On the Reproducibility Challenges of Federated Learning: Investigating the Gap between Simulation, Emulation and Real-World Deployments

Cédric Prigent*, Kate Keahey†, Alexandru Costan*, Loïc Cudennec‡, Gabriel Antoniu*
* University of Rennes, Inria, CNRS, IRISA - Rennes, France
† Argonne National Laboratory, Lemont, Illinois, USA
‡ DGA Maîtrise de l'Information, Rennes, France
{cedric.prigent, alexandru.costan, gabriel.antoniu}@inria.fr
{keahey}@mcs.anl.gov, {loic.cudennec}@intradef.gouv.fr

***Abstract*—Federated Learning (FL) is an emerging paradigm for decentralized training of Machine Learning models. It has been the subject of a large corpus of research due to its innovative approach to handling sensitive data. A common practice in the FL literature is to run simulations on a single compute node to assess the performance of FL algorithms. While simulation enables fast prototyping and validation of algorithmic concepts, it may face limitations in reproducing the real system's performance in heterogeneous environments such as the Computing Continuum, and particularly on resource-constrained Edge devices. Conversely, emulation on distributed testbeds offers more effective means to accurately reproduce the performance of real-world devices. However, to the best of our knowledge, no prior research has investigated the differences between simulation and emulation in FL experiments. In this paper, we study the complementarity of these approaches and discuss their respective challenges, as a first step towards reproducibility of FL experiments. We illustrate our study with a real-life application used as a baseline: an outdoor air quality forecasting framework with real-world sensors. Our results show that simulation can be used to accurately reproduce *model* performance metrics, while emulation can effectively reproduce the *system* performance of real-world experiments. Finally, we present a set of lessons learned on the challenges of FL reproducibility and the selection of experimental infrastructures for FL experiments and applications.**

***Index Terms*—Federated Learning, Reproducibility, Edge**

I. Introduction

Federated Learning (FL) enables decentralized training of Machine Learning (ML) models over distributed nodes without sharing the training inputs of the federated peers [1]. This allows the processing of sensitive data directly at the Edge of the network, where the data is produced, in many domains (*e.g.*, healthcare [2], smartphone applications [3], [4]). Progressively larger ML models used in FL achieve unprecedented accuracy. To efficiently train such models, larger compute infrastructures spanning the entire Edge-Cloud Continuum [5] and involving increasing number of peers are used to accelerate and enhance the FL process. Extensive research efforts propose new algorithms and systems for improving model convergence, clustering, aggregation, training efficiency, and security of FL [6] on these infrastructures.

Reproducing the performance of such algorithms and systems is essential for modeling, optimisation and analysis, since the training efficiency (in terms of both accuracy and execution time) highly depends on the underlying FL infrastructure: *(i)* understanding the performance of an FL system aids in optimization (*e.g.*, reproducing the execution can help identify bottlenecks and allow for comparisons between different FL strategies); *(ii)* since implementing optimisation strategies in current FL frameworks is often tedious, error-prone, and resource-intensive, reproducing system and model performance allows to build accurate performance models that can reduce the effort and resources required for evaluation.

A common practice in the FL community has been to run simulations on a single compute node to assess the performance of FL algorithms [6]. The reasons why FL simulations are prevalent are mainly due to the high cost of setting up real FL systems. This cost arises from the complexity of deployments, the need for reliable and secure configurations, and hardware requirements. Although simulations enable fast prototyping and validation of algorithmic concepts, they face limitations for modeling and studying the system overhead at a fine-grained level [6]. This is particularly true for heterogeneous and resource-constrained devices such as those at the Edge of the continuum [7]. Surprisingly, there is significant variability in model performance and accuracy across different simulators, even when applied to the same topology [8]. Moreover, the usability and scalability of those strategies in real-life deployments (typically involving hundreds of data owners, dynamic network conditions, geographic distribution, data and hardware heterogeneity) have yet to be assessed.

We identify two key issues that hinder the reproducibility of FL experiments when moving applications from simulation to production in a distributed FL deployment. The first issue is **resource heterogeneity across the Computing Continuum**. A major source of the discrepancy between simulation and reality stems from the high variance and volatility of the underlying infrastructure. In practice, FL involves hundreds of heterogeneous embedded devices with non-IID (non-independent and identically distributed) data. This makes the

simulated performance—such as convergence time, bandwidth consumption, and model quality—less accurate compared to real or emulated deployments. While, some previous works have investigated the FL performance at Edge (*e.g.*, on Raspberry Pi*s*) [9]–[12], they did not quantify nor identified the hypothetical gap that could exist between simulation, emulation and real-world deployments for reproducibility purposes.

The second issue is **modeling and reproducing complex FL runtime behaviors**. At execution time, communication (*i.e.,* transfer of the model weights to the centralized server) may overlap with computation (*i.e.*, local training at the clients) to hide other stages of the FL processing that may compete for bandwidth resources (*e.g.*, client clustering). While these optimizations can enhance efficiency, they also introduce additional overhead, which: 1) may reduce the overall throughput of the FL model, and 2) is difficult to accurately predict through simulation.

To address these issues, we propose to investigate the gap between simulation, emulation and real-world deployments for FL performance evaluation. We present a rigorous methodology to evaluate key FL metrics (*i.e.*, model convergence, training latency, system utilization) and identify how different infrastructures can reproduce different experimental patterns observed in a real-world experiment. We deploy a real-world sensor network for air quality forecasting and follow our methodology to compare the real-world training performance against different experimental setups previously used in the FL literature. We investigate the extent to which different platforms can reproduce the results obtained using the real-world deployment. Our contributions are summarized as follows:

1) We introduce a rigorous methodology to evaluate the performance of FL applications and workflows on different experimental platforms.
2) We design an experiment template for FL deployments, illustrated by building and deploying air-quality station prototypes to measure and forecast ambient air pollution on the UChicago campus. Those stations are used to train PM2.5 forecasting models at the Edge using FL.
3) We follow the proposed methodology to compare the performance of simulation, emulation and our real-world prototypes. We investigate how different hardware resources can be used to reproduce the real-world experiments by deploying FL on GRID'5000 (data centers) and CHAMELEON's CHI@Edge (raspberry pi*s*) testbeds.
4) We discuss the lessons learned from our experiments (*e.g.*, which infrastructure can be used to validate which experimental patterns), and emphasize key reproducibility aspects for FL experiments.

