# Privacy Preserving Neural Networks Training in HFL

NAIBO WANG, SHIQI ZHANG, National University of Singapore (NUS)

Federated learning is a distributed learning technique which enables the collaborative training of machine learning models among different organizations under the privacy restrictions. In this project, we focus on *Horizontal Federated Learning (HFL)* where each organization holds data with same features and a protocol of privacy-preserved federated averaging is designed. Specifically, we exploit semi-honest additive secret sharing protocol to preserve privacy and achieve distributed federated averaging with the help of a coordinator host. Furthermore, extensive experiments are conducted over 10 real world datasets and multiple training tasks. We use CNN, MLP and LSTM to show that our technique can work on different models.

CCS Concepts: • **Computing methodologies** → **Machine learning**; **Distributed computing methodologies**; • **Security and privacy** → **Privacy-preserving protocols**.

Additional Key Words and Phrases: Federated Learning, Distributed Learning, Privacy

## 1 INTRODUCTION

Many machine learning algorithms are data-driven, i.e., a more accurate model will be returned with more feeded data. However, in the real world, many datasets are sensitive (e.g. patient information) and are secretly kept in different organizations (e.g. hospitals).

**Horizontal Federated Learning.** Motivated by this, federated learning [1] has become a hot research topic in machine learning, which is a collaborative training scheme. Specifically, each organization contributes the local model trained by its own data to others and all local models are aggregated together as an enhanced global model. For *Horizontal Federated Learning (HFL)*, the dataset in each organization holds same features.

**Secure Multi-Party Computation.** However, the information (e.g. data and model) cannot simply be shared due to various policies and regulations, such as General Data Protection Regulation (GDPR) [2] and Personal Information Security Specification (PISS) [3]. Motivated by this, the federated learning systems are supposed to obey policies and regulations to protect privacy. In this project, *Secure Multi-Party Computation (MPC)* is applied to guarantee secret model sharing, which create methods for parties to jointly compute a function over their inputs while keeping those inputs private.

**Contributions.** In this project, we propose a horizontal federated learning framework with additive secret sharing. The detailed contributions are listed as follows.

- A federated average algorithm with secret sharing is proposed, which utilizes a coordinator host to select clients to share their local models. The coordinator will never know any client's local data and model, it can only get the averaging model each round.
- A semi-honest additive secret sharing protocol is applied for each secret party. Furthermore, even all secret parties collude with each other, they can not know the original model from every client because the model is transferred after weighted averaging.
- Comprehensive experiments are conducted over 10 real word datasets.
- Implemented other distributed federated learning algorithms such as FedBrain.

Author's address: Naibo Wang, Shiqi Zhang, naibo@comp.nus.edu.sg,s-zhang@comp.nus.edu.sg, National University of Singapore (NUS), Singapore.

