Efficient and Secure: Privacy-Preserving Federated Learning for Resource-Constrained Devices

Muhammad Ayat Hidayat
Information Science and Technology
ISEE, Kyushu University
Fukuoka, Japan
muhammad.971@s.kyushu-u.ac.jp

Yugo Nakamura
Information Science and Technology
ISEE, Kyushu University
Fukuoka, Japan
y-nakamura@ait.kyushu-u.ac.jp

Yutaka Arakawa
Information Science and Technology
ISEE, Kyushu University
Fukuoka, Japan
arakawa@ait.kyushu-u.ac.jp

***Abstract*—Federated learning has gained popularity as a distributed machine learning approach that provides security and privacy for data trained on local devices. However, vulnerabilities still exist in this approach, and common solutions such as encryption and blockchain techniques often suffer from high computation and communication costs, making them impractical for resource-constrained devices. To solve this problem, we propose a privacy-preserving federated learning system that leverages compressive sensing and differential privacy, specifically designed for devices with limited computational resources. In this paper, we demonstrate the capabilities of our proposed system in resource-limited environments. We outline the features, infrastructure, and algorithm of our proposed system, and simulate its performance using image datasets on a Raspberry Pi 4 and an Android smartphone in a cloud environment. Our approach offers a practical solution for secure and privacy-preserving federated learning in resource-constrained scenarios, with potential applications in various domains such as healthcare, IoT, and edge computing.**

***Index Terms*—privacy-preserving, federated learning, compressive sensing, differential privacy, resource-constraint devices**

I. Introduction

Federated learning (FL) involves a collection of devices working together to create and train a model, exclusively utilizing their individual data for training. Federated learning systems involve the execution of computations on clients, enabling communication between clients and servers. Typically, both communication and computation serve the common objectives of model training and exchanging model parameters. The clients are primarily responsible for performing computations and generating the local model, whereas the server assumes the role of computing and generating the global model, which is subsequently shared among the clients [1].

In addition, federated learning offers the ability to train models without exposing raw client data, making it well-suited for on-the-edge computing [2], which also prioritizes security and privacy. Furthermore, due to the collaborative nature of the federated learning system, the need for a high-performance central server can be minimized. This is because the load for training has been divided among the clients or local devices that participate in the learning process, resulting in lower computational costs for federated learning.

Because of the high privacy and lower computational cost, federated learning has been applied in many fields such as medical healthcare [3], autonomous vehicle [4], industry [5] and financial sector [6]. However, the security and privacy of federated learning systems have become major concerns, as various types of attacks such as "Poisoning Attacks", "Inference Attacks", and "Reconstruction Attacks" can compromise the confidentiality of the data used in the learning process [7]. To address these challenges, several techniques, such as homomorphic encryption [8] and blockchain [9], have been proposed to counteract attacks. However, these approaches often come with additional computational overhead, particularly for computationally constrained client devices like IoT devices [10], as every client needs to encrypt all the parameters that are uploaded.

In contrast, our proposed system employs differential privacy instead of encryption, aiming to minimize the computation resources required to run the system. However, using differential privacy introduces noise into the model, which can significantly impact model accuracy and communication cost. To address these challenges, we propose the use of compressive sensing to compress the local gradient before sending them to a central server to reduce the communication cost and applying adaptive clipping to adjust the variability of the noise added to the model, making it less biased and more accurate.

A notable feature of our proposed system is its minimal hardware resource requirements. In this demonstration, we will highlight how our system is capable of operating on affordable microcontrollers, like Raspberry Pi, and budget-friendly Android smartphones that have limited memory size and CPU power. By leveraging the benefits of differential privacy and addressing the limitations of communication cost, model accuracy, and bias, our approach presents a promising solution for federated learning in environments with limited resources.

II. System Overview

A. System Infrastructure

We consider this system a general FL system consisting of a central server and N-clients, as shown in Fig. 1. The server and

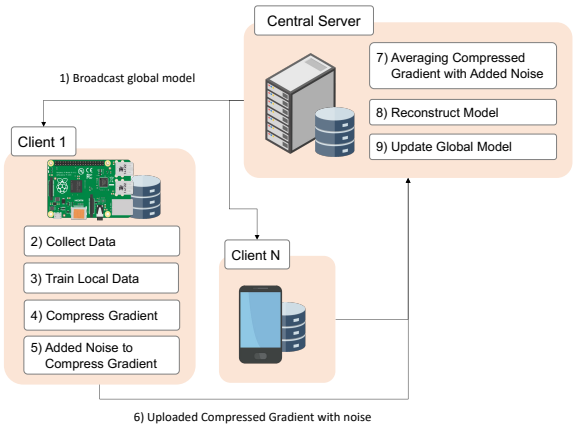


Fig. 1: System Infrastructure of the developed system

client have different processes. Here is a detailed description of the system:

Client consists of embedded and mobile devices. After receiving the global model broadcast by the central server, the client performs four activities, namely:

1) **Collecting Data**. In this process, the client collects data from a sensor or camera. In this demonstration, we use an image that has already been stored on the client's local device. The client preprocesses the image, converts it into an array, and then starts the training process.
2) **Train local data**, In this process, the client trains the model that has been broadcast by the central server using a locally-preprocessed image. This activity generates the local model.
3) **Compress gradient**, In this process, the client compresses the gradient resulting from the local training using compressive sensing [11]. We convert the gradient into a one-rank tensor by multiplying it with a random matrix using Discrete Cosine Transform (DCT).
4) **Added noise**, In this process, the client adds Gaussian noise to the compressed gradient, and the amount of noise is determined by the privacy budget with $\epsilon = 4.0$ and $\delta = 1e - 04$. Once the noise has been added, the compressed gradient will be sent to the server for aggregation.

Server, after receiving the compress gradient with noise from the client, the server performs three activities, namely:

1) **Averaging**, in this step, the server averages the compressed gradient and generates an unconstructed model. It will be used in the next step.
2) **Reconstructing the model**, in this process, the server reconstructed the global model using a recovery algorithm.
3) **Update the global model**, in this process, the server will update the global model based on the model that has been reconstructed in the previous step. This update will enhance the learning process in the next round or iteration.

III. Demonstration

The demonstration of the proposed system involves setting up a federated learning system for classifying digit images into ten classes: images with 0 to 9. The system is comprised of a central server, which runs a Python-based program on a cloud infrastructure, and multiple client devices, including a Raspberry Pi 4 Model B and an Android-based smartphone.

A. System Environment

The demonstration is performed on a central server equipped with 2GB of memory, 50GB of NVMe storage, and an AMD EPYC-Rome processor with a single core. The client consists of Raspberry Pi 4 model b with Quad core Cortex-A72 processor, 4 GB of memory, and 50 GB storage, for the mobile phone we use Redmi 10 with Cortex-A75 processor, 4GB of memory, and 64 GB storage. For the operating system, the Central server runs Debian 11.0 "Bullseye", Raspberry pi 4 model b with Raspberry-pi 64bit, and for mobile phones Android 11 "Red velvet cake". The simulation platform consists of Python 3.8 and Tensorflow 2.9.0, with Flower Federated Learning 1.5.0 as the FL framework, and Tensorflow Privacy 0.8.0 as the privacy library.

B. Demonstration requirement

System requirements are the configuration that a system must have in order for a hardware or software application to run smoothly and efficiently. For this demonstration, the minimum system requirements can be seen in Table I.

TABLE I: Minimum requirement for system implementation

Application	Software	Hardware
Python based	python >=3.8 Tensorflow >= 2.8 TFP >= 0.8.0 Linux os with 64 bit flwr >= 1.0	Processor with single core Memory>=2 GB Storage>=2GB(Free Space)
Android based	Tensorflow lite Android 11.0 or above SDK BT 30 or above	Processor with quad core Memory>=2GB Storage>=4GB(Free Space)

TFP: Tensorflow privacy, flwr: flower framework, BT: Build-tools

C. Demonstration step

The demonstration is divided into four steps: Initialization, loading the dataset, setting up the client, and the Learning process.

1) **Initialization**: In this step, the program on the central server is executed to allow client connections and aggregate the gradients received from them. The maximum number of clients, IP addresses, the number of rounds, and available ports are determined in this setup. This process results can be seen in Fig. 2a.
2) **Load Dataset**: In this step we load the dataset that will be used for the learning stage, instead of using live images from the camera, an open dataset called EMNIST is utilized. A proportional amount of these images is stored on each local device.
3) **Setting up client**: In this step, we configure connection setting to the central server which includes port and

IP address. For Android smartphones, we can set those configurations in the menu connection setup as seen in Fig. 2c, as for the Raspberry Pi we set the connection directly on Python code. For the connection to the server, we are using SSL to ensure a secure communication channel.

4) **Learning**: In this step, the training and learning process is initiated. Ten clients are used in this demonstration, consisting of Raspberry Pi 4 Model B and an Android-based smartphone. We start the process by executing the Python program on the Raspberry Pi as seen in Fig. 2b. As for Android, we can start the learning process by going to the menu LCPL and clicking the start training as seen in Fig. 2c. The detailed process in this step will follow the same procedure as depicted in Fig.1.