II. BACKGROUND AND MOTIVATION

A. Federated learning workflows

Federated Learning (FL) is usually deployed on large cohorts of heterogeneous and unreliable systems. In FL, private data remain local to the federated peers. Instead of running the training in centralized data centers, the FL server samples a subset of clients in federated rounds for local training of the ML model. Sampled clients receive a copy of the model and apply local training using their private data followed by aggregation and model averaging by the server [1]. The FL optimization goal is to minimize the global objective function:

$$F(w) = \sum_{i=0}^{N} F_i(w) \tag{1}$$

where N is the total number of clients and $F_i(w)$ is the local objective function of client i,

Weight initialization and federated fine-tuning. Most prior works have considered random weight initialization for FL tasks [1], [13]. However, many FL applications could benefit from proxy data available at the server for pre-training purposes. The pre-trained model could then be further optimized through federated fine-tuning in order to meet the global optimization goal with better convergence [13].

B. Related works

Simulation/Emulation of FL systems Simulation has been largely adopted in previous FL research works [1], [6], [14]. Existing tools mainly rely on optimizations to accelerate computations and run large client cohorts on a single node with minimal overhead [15]–[17]. Emulation can be used to more realistically mimic different system behaviors by deploying the FL clients on different physical or virtualised nodes and applying network or computation limitation rules. Some tools come with specific emulation support based on real-world system traces of mobile devices (*e.g.*, client speed and availability) [16], [18], while others let the users apply custom network emulation rules to the different nodes [19].

Performance evaluation of FL at the Edge. Some previous works investigated the performance and overhead of FL on Edge devices. [9], [10] measure training time, communication overhead, power consumption and memory usage of FL on Raspberry Pi (RPI) devices, highlighting: 1) the limitations of such devices for training models exceeding several millions of parameters, and 2) the impact of communication operations on their power consumption. The impact of a cooling mechanism on RPI training performance is studied in [12]. [11] investigated FL performance on a RPI testbed when emulating a constrained 7.65MB/s network bandwidth (representative of 4G/5G wireless communication) and showed that communication time represented nearly half of the end-to-end training time.

C. Challenges of reproducibility

We adopt the terminology proposed by ACM [20] *i.e.*, an experiment is deemed reproducible, if a **different team using different experimental setup** is able to reliably repeat the experiment and obtain the same measurement with stated precision on multiple trials. In addition to the broader challenges of scientific computing reproducibility, FL faces specific challenges in terms of reproducibility:

(C1) *Data accessibility.* Experiments reproducibility is limited by the inability to retrieve training data, whereas

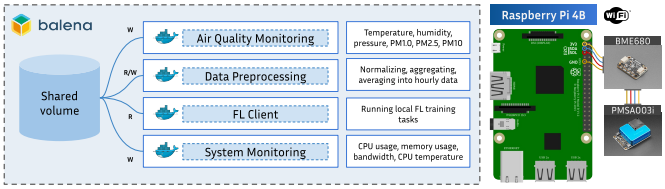


Fig. 1: Air quality station system architecture

statistical reproducibility can be limited by the lack of public and statistically similar datasets [21].

(C2) *Software randomness.* Random client selection approaches and random model initializations are commonly used in FL. While fixing the random seed can partially solve this problem, factors such as the order in which the clients connect to the federation or their volatility may affect the sampling approach in consecutive runs.

(C3) *System heterogeneity.* FL clients are usually heterogeneous and resource-constrained. In practice, they may also present very heterogeneous network access with volatility. Reproducing FL training on different experimental platforms might require system emulation and a fine-grained capture of federated client system metrics. In addition, the non-determinism in the hardware can add to the intricacies of reproducing training of Deep Learning models [22].

D. Problem statement

While FL has been the subject of numerous studies, it has been evaluated mainly through simulation. Few works have evaluated FL performance on Edge devices (*e.g.*, Raspberry Pi*s*). However, these evaluations typically rely on benchmarking datasets and do not include comparisons with simulation or emulation approaches. In this work, we aim at answering the following research questions:

(Q1) Can public datasets be used to reproduce FL results obtained from private data? (addressing challenge C1)

(Q2) How simulation results can predict the behavior of a real-world Federated Learning system? (addressing challenges C2-C3)

(Q3) How realistically can different experimental approaches model performance of Federated Learning at the Edge? (addressing challenge C3)

III. Case study: Outdoor Air Quality Forecasting

Air pollution is one of the greatest environmental risks for human health. According to the World Health Organization (WHO), in 2019, ambient air pollution was estimated to be responsible for a yearly 4.2 million premature death worldwide [23]. Fine particulate matter (PM) is one of the most prominent sources of risks for human health, due to their microscopic size. Some IoT sensor networks have been deployed over urban areas to measure the concentration of such pollutants and calculate air quality index [24], raising public awareness on air quality problems.

In order to observe the behavior of real-world devices in practice, we design our own air quality station prototypes and deploy them across the UChicago campus. We describe the complexity of this deployment in this section. We consider the significant effort and resources required for this process as representative of real-world FL deployments, highlighting the need to explore alternatives such as simulation and emulation.

Hardware. To measure outdoor air quality and enable in-situ data processing, we build our air-quality stations on top of Raspberry Pi 4B (RPI) single-board computers. RPI*s* come with general purpose I/O pins (GPIO) which can be used to connect the devices to external sensors. We extend our prototypes with two sensors connected through the I2C interface. BME680 sensors measure relative humidity, barometric pressure and ambient temperature, and PMSA003I sensors measure PM1.0, PM2.5 and PM10.0 which correspond to the concentration of particulate matter in the air. RPI 4B natively supports WiFi connectivity, allowing for greater flexibility regarding deployment locations. This architecture is simple and easily replicable. We conducted different tests on a first prototype running docker containers to ensure that the sensors worked as expected. After validation of the first prototype, we prepared 7 additional prototypes (for a total of 8 stations). Each device runs balenaOS, a minimal OS for running containers on embedded devices. We configure WiFi and ssh access directly in the balenaOS configuration files.