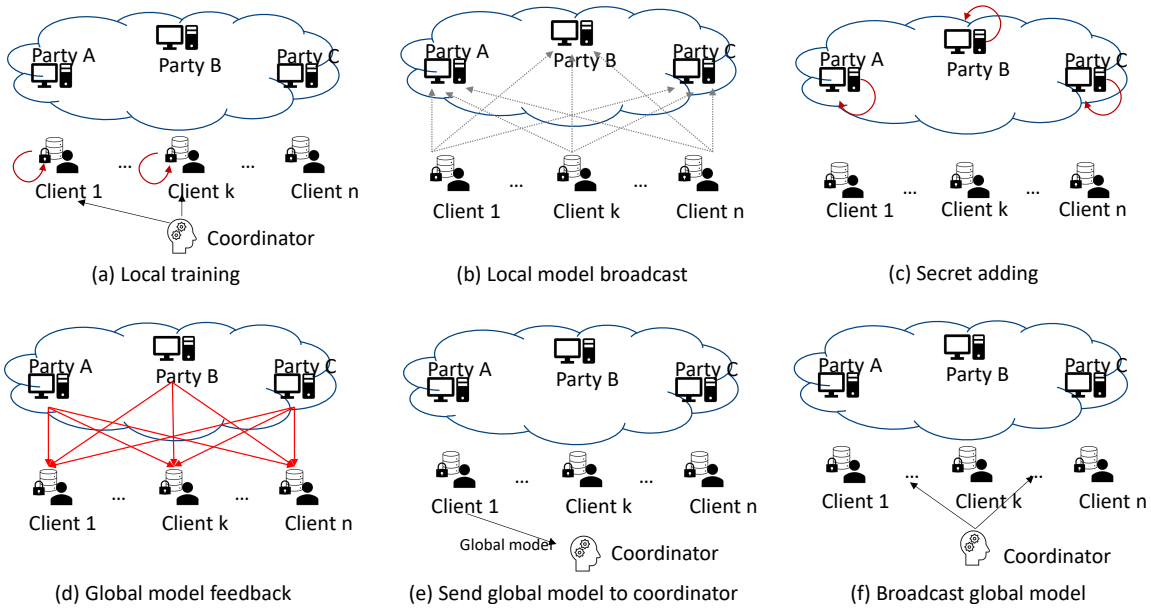


Fig. 1. Project Framework

## 2 METHODOLOGY

### 2.1 Overview

The framework of our privacy-preserved federated learning is shown in Figure 1. Specifically, a coordinator host firstly select $k$ clients out of all client for federated learning (Figure 1(a)). For the selected clients, they will use their own data to train a local model, which will be secretly broadcast to each of MPC party (Figure 1(b)). For the MPC party, each party will conduct secret adding locally and broadcast the result back (Figure 1(c)). For each selected client, it can reveal the global model by combining together each secret piece of model from the MPC party (Figure 1(d)). After that, without loss of generalization, the first selected client will send the global model back to the coordinator, by whom the global model will be further broadcast to other non-selected clients (Figure 1 (e)(f)).

### 2.2 Federated Averaging

Federated Averaging [4] (FedAvg) is a very popular as well as the baseline algorithm in federated learning. Every round the server select $k$ clients from all $n$ clients, and perform local stochastic gradient descent (SGD) training on each client. Then, all selected clients will send their local models to the server, and the server will average the weights of each client model to produce the global model. In this project, we combined FedAvg with Secret Sharing protocol to provide privacy preserving for all the local models from local clients (workers). The details of our algorithm is shown in Alg. 1.

As we can see from Alg. 1, the coordinator will never know any client's local data and model, it can only get the averaging model each round. And even all secret parties collude with each other, they can not know the original model from every client because the model is transferred after weighted averaging.

**Algorithm 1** Federated Averaging With Secret Sharing

**Require:** number of clients $K$; local minibatch size $B$; number of local epochs $E$; learning rate $\eta$, number of selected clients every epoch $m$, number of training samples $n_k$ for each client $k$.

**Coordinator executes:**
  **for** each round $t$ = 1, 2, ... **do**
    $S_t \leftarrow$ (random set of $m$ clients)
    $n \leftarrow \sum_{k \in S_t} n_k$
    **for** each client $k \in S_t$ **in parallel do**
      $w_{t+1} \leftarrow$ ClientUpdate($k$, $t$, $n$)
    **end for**
    broadcast $w_{t+1}$ to every client $k \notin S_t$
  **end for**

**ClientUpdate**($k, t, n$): // *Run on client* $k$
  **if** t = 1 **then**
    initialize $w_k$
  **end if**
  $Batch \leftarrow$ (split $\mathcal{D}_k$ into batches of size $B$)
  **for** each local epoch $i$ from 1 to $E$ **do**
    **for** batch $b_k \in Batch$ **do**
      $w_k \leftarrow w_k$ - $\eta \nabla(w_k; b_k)$
    **end for**
  **end for**
  broadcast $\frac{n_k}{n} w_k$ to all secret party $P_1$ to $P_3$.
  // parties will conduct $w \leftarrow \sum_{k \in S_t} \frac{n_k}{n} w_k$ by secret sharing.
  wait all parties to finish secret adding, and get global model $w$
  $w_k \leftarrow w$
  return $w$ to Coordinator

