ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

ON DEVICE FEDERATED LEARNING PERFORMANCE

Student: Kang Fu

Project Advisor: Professor Anne-Marie Kermarrec

Project Supervisor: Dr. Rafael Pires, Dhasade Akash Balasaheb

Semester Project
7th January 2022

Abstract

Nowadays the edge devices experience the explosive growth due to improving computation power. Numerous data has been generated to record the user activities. Service provider can utilize those data to further enhance the service quality. However, as the data is extremely larger, centralized servers face great challenge to train huge data. Besides, users are sensitive to personal information in application usage data. One problem occurs that how service provider improves service quality without user application data. Hopefully, this difficulty is addressed by federated learning in conjunction with decentralized learning systems. Federated learning is a machine learning technique that involves training an algorithm across numerous decentralized edge devices that keep local data samples without transferring them. However, one critical issue of federated learning is systems heterogeneity, which has a great impact on federated learning process. When devices are heterogeneous, many issues arise: there are stragglers in the training process, meaning devices may drop out suddenly. This is hard to model. Therefore, this project is aimed to collect system measurements like memory usage, power consumption and training time, with which we can finally profile device characteristics. To measure the system parameters, Android applications are designed to implement on-device federated learning and measured to find the device characteristic. The applications can test multiple image classification datasets to get better generalization. Though promising results are obtained, there's still a place for further improvements and research.

Contents

CHAPTER 1

INTRODUCTION

Mobile devices are generating an increasing amount of data on a daily basis as computational capability on them grows. These data can be utilized to improve the user experience on mobile devices, such as personalized recommendations and text message prediction. This contradicts with security of personal information, which is cared by users. That's why it is important to analyze how the service provider can improve service quality without knowing the data of the users. Federated learning is an attractive option. It is a new machine learning technique that trains models on distributed edge devices rather than sharing data with a centralized server[1–3]. Generally, edge nodes share model parameters with the server, such as weights and biases, and the server aggregates all of the received model parameters to assist edge nodes in improving model performance. This allows it to build a robust model without having to share the training data, safeguarding data privacy.

Although federated learning has good performance in handling data privacy, it still faces great challenges. System heterogeneity[4–6] is one of these issues. The heterogeneity results from the differences in client device hardware conditions (CPU, memory), network connections (3G, 4G, 5G, WiFi), battery power and the storage of each device in the federated learning network. Due to network restrictions, only a few devices may be active at any given time. In addition, the device may be subjected to unforeseen circumstances such as a lack of power or network connectivity, resulting in an instantaneous loss of connection. The formulation of the overall federated learning approach is influenced by the heterogeneous system architecture. Although modeling heterogeneity is tough, the scope of the project can be limited to focus solely on heterogeneous hardware. Measurement of memory usage, energy consumption, and training time can be done to profile the device characteristics when the edge devices participate in federated learning. Since the numbers will change from task to task, this measurement can be applied to a variety of learning activities and models. As a result, the device properties can be more accurately represented.

This project is designed to model the devices characteristics by measuring the training time, memory usage and power consumption. Although there are three Android devices involved, only two are modelled because one Android device serves as the server. To fulfill the function of on-device training of federate learning, Android applications are designed in Android Studio. There are four sections to the client applications. The initial step is to use the HTTP protocol to get the training dataset stored online. The model is sent or received using TCP socket programming in the second section. The typical deep learning implementation is used in the third step to further train the model using local data. The final step is to save each round's training time. The structure of the server application is similar. The initial step is to use the HTTP protocol to get the testing dataset stored online. Because there are multiple clients, the second element is multi-thread TCP socket programming to send or receive the model. The development of an initial model, which defines the architecture of a convolutional neural network, is the third step. The final step is to save each round's evaluation metrics. In terms of memory utilization measurement, Android Studio's embedded tool Profiler may give real-time memory usage. The power consumption is measured with the Powerspy device[7] and verified with the Android Debug Bridge[8], which is a versatile command-line tool that allows user communicate with a Android device. It has tools to provide information about system services like power consumption.

The rest of the report can be split into three parts. Chapter 2 introduces all the details about the implementation of on-device federated learning, including the basic setup, dataset partitioning and neural network topology. The evaluation of federate learning performance and device characteristics are covered in Chapter 3. A conclusion and future work in Chapter 4 brings this report to a close.

Chapter 2

Implementation

This section describes all the details about implementation. To begin, the implementation setup is described, including the client device specifications, required deep learning framework and measurement methods. The architecture of the server and clients is shown in Chapter 2.2, followed by a description of the entire process's working flow. The data is partitioned and the neural network is designed for each dataset. Last part is the User Interface of designed Android application. This project is implemented in Java and the corresponding code is available at `https://gitlab.epfl.ch/sacs/ondevice-fl-perf`.