Micro-services. We build our system following a micro-services architecture, decoupling the different building blocks of our application into isolated services. Micro-services are bundled in docker images which contain source codes and dependencies needed to run the application, facilitating their deployment across various infrastructures. Docker containers share a persistent volume for writing and reading data, enabling data transfer between services (*i.e.*, raw and processed data). Figure 1 gives a schematic view of our air quality station system architecture. Our experimental workflow is divided into 4 main phases for which we define micro-services: (1) the air quality monitoring service which collects raw air quality data using the BME680 and PMSA003I sensors through the I2C interface, (2) the data preprocessing service which reads raw data from the shared volume and prepares the local dataset for the FL client, (3) the FL client service which trains a PM2.5 forecasting model in collaboration with other devices, and (4) the system monitoring service which measures local system metrics in real-time (*e.g.*, CPU usage, memory usage).

Deployment. We face several conflicting challenges for deploying our devices: (1) the sensors require direct access to outdoor air, (2) the prototypes need to be protected from outdoor weather, and (3) the prototypes require a powering solution. Keeping deployment cost low while addressing these constraints can be challenging. Although waterproof enclosures and batteries address the problem of outdoor deployments, this would also increase the total cost of our deployment (*i.e.*, equipping each device with waterproof cases and batteries). While relying on waterproof enclosures seems essential for outdoor deployments, setting up powering solutions other than

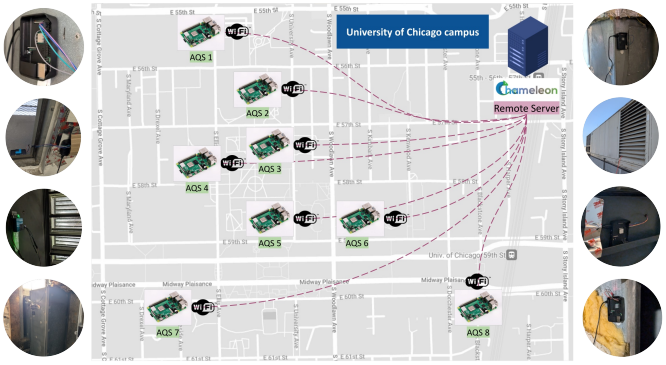


Fig. 2: Prototypes deployment on the UChicago campus.

batteries could be tricky. We also do not know to what extent different waterproofing solutions could affect the quality of our air quality measurements. An alternative solution is to target indoor deployment locations with direct access to outdoor air. With the help of technical staffs, we identified 8 fresh air intakes from the university buildings that could be used to deploy our devices. These fresh air intakes came with the benefit of having nearby access to powering solutions and protecting the devices from outdoor weather. A public WiFi available across the university campus provides easy connectivity for each device. Another advantage of deploying our prototypes in these fresh air intakes is that they are protected from pilfering and vandalism. The on-campus device deployment spanned over two days, with the mobilization of technical staffs to access several authorization-only areas. For each deployment, we verified the proper connection of devices to the WiFi (available across the university campus) and to the CHI@Edge [25] testbed (used to operate the devices). A map of the deployed air quality station prototypes is presented in Figure 2.

IV. Methodology and Experimental Plan

To answer our research questions, we deploy different FL training tasks (*i.e.*, different training settings and model architectures) and measure the training performance of each system using precise evaluation metrics addressing multiple optimization goals (*i.e.*, model performance, training round latency, and system utilization).

A. Training task

Models. Our objective is to train forecasting models that use time series data to predict ambient air quality (*i.e.*, PM2.5) for the next couple of hours. To study the impact of model complexity on the reproducibility of training performance, we train recurrent neural networks (RNNs) with different architectures. Each RNN is composed of LSTM layers followed by a fully connected output layer predicting PM2.5 concentrations for the next 4 hours. We describe our models in Table I.

Federated fine-tuning. We study the impact of two weight initialization approaches in FL: (1) random initialization (*i.e.*, training the model from scratch) and (2) centralized pre-training on public data (*i.e.*, federated fine-tuning).

TABLE I: LSTM Models

	Sequence length	Input size	LSTM layers	Hidden state size	Output size	Number of parameters	Size (MB)
1	24	(6)	1	64	(4)	18.4K	0.08
2	24	(6)	2	64	(4)	51.7K	0.21
3	24	(6)	1	128	(4)	69.6K	0.28
4	24	(6)	2	128	(4)	201.7K	0.81
5	24	(6)	1	256	(4)	270.3K	1.09
6	24	(6)	2	256	(4)	796.7K	3.19

B. Evaluation metrics

To conduct our study, we evaluate the experimental results along 3 performance axis: (1) model performance, (2) training round latency, and (3) system utilization.

In terms of **forecasting performance**, we report the federated MSE loss, which corresponds to the average MSE loss achieved by the model on the client local testsets. In addition, to evaluate the performance gain provided by federated fine-tuning, we report: (1) the loss improvement compared to FL from scratch, and (2) the number of rounds to achieve different target loss values.

In terms of **training round latency**, we report the average client computation time per round and average client communication time per round. We also present statistics about the client working time variations observed during consecutive training rounds (*i.e.*, heterogeneity statistics between clients).

In terms of **system utilization**, we report CPU usage, CPU temperature, memory usage and total network I/O observed across the different platforms when running our application.

C. Hyperparameters and model pre-training

FL hyperparameters. We run each FL experiment for 100 training rounds using 3 clients and 1 server. We set the client sampling rate to 1.0 (*i.e.*, every client participate in every round). When sampled, clients train the downloaded model for 1 local epoch over their training samples. We set a learning rate decay of 0.99 for training stability. For reproducibility purposes, we fix the random seed for all experiments. Finally, we conduct a hyperparameter search through simulation to fix the local learning rate in the range $\{0.01, 0.003, 0.001, 0.0003, 0.0001, 0.00003\}$. We set the learning rate to $1e-3$ for training the model from scratch and $3e-4$ for federated fine-tuning of the model.