### 2.3 Secret Sharing

The secret sharing procedure consists of *secret model broadcast (in Figure 1(b))*, *secret model aggregation (in Figure 1(c))* and *secret model feedback (in Figure 1(d))*. The whole procedure is based on semi-honest additive secret sharing protocol, which is widely accepted in SPDZ [5] and MASCOT [6]. Specifically, for additive secret sharing, suppose that a number $x \in [0, M-1]$ is shared among parties $P_1, ..., P_n$ by sending a random number $x_i \in [0, M-1]$ to $P_i$ such that $\sum_i x_i = x \mod M$. It is easily seen that the all parties together can reconstruct x while the view of any strict subset is indistinguishable to a random set of numbers. For implementation, MP-SPDZ [7] library is exploited. It provides 34 MPC protocol variants and all of which can be used with the same high-level programming interface based on Python. In the following sections, each step of secret sharing procedure will be introduced.

*2.3.1 Secret Model Broadcast.* Before broadcasting, there exists a preprocessing step, where each party $P_i$ generates and holds a random number $r_i$. For each party $P_i$, it sends its $r_i$ to every client and the detailed sending procedure will be introduced in Section 2.3.3. Each client can know $r = \sum_i r_i$ after collecting each $r_i$ from $P_i$. Suppose that a client holds data $x$, then it will broadcast $x - r$ to each party.

*2.3.2 Secret Model Aggregation.* Suppose that there are two clients who holds $x$ and $y$ respectively, then each party $P_i$ will receive $x - r$ and $y - r$ after the previous section. $P_i$ generates the secret values by

$$x_i = x - r + r_i; y_i = y - r + r_i.$$

After secret value generation, each party $P_i$ will do secret adding for each secret value $x_i + y_i$.

*2.3.3 Secret Model Feedback.* For each party $P_i$, it sends back the addition of secret number (e.g. $x_i + y_i$). For each client, it can reveal the sum of secret numbers by

$$\sum_i (x_i + y_i) \mod r.$$

Furthermore, to detect the dishonest party which will send false value, Beaver triple [8] is applied. Specifically, for each party $P_i$, besides secret value (in previous section), it also generates an authentication tag $w_i = x_i \cdot r_i$ and send triple $(x_i, w_i, r_i)$ to each client. For each client, beside that reveal the summation, it also check if $w = x \cdot r$ where $w = \sum_i w_i$ and $r = \sum_i r_i$.

## 3 PROGRAM IMPLEMENTATION AND DEMO

### 3.1 Program Implementation

In this project, we don't use existing Federated Learning frameworks such as PySyft or FATE. We implemented our platform and the network communication process all by ourselves. In this way, we can manipulate all the underlying structures of our programs and know deeply about how federated learning and secret sharing works.

We use RPC and Socket as the network communication protocol to conduct the distributed experiment.

You can run our program with the source code and README file we provided.

### 3.2 Demo Setting

We conducted an experiment to run our Federated Averaging algorithm on two real machines:

- *Machine A*: Naibo Wang's Laptop, whose IP address is: 172.26.186.87.
- *Machine B*: Shiqi Zhang's Server, whose IP address is: 172.28.176.153.

Table. 1 shows the details of the environment for the demo.

Table 1. Details of the demo environment.

| **Role** | **IP** | **Port** |
|---|---|---|
| Worker 0 | 172.26.186.87 (A) | 50001 |
| Worker 1 | 172.28.176.153 (B) | 50002 |
| Worker 2 | 172.28.176.153 (B) | 50003 |
| Worker 3 | 172.28.176.153 (B) | 50004 |
| Coordinator | 172.26.186.87 (A) | 50000 |
| Secret Party A | 172.28.176.153 (B) | 14001 |
| Secret Party B | 172.28.176.153 (B) | 14002 |
| Secret Party C | 172.28.176.153 (B) | 14003 |

