UNIVERSITY OF INFORMATION TECHNOLOGY

FACULTY OF COMPUTER NETWORK AND COMMUNICATION

Logo, company name

Description automatically generated

**FINAL REPORT**

Subject: Cryptography - NT219.N21.ANTN

Semester II (2022 – 2023)

**PRIVACY-PRESERVING USING HOMOMORPHIC ENCRYPTION ON FEDERATED LEARNING**

Lecturer : TS. Nguyễn Ngọc Tự

Student 1: 21520011 – Võ Nguyên Chương

Student 2: 21520757 – Nguyên Hữu Dương

**Hồ Chí Minh City, June 2023**

Acknowledgement

To complete the final course project, first of all, we would like to express our sincere and deep gratitude to Mr. Nguyen Ngoc Tu. He is the one who directly guides, gives suggestions and orientation to help us complete the project in the best way. Once again thank you very much sir!

Sincerely, Author Team

Võ Nguyên Chương

Nguyễn Hữu Dương

1. **Problem statement**
   1. **Introduction**

Machine learning (ML) is a widely used technique in almost all fields, where a computer system can learn from data to improve its performance. This technique is widely used in many application areas such as image recognition, natural language processing, and machine translation. Federated learning is a machine learning technique where the training data is distributed across multiple machines, and the learning process is performed in a collaborative manner. This technique can be used to improve the privacy and security of medical data.

Medical data is often highly sensitive and is often subject to data privacy and security concerns. For example, a person’s health information is often confidential and can be used to identify the person. Thus, it is essential to protect the privacy and security of medical data. Health Insurance Portability and Accountability Act (HIPAA) (US Department of Health and Human Services, 2014) and General Data Protection Regulation (GDPR) (The European Union ,2018) strictly mandate personal health information privacy. There are various methods to safeguard private information. Federated learning is one of the techniques that can be utilized for the protection of sensitive data during multi-party computation tasks. This technique can be used to improve the privacy and security of medical data by preventing the data from being centralized and vulnerable.

Keeping the data local is not sufficient for the security of the data and the ML model. However, there are several privacy attacks on ML models to get private data. For example, the attackers can use the gradient information of the deep learning model to get the sensitive information. Thus, the ML model itself should be protected from adversaries as well. Furthermore, the machine learning model needs to be protected from third parties (cloud computing server), and only the stakeholders (hospitals) have access to the global model. One of the solutions for this problem is homomorphic encryption-based model protection from the adversary collaborator. Homomorphic encryption is a technique where the data can be encrypted, and the operations can be performed on the encrypted data. This technique can be used to protect the ML model from adversaries.

This project proposes a privacy-preserving federated learning algorithm based on Logistic Regression teachnique in Machine learning for medical data using homomorphic encryption. The proposed algorithm uses a secure multi-party computation protocol to protect the deep learning model from adversaries. We evaluate the proposed algorithm using a real-world medical dataset and show that the proposed algorithm can protect the deep learning model from adversaries.

* 1. **Scenario**

Let's take the actual context as follows: each hospital in the process of medical examination and treatment generates its own patient data set. Hospitals have the ability to train machine learning models based on their datasets for medical diagnostic purposes.

However, just the amount of patient data in each hospital is not enough to create a quality model. So they want to develop their model by collaborating with many hospitals to increase the amount of training data for this model.