(Simulation/Emulation) Public dataset. In practice, FL is deployed to train models on data that could not be retrieved in centralized storage due to privacy or communication constraints. This hinders the ability to fully reproduce FL experiments using experimental testbeds. For this reason, we study how public data could be used to reproduce experiments relying on private data. We rely on the PURPLEAIR [24] sensor network API to download public air quality data from sensors deployed across Chicago that should be conceptually similar to the data collected at our real-world air-quality stations. We purposely download one month of data for each sensor to match the real-world dataset sizes and verify that the same number of samples is used in every experiment.

Model pre-training. We assume the existence of a publicly available and limited dataset that could be used to pre-

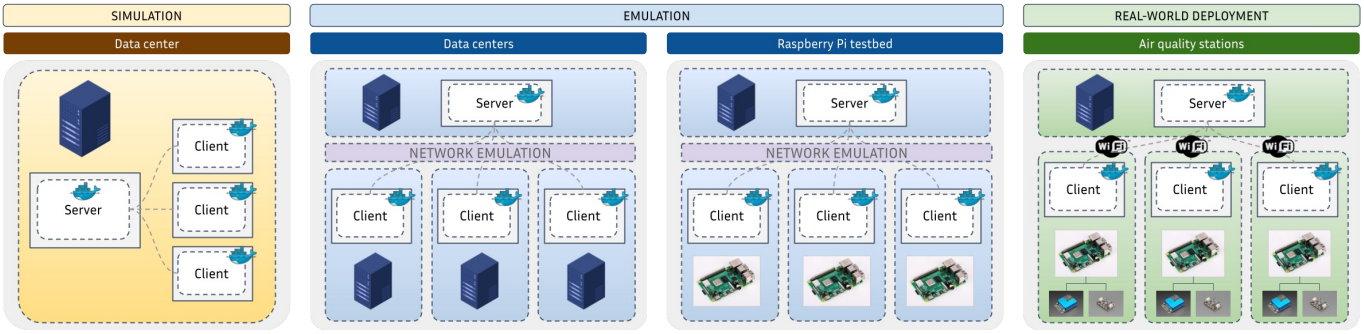


Fig. 3: Experimental platforms

train our forecasting models. To study the impact of such datasets on model convergence, we assume the availability of previous sensor data from Sydney and collect them using PURPLEAIR [24] API. We download one month of data coming from a sensor located in Sydney and use them to pre-train our forecasting models using centralized computing resources.

V. Experimental Platforms

Our objective is to understand how different platforms can be used to reproduce the performance of a real-world deployment. In this section we present the different platforms and settings considered in our study, which are selected based on their previous adoption in the FL literature (Figure 3).

A. Real-world deployment

We use our real-world air quality station prototypes described in section III to run the FL clients for our experiments. The FL server is deployed on a compute node from the CHAMELEON [26] testbed equipped with 2×Intel(R) Xeon(R) Gold 6240R CPU @ 2.40GHz, 192GB of memory and 10Gbps ethernet link. We use these experiments as baseline and set their performance metrics as the reproducibility objective for the other platforms.

B. Distributed testbed emulation

Using emulation, each federated peer is isolated on a different node where network and compute limitation rules may apply in order to replicate the real-world system performance. Since our objective is to reproduce FL performance, we only deploy 2 micro-services: (1) the FL client service (in order to reproduce FL training), and (2) the system monitoring service (for performance comparison purposes).

We consider two popular settings previously used for distributed emulations:

1) *Data centers deployment.* We use 4 nodes from the GRID'5000 experimental testbed. Each node is equipped with 2×Intel Xeon E5-2630 CPU @ 2.40GHz (8 cores), 128GB of memory and 10Gbps ethernet link. We use one node to deploy the FL server and 3 nodes to deploy the FL clients. In addition, we use the E2CLAB [19] deployment framework to configure the different nodes. We fix the data rate between the FL server and FL clients to 1Gbps.
2) *Raspberry Pis deployment.* We use the CHAMELEON [26] experimental testbed to deploy our experiments. We deploy the FL server on a compute node equipped with 2×Intel(R) Xeon(R) Gold 6240R CPU @ 2.40GHz, 192GB of memory and 10GBps ethernet link. FL clients are deployed on 3 Raspberry Pi 4B devices from CHAMELEON's CHI@Edge [25] testbed. They share the same hardware as the real-world prototypes (Quad core Cortex-A72@1.80GHz and 8GB of memory).

C. Simulation

We consider a last experimental setting which consists in simulation on a single compute node. This setting has been largely adopted in previous works to evaluate performance of FL systems [1], [6], [15], [16]. In this setting, the entire FL system (*i.e.*, clients and server) is deployed on a single node. As with the emulation experiments, we only deploy the FL client and system monitoring services. The difference with emulation is that all federated peers, including the server, are deployed and run concurrently on the same node.

We use the GRID'5000 [27] experimental testbed to simulate FL experiments. We use a single compute node equipped with 2×Intel Xeon E5-2630 CPU @ 2.40GHz (8 cores), 128GB of memory and 10Gbps ethernet link, and deploy the entire FL system (*i.e.*, server and clients) on it.

VI. Results

Our application is bundled in docker images, the source code and the experimental artifacts are publicly available [28] to facilitate the deployment and reproducibility of experiments across diverse infrastructures.

A. Model performance

In this section, we present the performance achieved by the different LSTM models when using two weight initialization approaches: random initialization and centralized pre-training.