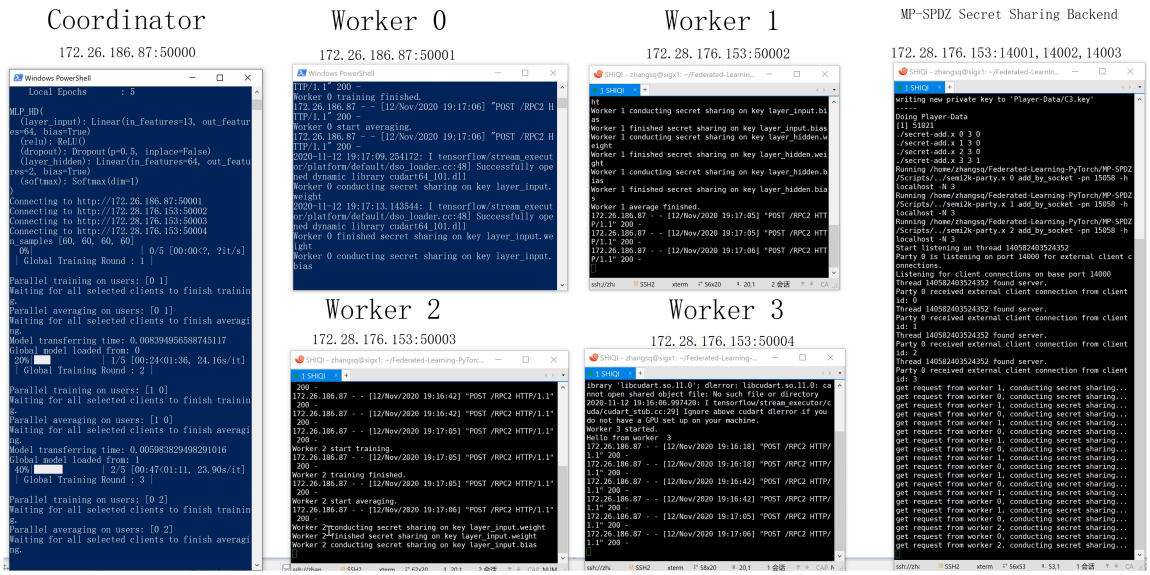


Fig. 2. Demo Screenshot.

### 3.3 Demo Display

As we can see from Fig. 2, after we have started the MP-SPDZ secret sharing backend, we started worker 0 to 3 one by one, and in the end we started the coordinator and we can see all the roles started to work and serve.

In Fig. 2, we can see that at every round, the coordinator will select 2 workers (clients) from 4 workers, and the selected two workers will train their models in parallel. After training, the selected workers will send their parameters to the secret sharing backend, and the backend will conduct secret sharing to get the sum results of the parameters. The whole process will continue until all reach the max epochs.

To know more about the program, please run our codes with different arguments.

## 4 EXPERIMENTS

### 4.1 Settings

**Datasets and tasks.** 10 real world datasets [9–17] are selected. Specifically, ChestXray, COVID19, MalariaCell and HAM10000 are the images from the healthcare respect. Mnist, Femnist and Cifar10 are the images of handwritten numbers, clothes and objects respectively. HeartDisease is the multidimensional human information. COVID19twitter contains the twitters about COVID19 and HeartBeat records the sounds of heart beats. All datasets are splited into both

IID and non-IID with 70% as the training set and 30% as the test set. We apply CNN, MLP and LSTM on different datasets to show that our program can work on different models. The details are listed in Table. 2.

**Metrics.** For the effectiveness, we measure the accuracy of test set. For efficiency, we measure the running time including the communication overhead for the program.

**Parameter settings.** Adam optimizer is applied. About the number of workers, 100 users are simulated and deployed in real environment with 5 users. The epoch is set from 100 to 300 and 10 users are selected per epoch. The number of local epoch, local batch size and learning rate are set to 5, 50 and 0.001 respetively.

**Platform.** All experiments are conducted over 72 core Intel(R) Xeon(R) Gold 6240 CPUs @ 2.60GHz and 2 NVIDIA RTX 2080Ti GPUs where each with 11GB CUDA Memory. The overall memory is 310GB. For the softwares, Python 3.8.1 and Pytorch 1.7.0 are used.

