Implementation of Federated Learning on Resource-constrained devices: Lessons learned

Thomas Tsouparopoulos
Department of Informatics
Athens University of Economics and Business
Athens, Greece

Iordanis Koutsopoulos
Department of Informatics
Athens University of Economics and Business
Athens, Greece

***Abstract*—In this paper, we implement and deploy the most widely used algorithm in Federated Learning (FL), i.e. Federated Averaging (*FedAvg*), on an experimental testbed. The testbed consists of Raspberry Pi devices (RPis) connected to a wireless network. We perform extensive evaluations in various real-world scenarios and provide insightful results and lessons learned. Specifically, we evaluate *FedAvg* on our testbed to investigate the effect of the following parameters on training: (*i*) number of users selected at each round, (*ii*) number of local gradient steps before communicating with the server, (*iii*) clients disconnecting from the server, (*iv*) data distribution heterogeneity across clients, and (*v*) mobility of users. Finally, we examine the impact of the wireless environment on the learning performance under varying network parameters.**

***Index Terms*—Federated Learning, Implementation, Raspberry Pi devices.**

I. Introduction

Due to the huge increase in the use of edge devices, like sensors and wearables, data are mostly produced in resource-constrained devices, with limited computational and connectivity capabilities [1]. FL is introduced as a distributed Machine Learning (ML) paradigm to take advantage of such data produced at the network edge. In FL, many clients (e.g., mobile devices) collaboratively train a ML model, under the orchestration of a central server without exposing any information about their training data [2]. When developing and experimenting with FL algorithms, most current research efforts, especially in the ML scientific community, are focused on simulation environments located on a single machine. Since all training parameters are transmitted over wireless links though, these setups do not reflect real-world FL systems.

In an IoT setting, clients might only be temporarily connected with the server, for a few communication rounds, or they might vary their distance from the wireless access point (AP) due to mobility. Such clients' behaviors can be validly simulated, in order to expound their effect on the learning performance, and this is one of the main aspirations of this work. Furthermore, the computational power (e.g., CPU, RAM etc.) and communication capacity (e.g., bandwidth) of resource-constrained edge devices, can exhibit extreme heterogeneity, and their effect in training needs to be examined. Consequently, the deployment of FL on actual hardware remains an under-explored area, as the main focus is centered around theoretical results driven by data experiments, where the data is conceptually but not physically distributed.

In this work, we cater to fill this gap by thoroughly evaluating *FedAvg* [2], the most widely used FL algorithm for benchmarking, and by examining all of its parameters, as well as repercussions that arise in the wireless setting, with the purpose of effectively studying their impact on the learning performance of FL. The main contributions of this work can be summarised as follows:

- We implement the *FedAvg* algorithm and demonstrate key software and hardware issues that arise from the deployment on RPis.
- We perform real experiments with RPis and investigate the impact of various FL algorithmic parameters on the learning performance.
- We explore and expound the effects of the volatility of the wireless environment on the FL procedure.

The paper is organized as follows: In section II, we outline an overview of the related work. In section III we demonstrate our implementation, cite the hardware and software issues we addressed and describe the experimental setup. In section IV we showcase and discuss results from our experiments. We conclude and suggest some extensions in Section V.

II. Related Work

FL Algorithms: *FedAvg* [2] was the first successful algorithm in the FL setting. It works by simply aggregating the model weights sent from the clients and producing a global model by averaging them. In [3], the authors prove that *FedAvg* suffers from the phenomenon of "client-drift" (i.e. each client "pulls" the global model towards their local model), when the data are non-identically and independently distributed (non-i.i.d), resulting in unstable and slow convergence. To alleviate the issue of heterogeneous local updates due to non-i.i.d local datasets, the authors in [4] propose adding a proximal term $\frac{\mu}{2}\|w - w^g\|^2$ to each local objective, where μ is a tunable parameter and w^g are the global weights sent at that communication round to the client. These approaches improve performance over *FedAvg*, but statistical heterogeneity caused by non-i.i.d data is still a core challenge of FL [5].

