project makes three contributions. First, it explains the development of the front-end for both user and node providers. The front end provides a user-friendly interface to use the platform. Second, the project introduces the core functionality of the back end. Unlike traditional back-end servers, the project's server incorporates both API requests and federated averaging training. Lastly, participating nodes will be simulated in a cloud environment. Using Google Cloud Platform (GCP), servers will be set up to receive training data and models. The result will

be sent back to the back-end server which is the main server and trained results will be returned to the user.

### **1.5 Research Gap and Significance**

This project will allow existing node providers that are wasting computational power to provide machine learning services to society. These node providers will be given tokens for their services. The project focuses on creating a sustainable environment where a user submits training datasets and node providers get tokens according to the training performance. Since such a concept of a platform where one can leverage federated average learning to train datasets without a cloud platform is novel the project will look more at the system cloud providers such as AWS or Google Cloud. The project will research and incorporate fundamental cloud computation methods to successfully develop the dApp platform.

### **1.6 Outline of the Report**

The report consists of 5 sections. Section 2 depicts the methodology of the project, Section 3 illustrates the current development progress and results, Section 4 discusses future implementation, and Section 5 concludes the overall report.

## **2. Methodology**

### **2.1 Overview of Methodology**

The platform consists of 3 major aspects: front-end, back-end, and a distributed machine learning algorithm. The combination of these three aspects allows users to train custom data using our platform and node providers to get tokens as a reward for providing computational power.

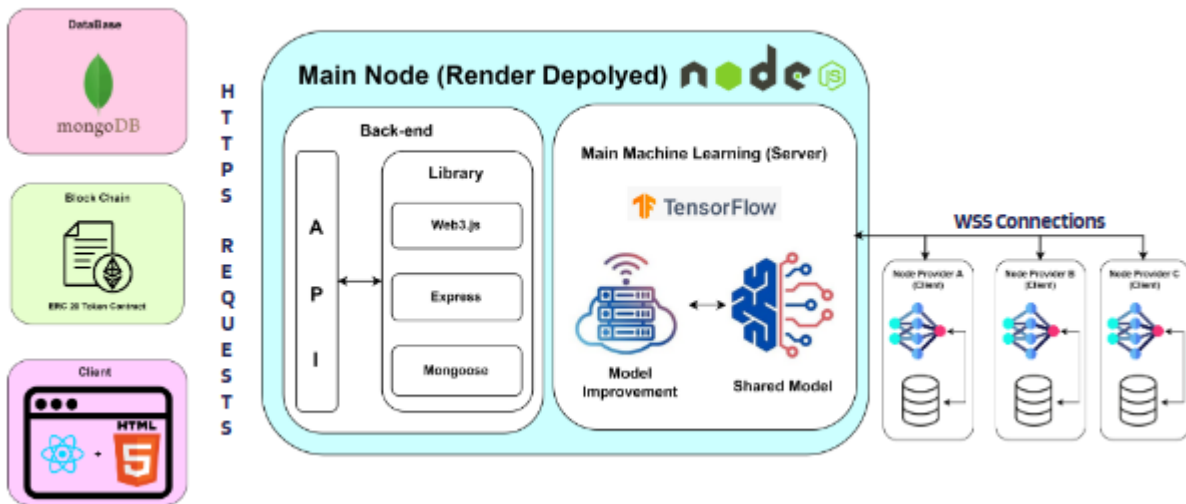
Section 2.2 summarizes our development platform. Sections 2.3 to 2.5 describe the procedures of how front-end, back-end, and distributed machine learning algorithms will be implemented.

## 2.2 Technologies and Platform

The development environment differs in the back-end and front-end. For the back-end, the entire codebase has been written in JavaScript with libraries such as Web3.js, express, and Mongoose. Using JavaScript allows the development of HTTPS requests and APIs for the database, blockchain, and the front-end. For the database, the project has adopted MongoDB as it is a non-schema database. The core of machine learning which is implemented in the back-end server is written in Python with a federated averaging library from TensorFlow. For the development of the front end, HTML and CSS are used with vanilla JavaScript.

## 2.3 Back End Development

Development of the back end remains the main root of the project. The back end consists of two major backbones: API requests management, and the main server for federated machine learning. The figure below shows the architecture design of the back end.



**Figure 1** Simple Architecture design of back end server that is deployed on render. Illustrates two major components in the Main server which are the API requests handler and the main machine learning server.

Unlike the traditional back end, the DML platform incorporates an additional layer to manage federated averaging machine learning training. 3 ports will be opened where one is for API requests management, one for WSS connection management with other node providers, and

the last for the Mongo database. Once the framework is set up, it will be deployed utilizing render.com which gives API strings that can be used.

### **2.3.1 API Requests Handler**

As shown in Figure 1, the project utilizes a traditional back-end design for API request management. Utilizing the express library framework, the project developed HTTPS requests for the front end, database, and blockchain for transaction management. An additional Web3.js library has been utilized to allow transactions via smart contract interactions. During the development stage, API will be checked utilizing Postman framework and will identify errors. Moreover, the platform will handle HTTP standard errors such as 404 or 401.

### **2.3.2 Main Node for Machine Learning**

For the development of the main server for the federated machine, additional network configuration is required for communication with node providers. A new port will be opened for websocket secure (WSS) connection with other node providers to transfer train data so that node providers can train and return the parameters back to the main server. The project plans to simulate 2 to 4 virtual machines with GPU. They will act as node providers. Set up will be done utilizing Google Cloud Storage. The setup allows machine-learning data sets to be trained in parallel. Once the main server receives the models from the node providers, aggregation, evaluation, and averaging will happen under the tensor flow federated learning library framework. After the loops of training are completed, created model parameters will be uploaded to Google Cloud Storage and a downloadable link will be sent to the user.

### **2.3.3 Database Management**

The database has been deployed using Mongo database. Mongo database with mongoose library in Java Scripts allows easier implementation. The database will have three main schemas: user information, node provider information, and tickets. User information will hold the login details of users. It will allow users to create accounts and log in accordingly. The schema for node providers will hold information about node providers, their wallet addresses, and the node's status. Lastly, tickets are information about training data the user has submitted after payment. It holds information about transaction hash, pay amount, list of nodes who will train the data, and

status of the training. Since the project wants the main server to start machine learning once the user sends the data to Google Cloud Storage, the team will open up a port that will stream changes in collections of tickets. Once the ticket is created after the user successfully pays it will trigger the main node to download the data and start training. Such a stream will be implemented utilizing Mongoose. watch() function which allows to watch for new changes (insertions, updates, replacements, deletions, and invalidations) in this collection.

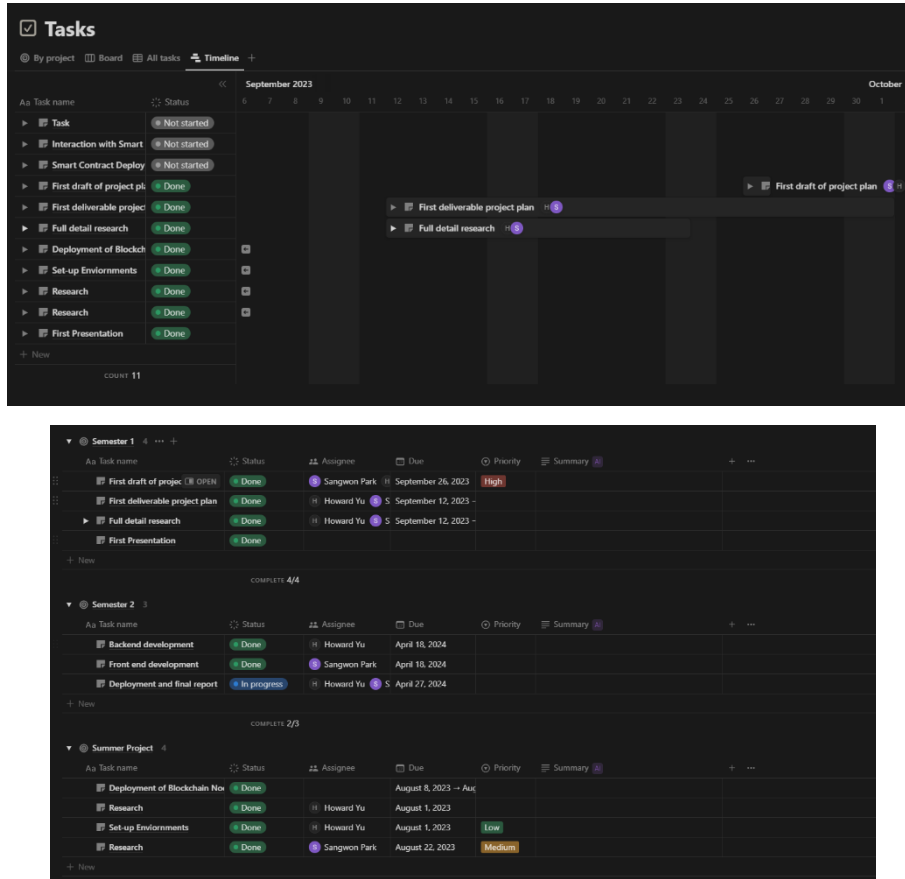
#### **2.4 Blockchain & ERC-20 contract**

The project will utilize the Proof of Stake (PoS) chain as it does not use much energy and provides fast transaction time. Given a timeframe for the project, the team will utilize the Polygon test net to deploy token contracts and create API for interactions instead of the actual main net.