Table 2. Dataset details.

| Dataset[9–17] | Datasize | Model | Description |
|---|---|---|---|
| Mnist | 70,000 (52.4M) | CNN | handwritten number |
| Femnist | 70,000 (52.4M) | CNN | clothes |
| CIFAR10 | 60,000 (177M) | CNN | objects |
| Chest Xray | 5,840 (1.14G) | CNN | xray images of chests |
| COVID19 | 2,905 (1.17G) | CNN | COVID-19 radiography images |
| MalariaCell | 27,558 (317M) | CNN | cell to detect malaria |
| HAM10000 | 10,015 (2.57G) | CNN (DenseNet) | skin images |
| Heart Disease | 303 (12K) | MLP | human info |
| COVID19_twitter | 2,756 (466K) | LSTM | tweets about COVID19 |
| Heart Beat | 832 (152M) | LSTM | heart-beat sound |

### 4.2 Effectiveness

We conducted experiments on multiple indicators to prove the effectiveness of our algorithm such as the training curves, max accuracy, communication cost, etc.

*4.2.1 Training Curve.* We got the training curves among all 10 datasets with these four indicators:

- $IID\_train$: training accuracy for IID setting.
- $IID\_test$: test accuracy for IID setting.
- $Non-IID\_train$: training accuracy for Non-IID setting.
- $Non-IID\_test$: test accuracy for Non-IID setting.

Fig. 3 shows the training curves [1] for all 10 datasets, we can see that overall, IID training accuracy > IID test accuracy > Non-IID training accuracy > Non-IID test accuracy. IID training curves are smooth, while Non-IID curves fluctuate a lot, which means IID settings are more stable to models. Meanwhile, different types of data show different training trends, so the data preprocess procedure is very important for model training. For sequence data (e.g., audio data), curve fluctuates more than other types of data, not as stable as images.All models training on 10 datasets will converge finally, but accuracy still can be improved for some models. We can also conclude that a good model structure as well as its hyper

[1]For heart_disease and heartbeat dataset, since the data amount is relatively small, so there is no Non-IID setting.

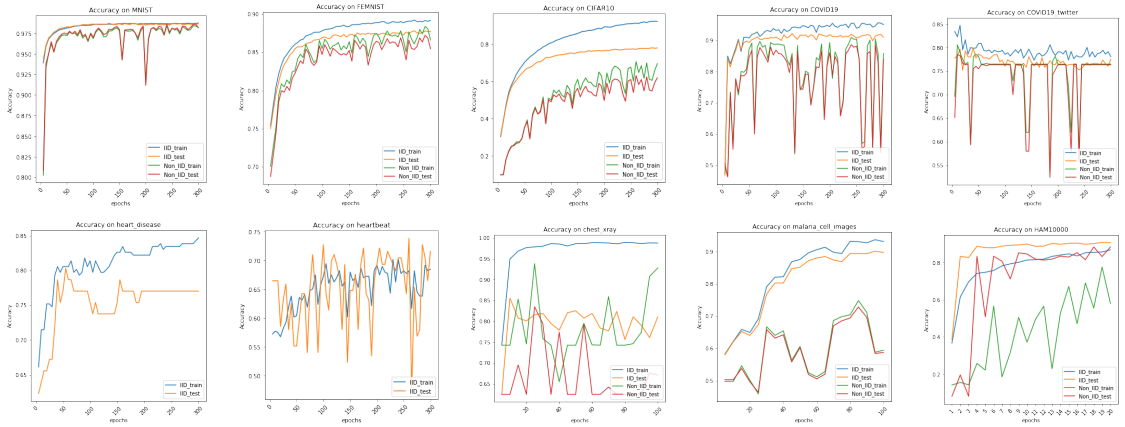


Fig. 3. Training Curves among all datasets.