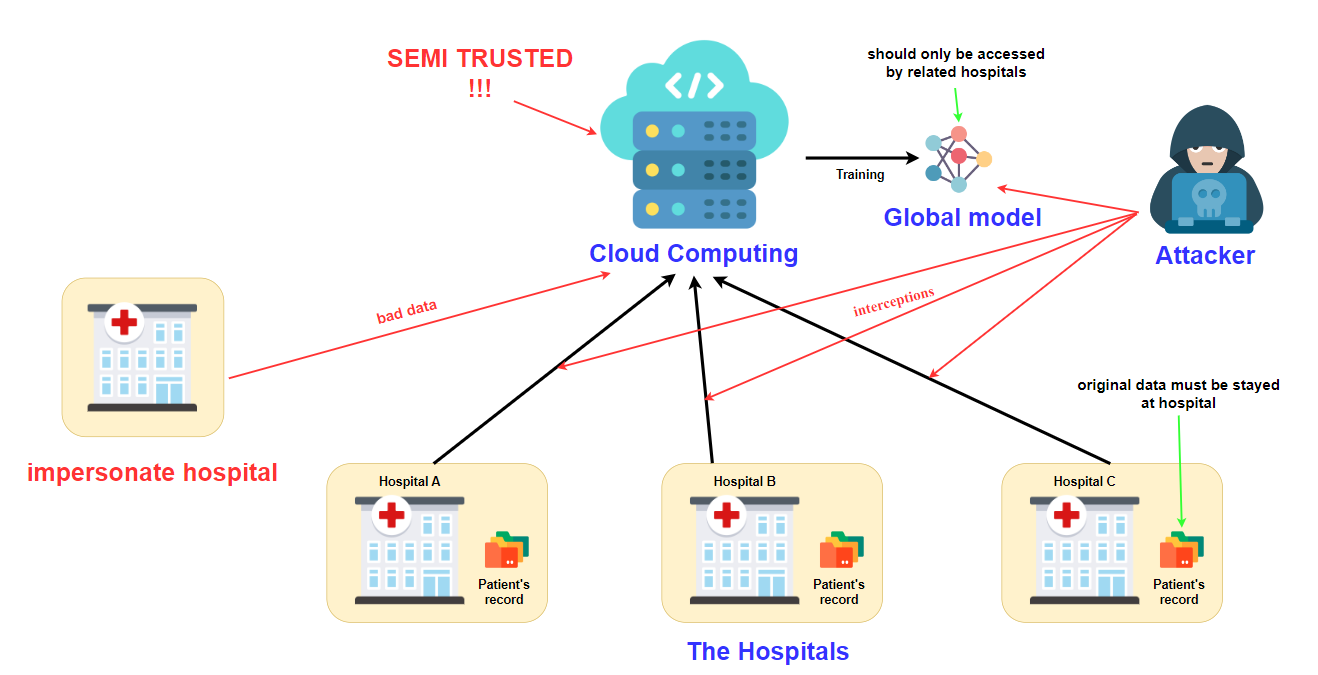


Figure 1.2. Threads to traditional Federated Learning system

The most important problem is that the patient data of hospitals cannot be gathered in one place, but must be distributed in each hospital (data cannot leave the hospital). Therefore, the Federated Learning model is suitable to solve the decentrialize datasets problem on machine learning.

Looking at Figure 1.2, we can see some threats to the traditional Federated Learning (FL) model:

* **Cloud computing:** This is definitely a semi-trusted party, in addition to providing computing power, it is also "curious" about the model parameter sets that the hospital side sends. Depending on the machine learning model, the characteristics of the patient data set can be traced back or predicted to the original characteristics, which compromises the privacy of patient data. Besides, an important thing that we need to mention is that the cloud server can completely steal the model and sell it to another party. The hospital side needs to take measures to prevent it, which the conventional FL model cannot do.
* **Attacker:** It can be anyone, including hospitals, where it performs interceptions, feeds data on the transmission lines from which to extract important information, retrieves features from the model's parameter.
* **Hospital:** As I mentioned before, it is possible for a patient to be an attacker and it can be harmful to other hospitals. In addition, the system needs to pay attention to prevent impersonating the hospital to send malicious data sets, leading to the training not achieving the desired results.
  1. **Security goals**

There are two main security goals that we need to work towards:

* Protect the privacy-preserving of medical data from the orther hospitals, attacker, cloud computing server (semi-trusted)
* Protect global model from any model steal attack (usually from the aggregator) and malicious datasets from impersonating hospitals.

1. **Preliminaries**

Before proposing the solution model, we will present the section preliminaries.

* 1. **Homomorphic encryption**

Nowadays data encryption is a common practice not only for enterprises but also individuals. It is meant to protect the privacy of the data. Data encryption is mostly done at rest when the data is stored and in transit when the data is transferred. However, data encryption is not popularly used when running or executing operations or computations.

Homomorphic encryption is an encryption method which allows arithmetical computations to be performed directly on encrypted or ciphered text without requiring any decryption. Outputs of the computations are also in encrypted form and provide identical or almost identical results when decrypted. This means that Homomorphic encryption allows data processing without disclosing the actual data.

A picture containing text, diagram, screenshot, line

Description automatically generated

Figure 2.1.1. Overview of homomorphic encryption.