ERC-20 contracts are standardized smart contracts that are fungible tokens. It allows developers to build products and services from ERC-20 token allowing easier interaction on blockchain for users at lower cost. Moreover, as ERC-20 tokens are standardized, exchanges with other tokens are flexible. The team will create a token under Openzeppelin's framework and deploy using hardhat. The coined token will be named the Decentralized Machine Learning (DML) token. Interaction on the front end will be done utilizing Web.js with meta mask and on the back end will be done solely with Web3.js.

#### **2.5 Production Management**

The main server which consists of machine learning and API requests handling will be deployed utilizing render.com. To allow continuous integration and continuous development (CI/CD) environments, the project used Git to manage versions and updates. Moreover,



**Figure 2** Screenshots of Project Management in Notion. Notion provides a simple environment to manage tickets and visualize project plans in a timeline and ticket manner.

render.com provides a simple environment where deployment can be easily done once the git main branch has been updated. Such an environment allowed a better CI/CD environment. Secret and confidential information such as database connection string, password, wallet's private key, and password for email has been managed in .env files. For production management, the team has used Notion to intuitively visualize remaining tasks and deadlines as shown in Figure 2.

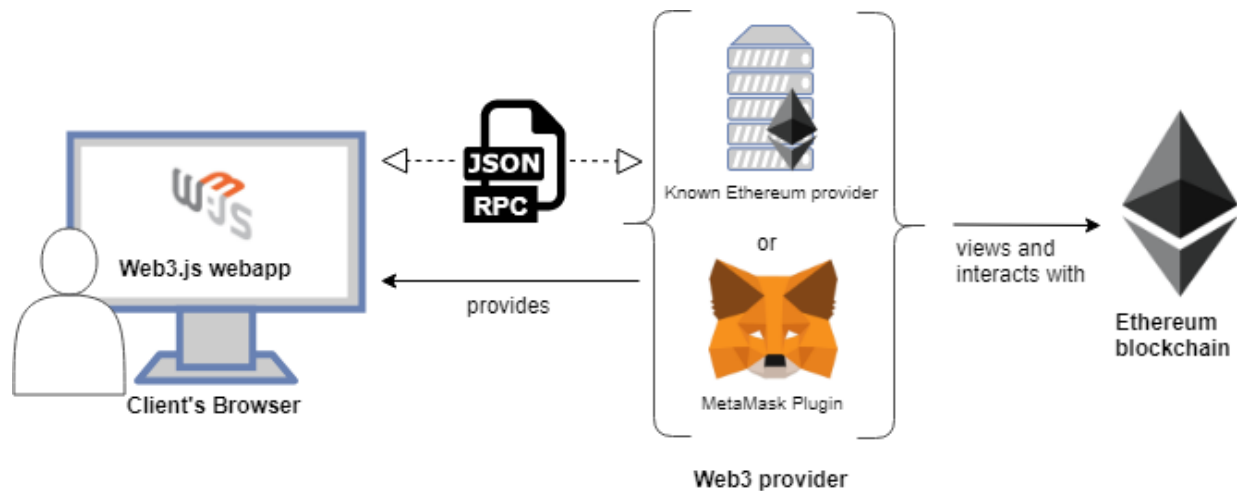
## 2.6 Front End Development

Multiple technologies were used in building the front-end module of the application. This section will provide information about the technologies that were used.

### 2.6.1 MetaMask

MetaMask is a cryptocurrency wallet and browser extension that allows users to securely manage their digital assets and interact with decentralized applications (dApps). MetaMask acts

as a bridge between web browsers and the blockchain and enables users to create and manage accounts. Furthermore, it helps store and transfer tokens, and seamlessly connect with dApps.

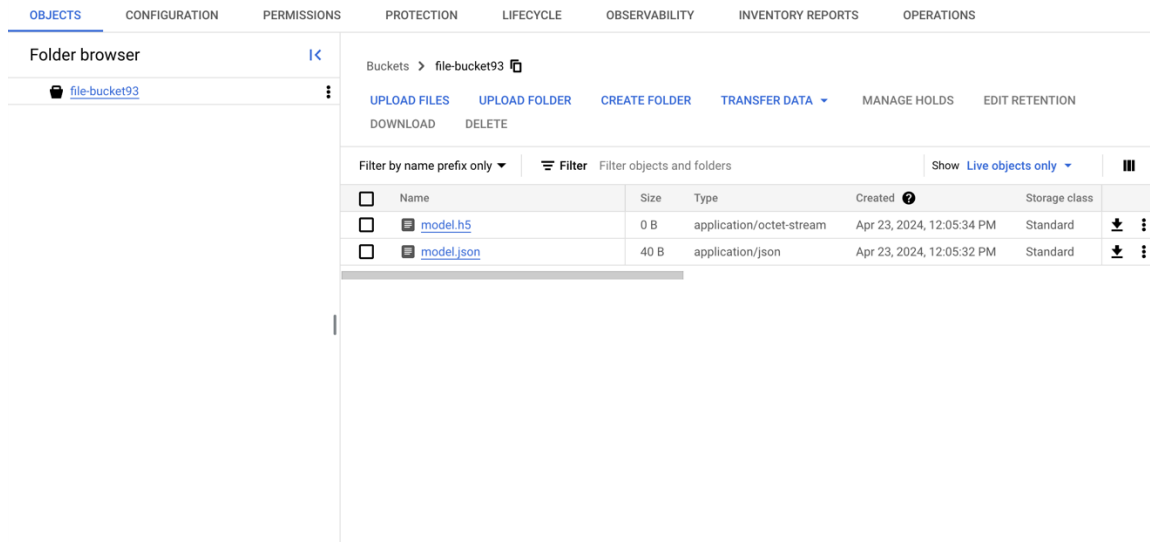


**Figure 3** Flow chart of how MetaMask is used to connect the client to the blockchain

MetaMask is extensively used in the application on the front end as it provides a gateway to connect the data being inputted from the client to the blockchain. Figure 3 shows how by utilizing MetaMask's API within the client's browser, the application can interact with Ethereum and other types of blockchain as well. A key library called Web3.js is also utilized to interact with the blockchain, and this is used to fetch accounts but also to transfer tokens.

## 2.6.2 Google Cloud Storage





**Figure 4** Google Cloud Storage Buckets are used to store data

Google Cloud Storage buckets are scalable and durable storage containers provided by Google Cloud Platform (GCP). The buckets serve as a central repository for storing and accessing various types of data, including objects, files, and media assets and the project will utilize the buckets due to their high availability, reliability, and global accessibility across programming languages and regions.

Since files are not stored inside a local server but instead on the cloud, buckets provide a modular way to upload and download files. Given that access to the buckets does not depend on programming languages and instead relies on specified credentials, it provides a secure way for both customers and the main server to retrieve and upload data.

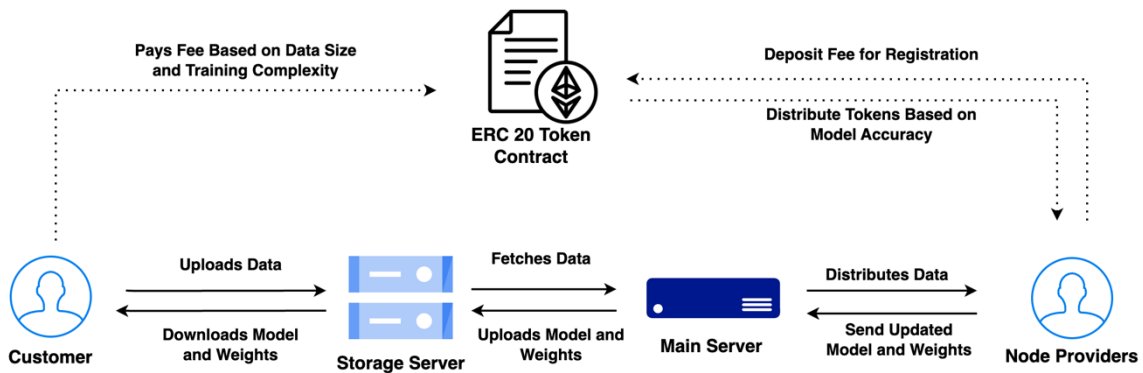
### 2.6.3 Vanilla JavaScript

Vanilla JavaScript instead of other external frameworks will be utilized due to the limited number of functions required for the platform. Because the customer can start the entire machine learning process by uploading data, the website will consist of specific instructions on how to login/register and perform payment to upload data. Specific JavaScript functions will be developed that dynamically enable/disable buttons on the customer's page, providing a smooth user interface. Additional functions will be developed so that once the data is fetched from the

local storage and uploaded to the Google Cloud, relevant alerts will be displayed on the screen to ensure that the customer is notified of the progress.

For the node providers, while there are plans to further improve the home page for providers, the current version of the project supports only registration to the platform. Further use of other external libraries such as React will be considered when additional features are needed.

## 2.7 Machine Learning Development

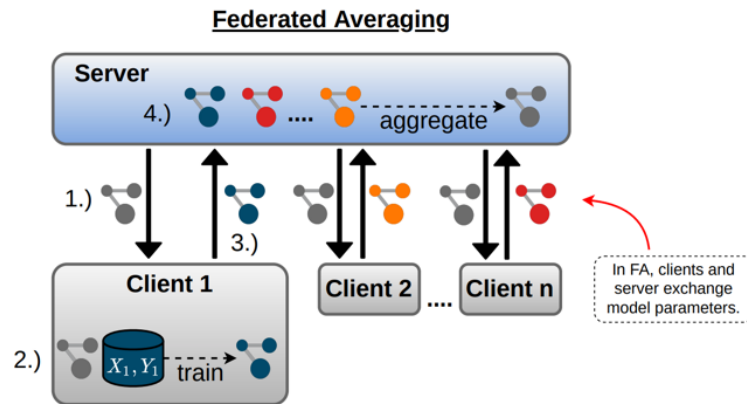


**Figure 5** Overall flow of the application

Figure 5 shows the overall flow of the application. The application will work as follows. Once a customer pays a fee according to the data size, the customer will be able to upload the data onto a cloud storage server. A server attached to the storage server will act as the central node and establish connections with the participating nodes on the network. Participating nodes that have TensorFlow's library of tools will be allocated customer data and an initial model. The node will use the local data to train the initial model. The updated parameters will then be sent back to the server for consolidation. Once the parameters are received, the central server aggregates the models and forms a global model through a technique known as federated averaging.