2.1 Implementation Setup

The implementation setup involves three Android devices(two Android phones and one Android tablet). In this project, these devices are used to implement on-device federated learning. One Android phone works as server to aggregate the model parameters. The remaining two devices server as clients, which store the data and train the model locally. The clients model specifications are shown on Table 2.1.

Clients	model	RAM	Core	Frequency	Battery
Tablet	Huawei Mediapad T3 10	2G	4	1.4GHz	4800 mAh
Phone	Alcatel 1c	1G	4	1.3GHz	2050 mAh

Table 2.1: The Specification of Client Devices

The Android applications for server and clients are developed on Android Studio. The main deep learning framework library is deeplearning4j[9]. This is a powerful library that provides wide support for deep learning algorithms for Java Virtual Machine. It is easy to build, train and deploy neural networks on Android applications since Android applications are written in Java language.

The motivation of this project is to measure the on-device federated learning performance, including training time, memory consumption and the battery usage. The training time can be measured using Java embedded time module. Memory consumption can be viewed by Profiler tool in Android Studio. This tool can provide real-time memory usage as well as the CPU and network usage. The battery usage is measured by PowerSpy device, which can be used to measure power consumption. Android Debug Bridge can also provide the log of power consumption of device during a period.

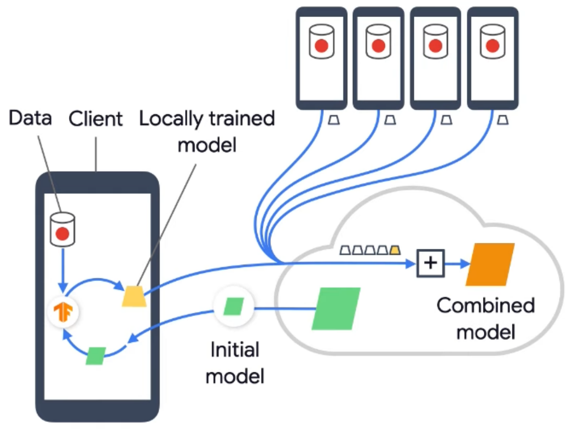


Figure 2.1: The Diagram of Federated Learning[10]

2.2 The Architecture of Server and Clients

Figure 2.1 shows the basic idea of the federated learning. The cloud and the mobile phones in the Figure 2.1 represent the server and clients respectively. Initially, the model is sent to all clients by the server. The clients then use their local data to train the received model and deliver the locally trained model back to the server. Once the server has received all of the models from all of the clients, it can integrate them into a new model. Following then, the server will broadcast the combined model to all clients until the evaluation of the combined model reaches the pre-defined target.

The architecture design of server and clients in this project follows the above introduction. The working flow of the server and clients is:

Step 1: Server generates a convolutional neural network model with zero weights and biases

Step 2: Server sends the model to all the clients by TCP socket.

Step 3: Clients receive the model from server and train the model using its own data

Step 4: Once the training process is finished, clients send the trained model back.

Step 5: Server aggregates the weights and biases and evaluates the combined model on the test dataset.

Step 6: If the combined model reaches the target accuracy, the federated learning process stops. If not, server will send the combined model to clients to continue training.

Step 7: Repeat Step 6

2.3 Data Partition in Clients

The main task of the experiment is image classification task. Convolution neural network models are implemented on three different image datasets (MNIST, EMNIST and CIFAR-10). The introduction of those datasets is followed.

- MNIST[11]
  The first dataset is MNIST digit handwritten images. It contains 10 classes, namely 0 to 9. The training dataset has 60,000 28x28 greyscale images and the testing one has 10,000 28x28 images. In

the training and testing dataset, the data has a balance distribution, meaning that each class contains the same number of images.

- EMNIST[12]
  The second dataset is EMNIST handwritten images. This is the extension of MNIST dataset. It contains 62 classes, namely 0 to 9 and both lower case and upper case character. The dataset has 814,255 images with the size of 28x28. In this project, the whole EMNIST dataset will not be used due to the restriction of the device storage and computation power.

- CIFAR-10[13]
  The third dataset is CIFAR-10. The CIFAR-10 dataset contains 60,000 32x32 color images in 10 different classes. The training data has 50,000 images and testing data is 10,000.The 10 different classes represent airplanes, cars, birds, cats, deer, dogs, frogs, horses, ships, and trucks. There are 6,000 images of each class, which shows that this is a balanced dataset.