Implementation of FL on edge devices: In order to effectively deploy FL over real-world IoT networks, it is necessary to investigate how the volatility of the wireless environment affects the performance of FL algorithms. In [6], the authors

ISBN 978-3-903176-48-5

introduce an adaptive client selection scheme that takes into consideration the clients' resources, however they perform simulation of the wireless environment to test its learning performance and training time. A similar work in terms of implementing and evaluating FL is [7], where the authors evaluate FL at an actual IoT testbed of RPis. However, they examine the learning performance of FL only in simulations. An adaptive FL algorithm that determines the best tradeoff between local computation and global communication rounds is proposed in [8], but the authors omit the effects of wireless network volatility. From the networking standpoint, the authors in [9] propose a communication scheme with adaptive resource optimization for FL at the edge and evaluate QoS metrics such as delay and packet loss in a simulation environment. Finally, a tutorial of implementation of FL on RPis is presented in [10], but no evaluations are provided, while a third-party library is used to accommodate the communication between the server and the clients.

III. Raspberry Pis' Experimental Setup

For our implementation we used 5 Raspberry Pi devices (version 4, with Quad core CPU and 4 GB RAM) as clients and a desktop computer as the server. One of our main aspirations was to build and use a setup that allows for an easy-to-follow-and-reproduce procedure and can be implemented across many types of resource-constrained edge devices. This allows us to focus only on the complexity of the deep learning models when considering computation time of such devices. The setup of our hardware testbed is shown in Figure 1.

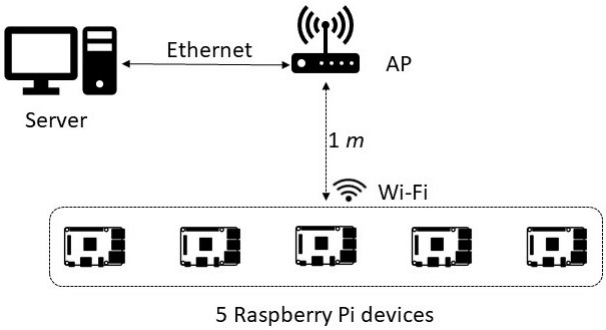

Fig. 1: The arrangement of our hardware setup. We use a desktop computer connected with Ethernet to the internet, and 5 RPis connected to the same network over Wi-Fi, with a distance of 1 meter from the AP.

A. Software settings

We follow the conventional FL procedure described in [2]. Edge devices train their local models using their own locally available data and then transmit the trained local models to the server. The server in turn generates a global FL model by aggregating the received local models and sends it back to the clients. The procedure repeats until a halting criterion is met.

For the implementation, the ML library PyTorch[1] was preferred. PyTorch does not have an official package in Python Package Index (official third-party repository of software) for ARM compatible devices, for versions 1.0 and above. To cater for this issue, a pre-compiled wheel (i.e. the standard built-package format used for Python distributions package) was used to install PyTorch on the RPis.

Remark: When dealing with memory-intensive tasks, the RPi system ran into "out of memory" errors or had to shut down other packages. To address this, the swap file is used as the virtual memory extension, in order to increase the system's total accessible memory beyond its hardware capabilities.

Finally, the inherent need of FL for concurrent management of multiple clients inevitably leads to the use of threads in Python. The computation phase is synchronous such that all clients have to finish solving their local problems before entering the communication phase to transmit their updates to the server. A natural difficulty arising in the context of a multi-threading environment, is race conditions, i.e. multiple clients accessing and updating the same parameters. Since in our case there are some global variables that are accessed concurrently by all clients, the use of locking mechanisms when updating them was necessary so as to guard against simultaneous access.

B. Communication setup

In a communication round of *FedAvg*, the server and the clients send and receive updated model parameters. We developed a TCP-based socket interface for the client-server connection. TCP was preferred to UDP because of guaranteed data delivery and packet retransmission. To establish a connection, a desktop computer equipped with one Python program (we refer to it as *server.py*) takes the role of the server, knowing in advance the number of clients that are going to be connected. Five RPis participate in the procedure as clients, with a *client.py* Python program respectively. The clients need to know in advance the IP address and the port of the server they are going to connect to.