### 2.7.1 Federated Averaging

While there have been numerous successful applications of deep learning that use stochastic gradient descent (SGD) for optimization, when SGD is applied to federated learning optimization, large rounds of training are required to produce good models on limited data [7]. To tackle this problem, Chen et al. [8] suggests that using large-batch synchronous SGD, where parameters of models are commonly initialized before distribution, outperforms asynchronous approaches where the parameters are independently initialized. When this approach is applied in a federated learning environment, studies show that combining the tuned parameters and averaging the models results in a significant decrease in loss [9].



**Figure 6** The system architecture and data flow for Federated Averaging. Federated Averaging sends updated models [11].

Therefore, this project will use Federated Averaging (FA) to combine models received from participating nodes to form a global model. The system architecture design for FA is shown below (Figure 6). The training is conducted in three key steps:

1. The central server chooses an initial global model and broadcasts the model to participating client nodes.
2. Nodes receive a model that has common initialization across all nodes but are trained using local data that results in an updated model.

3. The updated models are sent back to the central server to create an aggregated global model for the next round of iteration.

### **3. Project Development**

Section 3.1 describes the current back-end development progress. Section 3.2 illustrates the current front-end module of the application. Section 3.2 describes the current implementation of Federated Machine Learning on the platform and the user flow of the entire system.

#### **3.1 Back End Development**

The back end oversees fetching data from the Google Cloud server, distributing data to node providers to train, upload models and weights to the user, and manage rewards according to the model's accuracy. The report will depict three major functionalities in the back end which are interaction with blockchain, API requests handler, and database management.

##### **3.1.1 Interaction with Blockchain**

The role of blockchain in the project is to provide a clear and transparent payment system where users do not have to pay by entering banking details. The project first picked Proof of Stake chain which was Polygon Amoy testnet. Proof of stake is currently the most energy-efficient chain with the fastest transaction time; hence, for development purposes, the project decided to pick this chain. Utilizing Oppenzeppelin's framework, the project managed to create a token contract in Solidity. The project followed the standard ERC-20 token contract with ownable functionality and coined the name of the token as "Distributed Machine Learning Token" (DML). After creating the token contract, we deployed our token through the hardhat framework on Node.js.

From the front end side, payment of user and node providers are managed through the integration with meta mask. Users or node providers can sign the transaction and make payment through MetaMask. Payment is based on data size where 10Mb of data is 1 DML token. However, in the back end the role differs.

The server has a wallet address that is kept outside the code level. This wallet will manage rewarding tokens to node providers according to rating weight, distribute native tokens, which is ETH for the project scope, when malicious node providers are detected, and refund tokens if machine learning fails.

Unlike the front end's interaction, the back end cannot use a meta mask to sign the transaction. Instead, we directly use the hard wallet's private key and conduct the signature at the code level. Hence, it is important to keep the wallet's private key details safe and make sure the wallet's details for not hard coded in programming. The project has used .env files to manage private data such as the wallet's private key. Unless the hacker has direct access to the main server physically or via SSH, it is almost impossible to acquire the private key of the wallet at the network level.

Payment done in code level differs from front end too. First we need to grab token's Application Binary Interface (ABI). Since the project utilized ERC-20 token, we can utilize the standard ABI for ERC-20. Moreover, the project needs token's address. Given both information the team can now access function in the token contract such as transfer. To create a sign transaction the project need to create transaction Object which includes information such as gas limit, gas price, recipient, sender, nonce, and encoded transfer function of given token's ABI.

After creating transaction object it will be signed and send with wallet's private key. The logic of the code is given in Figure 7 below.

```
async function transfer(web3, sender, recipient, contract, amount) {
  try {
    // Get the number of tokens to transfer (convert from wei to token decimals)
    const tokenCrt = new web3.eth.Contract(TokenABI.abi, contract);
    const tokenDecimals = await tokenCrt.methods.decimals().call();
    amount = amount * Math.pow(10, tokenDecimals);
    amount = BigNumber(amount);

    // Transfer tokens from the owner's account to the recipient
    const gasPrice = await web3.eth.getGasPrice();
    const gasLimit = 150000;
    const nonce = await web3.eth.getTransactionCount(sender.address, "latest");

    const txObject = {
      gasLimit: Web3.utils.toHex(gasLimit),
      gasPrice: Web3.utils.toHex(gasPrice),
      from: sender.address,
      to: contract,
      nonce: nonce,
      data: tokenCrt.methods.transfer(recipient, amount).encodeABI(),
    };

    // Sign and send the transaction
    const signedTx = await web3.eth.accounts.signTransaction(
      txObject,
      sender.privateKey
    );
    const receipt = await web3.eth.sendSignedTransaction(
      signedTx.rawTransaction
    );
    return receipt.logs[0].transactionHash;
    console.log(receipt.logs[0]);
  } catch (error) {
    console.error("Error transferring tokens:", error);
  }
}
```

**Figure 7** Screenshot of the transferring ERC-20 token in back-end level. Note that after creating sing transaction with transaction object, transaction are signed and send with private key unlike front end where sign and send is done on MetaMask.

Other than the wallet's security concern, the imperative part is the algorithm for token distribution to node providers who have done the machine learning. To formulate a natural competitive market the project has adopted the following mathematical formula.

$$\text{rewarded token for } x = \frac{\text{Accuracy of Node } x}{\sum \text{Node Accuracy}} \times \text{Token Paid} \times 0.98$$

It is a simple equation that distributes 98% of the token users paid to train data according to the weight of one's accuracy from the total group. 2% is taken as a commission fee from DML platform. It is a sample business scenario the project has adopted to test if the platform can be further improved in the business model in the future hand.

To put it into an example let's look at Figure 7, 4 node providers have finished training the data successfully. the user paid 0.13 DML tokens, 0.11 DML (round down up to the second decimal place) tokens will be distributed after taking out the commission fee.

Node A has accuracy of 75%, node B 55%, node C 55%, and node D 95%. Then through the formula, node A will have:

$$\text{rewarded token for } A = \frac{0.75}{\sum(0.75 + 0.55 + 0.55 + 0.95)} \times 0.11 = 0.03 \text{ DML tokens}$$

```
Ready to proceed Machine Learning
{"status": "success", "data": {"nodeA": "success", "nodeB": "success", "nodeC":
"success", "nodeD": "success"}}
Machine Learning is done return status: success
[
'0xD48180db2C1572C7B04DCE59263c64784ECA2a68',
'0x09F65479B99f74C8eC8BD25ddCcd530A727f5614',
'0x307F7ff4D5e9CfdFD9CD6421D07f26a1197a9C22',
'0xF7F0aB9feE3Fff9bcF8bdAC7e42993fe67Ea7A64'
]
Transaction value: 0.13 tokens
0.03 tokens
0.02 tokens
0.02 tokens
0.04 tokens
Rewarded node providers accordingly: [
'0x7cadc092417857f739a6f65813b461723a7fe3506b539fc8c40fdc740046a3fc',
'0x51be1a862c3fe5fd4317bc64e5dc8772b8373d41c8bbeebd836d4a485d5177b3',
'0xd9ae472aa3762db2f53e08671ae14b2461b9383d161ff3e27e8b4490c9a31720',
'0x0c173ae7f8f5e192c92cff3767de551463a1152390cbb7cffff7aa5a9996ead67'
]
```

**Figure 8** Screenshot of the log text with successful training. Note that a total 4 nodes were utilized during the training and received tokens according to their accuracy.

The project understands that accuracy is not enough to truly measure computational power usage. The project plans to integrate more criteria into account to accurately reflect computational power usage and reward tokens accordingly. Further details will be discussed on future steps later in the report.

Note that the final deliverable for this project was not able to formulate WSS connections with other node providers mainly due to number limitations in Google Cloud Storage. The project was not able to set up more than one virtual machine with GPU and due to lack of supply, the connection towards that VM was extremely unstable. The team has requested 3 virtual machines with GPU but all requests have been denied. Hence, the project had no option but to simulate node providers in the local environment through multithreading. The project understands the deliverables are not as the team planned in methodology. However, the project understands such implementation can be done quickly once given the virtual machines with GPU. Framework for WSS connection training has been already set up. Such limitations will be discussed further in future steps later in the report.