Performance gain using centralized pre-training. We present model convergence metrics achieved in each scenario in Table II. We specifically focus on the results obtained in

TABLE II: Model convergence metrics (Reproducing experiments with **public data**)

	Random initialization				Model pre-training				Loss improvement			
	Sim	**DC**	**RPIs**	**RW**	**Sim**	**DC**	**RPIs**	**RW**	**Sim**	**DC**	**RPIs**	**RW**
Scenario	**Federated MSE Loss**											
1	2.60e-3	2.58e-3	2.63e-3	3.25e-3	2.50e-3	2.47e-3	2.52e-3	2.97e-3	-0.10e-3	-0.11e-3	-0.11e-3	-0.29e-3
2	2.75e-3	2.77e-3	2.68e-3	3.25e-3	2.35e-3	2.35e-3	2.28e-3	2.83e-3	-0.41e-3	-0.41e-3	-0.40e-3	-0.42e-3
3	2.53e-3	2.66e-3	2.56e-3	3.23e-3	2.32e-3	2.30e-3	2.32e-3	2.76e-3	-0.21e-3	-0.37e-3	-0.24e-3	-0.47e-3
4	2.46e-3	2.61e-3	2.60e-3	2.99e-3	2.31e-3	2.33e-3	2.37e-3	2.78e-3	-0.15e-3	-0.29e-3	-0.23e-3	-0.21e-3
5	2.36e-3	2.39e-3	2.44e-3	2.89e-3	2.40e-3	2.35e-3	2.40e-3	2.77e-3	+0.05e-3	-0.03e-3	-0.04e-3	-0.12e-3
6	2.43e-3	2.44e-3	2.30e-3	2.99e-3	2.41e-3	2.52e-3	2.42e-3	2.92e-3	-0.02e-3	+0.08e-3	+0.12e-3	-0.07e-3
Scenario	**Rounds to target loss**											
1	49	49	45	50	43	51	47	33	-6	2	2	-17
2	37	46	50	50	13	15	14	1	-24	-31	-36	-49
3	47	49	39	50	15	14	16	11	-32	-35	-23	-39
4	47	49	40	50	14	11	13	8	-33	-38	-27	-42
5	46	42	47	50	10	9	9	11	-36	-33	-38	-39
6	50	50	42	50	8	10	7	6	-42	-40	-35	-44

Sim (simulation), RPIs (Raspberry Pi testbed), DC (Data-centers testbed), RW (Real-world deployment)

TABLE III: Model convergence metrics (Reproducing experiments with **private data**)

	Random initialization			Model pre-training			Loss improvement		
	Sim	**RW**	**RSD**	**Sim**	**RW**	**RSD**	**Sim**	**RW**	**RSD**
Scenario	**Federated MSE loss**								
1	3.26e-3	3.25e-3	0.1%	2.95e-3	2.97e-3	0.4%	-0.31e-3	-0.29e-3	-4.9%
2	3.24e-3	3.25e-3	0.1%	2.79e-3	2.83e-3	1.4%	-0.46e-3	-0.42e-3	-6.0%
3	3.13e-3	3.23e-3	2.2%	2.81e-3	2.76e-3	1.2%	-0.32e-3	-0.47e-3	-26.4%
4	3.01e-3	2.99e-3	0.4%	2.78e-3	2.78e-3	0.0%	-0.23e-3	-0.21e-3	-5.6%
5	2.89e-3	2.89e-3	0.1%	2.76e-3	2.77e-3	0.3%	-0.13e-3	-0.12e-3	-3.1%
6	2.90e-3	2.99e-3	2.1%	2.95e-3	2.92e-3	0.7%	+0.05e-3	-0.07e-3	NA
	Rounds to target loss								
1	48	50	2.9%	31	33	4.4%	-17	-17	-0.0%
2	50	50	0.0%	1	1	0.0%	-49	-49	-0.0%
3	45	50	7.4%	10	11	6.7%	-35	-39	-7.6%
4	48	50	2.9%	8	8	0.0%	-40	-42	-3.4%
5	50	50	0.0%	10	11	6.7%	-40	-39	-1.8%
6	46	50	5.9%	6	6	0.0%	-40	-44	-6.7%

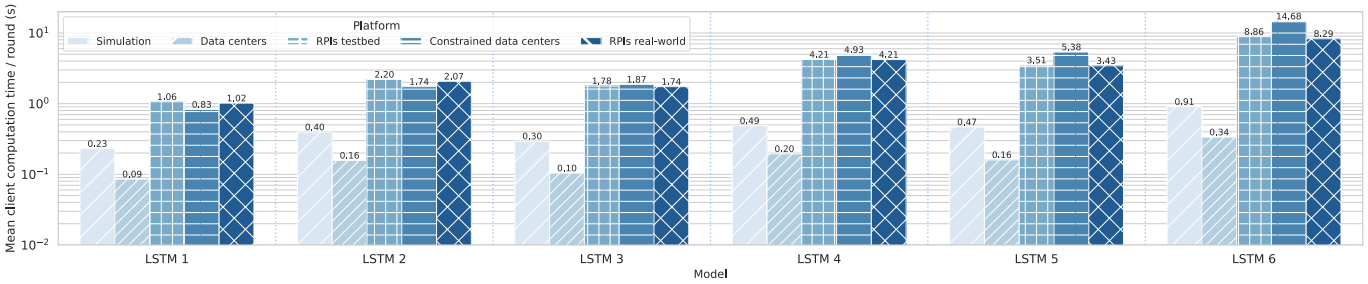
Sim (simulation), RPIs (Raspberry Pi testbed), DC (Data-centers testbed), RW (Real-world deployment)

the real-world deployment. Overall, we observe a performance gain in every scenario when using the centralized model pre-training approach: model convergence is improved and the final loss values are reduced (between -0.07×10^{-3} and -0.47×10^{-3} loss improvement across scenarios). Model pre-training coupled with federated fine-tuning can achieve high model convergence speed by reducing the total number of rounds needed to achieve a target loss by a range of $[17, 49]$ rounds, a significant speed-up resulting in substantial resource savings for constrained Edge devices (the target loss is fixed for each platform based on the performance of the random initialization approach).