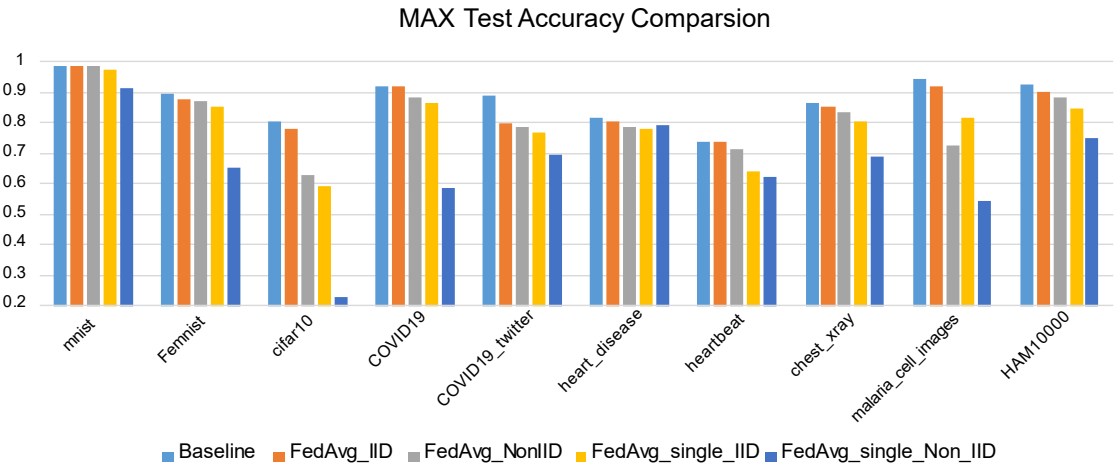


Fig. 4. Max Test Accuracy Comparison.

parameters are very important for training models. For HAM10000 datasets, since test set is relatively small than training set, so at the beginning training accuracy are lower than test accuracy.

*4.2.2 Max Accuracy.* Max Accuracy is the maximum test accuracy we get after we train the model for multiple rounds. We got max test accuracy in the following models:

- *Baseline*: model trained on a single machine with all dataset.
- *FedAvg_IID*: global model trained on federated setting with IID dataset partition.
- *FedAvg_Non_IID*: global model on federated setting with Non-IID dataset partition.
- *FedAvg_single_IID*: single model trained on federated setting with IID dataset partition.
- *FedAvg_single_Non_IID*: single model trained on federated setting with IID dataset partition.

Note the global model means the model after averaging per round. Single model means every client's local model before averaging per round, and we test all selected clients' single models on the test set to get the test accuracy.

Fig. 4 shows the comparison of max accuracy for all datasets. We can see from the graph that basically, the max accuracy of Baseline > FedAvg_IID > FedAvg_Non_IID > FedAvg_single_IID > FedAvg_single_Non_IID.

In some cases, the max accuracy of FedAvg_single_IID setting better than FedAvg_Non_IID, which proves the data distribution is a very strong influence factor in Federated Learning Scenarios. Also, we can see that the max accuracy of global model is obviously better than single model, which proves the needs for federated learning.

*4.2.3 Number of Users.* Now we look how the number of users will affect the model performance. We now have two settings:

- Setting A: 100 users, 10 users selected every epoch.
- Setting B: 5 users, 3 users selected every epoch.

We can see from Table. 3 that Setting B is better than setting A on accuracy. This is because each user from setting B has more data than setting A, thus provide better performance.

Table 3. Model accuracy for two different settings of number of users.

| **Dataset** | **Setting A Accuracy** | **Setting B Accuracy** |
|---|---|---|
| Femnist | 0.878 | 0.895 |
| CIFAR10 | 0.778 | 0.813 |
| COVID19 | 0.921 | 0.922 |
| MalariaCell | 0.920 | 0.936 |
| COVID19_twitter | 0.799 | 0.824 |

*4.2.4 Insights.* From the above experiments we can know that the IID setting performs better than Non-IID setting, and more stable for model training. Meanwhile, the data preprocess and model selection are very important for model training. Different user numbers and fraction selected can also affect model performance. Therefore, when conducting federated learning experiments, we need to very careful to adjust all these parameters to get the best model performance.