Another aspect back end that needs to be considered is when one of the node providers performs malicious action. For instance, the node provider tries to access the user's training data or alter some training parameters to result in a higher accuracy model through overfitting. Within this project's scope, methods to detect such malicious activities have not been implemented. During the development, the team soon realized detection of such security issues requires additional hardware configuration to simulate a Trusted Execution Environment (TEE). Such limitations will be discussed further in future steps. Yet, the project managed to implement the after steps. If malicious activities have been detected in one of the node providers, such node providers will get banned permanently and will not be able to perform any more training. Moreover, the deposit ETH that the node providers have paid to register will be distributed to non-malicious node providers as they have done their training. The above figure is the log from the main server when malicious activities are detected in one of the nodes.



```

Received data in cloud for user: howard9@connect.hku.hk
Ready to proceed Machine Learning
{"status": "fail", "data": {"nodeA": "fail", "nodeB": "fail", "nodeC": "success", "nodeD": "success"}}
Machine Learning is done return status: fail
Malicious activity detected in the node providers
malicious node provider nodeA has been banned
malicious node provider nodeB has been banned
user will get back token
transfer done
deposited coin will be distributed to other node provider
paying node provider : nodeC
reward amount: 0.001
paying node provider : nodeD
reward amount: 0.001

```

**Figure 9** Screenshot the log text of malicious activities conducted in nodes from the main server. Logs depicts once malicious activities are detected the server will automatically ban malicious nodes, and fully refund tokens to user.

Here, nodeA and nodeB had conducted malicious activities. Hence, the returned model has a chance of contamination where the model can be overfitted. As the model is unreliable, the main server will not return the model to the user instead will fully refund the token he or she has paid. Then nodeA and nodeB will be permanently banned and their deposited ETH has been distributed accordingly to nodeC and nodeD who have performed training accordingly. As nodeC and nodeD need to get rewards for training, rewards will be given from malicious nodes. Note that the above cases are simulation cases. The value of ETH is 0.001, which is very small. As the project utilized a Polygon test net, the generation of faucet tokens where limited to 0.2 ETH per account; hence, a small ETH value had to be used for testing purposes. However, the project expects to have a higher staked amount as a deposit to ensure node providers are not motivated to perform malicious activities.

### 3.1.2 API Requests

The project manages API requests to provide interaction for the front end. API requests can be divided into 3 sections: user login, node provider registration, and ticket creation.

Firstly, the user login has two API requests one is GET and the other is POST. When the user creates an account, a POST request with JSON body including user email, encrypted password, and username will be sent to the back end. Once received, the back end will create an

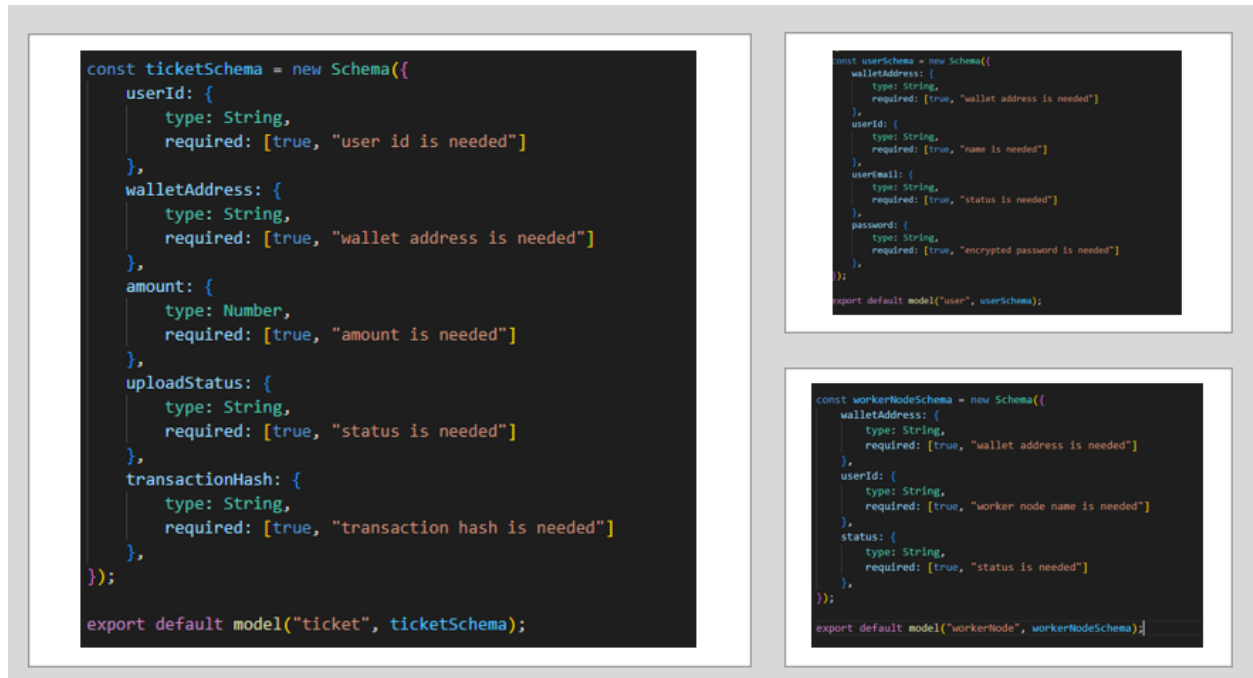
object under the user schema. When the platform checks whether it is a registered account or not, a GET request is made with the path variable key and value pair. The value will be used to check the object in the user schema. Password validation takes an extra step as in the database the platform only holds encrypted passwords to ensure security. Encryption is done utilizing SHA-256. SHA-256 is one function encryption meaning that no one can decrypt a user's password given the hash. This ensures that only the user can log in and the platform and the DML platform do not know the password of the user.

Secondly, the API request is for the registration of node providers. Like that of User login, it has one GET request and one POST request. POST request is used to create node providers information with status on Mongo database and GET request is used to check existing node providers.

Lastly, after the user makes a successful payment and data has been successfully uploaded to Google Cloud Storage, the front end requests to create a ticket in the database. The ticket holds information about the user's email, pay amount, transaction hash, uploaded data, and status. Once a ticket is created in the database listener in the back end will trigger the machine learning process. After the completion of the machine learning and if the training is successful, the back end will upload the created model to Google Cloud Storage and the user will get the downloadable link in his or her email. Also, rewards will be given to the node providers according to accuracy. If it was unsuccessful due to malicious activities in one of the node providers, the user will get a full token refund and the deposited amount from malicious node providers will be distributed to other nodes who have successfully trained as a reward.

### 3.1.3 Database Schema

The database holds three schemas also known as collection in Mongo database. The template for the 3 collections is shown in Figure 10.



**Figure 10** Screenshots of the schema layouts for ticket, user, and node providers. One on the left is for a ticket, the top right is for the user, and the bottom right is for node providers.

Note that the current key and value pairs for each schema are for this first production model. In the future development for concise and suitable key-value pairs will be added and modified.

Other than the implementation of schema there has been an additional feature the team has utilized. The team has opened an additional listener port where the project monitors changes in the ticket schema. If any objects are added, deleted, or modified, the database will give an update directly to the back-end server. If the notification is related to the addition of a new object in the collection, the back end will trigger the machine learning process after downloading user data in Google Cloud Storage. Databases listener also known as `.watch()` allowed the whole process in the back end to be conducted automatically. During the whole process error handling process has been managed. If there is any error encountered in machine learning, payment in blockchain, or sending model via email, such process will notify both back-end user admin.

## 3.2 Front End Development

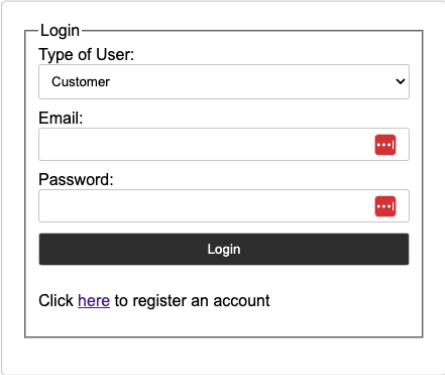
### 3.2.1 Overview of Front End Development

This section explains the front-end features of the current version of the platform and who it ties into the entire user flow. The front-end application aims to provide both customers and node providers with a centralized website to access the platform. To accomplish this, the web application is built using HTML/CSS for the client side and JavaScript for dynamic content. The current version of the platform offers three key features, namely transferring of data across servers, machine learning of datasets, and payout based on accuracy.

### 3.2.2 Authentication Flow

When a user first accesses the web page, they are greeted by a login page.

#### Decentralized Machine Learning Platform



Login

Type of User:  
Customer

Email:

Password:

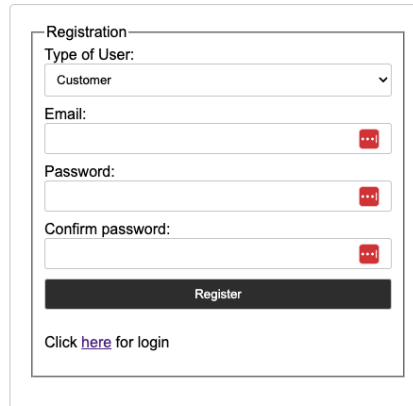
Login

Click [here](#) to register an account

Figure 11 Login page

If the user already has either a customer or provider account, the user can provide the email account that they used to register with and the password. If the user has not yet registered, clicking the hyperlink below the login button takes the user to a separate page for registration.

## Decentralized Machine Learning Platform



The image shows a registration form titled "Registration". It contains the following fields and elements:

- Type of User:** A dropdown menu with "Customer" selected.
- Email:** A text input field with a red eye icon to toggle visibility.
- Password:** A text input field with a red eye icon to toggle visibility.
- Confirm password:** A text input field with a red eye icon to toggle visibility.
- Register:** A black button with white text.
- Click [here](#) for login:** A link below the register button.