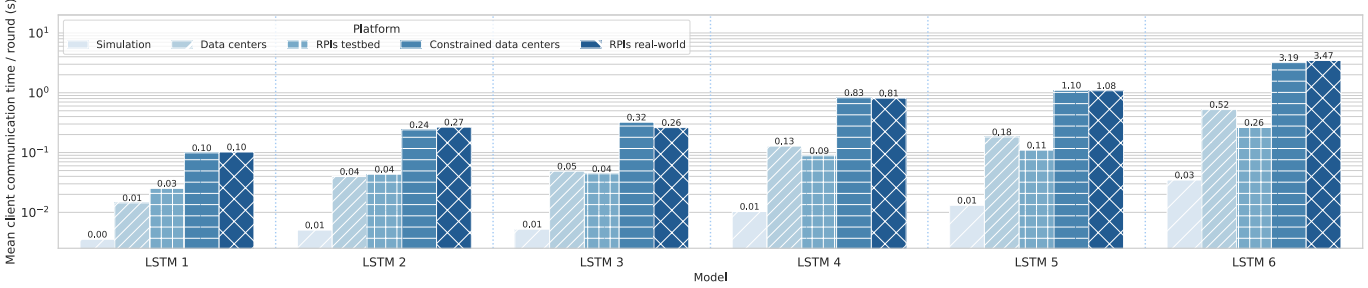
Reproducing experiments with public data. In practice, data used during FL training is private, hindering their retrieval for reproducibility purposes. Therefore, we investigate how public data could replace private data to reproduce real-world experiments. Simulation and emulation results when using public data are presented in Table II. A first discrepancy between the simulation/emulation and real-world deployment is the gap between their achieved MSE loss. We notice a non-negligible average loss difference of 0.56×10^{-3} (corresponding to 14% relative standard deviation) across scenarios. We observe a second discrepancy regarding the number of training rounds saved by using federated fine-tuning. While the real-world and simulation/emulation experiments present similar results in terms of convergence speed for scenarios $2-6$, experiments from the first scenario exhibit higher divergences. The simulation/emulation present almost no benefit from using federated fine-tuning whereas the real-world deployment saves 17 training rounds.

One possible reason for these performance variations is the variability in statistical distributions between the private and public datasets used for the real-world and simulation/emulation experiments. The real-world (*i.e.*, private) data might be presenting more challenging tasks to forecast, specifically in the testing set, in comparison to the public data, resulting in higher testing loss. Despite discrepancies in the final loss values, we observe similar outcomes regarding the improved convergence speed of the model between the real-world and simulated/emulated experiments.

Reproducing experiments with private data. Results of simulation when reproducing the experiments using the real (in practice, private) data are presented in Table III. We present the relative standard deviation (*i.e.*, RSD) measuring the variation between the real-world and simulation results. Regarding MSE loss achieved across scenarios, we observe accurate reproducibility of experiments with very small RSDs (*i.e.*, 0.1% to 2.2%). However, we observe higher variations in terms of loss improvement (*e.g.*, up to -26.4%) which is the consequence of small variations in the loss achieved by the different weight initialization approaches. For instance, in scenario 3, small variations in the final loss led to improved performance with random initialization but decreased performance with model pre-training compared to the real-world setting. This resulted in reduced performance gains and higher RSD. Regarding training efficiency (*i.e.*, rounds to target loss),



(a) Average computation time per round



(b) Average communication time per round

Fig. 4: Client working time

we observe that simulation can accurately reproduce the real-world experiments results.

Discussion: *Reproducing FL model convergence*

In this set of experiments we investigated the reproducibility of model performance using public and private data. Unsurprisingly, the use of real (*i.e.*, private) data resulted in accurate reproducibility of model convergence regardless of the platform, showing only slight variations in the model convergence results. However, in practice the reproducibility of FL experiments with private data is not feasible. Therefore we also studied how public data could be used for reproducibility. We showed that some divergences could arise due to the client data distributions (*e.g.*, resulting in loss variations). Nevertheless, the same conclusions could be drawn regarding the efficiency of federated fine-tuning through its conceptual validation.

B. Training round latency

Client local training time. We present the average computation time of federated clients in Figure 4a. We observe that Raspberry Pi*s* from the experimental testbed achieve approximately the same computation time as the real-world prototypes in every scenario. This is an expected result as RPI*s* from the experimental testbed share the same hardware as the real-world prototypes. In turn, the data centers simulation/emulation fail to reproduce the client computation times. The simulation takes as little as $11\% - 22\%$ of the time required by the real-world devices to run the local FL computations. The data centers emulated devices show even greater discrepancies, requiring only $4\% - 9\%$ of the computation time needed by the real-world devices. Note that while computation time increases with model complexity across all platforms, the rates of increase vary significantly. For instance, there is an $8.1\times$ increase between the first and last scenarios for the real-world devices, while the data center emulated devices show a $3.7\times$ increase and the simulation exhibits a $3.9\times$ increase.

Client communication time. We present the average client communication time in Figure 4b. Despite a noticeable communication time increase in consecutive scenarios across all platforms, the simulation/emulation experiments result in communication times that differ by orders of magnitude compared to the real-world deployment. On average, the simulation achieves communication times that are two orders of magnitude smaller than those of the real-world deployment ($25\times$ to $99\times$ difference) while the emulation results in a one order of magnitude decrease ($4\times$ to $13\times$ difference).