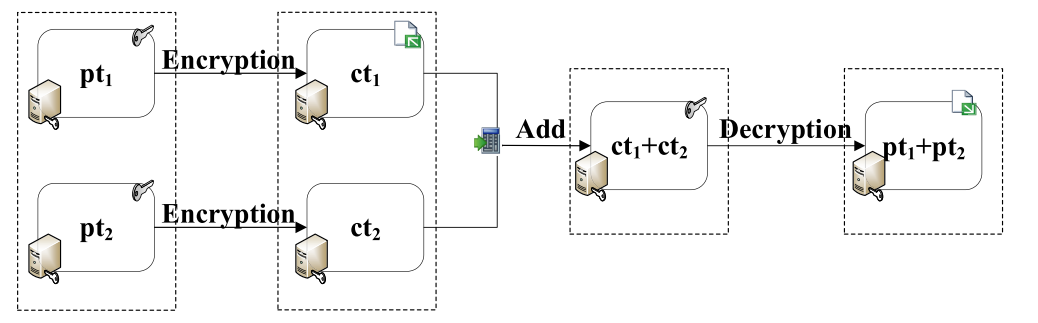


Figure 2.1.2. Addictive homomorphism.

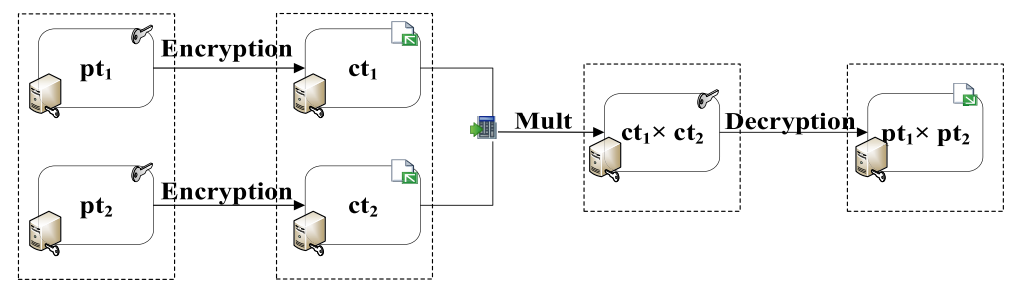


Figure 2.1.3. Multiplicative homomorphism.

Homomorphic encryption can be used for privacy-preserving outsourced storage and computation. This allows data to be encrypted and outsourced to commercial cloud environments for processing, all while encrypted. There are several types of homomorphic encryption:

* Partially homomorphic encryption is homomorphic encryption that supports only one homomorphic operation, either addition or multiplication, with unlimited number of times.
* Somewhat homomorphic encryption schemes allow both addition and multiplication but only for a limited number of times.
* Leveled fully homomorphic encryption supports the evaluation of arbitrary circuits composed of multiple types of gates of bounded (pre-determined) depth.
* Fully homomorphic encryption (FHE) supports both addition and multiplication operations with unlimited number of times.
  1. **Federated Learning**

Federated learning is a machine learning technique that enables multiple parties to build and train a common machine learning model without exchanging or sharing data. Each party (client) stores and processes their own dataset (local dataset) while there is a common model shared with all parties (clients). In this case each client trains the common model using local dataset and sends trained model to a centralized server. The server then aggregates the models received from all the clients and distributes the aggregated model back to the clients.