### 4.3 Efficiency

This section will analyze the efficiency of our FedAvg algorithm with secret sharing.

*4.3.1 Secret Sharing.* Then we look at the computation cost for secret sharing, we can see from Table. 4 that the secret sharing version always takes 5 to 10 times as long as the one not using secret sharing, it is relative a liner relationship.

Table 4. Time Cost for Secret sharing

| **Dataset** | Sharing size | **Secret sharing time** | **Without secret sharing time** |
|---|---|---|---|
| HeartDisease | 0.35MB | 215.47s | 20.88s |
| Femnist | 7.50MB | 1107.81s | 164.24s |
| Mnist | 7.50MB | 1002.75s | 192.66s |
| HeartBeat | 17.38MB | 1647.14s | 36.92s |

*4.3.2 Communication Cost.* And the last indicator we compared is the communication cost. As shown in Table. 5, compared to simulation environment in a single machine, the network communication is not obvious huge. It's because in Local Area Network (LAN), the average transferring speed is around 12MB/s (compared with local disk writing speed 300MB/s on a single machine), so it will takes less than 3 seconds to finish the transferring. Therefore, Compared with training time for each clients, the transferring time can be ignored for both small and big models.

Table 5. Communication cost in real distributed version.

| Dataset | Model Size | Network Transferring Time | Training time per epoch |
|---|---|---|---|
| Mnist | 88KB | 0.002s | 1.38s |
| COVID19 | 1MB | 0.017s | 6.25s |
| CIFAR10 | 13.6MB | 1.37s | 37.59s |
| HAM10000 | 27.8MB | 2.59s | 21min |

*4.3.3 Insights.* From the efficiency analysis we can know that the process of secret sharing causes large time overhead. E.g., for mnist dataset, we need overall 1000 seconds to compute a model with size of 7.5MB. Also, we should note that since secret sharing is lossless computing, so the model accuracy remains the same as non-secret-share setting. Meanwhile, communication time can be ignored when the model is relatively small in LAN.

### 4.4 Other distributed federated learning algorithm

Besides Alg. 1, we designed to conduct FedAvg with Secret Sharing, we also implemented the following three algorithms, which are decentralized version of federated learning:

- $BrainTorrent$ ($FedBrain$): an decentralized federated learning algorithm which presents a highly dynamic peer-to-peer environment, where all centers directly interact with each other without depending on a central body. All the local clients will maintain a version matrix to load and averaging other client's model. For the details of the algorithm, please refer to the original paper [18].
- $Federated\ Brain\ version\ 2$ ($FedBrain\_v2$): a variation of the $FedBrain$ algorithm. Instead of averaging the models after the training of all selected local clients, here we average the models first and train the selected client's model only to reduce the training time.
- $Federated\ Distributed\ Random$: This algorithm just random select $k$ clients to train based on their local models in order for every round, the $i$-th model is initialized from the $i-1$-th model.

Table 6. Model performance on CIFAR10 for three decentralized FL algorithms on 100 users.

| Algorithm | Training Rounds | Training Accuracy | Test Accuracy |
|---|---|---|---|
| FedBrain | 326 | 0.742 | 0.716 |
| FedBrain_v2 | 184 | 0.947 | 0.764 |
| Federated Distributed Random | 180 | 0.862 | 0.779 |

We didn't conduct too much analysis on these algorithms, only tested them on CIFAR10 dataset with the same CNN model as Alg. 1. We can see from Table. 6 that our variation of $FedBrain$ performs better than the original algorithm. Meanwhile, the $FederatedDistributedRandom$ got the best test accuracy, but note that this algorithm can not train the

local clients in order, so it will be very time-consuming to finish training since $FedBrain$ and $FedBrain_v2$ can run in parallel (theoretically, we performed them in simulation).

And also, these three algorithms can be combined with secret sharing to realize privacy preserving.