Emulating resource constraints in data centers. We investigate how emulating resource constraints can help reproduce real-world experiments in a data center setting. Based on the results obtained in the previous experiments, we set a network limitation between the FL server and the clients, with a data rate of 12.5MB/s. In addition, in order to mimic the computation capabilities of RPI*s*, we limit the CPU usage to 0.1 CPU core per client. We observe that by setting CPU limitations, one can reach computation times that are closer to the real-world computation times. However, while succeeding in reproducing the real-world computation times for scenarios $1 - 4$ ($0.81\times$ to $1.17\times$ of the RPI computation times), the limitation on computing resources becomes too restrictive for the more complex models encountered in scenarios 5 and 6, leading to $1.57\times$ and $1.77\times$ longer computation times compared to the RPI. Regarding communication times, the data rate limits set for these experiments result in close

TABLE IV: Client working time variations across training rounds for the last evaluation scenario

	Absolute working times (s)			Standard deviation (s)			Relative standard deviation		
Platform	**min**	**max**	**mean**	**min**	**max**	**mean**	**min**	**max**	**mean**
Simulation	0.4	1.8	0.9	0.1	0.8	0.3	5.4%	89.6%	29.6%
Data centers	0.5	1.4	0.9	0.0	0.7	0.2	2.4%	102.2%	21.3%
DC constrained	16.4	19.9	17.9	0.2	1.7	0.8	0.9%	9.3%	4.6%
RPI testbed	8.3	10.8	9.1	0.1	1.2	0.5	1.1%	12.5%	5.1%
RPI real-world	9.3	19.2	11.8	0.8	4.3	2.0	6.8%	30.6%	16.8%

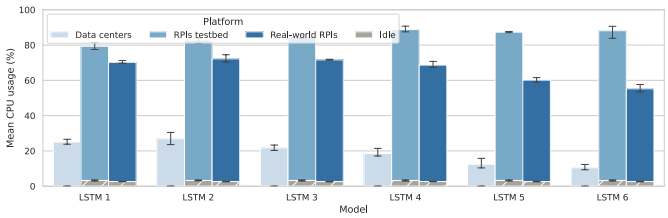


Fig. 5: CPU usage

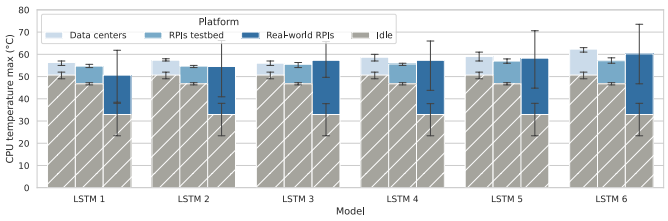


Fig. 6: CPU temperature

reproduction of communication times across all scenarios ($0.89\times$ to $1.23\times$ the RPI communication time).

Training time variance. We present statistics on the variations in client working time for the last training scenario in Table IV. Overall, we observe high heterogeneity in the data centers simulation/emulation (up to 102.2% RSD). However, this only represents slight absolute variations (0.3s average standard deviation) compared to the real-world devices, which experience an average standard deviation of 2.0 seconds (the highest absolute variations), and an average 16.8% RSD. The constrained data centers and RPI testbed devices, which present similar end-to-end training times to the real-world devices, fail to reproduce the heterogeneity pattern, exhibiting homogeneous end-to-end training times ($4.6\% - 5.1\%$ RSD).

Discussion: *Reproducing FL training latency*

In this set of experiments we observed that simulation/emulation on data centers without proper configuration (*i.e.*, setting computing resources and network limitations) cannot reproduce the performance of the real-world prototypes. In turn, RPI testbeds can be used to reproduce more faithfully the local training times of FL clients. However, applying additional networking rules is necessary to reproduce the real-world communication performance. Finally, none of the platforms reproduced the level of heterogeneity of the real-world devices. Exploring more advanced emulation with finer granularity would be required to match the real-world device heterogeneity.

C. System utilization

Mean CPU usage. Figure 5 presents the average CPU usage of the devices during FL training. Although similar CPU usage was expected for the real-world devices and RPI testbed devices, in practice we observe significant differences in their utilization. The CPU usage of the real-world devices drops as the model complexity increases, whereas the RPI testbed devices see their CPU usage increase with model complexity. This discrepancy is explained by the total communication times observed in Figure 4b. The real-world devices exhibit high communication times compared to the testbed devices, which translates into higher communication to computation ratio (and CPU usage). We observe a similar trend for the data centers, with an average CPU usage decreasing with model complexity (*i.e.*, higher communication to computation ratio).

CPU temperature. Figure 6 presents the CPU temperature of the different platforms during FL training. We observe high variability across platforms in terms of CPU idle temperature. The real-world prototypes present heterogeneous CPU temperature in idle state ($23°C$ to $38°C$), whereas the testbed devices present homogeneous but higher CPU temperature (ranging from $49°C$ to $52°C$ for data centers and $46°C$ to $47°C$ for RPIs). This can be explained by the different locations where the devices are deployed. Regarding CPU temperature during FL training, the real-world devices experience high temperature increase (up to $35°C$), which differs from the testbed devices presenting relatively stable temperatures, likely due to the cooling solutions typically present in testbeds.

Memory usage. Figure 7 presents the memory usage of the application when running on the different platforms. We report the memory usage increase when switching from the devices idle state to the FL training phase. Overall, the different platforms reach similar memory usage across scenarios. Note that, while the data centers emulation seems to present lower memory usage variations across scenarios, this might be due to our measurement procedure. At the device level, we monitored relative memory usage and then converted it to absolute memory usage, which may have caused some loss of precision.

Network I/O. We measure the total uploads and downloads of the federated clients and present the results in Table V. While network I/O increases across scenarios, we observe no discrepancy between experimental platforms. Note that client downloads are twice as high as uploads, as federated evaluation only requires clients to download the model.