**Figure 12** Registration page

The user inputs four fields as shown in Figure 12:

1. Type of User
2. Email
3. Password
4. Confirm password

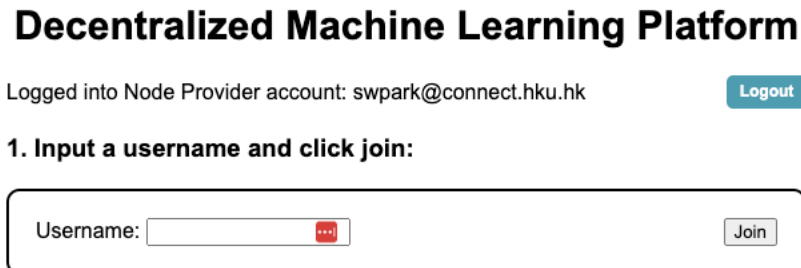
Once the user clicks on the register button, the 'register()' function inside the Javascript code is called using a POST request which checks the database for any duplicate email. If not, the information is passed on to the database to be inserted and the user is taken to a page according to the user type he or she has selected.

If the user already has registered before and after logging in did not close out the browser, the login session is recorded inside the session variables. Once the user logs in, the application sets a 'user' session variable with the user's email to make sure that the particular user is authenticated. So, whenever the user needs to create a new tab, the user is already logged in with that particular account.

When the user clicks the logout button inside the web page, the 'logout()' function will be called and the session variables will be reset so that the user cannot access additional private web pages without being authenticated again. The user will be then redirected to the login page.

### 3.2.3 Node Provider Page

For node providers who wish to join the platform, they will be redirected to the node provider page after successfully creating an account and logging in.



**Decentralized Machine Learning Platform**

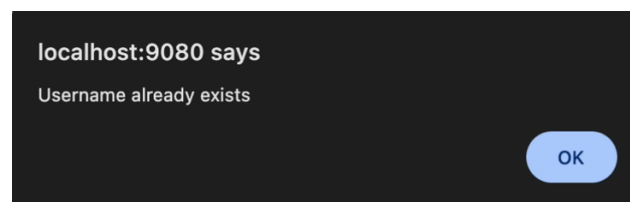
Logged into Node Provider account: swpark@connect.hku.hk [Logout](#)

**1. Input a username and click join:**

Username:  [Join](#)

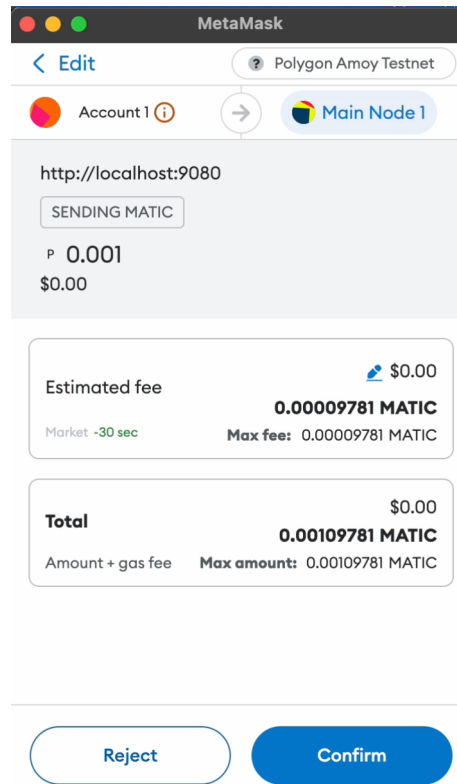
**Figure 13** Page for node providers to join the platform

Figure 13 shows the page for node providers. There is one input field for users to put their username in. The application recommends that users put their email address into the username to make sure that the platform has unique usernames. The current version of the application requires users to manually type in their email address and this is for the application to test whether there are duplicate usernames.



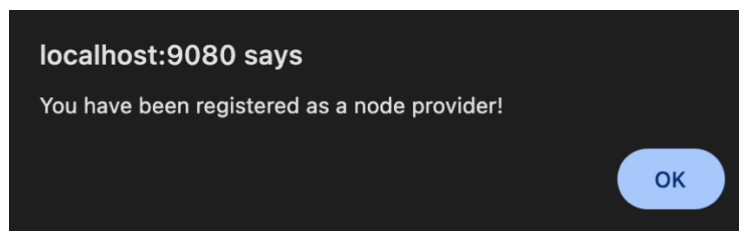
**Figure 14** An alert shows up if the user tries to register for duplicate usernames

Once the user types in a username and clicks the Join button, two functions mainly ‘getUserFunction()’ and ‘joinFunction()’ will be triggered. The ‘getUserFunction()’ fetches the data from the text input field and the ‘joinFunction()’ calls this to then pass the result through an API. This GET request checks if there is an existing entry with the same username. Figure 14 shows that if a user looks to register multiple times with the same username, an alert will show up signaling an error. However, if there is no username of the same name on the database, the ‘joinFunction()’ will move on to insert the user into the system.



**Figure 15** A MetaMask alert instructs how much MATIC is needed to join the platform

To ensure that node providers who register into the system understand that there are repercussions if they act maliciously, the platform takes a deposit from the node providers when they register. A MetaMask notification will be shown to let the users know about the deposit and only when they pay the deposit will the application register the information to the back end.



**Figure 16** An alert tells the node provider that it has been successfully registered

The registration process is complete for node providers once they see the alert saying that they have been registered. Now, node providers can wait for customers to upload data and start using their compute services to perform machine learning.

### 3.2.4 Customer Page


Once the user selects to log in or register as a customer, 'display\_customer\_page()' function will be called to bring up the dedicated customer page to the user.

## Decentralized Machine Learning Platform

Logged into Customer account: swpark@connect.hku.hk

Logout

### 1. Choose a file to run ML on:

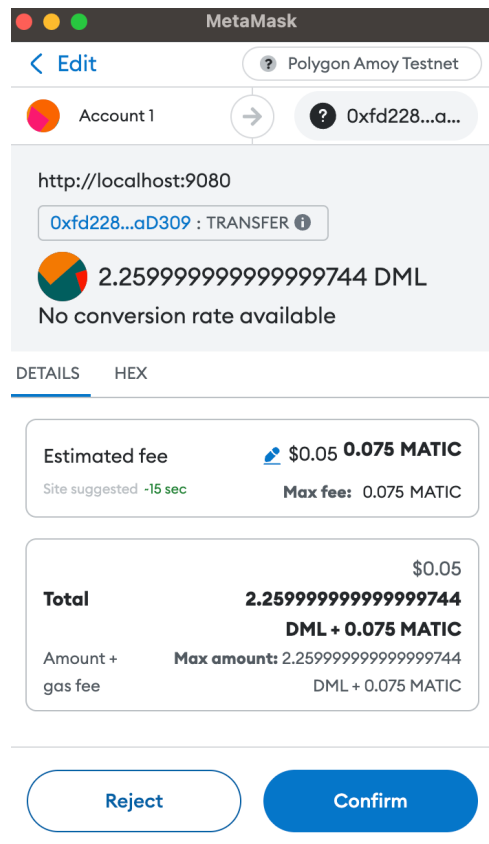


The screenshot shows a web interface for a customer. At the top, it says "Decentralized Machine Learning Platform". Below that, it indicates the user is logged in as "swpark@connect.hku.hk" with a "Logout" button. The main heading is "1. Choose a file to run ML on:". Underneath, there is a file selection area with a "Choose File" button, the text "No file chosen", a "Pay" button, and an "Upload" button.

Figure 17 Customer page before data upload

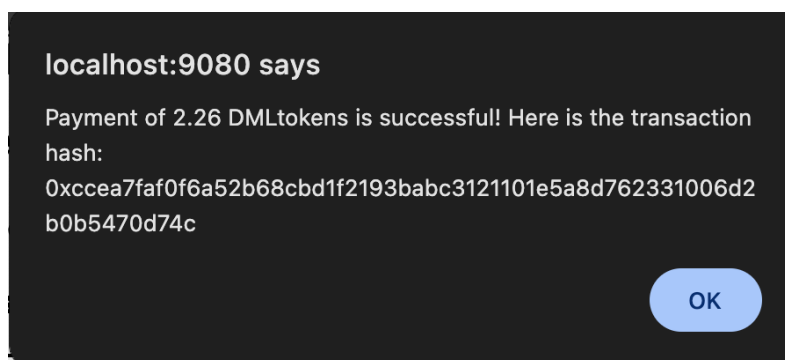
The email account of the customer will be displayed on the top along with the log-out button. Below, there are three buttons, Choose File, Pay, and Upload. At this stage, the customer can select a file to run machine learning on. Notice that the Upload button cannot be clicked, and the Pay button has to be pressed first before moving on to data upload. Once the data has been selected by clicking on the Choose File button, the user will have to press the pay button which will trigger the function 'payFunction()' in the script.js JavaScript file. This file serves as the code needed for communicating with the main server and MetaMask. MetaMask is needed for recording the transaction onto the blockchain and for fetching the information from the blockchain. The 'payFunction()' calculated the amount of DML tokens, tokens that the project uses to pay for transactions, needed to upload the specified amount of data. Here, we use the logic of 1 DML token per 10 Megabytes of data. So, for example, once we upload data of file size 23.1 MB, we are required to pay 2.26 DML plus any additional gas fees. Because the project is currently using the Polygon Amoy testnet, the gas fees are paid using MATIC.





**Figure 18** MetaMask notification of a pending transaction

A notification like Figure 18 will appear in the top right-hand corner and the customer will be able to pay using his or her account. Once the user presses confirm, the function moves on by recording the transaction onto the blockchain and the DML tokens will be released from the account to the deployed contract.