## 5 CONCLUSION AND FUTURE WORK

In this project, we propose a horizontal federated learning framework with additive secret sharing. The proposed federated average algorithm utilizes a coordinator host to select clients to share their local models. The coordinator will never know any client's local data and model, it can only get the averaging model each round. Furthermore, the data and model are secured by additive secret sharing. We also conduct comprehensive experiments over 10 real datasets and analyzed the model performance as well as the efficiencies of our algorithm. In the end, we briefly introduced other distributed federated learning algorithms such as FedBrain.

In the future, we can deploy the program on Internet at various platforms, and test communication costs and detect other latent problems (such as drop off).

## ACKNOWLEDGMENTS

The secret sharing backend is implementated by the authors of MP-SPDZ [7] library. For the federated learning algorithms, we reused a little code from AshWinRJ's repository [2] but most of the codes are overwritten and complemented.

Also, all our datasets come from the public source on the internet such as Kaggle, which are already referenced.

## REFERENCES

[1] Lei Shi, Liang Zhao, Wen-Zhan Song, Goutham Kamath, Yuan Wu, and Xuefeng Liu. Distributed least-squares iterative methods in networks: A survey. *arXiv preprint arXiv:1706.07098*, 2017.

[2] 2018 reform of eu data protection rules.

[3] Sarah Wang Han and Abu Bakar Munir. Information security technology-personal information security specification: China's version of the gdpr. *Eur. Data Prot. L. Rev.*, 4:535, 2018.

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

[5] Ivan Damgård, Valerio Pastro, Nigel Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In *Annual Cryptology Conference*, pages 643–662. Springer, 2012.

[6] Marcel Keller, Emmanuela Orsini, and Peter Scholl. Mascot: faster malicious arithmetic secure computation with oblivious transfer. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 830–842, 2016.

[7] Marcel Keller. MP-SPDZ: A versatile framework for multi-party computation. Cryptology ePrint Archive, Report 2020/521, 2020. https://eprint.iacr.org/2020/521.

[8] Donald Beaver. Correlated pseudorandomness and the complexity of private computations. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 479–488, 1996.

[9] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[10] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.

[11] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[12] Daniel S Kermany, Michael Goldbaum, Wenjia Cai, Carolina CS Valentim, Huiying Liang, Sally L Baxter, Alex McKeown, Ge Yang, Xiaokang Wu, Fangbing Yan, et al. Identifying medical diagnoses and treatable diseases by image-based deep learning. *Cell*, 172(5):1122–1131, 2018.

[13] Roman Kalkreuth and Paul Kaufmann. Covid-19: a survey on public medical imaging data resources. *arXiv preprint arXiv:2004.04569*, 2020.

[14] Sivaramakrishnan Rajaraman, Sameer K Antani, Mahdieh Poostchi, Kamolrat Silamut, Md A Hossain, Richard J Maude, Stefan Jaeger, and George R Thoma. Pre-trained convolutional neural networks as feature extractors toward improved malaria parasite detection in thin blood smear images. *PeerJ*, 6:e4568, 2018.

[2] https://github.com/AshwinRJ/Federated-Learning-PyTorch

[15] Philipp Tschandl, Cliff Rosendahl, and Harald Kittler. The ham10000 dataset, a large collection of multi-source dermatoscopic images of common pigmented skin lesions. *Scientific data*, 5:180161, 2018.

[16] Yiqi Deng and Peter J Bentley. A robust heart sound segmentation and classification algorithm using wavelet decomposition and spectrogram. In *Workshop Classifying Heart Sounds, La Palma, Canary Islands*, pages 1–6, 2012.

[17] Robert Detrano, Andras Janosi, Walter Steinbrunn, Matthias Pfisterer, Johann-Jakob Schmid, Sarbjit Sandhu, Kern H Guppy, Stella Lee, and Victor Froelicher. International application of a new probability algorithm for the diagnosis of coronary artery disease. *The American journal of cardiology*, 64(5):304–310, 1989.

[18] Abhijit Guha Roy, Shayan Siddiqui, Sebastian Pölsterl, Nassir Navab, and Christian Wachinger. Braintorrent: A peer-to-peer environment for decentralized federated learning. *arXiv preprint arXiv:1905.06731*, 2019.