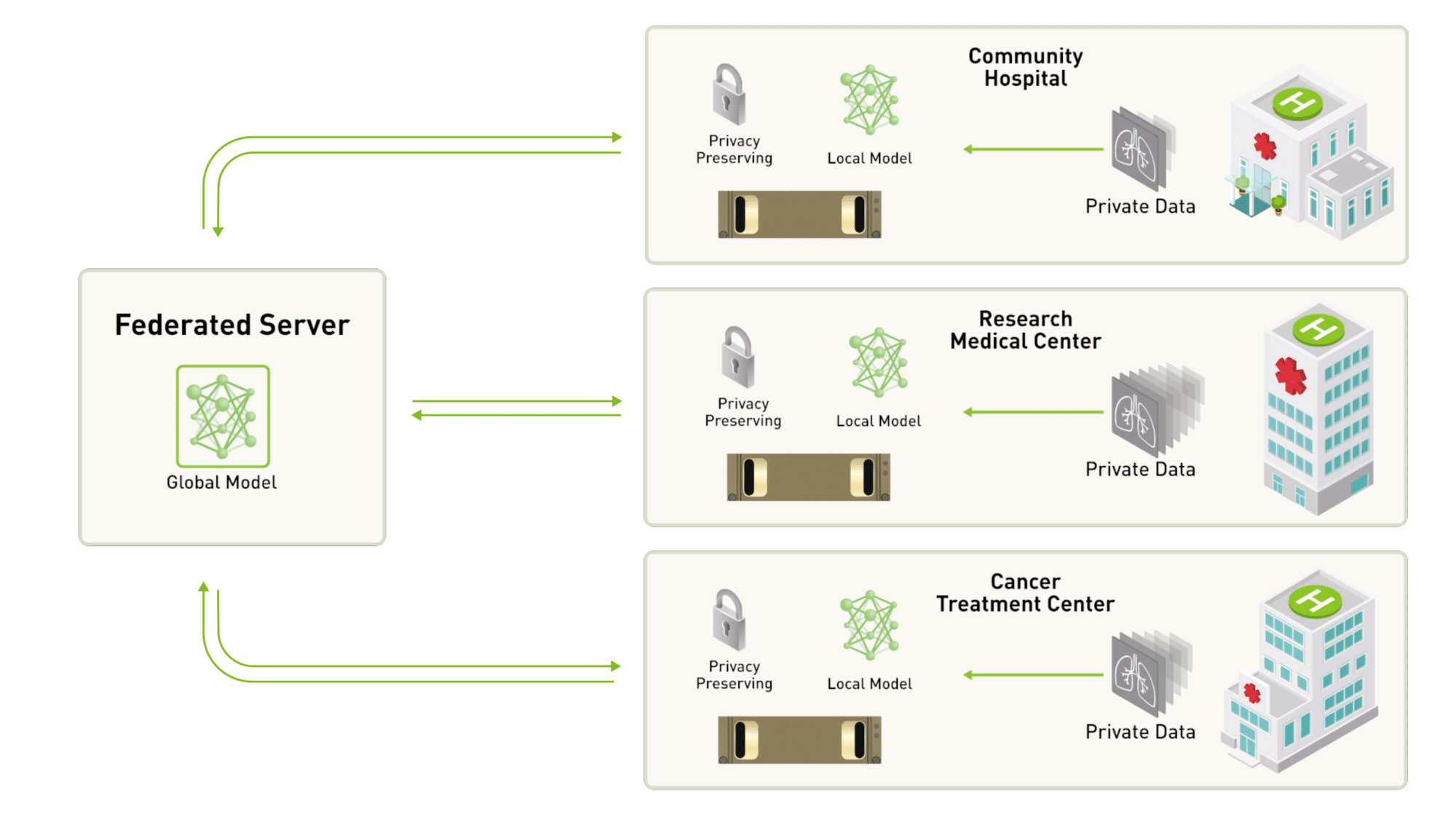


Figure 2.2. A centralized-server approach to federated learning.

Federated learning addresses data security and privacy issues since it doesn’t require access to the dataset of each client, nor requires the dataset to be distributed. The local dataset itself doesn’t have to be identically distributed and can be heterogeneous. This behaviour makes Federated Learning more popular in healthcare applications. Federated Learning enables health institutions to form and train a common model without transferring sensitive patient data out.

There are several types of Federated Learning setting:

* Centralized federated learning. In this setting, a central server is used to populate and aggregate models from participating clients during the learning process. A global common model is pushed from the server down to the clients.
* Decentralized federated learning. In this setting, participating clients coordinate among themselves to obtain a global common model.
* Heterogeneous federated learning. In this setting, participating clients come from different technical platfrom, e.g., PC and mobile phones, with own local dataset and model while obtaining single global model.

In this work, centralized federated learning setting is implemented, to demonstrate model aggregation by single centralized server.

1. **Proposed solution system model**

This section gives a high-level system overview of the proposed homomorphic crypto-scheme-based privacy-preserving federated learning training method. The proposed privacy-preserving scheme is a two-phase approach: (1) local model training at each client and (2) encrypted model weight aggregation at the server.

**A diagram of a model weight aggregateor

Description automatically generated with low confidence**

Figure 3. Overall system overview of the proposed method

Here are the detailed steps:

* **Step 1: Initialization between parties**
* Each hospital needs to agree on a machine learning algorithm as well as a set of datasets’s features.
* Stakeholders (hospitals and cloud computing servers) agree on a shared homomorphic algorithm then generate public and private key pairs. The hospital will keep the private key and the cloud will keep the public key.
* Each pair of cloud computing server and hospital will agree on an AES key, note that this key is used separately between each hospital. This helps prevent information theft between hospitals.
* **Step 2: Client local training**
* Each client trains the local machine learning with their private dataset. The training process will generate a set of parameters.
* The trained model’s weight parameters then will be encrypted with homomorphic private key – this key is the same between those hospitals.
* After doing homomorphic encryption, we will encrypt one more layer using AES algorithm, this key will be used separately for each hospital.
* **Step 3: Model Aggregation in cloud computing server**
* The server collects all encrypted weight parameters from the clients. The server will first decrypt AES with the key corresponding to each hospital.
* After doing AES decryption, it calculates the average weight value (using Federated Learning Aggregation algorithm) of each encrypted weights by using homomorphic encryption public key.
* Finally, the global model’s weight will be encrypted with AES key and send back to coressponding hospital.
* **Step 4: Client update new model**
* After receiving encrypted model from server, each hospital will use their AES key to decrypt it, and use the shared HE private key to decrypt it one more time to get the final global model parameters
* After updating new model, each hospital perform local training again, and so on…

**There are some clear advantages with this approach:**

* The aggregator doesn’t get access to any of the model updates => preventing potential inference attacks.
* Only hospitals get access to global models, preventing any model steal attack from the aggregator.
* The hospitals operate over the plain-text model, so the type of model we can train is not restricted.
* More efficient and accurate.

1. **Implementation**

We have implemented our proposed protocols and the classifier training phase in Python by using the Pytorch libraries for the model building and the openFHE library for the homomorphic encryption implementation. To show the training phase time performance of the proposed protocols, we tested breast cancer public dataset. Here are the details of the sources we use and the planning.

* 1. **Library and framework**
* **Pytorch:** is a machine learning framework based on the Torch library, used for applications such as computer vision and natural language processing, originally developed by Meta AI and now part of the Linux Foundation umbrella.
* **openFHE:** is an open-source cross platform software library that provides implementations of fully homomorphic encryption schemes. OpenFHE is a successor of PALISADE and incorporates selected design features of HElib, HEAAN, and FHEW libraries.
* **Numpy:** is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays. This use for data preprocessing.
  1. **Machine learning algorithm**

**Dataset and data preprocessing:**

* Our chosen dataset has 32 features, here is the fifth sample:

**A screenshot of a computer

Description automatically generated with low confidence**

We define some functions in order to randomly split this dataset to:

* Training dataset (80%)
* Testing dataset (20%)

A screenshot of a computer program

Description automatically generated with low confidence

Here is the result after preprocessing data:

A screenshot of a computer program

Description automatically generated with low confidence

We will use Logistic Regression model for breast cancer diagnosis. Firstly, we define our machine learning model, in this case we use Pytorch framework:

* Define some dependencies module:

A picture containing text, font, screenshot, white

Description automatically generated

* Define Logistic Regression class:

A screenshot of a computer code

Description automatically generated with medium confidence

In our “forward” pass of the PyTorch neural network (really just a perceptron), Logistic regression can also be visualized as a network of features feeding into a single logistic function, the visual representation and corresponding equations are shown below:

A diagram of a network

Description automatically generated with medium confidence

Figure 4.2.1. Overview of logistic regression

**Sigmoid function**

The sigmoid function is extremely useful for two main reasons:

* It transforms our linear regression output to a probability from 0 to 1. We can then take any probability greater than 0.5 as being 1 and below as being 0.
* Unlike a stepwise function (which would transform the data into the binary case as well), the sigmoid is differentiable, which is necessary for optimizing the parameters using gradient descent (we will show later).

A picture containing text, screenshot, plot, line

Description automatically generated

Figure 4.2.2. Sigmoid function

### Training process

Firstly, we should assign some hyper-parameters.

A picture containing text, font, screenshot, white

Description automatically generated

Parameter Definitions:

* **Epoch**: Indicates the number of passes through the entire training dataset the network has completed.
* **learning\_rate**: A tuning parameter in an optimization algorithm that determines the step size at each iteration while moving toward a minimum of a loss function.

Next, we define our main training model process:

A screenshot of a computer program

Description automatically generated with medium confidence

In this above code, we introduce two important functions: the Loss Function and the Optimizer:

**Binary Cross Entropy Loss Function**

A picture containing text, font, screenshot, algebra

Description automatically generated

**Stochastic Gradient Descent Optimizer**

A picture containing text, screenshot, font

Description automatically generated

Demo our training process with logistic regression:

A picture containing text, font, screenshot, line

Description automatically generated

Model parameters after training process:

A screenshot of a computer code

Description automatically generated with low confidence

Virtualize record of training process:

A screenshot of a computer program

Description automatically generated with low confidence

|  |  |
| --- | --- |
|  |  |

**Evaluating the Model**

**A picture containing text, font, screenshot, line

Description automatically generated**

* 1. **Homomorphic encryption with openFHE**

Fully Homomorphic Encryption (FHE) is a powerful cryptographic primitive that enables performing computations over encrypted data without having access to the secret key. OpenFHE is an open-source FHE library that includes efficient implementations of all common FHE schemes:

* Brakerski/Fan-Vercauteren (BFV) scheme for integer arithmetic
* Brakerski-Gentry-Vaikuntanathan (BGV) scheme for integer arithmetic
* Cheon-Kim-Kim-Song (CKKS) scheme for real-number arithmetic (includes approximate bootstrapping)
* Ducas-Micciancio (DM) and Chillotti-Gama-Georgieva-Izabachene (CGGI) schemes for evaluating Boolean circuits and arbitrary functions over larger plaintext spaces using lookup tables.

OpenFHE also includes the following multiparty extensions of FHE:

* Threshold FHE for BGV, BFV, and CKKS schemes
* Proxy Re-Encryption for BGV, BFV, and CKKS schemes

In our project we will demo three homomorphic encryption schemes: BGV, BFV, CKKS, the implement for multiparty clients and server.

Each scheme has three main processes:

* **key\_generation.cpp:** this program will set up parameters for crypto scheme, then generate public and private key pairs. Finally it serialize each key to file:
* **crypto\_context.txt:** this file is used in both server and client side. This store the crypto parameters of the scheme that has been agreed before between client and server.
* **public\_key.txt:** this file stores the public key which is used by the server.
* **private\_key.txt:** this file stores the private key which is used by all clients.
* **client.cpp:** this program will process some function like: load crypto context, load private key, perform encryption to ciphertext, serialize ciphertext to file.
* **server.cpp:** this program will process some function like: load crypto context, load public key, perform homomorphic operation, perform federated learning aggregation, serialize ciphertext to file.

Here are the details in each program, in this case we use bfv scheme as an example, the other scheme is similar:

* Include dependencies:

A screen shot of a computer code

Description automatically generated with low confidence

* + 1. **Key generation:**
* Define paths – where to store something like crypto context, key pair, ciphertext:

**A screen shot of a computer code

Description automatically generated with low confidence**

* Define CryptoContext and KeyPair global object:

**A black background with white text

Description automatically generated with low confidence**

* Setup BFV CryptoContext:

**A screenshot of a computer program

Description automatically generated with medium confidence**

* Serialize CryptoContext To File:

A screen shot of a computer code

Description automatically generated with low confidence

* Serialize public key and private key To File:

A screen shot of a computer program

Description automatically generated with low confidence

* Serialize multiplication key To File:

A picture containing text, screenshot, font, software

Description automatically generated

* + 1. **Client side**
* Define CryptoContext and KeyPair global object:

**A screenshot of a computer code

Description automatically generated with low confidence**

* Load CryptoContext from file:

**A screen shot of a computer

Description automatically generated with medium confidence**

* Load key pair from file:

**A screen shot of a computer screen

Description automatically generated with low confidence**

* Encrypt and serialize ciphertext to file:

**A picture containing text, screenshot, font

Description automatically generated**

* Load ciphertext from file then decrypt it:

**A screen shot of a computer program

Description automatically generated with low confidence**

* + 1. **Server side**
* Define CryptoContext, PublicKey and vector ciphertext from clients:

A screen shot of a computer code

Description automatically generated with low confidence

* Load multiplication key from file:

A screen shot of a computer code

Description automatically generated with low confidence

* Load clients’s encrypted weight:

A picture containing text, screenshot, font

Description automatically generated

* Server aggregator, this will perform federated learning algorithm by using homomorphic operator:

A picture containing text, screenshot, font

Description automatically generated

* 1. **Demo our proposed model system**

In this section we will demostrate our proposed model system. In this case, we will demo federated learning process on 4 hospital to produce a global model for prediction breast cancer.

* + 1. **Datasets**

We use some real-world public datasets, those datasets has the same 32 features:

A screenshot of a computer

Description automatically generated with low confidence

* <https://www.kaggle.com/datasets/yasserh/breast-cancer-dataset>
* <https://www.kaggle.com/code/a3amat02/breast-cancer-classification/input>
* <https://rstudio-pubs-static.s3.amazonaws.com/344010_1f4d6691092d4544bfbddb092e7223d2.html>
  + 1. **Client side**

Import openFHE library. Then run generate\_key(), this is just a python wrapper to run openfhe-lib/build/key\_gen. After running these files, it will create openfhe-lib/data folder that holds:

* crypto\_context.txt : contain CryptoContext object.
* public\_key.txt : contain public key.
* private\_key.txt : contain private key.
* mult\_key.txt : contain multiplication key.