In terms of dataset separation, the training data is partitioned into two clients while the testing data are all kept in the server for the simulation purpose. Indeed, the testing data is also stored at the client side in the real-life scenario. Generally, there are two ways of data partition. One way is the independent and identically distribution(IID), which means the images of all clients have the same probability distribution and all are mutually independent. In this case, the images of each client are selected by the same number in each class to simulate the IID partition. The other way is non-IID[14], which means the images of each client can be arbitrarily different. Both scenarios will be evaluated in this project. With regard to the non-IID, an extreme partition method is applied that one client will get all the images of half classes and the other client will get the remaining classes. Actually, non-IID partition is the most common situation in real-life application because each client represents a unique user and has user-related data[15]. For CIFAR-10, the 10 different classes have been renamed as 0 to 9 in Table 2.2.

Due to the limitation of device storage room and computing power, EMNIST dataset used in this project is the part of the whole EMNIST dataset. 148066 of 697932 images are selected as training dataset and 14575 of 116323 images are tested.

non-IID	MNIST	EMNIST	CIFAR-10
Tablet	0,2,4,6,8	upper_case character + 0,2,4,6,8	0,2,4,6,8
Phone	1,3,5,7,9	lower_case character + 1,3,5,7,9	1,3,5,7,9

Table 2.2: The non-IID partition of Training Data

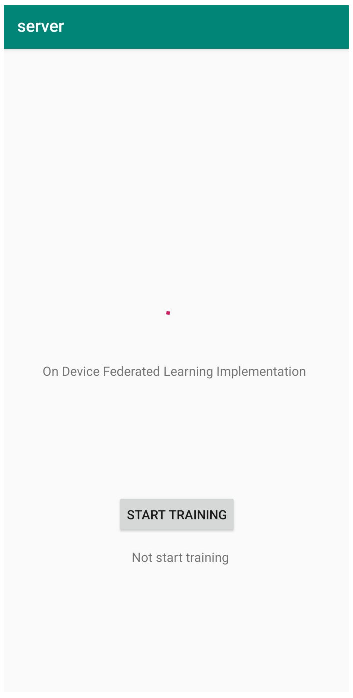
2.4 NEURAL NETWORK DESIGN FOR DIFFERENT DATASETS

From section 2.1, Convolutional Neural Network(CNN) models are applied to handle image classification task. Image classification involves the extraction of features from the image to observe some patterns in the dataset. Artificial Neural Network(ANN) can also serve the purpose of image classification, but it would be very computation-intensive since the trainable parameters are extremely large. Compared to ANN, CNN is very effective in reducing the number of parameters without losing on the quality of models. One important method is to use filters to exploit the spatial locality of a particular image by enforcing a local connectivity pattern between neurons.

Since the EMNIST dataset is the extension of MNIST dataset with more classes, the same convolutional neural network model can be applied. The only difference is the output number of classes. The MNIST dataset has only 10 classes while EMNIST has 62 classes. The CIFAT-10 dataset has large and complex images than MNIST/EMNIST dataset. Therefore, the model for CIFAR-10 dataset will be more complicated. The parameters of CNN model[16] can be viewed from Table 2.3.

Dataset	Parameters	Input	Conv1	Pool1	Conv2	Pool2	FC1	FC2	FC3
MNIST	Kernel size Strides	28x28x1	5x5x20 1x1	2x2 2x2	5x5x50 1x1	2x2 2x2	500	10	-
EMNIST	Kernel size Strides	28x28x1	5x5x20 1x1	2x2 2x2	5x5x50 1x1	2x2 2x2	500	62	-
CIFAR-10	Kernel size Strides	32x32x3	3x3x16 1x1	3x3 2x2	3x3x64 1x1	4x4 4x4	384	192	10