**Figure 19** Alert describing that the transaction is successful along with the transaction hash

Once the alert like in Figure 19 shows up, the user will now be able to click the Upload button on the main page.

```

async function fileUpload() {
  var input = document.getElementById("fileinput");
  const file = input.files[0];
  console.log(file);
  const fileContentss = await readFileSyncBrowser(file);
  console.log(fileContentss);
  const bucketName = "file-bucket93";

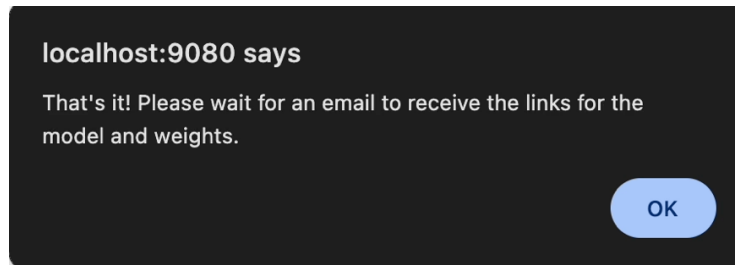
  const projectId = "intrepid-stock-411303";

  const uploadURL = `https://storage.googleapis.com/upload/storage/v1/b/${bucketName}/o?uploadType=media&name=${encodeURIComponent(
    file.name
  )}`;
  try {
    await fetch(uploadURL, {
      method: "POST",
      headers: {
        Authorization:
          "Bearer",
        "Content-Type": "text/plain",
      },
      body: fileContentss,
    })
    .then((response) => {
      if (response.ok) {
        alert(
          "That's it! Please wait for an email to receive the links for the model and weights."
        );
        console.log("File uploaded successfully");
      } else {
        console.error("Error uploading file:", response.status);
      }
    })
    .catch((error) => {
      console.error("Error uploading file:", error);
    });
  }
}

```

**Figure 20** Code example of using POST request for Google Cloud Storage

Once the customer clicks on the Upload button, the following code in Figure 20 will execute. Using HTTP POST requests and the API provided by Google Cloud, the project uploads the file to the predefined bucket in Google Cloud Storage. Authorization of the request is through a bearer token unique to the project and user while the body content is defined through a function called 'readFileSyncBrowser()' which takes the selected file and reads the contents through a buffer. The uploaded data will keep the name of the file and details of the user, wallet address, and status of the transaction as 'Uploaded', will be sent to the back end server through a POST request. This collection of data is called a ticket and it ensures that the transactions and data from the front end module are communicated to the back end server and eventually recorded in the database.



**Figure 21** Alert that indicates uploading of data has been completed

Once the data has been uploaded and a ticket sent to the back end, the alert shown in Figure 21 pops up. This indicates that the process for uploading data and paying for the machine learning service has been finished and the user now should wait for an email to download the model and weights.

### **3.3 Distributed Machine Learning Development**

#### **3.3.1 Overview of Distributed Machine Learning Development**

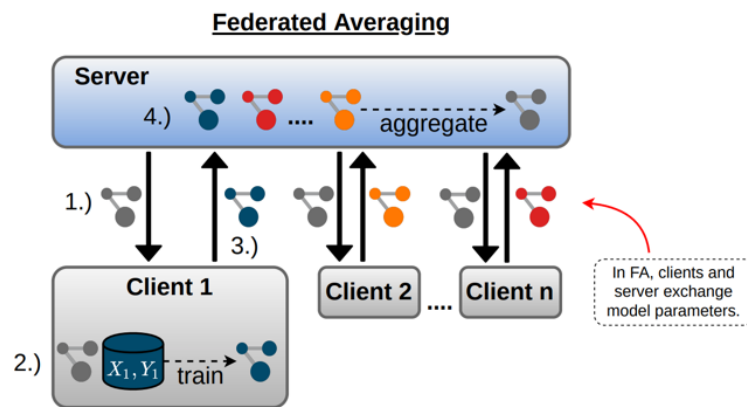
The machine learning process for the application uses Python and TensorFlow to streamline the entire operation. Once data is uploaded to the cloud server, the back end fetches the data and stores it into the main server. The project consists of multiple Python files including, 'emnist\_fedavg\_main.py' which is comprised of the machine learning execution logic of the server (main server), 'simple\_fedavg\_tf.py' which handles computation as clients (node providers), and 'simple\_fedavg\_tff.py' which handles the orchestration strategy of the server and clients. The code here is based on a TensorFlow Federated resource provided by Google and changes were made to the code to fit the project [10].

#### **3.3.2 Node Providers**

The node providers are represented in the 'simple\_fedavg\_tf.py' file which consists of functions that a client is expected to execute. The most important function is 'client\_update()' which takes in a local model and a dataset specified by the main server to perform machine learning. The function returns the weights delta, client weights, and model outputs.

### 3.3.3 Main Server

The ‘`emnist_fedavg_main.py`’ file acts as the main server. This file collects the data, splits the data according to the number of node providers, and runs iterations to collect the model outputs from the node providers. The file relies on functions defined in the ‘`simple_fedavg_tff.py`’ file to collect model outputs and weights from the clients. The function ‘`build_federated_averaging_process()`’ takes in a Keras model and server and client optimizers as arguments to create a TensorFlow Federated iterative process. This process forms the basis of the server and subsequent iterative processes across the client model outputs increase the accuracy of the server state. It is in this function that ‘`client_update()`’ from the client server gets executed to produce model outputs.



**Figure 22** The system architecture and data flow for Federated Averaging. Federated Averaging sends updated models. Figure adapted from [11].

Figure 22 displays the process for federated averaging, an algorithm that combines the model parameters from the node providers and aggregates them to form a global model.

```

for round_num in range(FLAGS.total_rounds):
    sampled_clients = np.random.choice(
        train_data.client_ids, size=FLAGS.train_clients_per_round, replace=False
    )
    sampled_train_data = [
        train_data.create_tf_dataset_for_client(client)
        for client in sampled_clients
    ]
    server_state, train_metrics, cw, mo = iterative_process.next(
        server_state, sampled_train_data
    )
    print("=====")
    print(f'Round {round_num}')
    print(f'\tTraining metrics: {train_metrics}')
    if round_num % FLAGS.rounds_per_eval == 0:
        server_state.model.assign_weights_to(keras_model)
        accuracy = evaluate(keras_model, test_data)
        print(f'\tValidation accuracy: {accuracy * 100.0:.2f}%')

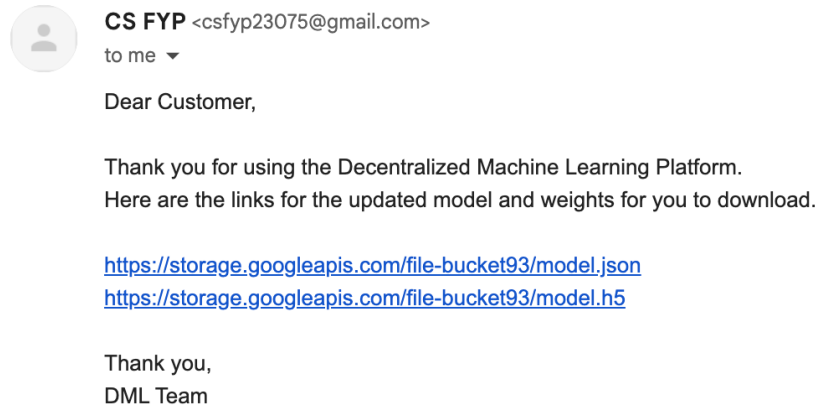
```

**Figure 23** Code to run multiple iterations of federated machine learning

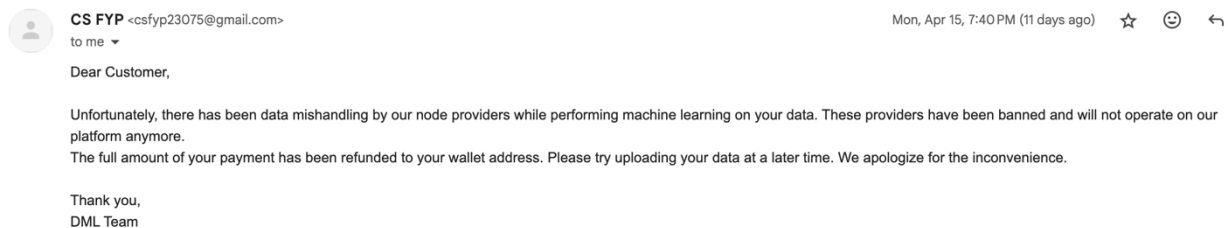
Figure 23 shows part of the code where the dataset for clients is separated and through the iterative process, the server state is updated, and model outputs are returned. At the final iteration, the server state's weights are assigned to the original model to create an updated model. This model is the result that customers will be able to download and can later use TensorFlow to load back the model.

### 3.3.4 Uploading Model and Notifying Customers

For customers to get the model, the application uses Google Cloud Storage and SMTP to email the customers. After saving the model as a JSON file and the weights as an H5 file, the program uses the Google Cloud Storage Python library to upload both files into a bucket.



**Figure 24** If machine learning is successful, customers receive an email with links to download the model and weights



**Figure 25** If machine learning is not successful, then the entire amount the customer initially paid is refunded

The final process after uploading the files to Google Cloud Storage is to notify the customers with the links to the model and weights. The main server uses a function called 'sendEmail()' which uses SMTP and sends an email to the client. The parameters of the function will be defined by the result of the machine learning process. If the process is completed, then, an email like Figure 24 will be sent to the customer. The customer will have to click on the link to download the content at which point the customer can locally load the model using TensorFlow to perform additional machine learning. If the process fails due to malicious activity in the workflow, then an email like Figure 25 will be sent. The email describes that there has been data mishandling by node providers and that the application will refund the entire amount back to the customer.