Remark: The *recv()* function of Python requires that we specify in advance the number of bytes we are going to receive. Since we are exchanging multiple files of various sizes, it would be a waste of bandwidth to naively provision and set a very large constant number of bytes for the whole FL process. Instead, we prefix the data we are going to send with a "header" of 4 bytes, which contains the size of the actual message. Then, the programs on the clients and the server first read the 'header' indicating the number of bytes to expect and then appropriately set the argument of *recv()* to that number.

C. Testbed Deployment and Execution

Once the respective programs on both sides are initialized, the following steps take place:

1) **Connection establishment**: The server starts listening for a prespecified number of connections. Clients in turn, load their local datasets and connect to the server, using its IP address and the Port number, which are known to them in advance.
2) **Training instructions broadcast**: For every connected client, the server receives the size of its local dataset and

[1]https://pytorch.org/

sends back all the hyperparameters required for training, e.g., the number of global rounds and the number of local epochs.

3) **Clients computation**: At each communication round, a fraction C of clients are selected for participation. Each selected client locally computes an update to the model by executing the training instructions, following for example the update rule of Stochastic Gradient Descent.
4) **Clients broadcast**: Clients send their updated local models to the server.
5) **Server aggregation**: Once all the clients in the current round have sent their local model updates, the server updates the shared model based on the aggregated update computed with the *FedAvg* algorithm.
6) **Server broadcast**: The server sends the updated central model to all the participating clients.

Steps 3 through 6 are repeated until a halting criterion is met, for example until central model convergence or the maximum number of communication rounds is reached.

D. Implementation details

Dataset split and distribution: In this work we evaluate our implementations on the CIFAR-10 dataset [12]. To study the FL optimization, we also need to specify how the data are distributed over the clients. We study two ways of partitioning the dataset in terms of the classes (labels): (*i*) i.i.d, where all the training data are simply shuffled and partitioned into 5 parts and are then distributed to the clients; (*ii*) non-i.i.d, where we first sort the data by class and assign training data from $c = 2$ or $c = 8$ classes to each client. For instance, when $c = 2$ each client will only have training examples of two classes out of the ten available. Thus, this lets us explore the efficiency of our methods on highly non-i.i.d data. Both of these partitions are balanced in terms of size, however. In all cases, there is no overlap in the training data between clients, and both the server and clients have the same test dataset to evaluate on, which is the default test partition the dataset comes with.

ML model: For our experiments we used a simple Convolutional Neural Network (CNN) model which consists of one 5x5 convolution layer with 6 channels, followed by batch normalization and a 4x4 max-pooling, and then 3 fully connected layers with 216, 120 and 64 units respectively, with Leaky ReLu activation function and Dropout. The final layer of the network is a softmax layer that outputs a probability distribution over the 10 classes of the dataset. We use as local solver for all clients, the Stochastic Gradient Descent (SGD) optimizer, with a learning rate 0.001 and momentum 0.9, which we keep fixed throughout all our experiments. As a loss function, we use cross-entropy loss.

IV. Results

In this section, we evaluate the impact of the following parameters on the performance of our implementation of the *FedAvg* algorithm: (*i*) Communication and computation time at the clients; (*ii*) Data distribution heterogeneity across clients; (*iii*) The fraction, C of clients selected for training at each communication round; (*iv*) Mobility of clients.

TABLE I: Network statistics that affect the communication time at each global round of the FL training procedure.

Distance from AP	Bandwidth	Signal strength	Link speed
1 *m*	36,3 Mbits/sec	-45 dBm	130 Mbits/sec
12 *m*	30,9 Mbits/sec	-51 dBm	110 Mbits/sec
30 *m*	18,7 Mbits/sec	-70 dBm	52 Mbits/sec

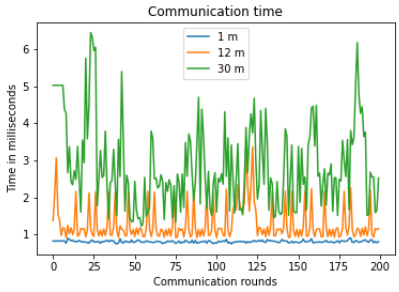

Fig. 2: Communication time of the 5 RPis vs. distance, for distances of 1 *m*, 12 *m* and 30 *m* from the AP.