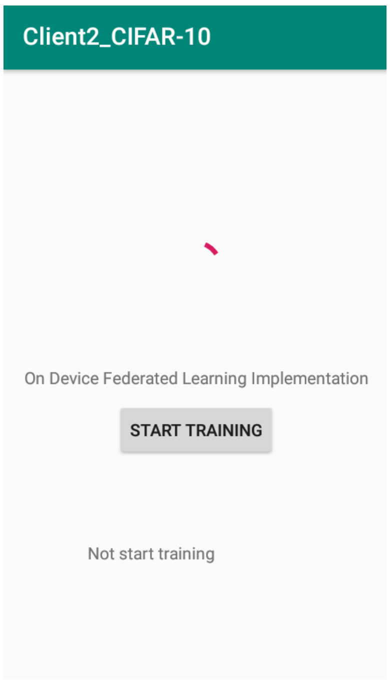
Table 2.3: The Parameters of CNN Model


server

On Device Federated Learning Implementation

Not start training

Figure 2.2: The UI of Server App


Client2_CIFAR-10

On Device Federated Learning Implementation

Not start training

Figure 2.3: The UI of Client App

2.5 USER INTERFACE OF APPLICATION

The user interface(UI) is an important aspect of application design. Here, the most basic design has been used. The device role and dataset used in this implementation are indicated by text at the top of the application. In the midst of the application, a progress bar indicates if the application is operating or not. Underneath the progress bar is a button. The federated learning process begins when this button is pressed. The output information is displayed at the bottom.

Chapter 3

Evaluation

This section covers the evaluation of the federated learning process and device characteristics. As Chapter 2.1 shows, three different datasets have been applied to investigate the device system characteristics. In this Chapter, various mobile usage scenarios are examined to determine how the device handles federated learning. One scenario is the idle state of client devices, which represents the time that users do not use the devices for a long time like sleeping at night. As a result, the devices are exclusively engaged in the federated learning process. The other scenario is the busy state of client devices, which depicts when people are using their devices for business or enjoyment while federated learning is running in the background. The Youtube videos are played to simulate the busy state.

Different machine learning methods for training are assessed in addition to various mobile usage scenarios. Full-image training is one way, in which the neural network trains all of the images it possesses for one epoch. This method is most commonly used in centralized machine learning with IID data partitioning, where huge graphics processing units can help speed up the training process. The second option is batch-image training, which means the neural network will train in multiple batches (the number of images in each batch is determined by batchsize) rather than using all of the data for one epoch. This strategy is commonly employed in federated learning since edge devices have limited computation capacity, and this method can drastically reduce training time. The non-IID data stored in clients is also an important factor to use batch-image training. Since the data is non-IID, the longer time consumed to train the model will result in model drifting towards its own local optima. This is harmful for the performance of the aggregated model. It is better to have quick communication with servers and do more aggregation to enhance the model generalization. However, there is a trade-off that more communication with servers means more transmission and aggregation time cost and can increase the total time to reach the target accuracy.

3.1 MNIST Evaluation

The MNIST image classification is the simplest task in this project. Thus the target accuracy is set to 0.95. Table 3.1 presents the results of different scenarios of MNIST dataset implemented on two clients. The CNN model is powerful and it needs only one round to reach the target accuracy for the IID data partitioning using full-image training. There are more exchange rounds of model if the non-IID partition is applied. Figure 3.1 demonstrates the accuracy change as well as total training time over the number of exchange rounds. The full-image training method needs 3 rounds while the batch-image training requires 17 rounds. Take the phone as an example, the batchsize is set to 64. Therefore, there are 469 iterations in one training epoch as each client has 30,000 images. The training time of full-image training method (469 iterations) of phone is 1013 seconds. The batch-image training uses only 25 iterations as a round

Devices	State	Partition	Batch Training	Memory Usage(MB)				CPU Rate	Training Time(s)	Power Consumption(mAh)
				Start App	Load Data	Training	Waiting for New Model			
Tablet	Idle	IID	No	81	123	142	125	85%	474	44.25
		Non-IID	No						764	47.78
		Non-IID	Yes 25						36	6.18
	Busy	IID	No	67	118	125	85	25%	1495	46.28
		Non-IID	No						1503	50.32
		Non-IID	Yes 25						82	10.12
Phone	Idle	IID	No	38	85	137	116	80%	876	28.9
		Non-IID	No						1013	38.47
		Non-IID	Yes 25						70	3.75
	Busy	IID	No	35	73	88	80	20%	2325	35.56
		Non-IID	No						2221	43.98
		Non-IID	Yes 25						125	4.95