For the demonstration, we will use 100 communication rounds, 10 local clients with local epochs=1, learning rate=0.01, and compression ratio=[0.3, 0.5, 0.8]. For the model, we will use CNN with the Softmax activation method.

D. Demonstration results

The outcomes of this demonstration are categorized into two categories: local results and global results. The local results show the accuracy and loss of the model at the client level, which can be observed in Fig. 2b and Fig. 2e. On the other hand, global results showcase the model accuracy, loss, and CPU time from the aggregation process on the central server level as illustrated in Fig. 2a. The summary of the results can be seen in Table II.

TABLE II: Demonstration results

Compression ratio (CR)	CPU time (in seconds)	Accuracy	Loss
0.3	180.19	51.03 %	1.85
0.5	181.73	78.79 %	0.71
0.8	195.71	94.52 %	0.20

*CR 0.3 means that the compressed only have 30% of the size of the uncompressed ones.

Based on Table II, the relationship between accuracy and compression ratio is evident. As the compression ratio decreases, so does the achievable accuracy. For instance, when CR is 0.8, the model achieves a significantly higher accuracy of 94.52 % compared to the accuracy of 51.03% and 78.79% for CR values of 0.3 and 0.5, respectively. On the other hand, lower compression ratios result in faster processing times compared to higher compression ratios. The fastest processing time is achieved with a CR of 0.3, taking only 180.19 seconds.

IV. Conclusion

Our federated learning system, as showcased in the demonstration, can effectively operate on devices that have restrictions in terms of CPU power, memory, and network bandwidth, while maintaining high levels of privacy protection and model accuracy.

V. Challenges and future works

The system we proposed still has some potential challenges, below are the examples:

1) Storage: For Android, the application size is large because the dataset used during the learning process is included in the application package. Therefore, if we use a large dataset, the application size will increase. In future research, we aim to utilize a live dataset without embedding it in the application package.
2) Network latency: For the demonstration, we simulate the system using ideal conditions in terms of network latency. However, we also want to determine whether our proposed system can perform well in high network latency scenarios.
3) Data distribution: For the demonstration, we employ a balanced or i.i.d dataset. In the future, we also plan to simulate our proposed system with unbalanced non-i.i.d datasets to assess its resilience with real-world data.

Acknowledgement

This research was partially supported by JSPS KAKENHI Grant Number JP19KT0020 and JST, PRESTO Grant Number JPMJPR21P7.

References

[1] A. Brecko, E. Kajati, J. Koziorek, and I. Zolotova, "Federated learning for edge computing: A survey," *Applied Sciences*, vol. 12, no. 18, 2022. [Online]. Available: https://www.mdpi.com/2076-3417/12/18/9124

[2] Q. Li, Z. Wen, Z. Wu, S. Hu, N. Wang, Y. Li, X. Liu, and B. He, "A Survey on Federated Learning Systems: Vision, Hype and Reality for Data Privacy and Protection," *IEEE Trans. Knowl. Data Eng.*, pp. 1–1, 2021.

[3] S. Moon and W. Hee Lee, "Privacy-preserving federated learning in healthcare," in *2023 International Conference on Electronics, Information, and Communication (ICEIC)*, 2023, pp. 1–4.

[4] J.-S. Kim, "Design of federated learning engagement method for autonomous vehicle privacy protection," in *2022 Joint 12th International Conference on Soft Computing and Intelligent Systems and 23rd International Symposium on Advanced Intelligent Systems (SCIS ISIS)*, 2022, pp. 1–2.

[5] M. A. Paramartha Putra, M. Verana, G. A. Sampedro, D.-S. Kim, and J.-M. Lee, "3dfed: A secure federated learning-based system for fault detection in 3d printer industry," in *2022 13th International Conference on Information and Communication Technology Convergence (ICTC)*, 2022, pp. 1459–1462.