A. Effect of Communication and Computation time

Similar to other ML schemes, one of the most critical performance metrics of FL is the time required to converge to a predefined accuracy level. However, different from conventional ML approaches, the time required for a round of FL training includes not only the computation time, but also the communication time of all participating devices. Here, we define as *communication time*, the time required for a RPi to send the local model to the server. Since all training parameters are transmitted over wireless links, the quality of training will be affected by factors such as bandwidth and latency. To capture the effects induced by communication, we first placed 5 RPis in three different distances from an AP and measured the network statistics that affect the FL procedure. Specifically, we used the Wi-Fi analyzer[1] application to measure the signal strength and link speed of the Wi-Fi connection. For the TCP throughput, we used the iperf3[2] program with the RPis as clients and the desktop PC as the server. In all cases, we took the average over 5 measurements from all RPis. The results of the measurements at the different client-server distances of 1*m*, 12*m* and 30*m* can be seen in Table I. As the distance from the AP increases, there is a noticeable decrease in all the networks measurements resulting in a considerable increase in the communication time as well. Specifically, on average, we observe a 60% increase in communication time going from close to the AP to 12 *m* distance, and a 120% increase at 30 *m* distance from the AP (Fig. 2). Such observations are important in larger-scale FL systems where the different client-server distance plays a role in performance. Another important factor to consider is the mobility of the clients. Any IoT

[1]https://play.google.com/store/apps/details?id=abdelrahman.wifianalyzerpro&hl=en&gl=US

[2]https://iperf.fr/

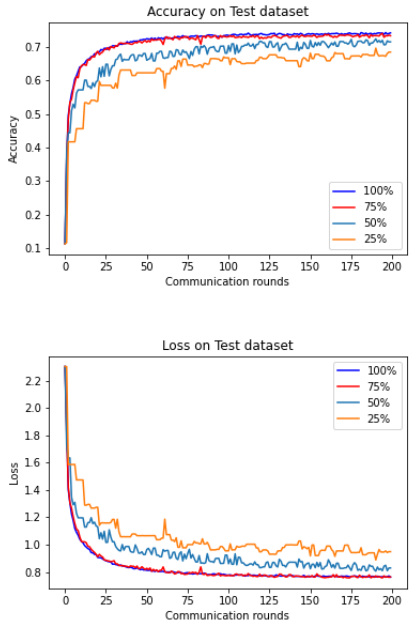

Fig. 3: The performance of the global model in terms of *(top graph)* test accuracy and *(bottom graph)* loss, when clients participate with a probability of $p = 25\%, 50\%, 75\%\, and\, 100\%$ at each communication round.

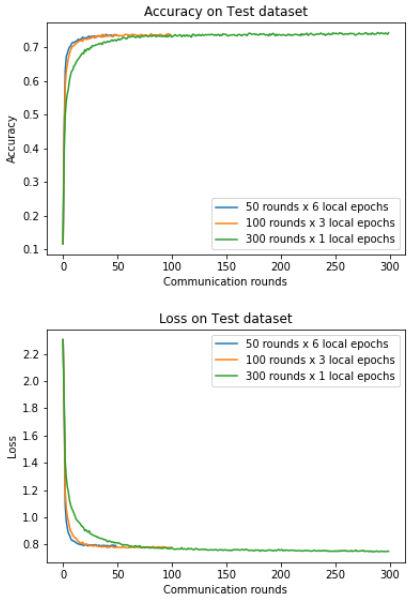

Fig. 4: The effect of number of local training epochs E of the clients *(top graph)* on the test accuracy and *(bottom graph)* loss of the global model at each communication round.

participant may disconnect from the network in the middle of the training phase or during the interaction with the server e.g., due to wireless channel quality fluctuation, hindering as a result the global model's learning performance. Besides, maintaining a constant connection with the server might be expensive or even infeasible. Such situations that consider both client mobility and bandwidth issues have not yet been explored [13].