Table 3.1: Evaluation Result of MNIST dataset

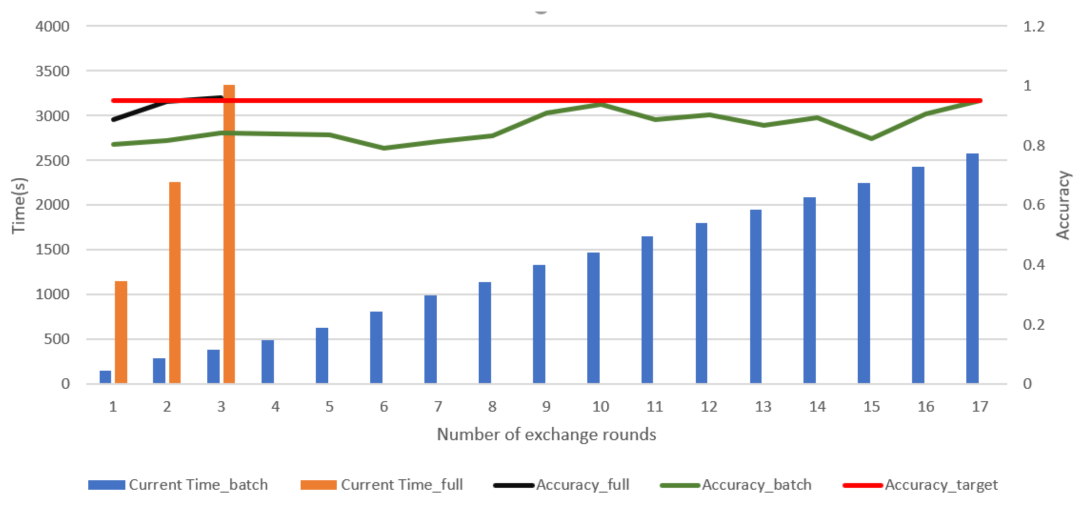


Figure 3.1: The Accuracy and Total Training Time across Rounds in terms of MNIST Non-IID Partition

and its training time is 70 seconds. It can be seen by Figure 3.4 that the batch-image training method can reduce the training time, which is an advantage of federated learning. This is related to the characteristics of non-IID data. Since the data in clients are often non-IID, full-image training could drift the model toward the global optimal value, which may not have a good performance in the evaluation of total dataset. Batch-image training randomly chooses small number of data and allows for a more robust and efficient convergence, avoiding local minima. Different data partitions might affect training time, and it has been discovered that non-IID partitions take longer to train. It can be seen that the tablet's training time in a busy condition is nearly three times longer than in an idle one. From the perspective of devices, the tablet has a short training time than phone in all scenarios.

The power consumption in Table 3.1 is calculated on a per-round basis. Long training time leads to higher power usage for each device. It can be shown by Figure 3.3 that tablet consumes more energy than phone in all scenarios. The devices in busy state requires a little more energy to finish the training than that in idle state. The difference between them isn't significant.

In terms of memory usage, the default operation executioner in Android Studio shows that both devices are 4 cores, but memory are different. The tablet is using 0.5GB while the phone is 0.3GB. This difference may result from the hardware difference since tablet has a larger RAM than phone. When the application starts, tablet consumes also two times memory than phone. After the training dataset is loaded, it is obsevered that about 40MB has been increased in memory of both devices. The memory continues to go up when the training process starts and drop to the level of finishing loading dataset once the training is finished and the clients wait for new model. Since the device is in idle state, this application occupies

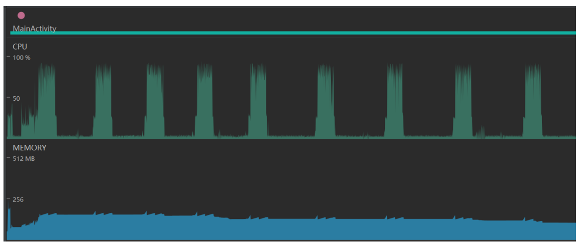


Figure 3.2: The Real-time Memory Usage

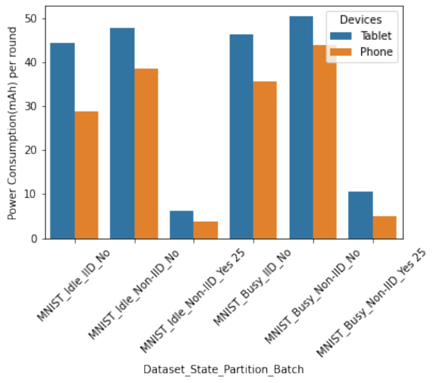


Figure 3.3: The Power Consumption per Round

most of CPU resources. Differences occur if the devices are in busy state, for instance, users are watching Youtube video while training the model. The memory usage in training decreases a little and the CPU rate drops by about 60 percent. This is normal because the user activities are consuming resources. There is an noticeable phenomenon that the memory usage at the beginning of training is very high and slightly drop down as the exchange rounds become large, which is presented by Figure 3.2.