### 3.3.5 Summary

To conclude the description of machine learning, the Decentralized Machine Learning platform utilizes Python and TensorFlow to perform federated averaging across different node providers. It provides an alternative solution for block miners to utilize their computing power to provide machine learning services for different customers.

## 4. Project Schedule and Future Work

Section 4.1 discusses the project schedule that has been accomplished so far. Section 4.2 discusses the future work and improvements of the project. The project can be further developed by improving security, deploying cloud servers, and experimenting with different algorithms.

### 4.1 Project Schedule

Objective	Deadline	Details	Status
Preliminary Blockchain Network Setup	November 15th 2023	Set up blockchain network along with servers	Completed
Integration of Distributed Machine Learning system	Dec 31st 2023	Integrate Google Cloud Platform with TensorFlow	Completed
Peer to Peer Network Implementation	February 15th 2024	Develop a fully functional network layer that allows large data to set to be transferred at high-speed rate	Completed
Pivot to develop dApp	February 28th 2024	Build application architecture and customize previously built servers for dApp	Completed
Development of Back-end	March 15th 2024	Build back end server to integrate the database and machine learning code	Completed
Development of Front-end	March 20th 2024	Build front end for the user interface	Completed

Testing of the platform	April 7th 2024	Testing entire platform and automating entire process	Completed
Deployment	April 15 <sup>th</sup> 2024	Deploy the platform	Completed

**Table 1** Project Time Schedule. It outlines the objective, deadline, and status of the tasks required to finish the project.

Although the project had to be shifted and deviated from the original plan, the new project has been entirely completed with the website able to be used in localhost and the back-end APIs deployed on render. Shifting from constructing a blockchain with a new consensus layer to developing a dApp platform seems like a different project, yet the motivation of the project remains the same. Simulate an environment where node providers can earn tokens as a reward for performing machine learning as proof of useful work and users can train data utilizing our services.

## 4.2 Future Work

### 4.2.1 Overview of Future Work

The following section describes further plans to improve and develop the project. Section 4.2.2 describes further work needed for efficient communication between node providers and the server, Section 4.2.3 mentions the further work needed for data security and encryption, 4.2.4 talks about the work needed for cloud deployment, and lastly, 4.2.5 discusses further work needed for improvement of federated machine learning.

### 4.2.2 Back End

The major limitation faced by the current back-end server is the lack of GPU power from the deployment platform. Currently, the back end is deployed on render.com. However, with the current plan we subscribe to we are given limited computational power and storage. Most importantly they did not provide GPU in the deployed server. Training machine learning in such an environment was not enough to run a federated averaging model in the deployed server. Therefore, the project deployed only the API management in the server and utilized our



computational power to perform machine learning. This was possible as we only needed to open the port and listen to changes in the database. Such limitations can be easily overcome by purchasing better deployment servers with GPU and storage devices. The team will look more at cloud server providers such as Lambda labs which provides servers with GPU, especially for machine learning purposes.

Moreover, another limitation was the setup of WSS connections with the node providers. The first limitation was that Google Cloud Platform did not provide up to 2 or more virtual machines with GPU. They limited the quota by one and due to high demand, SSH connection to that virtual machine was extremely unstable. Hence the project has simulated 4 nodes in the local environment through multithreading. As the sample dataset did not require high specification training was able to be performed. In the future, the team plans to set up multiple virtual machines with GPU, establish WSS connections, and train data in a distributed manner.

#### **4.2.3 Security Measurements**

During the development of the project, the team realized that implementation of detecting “malicious activities” in node provider's server remains a difficult challenge. For instance, how can the main server know that the node provider tries to read or modify data sets or training? How can such detection happen without having the authority over physical computational power?

Identifying such limitations, the project has researched two major aspects: secure federated averaging and a trusted execution environment. One way to ensure the security of data is by passing encrypted or non-readable data to the node providers. As such data has no value for node providers, they have no motivation to intercept the data sets. Such a method is done through a differential privacy technique which obfuscates the exchanged messages by properly adding Gaussian noise to Stochastic Gradient Descent (SGD) and allows it to maintain a convergence rate of  $1/t$  where  $t$  is the total number of SGD from worker nodes [13]. A detailed algorithm for secure federated averaging is shown below in the figure.

---

**Algorithm 1** Secure FedAvg

---

```

1: Input: Initial model  $\bar{w}_0$  and step size  $\eta_0$ . The PS broad-
   casts  $\bar{w}_0$  to all clients ( $\mathcal{S}_0 = [N]$ ).
2: for  $t = 0, 1, \dots, T - 1$  do
3:   Client side:
4:   for  $k \in \mathcal{S}_t$  in parallel do
5:     if  $\text{mod}(t, Q) = 0$  then
6:       Set  $w_t^k = \bar{w}_t$ .
7:     end if
8:     Sample a mini-batch  $\xi_t^k$  from  $\mathcal{D}_k$  and calculate the
       local gradient  $g_t^k = \nabla F_k(w_t^k; \xi_t^k, b)$ .
9:     if  $\text{mod}(t + 1, Q) \neq 0$  then
10:       $w_{t+1}^k \leftarrow w_t^k - \eta_t g_t^k$ ,
11:     else if  $\text{mod}(t + 1, Q) = 0$  then
12:       $w_{t+1}^k \leftarrow (w_t^k - \eta_t g_t^k) + z_t^k, z_t^k \sim \mathcal{N}(\mathbf{0}, \sigma_{t,k}^2 \mathbf{I}_M)$ .
13:      Send  $w_{t+1}^k$  to the PS.
14:     end if
15:   end for
16:   for  $k \notin \mathcal{S}_t$  in parallel do
17:      $w_{t+1}^k = w_t^k$ .
18:   end for
19:   Server side:
20:   if  $\text{mod}(t + 1, Q) = 0$  then
21:      $\bar{w}_{t+1} = \frac{N}{K} \sum_{k \in \mathcal{S}_t} p_k w_{t+1}^k$ .
22:     Select a subset of clients  $\mathcal{S}_{t+1}$  by sampling without
       replacement, and broadcast  $\bar{w}_{t+1}$  to all clients.
23:   end if
24: end for

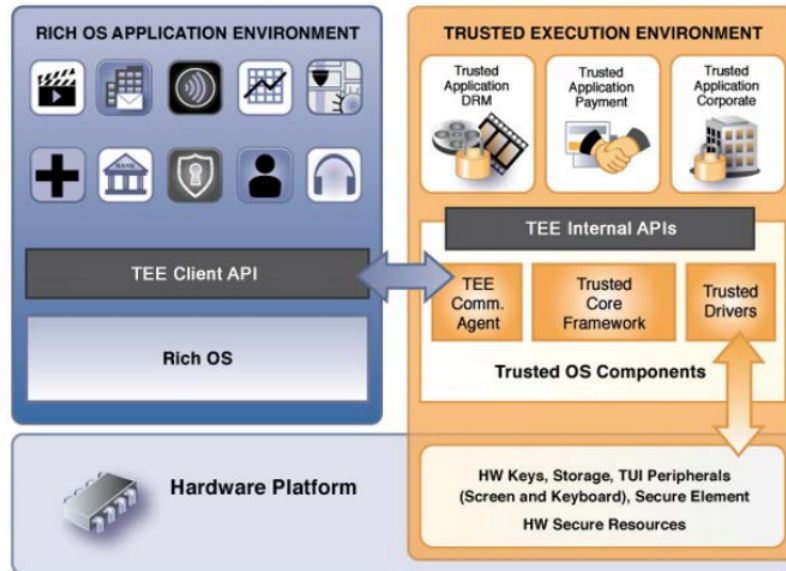
```

---

**Figure 26** Algorithm of Secure FedAvg. Illustrates the process of adding Gaussian noise for data that has been send from main server. Allows worker nodes to have noised data where it itself does not reveal original data of the user [15].

By adding extra noise and keeping the convergence rate to  $1/t$ , it allows training to be done on encrypted data and returned parameters can be used for model prediction for original datasets.

However, such a method does not reveal the fact that node providers read the datasets, which remains a vulnerability point. Such areas can be checked from the main server by setting Trusted Execution Environment (TEE) on node providers. TEE is an environment to execute code in an isolated environment through hardware setup providing the confidentiality and integrity of code and data [14]. As shown in the figure below, the application will run in a trusted environment that has limited access to the hardware. Data applications sent to TEE can only be trained within that environment and cannot be breached for reading or writing.



**Figure 27** Sample diagram showing the architecture design of TEE. Left is the application API request, and such is process in TEE with internal API. Note that TEE has it's own limitation of hardware spaces allowing a secure environment for code execution [14].

Through TEE, the main server can now know whether the node providers try to read or write the user's dataset. Yet, the major problem with TEE is that the project needs extra hardware configuration such as Intel SGX or AMD SEV. Without such hardware, TEE cannot be implemented. This limits the joiners of node providers which might lead to supply and demand problems of the platform. If there are not a lot of node providers, competition will not be performed, and users will not be encouraged to use the platform. There are other ways to perform isolated environments for the execution of code, yet detection of such malicious activities remains a main challenge to be studied and researched.

#### 4.2.4 Reward & Payment System

The major limitation of the reward and payment system for the current platform is that it does not accurately reflect the value to be paid and rewarded. For instance, as shown in the table below.

Current Scheme	
Payment Scheme	Reward Scheme
1 Token per 10Mb	Accuracy weighting system

**Table 2** Outlines the current payment and reward schema for DML platform.

The user pays according only to the size of the training data set and the reward is given solely based on the accuracy of the node providers. However, the objective of the project is to reward tokens based on computational power usage; hence, payment should be an estimate of how much computational power will be used and the reward should be the reflection of the computational power. As the size of data and accuracy are not enough to reflect computational power the project plans to change the scheme accordingly in the future.