To emulate these conditions, we implemented the standard *FedAvg* algorithm on our testbed and allowed each client to participate in a communication round by following a *Bernoulli* probability distribution with four different values of p, $p = 25\%, 50\%, 75\%\, 100\%$; with $p = 100\%$ being the baseline case of all clients participating at every round (Fig. 3). When a client participates in a communication round, it is able to send its local model update to the server and receive the resulted global model, while on the contrary if a client does not participate in a communication round, it can only train on its local data based on a stale global model. We observe that even with a participation rate of $p = 75\%$ per client, the global model is able to converge without any hampering in learning performance. On the other hand, when $p = 50\%$ or 25%, there is a significant drop in learning performance, as well as many oscillations (especially in the latter case) in the plot, showcasing a difficulty in convergence.

Finally, one of the key questions regarding the deployment of FL is whether we should spend *more time on computation and less on communication updates to achieve high learning accuracy, or vice versa*. This is an important issue, given that different devices have different computation capacity and communication bandwidth. We study three scenarios where the clients train for E = 1, 3 or 6 local epochs before sending the results to the server for aggregation and we present the results in Figure 4.

This scenario explores the tradeoff between computational and communication cost. Each local update consumes computational resources of the edge device, while each communication round consumes communication resources of the network. For a fair comparison of the 3 cases, we reduced the communication rounds as we increased the local epochs, to ensure that each client will perform 300 steps of SGD on its local training data in total. The results showcase that training for more epochs locally, increases the learning performance early on, with noticeably less communication overhead.

Specifically, at global communication round 30, the test accuracy for E = 1, 3 and 6 epochs is 70.49%, 73.72% and 73.77% respectively. This inevitably comes with a computational cost, as each training round in our testbed required on average 215 seconds to complete. Thus, if we set the training time T = (number of communication rounds) $\times$ (number of local rounds)$\times$215sec, then in order to reach the 30th communication round, it would require 107, 322 and 645 minutes for the cases E = 1, 3 and 6 respectively.

B. *Effect of data distribution heterogeneity*

In the FL setting, not all clients might have a representative sample from the overall distribution of data, i.e., they might not have data from all classes. We created a scenario for a non-i.i.d data distribution in the case of missing labels, by distributing c = 2, 8 or 10 classes (labels) to each client, in equal volumes, and we compared the performance of the global model in these 3 cases (Fig. 5). In the cases c = 8 and

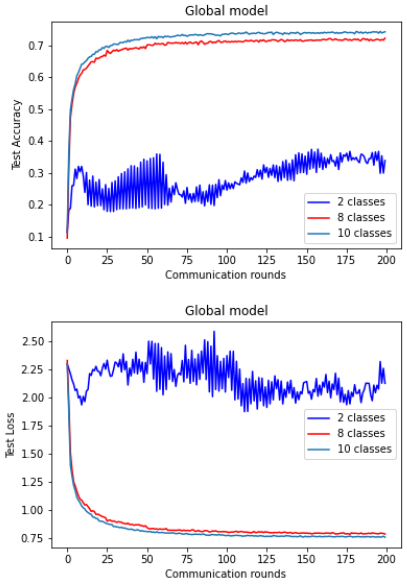

Fig. 5: The performance of the global model in terms of test accuracy *(top graph)* and loss *(bottom graph)* for different distribution of classes in each client's training dataset.

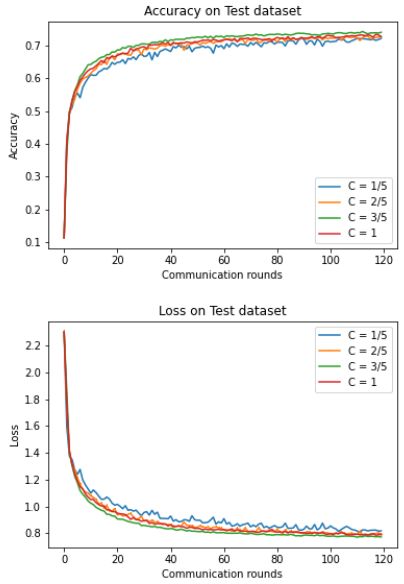

Fig. 6: The effect of the fraction of clients participating in each communication round for the i.i.d case.