3.2 EMNIST EVALUATION

Devices	State	Partition	Batch Training	Memory Usage				CPU Rate	Training Time(s)	Power Consumption(mAh)
				Start App	Load Data	Training	Waiting for New Model			
Tablet	Idle	Non-IID	Yes 25	83	151	160	145	90%	30	9.17
	Busy			63	84	121	111	25%	79	10.74
Phone	Idle			35	95	136	125	50%	71	4.68
	Busy			33	72	100	90	20%	137	5.34

Table 3.2: Evaluation Result of EMNIST dataset

The EMNIST dataset has more images and classes compared to MNIST dataset. Each client has 74,033 image.The more images means more training time. The implementation proves that full-image training in phone using IID EMNIST dataset per round takes approximately 50 minutes, which is not acceptable for federated learning. Consequently, batch-image training method is applied. The number of batch is chosen to be 25 and the batchsize is 64. Thus, 1600 images are selected to be trained as one epoch. The target accuracy is set to 0.78.

From Figure 3.4, it takes 26 exchange rounds to reach the target accuracy. The training time of each round for tablet and phone in idle state is 30 seconds and 71 seconds. In this dataset, the training time of devices in busy state is about 2.6 times than that in idle state.

Power usage is similar to that tested in MNIST dataset. The power consumed by tablet is almost twice of that of phone. Compared to MNIST dataset evaluation result in Figure 3.5, EMNIST dataset will cost a little more energy than MNIST dataset in each round.

The memory usage at the beginning of application of EMNIST dataset is similar to that of MNIST dataset. There is difference in loading the dataset that the memory increase by 60MB in EMNIST data while 40MB in MNIST data. This is reasonable that the EMNIST data is larger than MNIST data. Therefore, the memory required for training also increased.

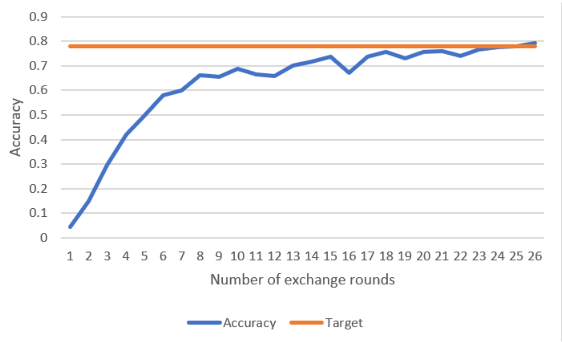


Figure 3.4: The Accuracy over Exchange rounds

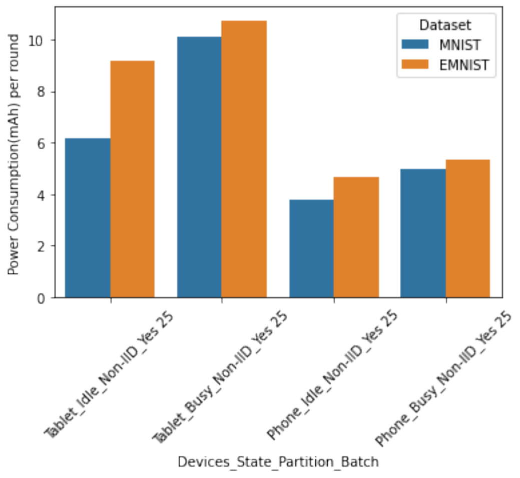


Figure 3.5: The Power Consumption of MNIST and EMNIST Dataset

Devices	State	Partition	Batch Training	Memory Usage				CPU Rate	Training Time(s)	Power Consumption(mAh)
				Start App	Load Data	Training	Waiting for New Model			
Tablet	Idle	IID	Yes 25	82	104	138	124	85%	31	9.32
	Busy			65	90	117	110	25%	72	11.73
Phone	Idle			35	70	131	122	55%	74	5.55
	Busy			32	57	99	87	20%	122	6.21

Table 3.3: Evaluation Result of CIFAR-10 Dataset

3.3 CIFAR-10 EVALUATION

The image in CIFAR-10 dataset has the size of 32x32x3, which is more complex than that in MNIST/EMNIST dataset. Each client has 25,000 images. First, the IID partition of dataset has been evaluated. It takes two exchange rounds for full-image training to reach the target accuracy(0.6)[17]. The total training time is 1959 seconds, which is much less than batch-image training, consuming 3436 seconds. This is normal in IID partition since the IID partition can be viewed as a small dataset of centralized learning. The total time of batch-image training may be dominated by the communication latency and the evaluation time at server side.

The training time for each round is longer than that of MNIST/EMNIST dataset because the neural network model for CIFAR-10 is more sophisticated. When devices are in busy state, the training time is almost twice than that in idle state, as is shown by Table 3.3.