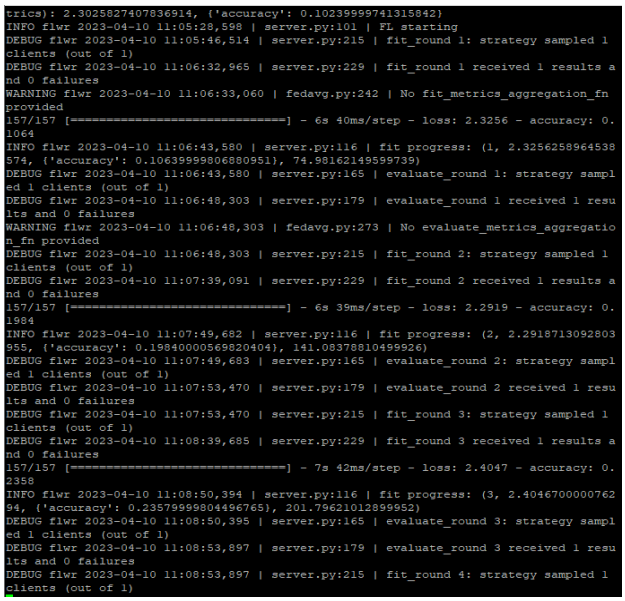
[6] K. Cheng, T. Fan, Y. Jin, Y. Liu, T. Chen, D. Papadopoulos, and Q. Yang, "Secureboost: A lossless federated learning framework," *IEEE Intelligent Systems*, vol. 36, no. 6, pp. 87–98, 2021.

[7] H. S. Sikandar, H. Waheed, S. Tahir, S. U. R. Malik, and W. Rafique, "A detailed survey on federated learning attacks and defenses," *Electronics*, vol. 12, no. 2, 2023. [Online]. Available: https://www.mdpi.com/2079-9292/12/2/260

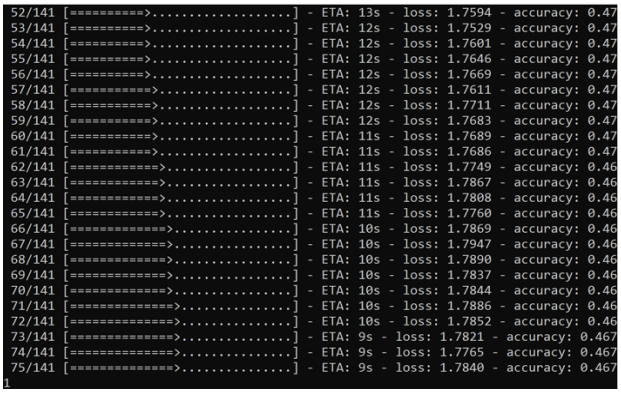
[8] H. Kurniawan and M. Mambo, "Homomorphic encryption-based federated privacy preservation for deep active learning," *Entropy*, vol. 24, no. 11, 2022. [Online]. Available: https://www.mdpi.com/1099-4300/24/11/1545

[9] D. Li, D. Han, T.-H. Weng, Z. Zheng, H. Li, H. Liu, A. Castiglione, and K.-C. Li, "Blockchain for federated learning toward secure distributed machine learning systems: a systemic survey," *Soft Computing*, vol. 26, no. 9, pp. 4423–4440, May 2022. [Online]. Available: https://doi.org/10.1007/s00500-021-06496-5

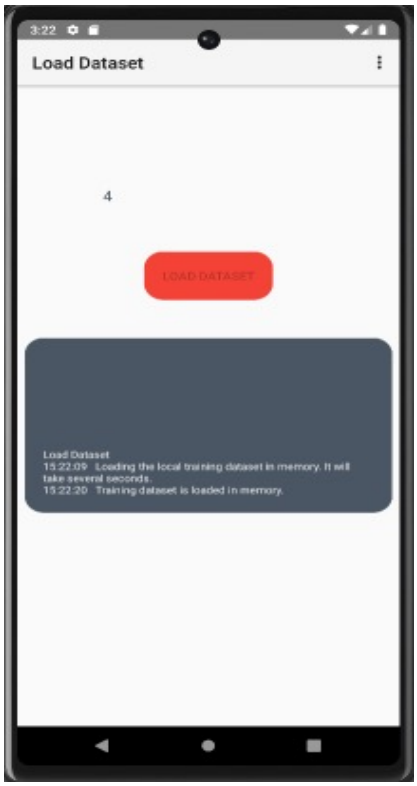
[10] V. Mothukuri, R. M. Parizi, S. Pouriyeh, Y. Huang, A. Dehghantanha, and G. Srivastava, "A survey on security and privacy of federated learning," *Future Generation Computer Systems*, vol. 115, pp. 619–640, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167739X20329848



(a) Server state when accepting gradient from the client



(b) Raspberry Pi state when connecting to the server



(c) Smartphone state when loading the dataset

(d) Setting up a connection to the cloud server from mobile apps

(e) Local training to produce a local model

Fig. 2: System prototype results

[11] R. Kerkouche, G. Ács, C. Castelluccia, and P. Genevès, "Compression boosts differentially private federated learning," in *2021 IEEE European Symposium on Security and Privacy (EuroSP)*, 2021, pp. 304–318.