10, the global model manages to generalize, and it achieves an accuracy of 72.29% and 74.73% respectively. The extreme case of non-i.i.d distribution of data, where each client has only 2 out of the 10 classes available for training, does not allow the global model to generalize to unseen data, and it reaches a maximum test accuracy of only 37.39%. The same observations hold for the test loss.

C. *Effect of number of users selected in each training round*

In real-world scenarios, it is unrealistic to expect that all clients participate in the FL procedure, since some may not be always active. For this reason, we randomly select a fraction C = 1/5, 2/5, 3/5 and 5/5 of clients (i.e., 1, 2, 3, or 5 RPis) to participate at each communication round, and perform FL training for two different cases of data distributions, namely i.i.d i.e., c = 10 (Fig. 6) and non-i.i.d i.e., c = 2 (Fig. 7). Clients selected at each round exchange model updates with the server, while the rest remain idle or turned off.

For the i.i.d case, the fractions C = 1/5, 2/5, 3/5 and 5/5 achieve test accuracy of 72.22%, 74.51%, 74.94% and 74.73% and test loss of 0.825, 0.816, 0.801 and 0.807 respectively. The results showcase a relatively similar performance in test accuracy and loss for C = 2/5, 3/5 and 5/5. Thus, the case of C = 3/5 is the optimal one, since it achieves the best accuracy and loss with less communication overhead compared to the other cases. On the contrary, for C = 1/5, the performance is much worse. When examining the non-i.i.d case with fractions of C = 1/5, 3/5 and 5/5, we observe a significant drop in learning performance when client participation is low, i.e. when only one RPi participates at each round, which inevitably mandates high participation. For the non-i.i.d case, we observe the expected fluctuations in accuracy and loss as well.

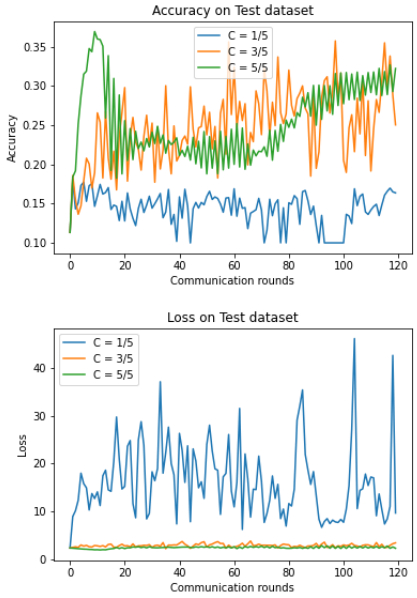

Fig. 7: The effect of the fraction of clients participating in each communication round for the non-i.i.d case.

D. *Impact of departing clients*

A critical aspect of FL training we should also pay attention to, especially in an IoT environment, is when clients that initially participate in the training process, disconnect from the server permanently e.g., because they leave the area where the server resides. Most recent works consider that all FL participants maintain a continuous connection with the server, which is an unrealistic scenario in the real-world IoT environment [13]. In our experiments, we disconnect at communication round 59 (i.e. halfway through training) 1, 2 or 3 RPis, and we show the effects on the test loss and accuracy

Fig. 8: The performance of the global model in terms of test accuracy *(top graph)* and loss *(bottom graph)*, when 1,2 or 3 clients disconnect at training round 59.

of the global model in Figure 8.

The results indicate a noticeable drop in performance, which is proportional to the number of devices that disconnect. Specifically, when we disconnect 1, 2, or 3 devices, the test accuracy of the global model drops by about 5%, 6% and 7% respectively, indicating a linear dependence of the reduction in terms of number of disconnected devices (about 1% for each additional device disconnected). The test loss in accordance, increases by 0.12, 0.19 and 0.27 respectively. Finally, while in the first two cases the learning performance of the global model shows an incline to stabilize, in the latter case, where more than half the clients disconnect from the server, it does not and keeps deteriorating.

E. Lessons learned from our experiments

The main takeaways from our experiments are as follows:

- In order to save communication costs, training with only a random subset of e.g. 3 out of 5 clients at each round seems to offer the best results in terms of both accuracy and loss, while reducing the communication overhead by about 40% compared to full client participation.
- When IoT devices fail to establish a connection with the server for at least 75% of the total communication rounds and train based on a stale global model, they significantly hinder its ability to converge.
- When more than half of the clients permanently disconnect during training, not only does the learning performance never recover, but it also continues to drop.
- When increasing the distance from the AP, the communication time (i.e. the time required for a client to exchange model updates with the server) is increased as well. However, the computational delay on resource-constrained IoT devices, that heavily affects the total time required for the FL procedure, since it is orders of magnitude greater than communication time.