Then, we start by creating the Client class that simulates the computers of each hospital. This just rewrite the Logistic Regression process that we have implemented previously.

**class** Client:

**def** \_\_init\_\_(self, name, data\_url, enc\_file, n\_features, iters):

self**.**id **=** name

self**.**enc\_file **=** enc\_file *# place wher clients save encrypted weights*

*# split data into train and test*

self**.**X\_train, self**.**Y\_train, self**.**X\_test, self**.**Y\_test **=** self**.**preprocessing(data\_url)

*# define local training model*

self**.**local\_model **=** LogisticRegression(n\_features)

*# some helpfull stuffs*

self**.**decide\_vectorized **=** np**.**vectorize(self**.**decide)

self**.**to\_percent **=** **lambda** x: '{:.2f}%'**.**format(x)

self**.**num\_epochs **=** iters

self**.**accuracies **=** []

self**.**losses **=** []

**def** preprocessing(self, data\_url):

df **=** pd**.**read\_csv(data\_url)

*# Replace "M" with 1 and "B" with 0 at "diagnostic" column*

df["diagnostic"] **=** (df["diagnostic"] **==** "M")**.**astype(int)

*# split dataframe to train and test df*

df\_train, df\_test **=** np**.**split(df**.**sample(frac**=**1), [int(0.8 **\*** len(df))])

*# scaling and convert to tensor context*

train, X\_train, Y\_train **=** scale\_dataset(df\_train, **True**)

test , X\_test , Y\_test **=** scale\_dataset(df\_test , **False**)

**return** X\_train, Y\_train, X\_test, Y\_test

**def** decide(self, y):

**return** 1. **if** y **>=** 0.5 **else** 0.

**def** compute\_accuracy(self, input, output):

prediction **=** self**.**local\_model(input)**.**data**.**numpy()[:, 0]

n\_samples **=** prediction**.**shape[0] **+** 0.

prediction **=** self**.**decide\_vectorized(prediction)

equal **=** prediction **==** output**.**data**.**numpy()

**return** 100. **\*** equal**.**sum() **/** n\_samples

**def** local\_training(self, debug**=True**):

n\_samples, \_ **=** self**.**X\_train**.**shape

*# define criterion function and set up optimizer*

criterion **=** torch**.**nn**.**BCELoss(reduction**=**'mean')

optimizer **=** torch**.**optim**.**SGD(self**.**local\_model**.**parameters(), lr**=**0.01)

*# main process*

**for** epoch **in** range(self**.**num\_epochs):

optimizer**.**zero\_grad()

*#### Compute outputs ####*

prediction **=** self**.**local\_model(self**.**X\_train)

*#### Compute gradients ####*

loss **=** criterion(prediction**.**squeeze(), self**.**Y\_train)

loss**.**backward()

*#### Update weights ####*

optimizer**.**step()

*# compute accuracy and loss*

train\_acc **=** self**.**compute\_accuracy(self**.**X\_train, self**.**Y\_train)

train\_loss **=** loss**.**item()

self**.**losses**.**append(train\_loss)

self**.**accuracies**.**append(train\_acc)

*#### Logging ####*

**if** debug **and** (epoch **+** 1)**%50** == 0:

print('[LOG] Epoch: %05d' **%** (epoch **+** 1), end**=**"")

print(' | Train ACC: %s' **%** self**.**to\_percent(train\_acc), end**=**"")

print(' | Loss: %.3f' **%** train\_loss)

**def** encrypted\_model\_params(self):

model\_weights **=** self**.**local\_model**.**linear**.**weight**.**data**.**squeeze()**.**tolist()

model\_bias **=** self**.**local\_model**.**linear**.**bias**.**data**.**squeeze()**.**tolist()

model\_params **=** model\_weights **+** [model\_bias]

encrypt\_weights(model\_params, self**.**enc\_file)

**def** decrypted\_model\_params(self):

params **=** decrypt\_weights("/enc\_aggregator\_weight\_server.txt")

*# convert float to tensor context*

W **=** Variable(torch**.**tensor([params[:**-**1]], dtype **=** torch**.**float32))