The memory usage when starting application is nearly identical with the evaluation in MNIST/EMNIST dataset. The difference lies in loading the dataset. The memory increase only by 30MB, which is less than both MNIST and EMNIST dataset. The memory usage is very similar to MNIST dataset evaluation result, which may indicate that the memory usage is related to the images quantity that client stores.

The power consumption for each round is similar to that of EMNIST dataset. The reason may be they are all use batch-image training with batch number 25 and same batchsize 64. The tablet has almost double power usage than the tablet.

3.4 DEVICES CHARACTERISTICS

The evaluation results of three dataset provide the insight of device characteristics. Firstly, tablet has a more powerful CPU than that of phone. Consequently, the tablet requires less training time than the

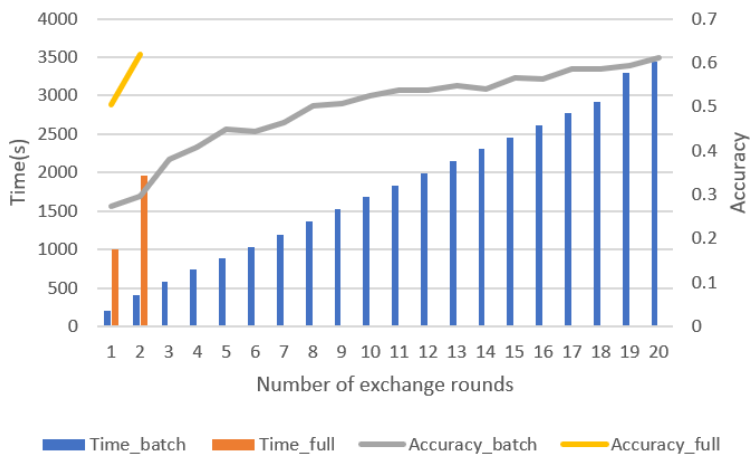


Figure 3.6: The Accuracy Change and Total Training Time over Exchange rounds using CIFAR-10 IID Partition

phone, but more power consumption. From all the evaluation results, it is seen that the training time is almost identical when the batch-image training method is applied. It can be concluded that the average time to train one image regardless of size for a tablet in idle state is 20 milliseconds, whereas the average time for a phone is 45 milliseconds. The tablet is twice faster than phone. In the busy state, training time is roughly 2.5 times longer than that in the idle state. The explanation for this has to do with the CPU utilization rate. When devices are busy, which means that other user activities require CPU resources, the amount of resources available for federated learning decreases, thus increasing the training time.

Furthermore, CPU has a close relationship with power consumption. The model training, which requires the CPU to do repeated gradient calculations, is the most power-intensive element of this program. In virtually all cases, the tablet consumes more power than the phone. The size of the gadgets determines this. Because a tablet is larger than a phone, it requires more electricity to operate correctly. Compared with results of three datasets, CIFAR-10 model requires a little more energy than EMNIST model. It can be inferred that the power usage is related to model complexity since the CIFAR-10 model just has one more fully-connected layer than EMNIST model.

When it comes to memory utilization, once the application starts to run, the memory usage will begin to increase. The phone has 1 gigabyte of RAM, while the tablet has 2 gigabytes. As a result, tablet memory utilization is larger than phone memory usage. One striking phenomenon is that as the number of exchange rounds increases, the memory consumption of training decreases. One probable explanation is that the application needs to load the needed libraries during the first few training rounds.

CHAPTER 4

CONCLUSION

In this project, the task is to profile the device characteristics in terms of training time, memory usage and power consumption. Since this is on-device measurement, Android applications are designed and implemented to realize the process of federated learning. Then, system measurements are done using various tools. Three different datasets are tested to get the better generalization of device properties. The state of edge devices, data partitioning and batch training method are considered to form multiple test scenarios. In this way, the device attributes can be fully developed.

It is observed that tablet has a shorter training time than phone because the tablet has a high specification on CPU frequency and RAM. The average time to train one image regardless of size for a tablet in idle state is 20 milliseconds, whereas the average time for a phone is 45 milliseconds. The tablet is twice faster than phone. This can also infer that the RAM could impact the training time as the RAM of tablet is also two times than that of phone. Furthermore, it has been discovered that the adoption of mini-batch training can efficiently reduce the training time, especially when the dataset is non-IID partition. The explanation is that the using full-batch training on non-IID dataset could drift model toward to its local optima instead of global optima from the server side.