V. Conclusions and Future Work

In this paper, we effectively evaluated our implementation of the *FedAvg* algorithm in various real-world scenarios and captured the negative effects induced by communication, which are ignored in simulation studies. Regarding the extensions of this work, we can explore the applicability of other FL solutions (e.g., compression techniques, model pruning etc.), as well as consider more resource-constrained devices, building on recent advances on *TinyML* paradigm [11] Energy consumption optimization is another under-explored aspect that can be considered. Finally, a large-scale deployment with more RPis, some of which could be mobile would help expose new challenges for a practical FL framework.

Acknowledgment

The research project was supported by the Hellenic Foundation for Research and Innovation (H.F.R.I.) under the "1st Call for H.F.R.I. Research Projects to support Faculty Members & Researchers and the Procurement of high-cost research equipment grant" (Project Number: HFRI-FM17-352, Project Title: Wireless Mobile Delay-Tolerant Network Analysis and Experimentation, Project acronym: LEMONADE).

References

[1] Y. Mao, C. You, J. Zhang, K. Huang and K. B. Letaief, "A Survey on Mobile Edge Computing: The Communication Perspective," *IEEE Communications Surveys & Tutorials*, 2017.

[2] B. McMahan, E. Moore, D. Ramage, S. Hampson and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," *Artificial intelligence and statistics*, PMLR, 2017.

[3] X. Li , K. Huang, W. Yang, S. Wang and Z. Zhang, "On the convergence of fedavg on non-iid data," 2019. [Online]. Available: arXiv:1907.02189.

[4] T. Li, A. K. Sahu, M. Zaheer, M. Sanjabi, A. Talwalkar and V. Smith "Federated Optimization in Heterogenous Networks," 2018. [Online]. Available: arXiv:1812.06127.

[5] T. Li, A. K. Sahu, A. Talwalkar and V. Smith, "Federated Learning: Challenges, Methods, and Future Directions," *IEEE Signal Processing Magazine*, vol. 37, no. 3, pp. 50-60, 2020.

[6] T. Nishio and R. Yonetani, "Client Selection for Federated Learning with Heterogeneous Resources in Mobile Edge," 2019. [Online]. Available: arXiv:1804.08333.

[7] Y. Gao, M. Kim, S. Abuadbba, Y. Kim, C. Thapa, K. Kim, S. A. Camtep, H. Kim and S. Nepal, "End-to-End Evaluation of Federated Learning and Split Learning for Internet of Things," *International Symposium on Reliable Distributed Systems (SRDS)*, 2020.

[8] S. Wang, T. Tuor, T. Salonidis, K. Leung, C. Makaya, T. He, and K. Chan, "Adaptive federated learning in resource constrained edge computing systems," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 6, pp. 1205-1221, 2019.

[9] P. Tam, S. Math, C. Nam and S. Kim, "Adaptive Resource Optimized Edge Federated Learning in Real-Time Image Sensing Classifications," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 14, 10929-10940, 2021.

[10] B. Barida, O. E. Taylor and C. L. Nwagbo, "A novel approach for federated machine learning using Raspberry Pi," *Global Journal of Engineering and Technology Advances*, vol. 6, no. 3, 2021.

[11] M. Lootus, K. Thakore, S. Leroux, G. Trooskens, A. Sharma, H. Ly, "A VM/Containerized Approach for Scaling TinyML Applications,", *https://arxiv.org/abs/2202.05057*.

[12] A. Krizhevsky,"Learning Multiple Layers of Features from Tiny Images," 2009.

[13] A. Imteaj, U. Thakker, S. Wang, J. Li,and M. H. Amini, "A survey on federated learning for resource-constrained iot devices," *IEEE Internet of Things Journal*, vol. 9, no. 1, pp. 1-24, 2021.