B **=** Variable(torch**.**tensor( params[**-**1], dtype **=** torch**.**float32))

self**.**local\_model**.**linear**.**weight **=** torch**.**nn**.**Parameter(W)

self**.**local\_model**.**linear**.**bias **=** torch**.**nn**.**Parameter(B)

**def** plot\_graphs(self, diagnosis\_title **=** 'Breast cancer'):

plt**.**plot(self**.**losses)

plt**.**title(f"{diagnosis\_title} - Training Loss")

plt**.**xlabel("Iterations")

plt**.**ylabel("Training Loss")

plt**.**show()

plt**.**plot(self**.**accuracies)

plt**.**title(f"{diagnosis\_title} - Training Accuracy")

plt**.**xlabel("Iterations")

plt**.**ylabel("Training Accuracy (Percent %)")

plt**.**show()

**def** print\_result\_after\_training(self):

print('Model parameters:')

print(' | Weights: %s' **%** self**.**local\_model**.**linear**.**weight)

print(' | Bias: %s' **%** self**.**local\_model**.**linear**.**bias)

self**.**plot\_graphs()

**def** evaluating\_model(self):

test\_acc **=** self**.**compute\_accuracy(self**.**X\_test, self**.**Y\_test)

print('[+] Testing Accuracy = {}'**.**format(self**.**to\_percent(test\_acc)))

* + 1. **Server side**

We define some functions to train the machine learning model in a federated way while keeping track of the training loss and the training accuracy, for each hospital separately.

A picture containing text, font, screenshot, line

Description automatically generated

The whole process is done in a server aggregator, in 1000 iterations (we can vary the number of iterations.) At each iteration.

A picture containing text, screenshot, font

Description automatically generated

A screenshot of a computer program

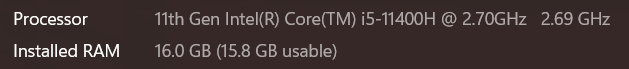
Description automatically generated with low confidence

A picture containing text, font, screenshot

Description automatically generated

1. **Summary**
   1. **The results**

To demo our system model, we use virtual machine to virtualize client side, here is the configuration of VM:



* Federated learning with BFV scheme:

|  |  |
| --- | --- |
|  |  |

* Federated learning with BGV scheme:

|  |  |
| --- | --- |
|  |  |

* Federated learning with CKKS scheme:

|  |  |
| --- | --- |
|  |  |

|  |  |  |  |
| --- | --- | --- | --- |
|  | BFV scheme | BGV scheme | CKKS scheme |
| Training Time | ~ 12 mins | ~ 14 mins | ~ 20 mins |
| Accuracy | ~ 87.76% | ~ 89.81% | ~ 85.71% |

**Compare different schemes.**

As you can see, the BFV scheme is the most optimal, and the BGV gives the most accurate results. Although the CCKS scheme supports approximation on real numbers, it is suitable for machine learning models, but it takes more time and is less accurate than integer schemes.

For more details, we left our source code of project here:

<https://github.com/idk-wh0am1/Federated-Learning-meets-Homomorphic-Encryption/tree/main>

* 1. **Conclusion**

Privacy preserving become an essential practice of healthcare institutions as it is mandated by both EU and the US. Federated learning and homomorphic encryption will play a critical role in maintaining data security and model training. Benefitting from both techniques, the proposed model achieves compatitive performance while there is a significant trade-off for the execution time and number of clients. The classification metrics, i.e., accuracy, F1. precision and recall, reaches over %80 using both encrypted and plain data for each federated learning case.

Privacy attacks will cause immense damage to the security and privacy of the patient information. This will hinder the advancement in healthcare using data-driven models. Therefore, it is indispensable to take imperative steps to strengthen not only the safety of the information but also the way data is processed. This study demonstrated that federated learning with homomorphic encryption could be successfully applied to enhance data-driven models by eliminating and minimizing the share of the sensitive data. It is envisioned that this study could be useful for the scientists and researchers working on sensitive healthcare data in multi-party computation settings.

1. **References**

[1] Kim, A., Song, Y., Kim, M., Lee, K., & Cheon, J. H. (2018). Logistic regression model training based on the approximate homomorphic encryption. *BMC medical genomics*, *11*(4), 23-31.

[2] Wibawa, F., Catak, F. O., Kuzlu, M., Sarp, S., & Cali, U. (2022, June). Homomorphic encryption and federated learning based privacy-preserving cnn training: Covid-19 detection use-case. In *Proceedings of the 2022 European Interdisciplinary Cybersecurity Conference* (pp. 85-90).