The memory continues to go up once the Android application starts. Four stages of memory usage are recorded. The starting point of application for tablet requires more memory than that of phone since the tablet has a larger size and complex structure. The memory increase during the data loading is almost identical because each client have same number of images. The memory consumption of training lowers as the number of exchange rounds increases. This can be explained that the application needs to load the required libraries during the first few training rounds.

The power consumed by the Android application mainly comes from the model training part. Therefore, the complexity of neural network model dominates the total power usage. Consequently, the CIFAR-10 dataset has the highest power usage than MNIST/EMNIST dataset. Although the devices in busy state takes much longer training time than that in idle state, the power consumed is close in both states.

All in all, the designed Android Application can fulfill the system measurement tasks and profile the device characteristics using the measurement results. Therefore, the exhibited experiments produce positive results and provide information about the device's features. However, this project has the potential to be expanded and some future implementation possibilities are given in the next part.

4.1 FUTURE WORK

All the experiments done in this project is the simulation of federated learning. In real-life application, the federated learning could even have millions of edge devices instead of two devices in this project. One direction of development lies in adding more clients to further approach the federated learning simulation. Besides, the communication method used in this project is socket programming, which requires the devices and server are in the same Local Area Network(LAN). If possible, the server can be implemented on the cloud or physical cluster instead of one Android device. Additionally, the user interface of this Android application can be further polished. Currently, the application cannot accept user input and one dataset represents one application, which is cumbersome. Dataset input options can be added to allow users to fully interact with application.

Bibliography

[1] Qinbin Li et al. ‘A Survey on Federated Learning Systems: Vision, Hype and Reality for Data Privacy and Protection’. In: *IEEE Transactions on Knowledge and Data Engineering* (2021), pp. 1–1. ISSN: 2326-3865. DOI: 10.1109/tkde.2021.3124599. URL: http://dx.doi.org/10.1109/TKDE.2021.3124599.

[2] Andrew Hard et al. *Federated Learning for Mobile Keyboard Prediction*. 2019. arXiv: 1811.03604 [cs.CL].

[3] Daniel J. Beutel et al. *Flower: A Friendly Federated Learning Research Framework*. 2021. arXiv: 2007.14390 [cs.LG].

[4] Xingyu Li et al. *FedLGA: Towards System-Heterogeneity of Federated Learning via Local Gradient Approximation*. 2021. arXiv: 2112.11989 [cs.LG].

[5] Amirhossein Reisizadeh et al. *Straggler-Resilient Federated Learning: Leveraging the Interplay Between Statistical Accuracy and System Heterogeneity*. 2020. arXiv: 2012.14453 [cs.LG].

[6] Zheng Chai et al. ‘Towards Taming the Resource and Data Heterogeneity in Federated Learning’. In: *2019 USENIX Conference on Operational Machine Learning (OpML 19)*. Santa Clara, CA: USENIX Association, May 2019, pp. 19–21. ISBN: 978-1-939133-00-7. URL: https://www.usenix.org/conference/opml19/presentation/chai.

[7] URL: https://www.alciom.com/en/our-trades/products/powerspy2/.

[8] URL: https://developer.android.com/studio/command-line/adb.

[9] URL: https://github.com/eclipse/deeplearning4j.

[10] URL: https://www.youtube.com/watch?v=X8YYWunttOY&t=152s.

[11] Li Deng. ‘The mnist database of handwritten digit images for machine learning research’. In: *IEEE Signal Processing Magazine* 29.6 (2012), pp. 141–142.

[12] Gregory Cohen et al. *EMNIST: an extension of MNIST to handwritten letters*. 2017. arXiv: 1702.05373 [cs.CV].

[13] Alex Krizhevsky, Vinod Nair and Geoffrey Hinton. ‘CIFAR-10 (Canadian Institute for Advanced Research)’. In: (). URL: http://www.cs.toronto.edu/~kriz/cifar.html.

[14] Yue Zhao et al. *Federated Learning with Non-IID Data*. 2018. arXiv: 1806.00582 [cs.LG].

[15] Peter Kairouz et al. *Advances and Open Problems in Federated Learning*. 2021. arXiv: 1912.04977 [cs.LG].

[16] Georgios Damaskinos et al. ‘FLeet: Online Federated Learning via Staleness Awareness and Performance Prediction’. In: *CoRR* abs/2006.07273 (2020). arXiv: 2006.07273. URL: https://arxiv.org/abs/2006.07273.

[17] Jianyu Wang et al. ‘Tackling the Objective Inconsistency Problem in Heterogeneous Federated Optimization’. In: *CoRR* abs/2007.07481 (2020). arXiv: 2007.07481. URL: https://arxiv.org/abs/2007.07481.