Improved Scheme	
Payment Scheme	Reward Scheme
Calculated based on Data size + Complexity of the model	Weighting system based on: <ul style="list-style-type: none"> <li>• Energy usage</li> <li>• Duration</li> <li>• Accuracy</li> <li>• F1 score</li> <li>• GPU usage</li> <li>• Number of process data</li> </ul>

**Table 3** Outlines the improved scheme for the payment and reward system in the future of the DML platform.

The team is still conducting heavy research on what criteria should be chosen to effectively estimate computational power in payment schemes and reflect on reward schemes. The key goal is to create a competitive market nature that motivates node providers to have better computational setups to get more tokens so that users can train data sets in a high computational environment.

#### **4.2.5 Message Transfer**

Current implementation of data transfer relies on Secure Copy Protocol (SCP) that is inside the Google Cloud Storage library. While transferring files using SCP does have advantages such as security and simplicity, however, it lacks interactivity which is especially important when transferring messages and not data across multiple node providers. Especially when only model parameters need to be shared, SCP lacks speed. Another alternative can be the Pub/Sub protocol provided by Google Cloud. Considering that Pub/Sub provides topics that any server can subscribe to presents a cost-effective and scalable solution for many more node providers.

#### **4.2.6 Improved Security**

Data that is currently uploaded to the cloud server and then distributed to the different node providers does not have any security mechanism that prohibits node providers from maliciously modifying or retrieving customers' data. The project can be further developed by implementing homomorphic encryption into federated learning. There are clear benefits to this. By using public and private keys for node providers, the main server, and customers, customers would have the utmost confidentiality of their data. Encryption would allow the full trust of customers and customers would be able to upload sensitive data to perform machine learning. There are multiple studies including an IBM study [12] where it states that there would be minimal disruption to the training time using encrypted data. However, because there has not yet been any commercial application of federated learning on homomorphically encrypted data, potential breakthroughs can significantly improve the security of the project.

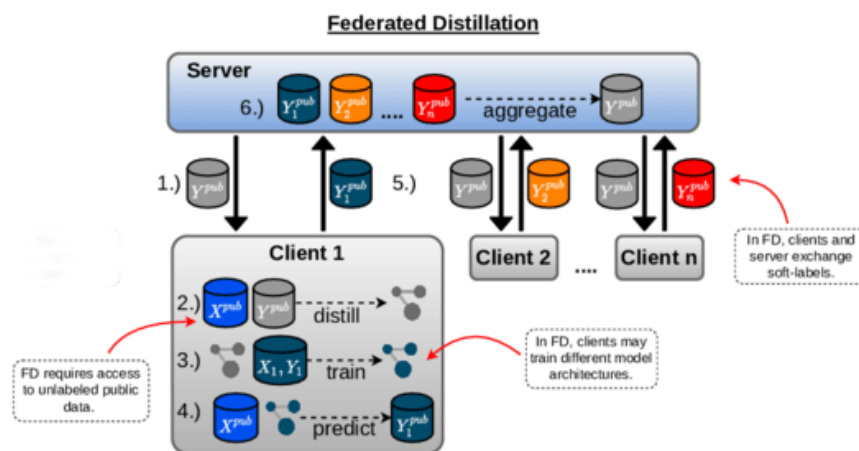
#### **4.2.7 Cloud Servers**

In our current version of the project, all the machine learning files are stored on one server and handle distributed computation in a simulated environment. While 'emnist\_fedavg\_main.py', which provides the code to iterate through multiple rounds of federated learning, can be stored locally in the main server, files that need to be executed by the client would be hosted in a cloud server that acts as a node provider. To truly replicate an environment where actual node providers register to the system, changing the code so that

machine learning can take place either on private data or public data is needed. In the case of private data, sharing of large amounts of data will be a bottleneck so developing an efficient data pipeline would be needed while for public data, exchanging model parameters securely through Pub/Sub would have to be developed as well.

#### 4.2.8 Distributed Machine Learning

The current version of machine learning utilizes datasets the customer has uploaded to perform machine learning. However, due to the abundance of public datasets on the Internet, applying different sources of datasets that are similar in context can bring increased diversity and a bigger sample size for training.



**Figure 28** The system architecture and data flow for Federated Averaging. Federated Averaging sends updated models. Figure adapted from [11].

This also works in conjunction with a different optimization algorithm called Federated Distillation. The way the algorithm works is as follows:

1. The central server chooses an initial subset of soft-labels from the public dataset and broadcasts it to the participating client nodes.
2. Nodes receive the soft-labels and update the local model by putting through/distilling the soft-labels according to the model parameters.
3. The model is then trained using the local data to produce an improved model.
4. The public data is then used against the updated model. This results in a newly

created set of soft-labels to send back to the central server.

5. The soft-labels are aggregated, and the global model is updated according to the soft-labels. This process is iterated until the central server decides the model is accurate enough.

While the algorithm had been researched at the start of the project, due to the scope of the project being focused on private data, federated distillation was not implemented into the project. However, should the customer look to utilize public datasets along with private data, then federated distillation can be used to improve the federated learning process.

The challenge would be to convert the existing machine learning pipelines to accommodate the public datasets. Once the customer has selected a public dataset to train with the private data, the platform would have to consider data compatibility, data quality, and biases inherent with the public dataset so that it does not significantly affect the model in one direction or another.

Another modifiable component would be to provide a customizable option for customers. While current development has certain parameters defined in the program, adding this feature would provide users the flexibility to produce even more accurate models. As such, the user interface would have to be developed to receive extra inputs for machine learning such as learning rate, batch size, and number of rounds.

## **5. Conclusion**

To tackle the problem of high energy consumption in blockchain with the Proof of Work consensus algorithm, the project aims to convert the wasted energy to perform machine learning tasks. The project is expected to manage and convert wasted energy to useful energy via a decentralized application. The project consists of three components. First, the project includes a front-end interface for both user and node providers. Users can freely register and log in as both customers and node providers. Second, the project has a main server that accepts API requests from both the user interface and node providers. The project's server accepts these API requests for data transfer and communication. Lastly, participating nodes are simulated in a cloud

environment. Using Google Cloud Platform (GCP), servers are set up to receive training data and models.

The project introduces a new form of decentralized application that provides machine learning services to customers. Formulated from the mixture of Proof of Work, Proof of Stake, and computational power rating system, the blockchain allows the user to train custom data using the system, and node providers receive tokens based on accuracy as a reward. Development of the platform has been fully completed and the entire process from uploading data to notifying the customer of the results has been fully automated. The Decentralized Machine Learning platform provides machine learning services to a wide range of customers while preserving the market for computer power providers to earn tokens.



## References

- [1] Stanford Graduate School of Business, "Popular Blockchain Use Cases Across Industries", 2022.
- [2] IBM, "Blockchain Consulting and Services – IBM Blockchain", 2022.
- [3] J. W. Kirkwood, "From work to proof of work: Meaning and value after blockchain," *Critical Inquiry*, vol. 48, no. 2, pp. 360-380, 2022.
- [4] "Cambridge Bitcoin Electricity Consumption Index (CBECEI)," Cambridge Centre for Alternative Finance, Cambridge, UK, 2023.
- [5] Jeffrey Dean, Greg S. Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Quoc V. Le, Mark Z. Mao, Marc'Aurelio Ranzato, Andrew Senior, Paul Tucker, Ke Yang, and Andrew Y. Ng. 2012. Large scale distributed deep networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1 (NIPS'12)*. Curran Associates Inc., Red Hook, NY, USA, 1223–1231.
- [6] M. Ball, A. Rosen, M. Sabin, N. Prashant, and Vasudevan, "Proofs of Useful Work," 2021.
- [7] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, 2015.
- [8] Jianmin Chen, Rajat Monga, Samy Bengio, and Rafal Jozefowicz. Revisiting distributed synchronous sgd. In *ICLR Workshop Track*, 2016.
- [9] M. H. Brendan, E. Moore, D. Ramage, S. Hampson, and Blaise, "Communication-Efficient Learning of Deep Networks from Decentralized Data," *arXiv (Cornell University)*, Feb. 2016, doi: <https://doi.org/10.48550/arxiv.1602.05629>.
- [10] The TensorFlow Federated Authors. (2018). TensorFlow Federated (Version 0.76.0) [Computer software]. <https://github.com/tensorflow/federated>
- [11] F. Sattler, A. Marbán, R. Rischke, and W. Samek, "Communication-Efficient Federated Distillation," *arXiv (Cornell University)*, Dec. 2020, doi: <https://doi.org/10.48550/arxiv.2012.00632.6>
- [12] Baracaldo, N., & Shaul, H. (2022, December 16). *Federated learning meets Homomorphic encryption* | *ibm research blog*. Federated Learning meets Homomorphic Encryption. <https://research.ibm.com/blog/federated-learning-homomorphic-encryption>

- [13] Y. Li, T. -H. Chang and C. -Y. Chi, "Secure Federated Averaging Algorithm with Differential Privacy," 2020 IEEE 30th International Workshop on Machine Learning for Signal Processing (MLSP), Espoo, Finland, 2020, pp. 1-6, doi: 10.1109/MLSP49062.2020.9231531 .
- [14] N. Hughes, "An introduction to the trusted execution environment for Mobile Services Security," Account Payments Specialists, <https://www.account.com/an-introduction-to-the-trusted-execution-environment-for-mobile-services-security/> .

## **Appendices**