TABLE V: Total network I/O

	Downloads (MB)						Uploads (MB)					
	S1	**S2**	**S3**	**S4**	**S5**	**S6**	**S1**	**S2**	**S3**	**S4**	**S5**	**S6**
Data centers	16	44	59	170	228	668	8	22	30	86	116	340
RPI testbed	16	46	60	172	230	676	9	24	31	88	119	348
DC constrained	16	45	60	171	229	673	8	22	30	86	116	340
Real-world	16	45	60	172	231	678	8	23	30	87	116	341

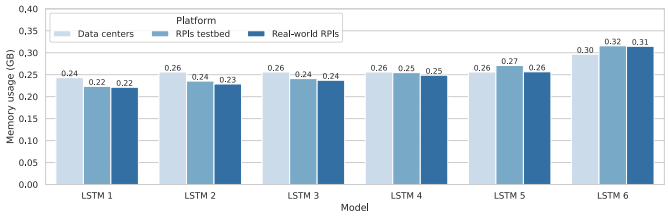


Fig. 7: Memory usage

Discussion: *Reproducing FL system utilization*

In this set of experiments, we showed that memory usage and total network I/O could be accurately reproduced across emulated platforms (data centers and RPIs). The CPU usage presented more variability. The data centers which over allocated CPU resources presented very low CPU utilization. In turn, the RPI testbed devices presented high CPU utilization, close to the real-world devices for the lighter models. However, with larger models requiring more bandwidth for communication, we observed a reduced CPU usage for real-world devices as communication time increased. Applying additional network constraints on the RPI testbed devices could help reproduce more accurately the CPU usage. Finally, the analysis of CPU temperature revealed discrepancies between the real-world and testbed devices. The testbed devices, which are generally supported by cooling systems, present stable temperature. While this ensures proper operation, it may not reflect the real state of devices deployed in the wild. Consequently, estimating the risk of system malfunction due to temperature increases in a testbed environment may not be straightforward.

VII. Lessons Learned from our Experiments

A. Which infrastructure can be used to validate which experimental aspects?

Model performance. We observed that regardless of the infrastructure, the model convergence could be fairly reproduced in the presence of the private client data. When it is impossible to reuse the private data, public data could still be used to partially reproduce the behavior of the application.

Training latency. Simulation can be used to model an application's behavior; however, without additional hardware emulation, it can not accurately reproduce the training latency of Edge devices. Conversely, testbed devices with specific hardware (*i.e.*, RPI) or using hardware emulation can accurately reproduce the computation time of the real-world devices. Using additional network emulation can help in reproducing the client communication time.

System utilization. Simulation does not provide an accurate mapping of different parts of a compute node's total system utilization to the various running processes (*i.e.*, the FL clients), limiting the reproducibility of system metrics. In turn, emulation using distributed infrastructures enables the monitoring of system metrics at the client level by isolating federated peers on different computing nodes. While some metrics can be easily reproduced (*e.g.*, memory usage), over-allocation of computing resources can limit the reproducibility of others (*e.g.*, CPU utilization). Therefore, for a fine-grained study of system metrics, deployments on specialized hardware are advised.

We summarize our findings in Table VI.

B. Relevant FL performance metrics for testbed environments

Based on our findings, we propose a set of relevant FL performance metrics to use on experimental testbeds that could reliably translate to Edge devices.

Same hardware. Using the same hardware in a testbed environment allows for fair comparison of system metrics (*i.e.*, computation time, CPU utilization, memory usage). In this context, computation time and power consumption metrics can be directly studied to predict the performance of the real-world device. In addition, network emulation can be used in testbed environments, enabling the analysis of communication time in constrained settings. Some interesting metrics for FL performance in this context include the computation time relative to accuracy/loss and the communication time relative to accuracy/loss. Additionally, computation time and communication time per round can provide useful insights into the application overhead.

Different hardware. Reproducing system metrics across different hardware can be challenging. Therefore, we recommend using metrics that are independent of the hardware. Instead of relying on absolute computation time to assess algorithm overhead, using the number of training rounds can provide a reliable performance indicator that is independent of the experimental platform. In this context, some useful performance metrics include rounds to accuracy/loss, total network I/O and network I/O per round.

C. On the challenges of real-world deployments

Deployments of devices in real-world require to address diverse (possibly conflicting) challenges: protection against outdoor weather and pilfering, access to powering solution, network connectivity, getting deployment authorizations. We addressed the above-mentioned challenges by deploying our devices into fresh air handlers (a setup easily reproducible across university campuses). We also designed our prototypes

TABLE VI: Experimental patterns reproducibility

Platform	Model convergence	Computation time	Network I/O time	CPU usage	CPU temperature	Memory usage	Total network I/O
Simulation	✓	✗	✗	✗	✗	✗	✗
Data centers	✓	(partially)	(network emulation)	✗	✗	✓	✓
Raspberry Pis	✓	✓	(network emulation)	✓	✗	✓	✓
Real-world	✓	✓	✓	✓	✓	✓	✓

with a simple architecture (using popular hardware solutions), which makes our deployment easily reproducible.

During their deployment periods (*i.e.*, when collecting air quality measurements), we encountered several **network connection problems** with 5 of our prototypes. The re-deployment of such devices requires finding new locations that fulfill the different constraints. It may also be dependent on technical staff that have special authorizations to access the deployment locations. As a consequence, we were not able to re-deploy the different devices over the duration of our study. We have been limited to study the 3 remaining (functioning) devices.

Despite the limited scale of our experiments, we highlighted some notable differences between testbed and real-world experiments: (1) some metrics are dependent on the device deployment location (*e.g.*, CPU temperature), (2) despite an access to a public wifi, the prototypes could experience high heterogeneity (including volatility with network connection problems), (3) emulation can help understand the performance of Edge devices but may have limitations in modeling device failures, such as those caused by overheating or network volatility.

VIII. Conclusion

This paper contributes to the field of FL by making a first step towards understanding the quality trade-offs between simulation and emulation of FL deployments. We argue this is a foundational element for addressing the challenges posed by practical reproducibility for FL enabled scientific exploration. Extensive experiments with an air-quality forecasting FL system on the Grid'5000 and Chameleon testbeds underscore the strengths and limitations of both approaches for the reproducibility of real-world FL experiments. As a notable result, the cost of real deployments (understood as device placements) plays a crucial role, depending on the research goals. When that cost is still too high one should continue to evaluate using simulation, which provides good results for reproducing *model* based performance metrics (*e.g.*, convergence, accuracy). Conversely, emulation on real testbeds allows more accurate reproducibility of *system* related metrics (*e.g.*, execution time, CPU and memory usage).

Acknowledgment

This work was funded by the ENGAGE Inria-DFKI project. This work was also (partially) supported by a French government grant managed by the Agence Nationale de la Recherche under the France 2030 program, reference "ANR-23-PECL-0007" (PEPR CLOUD - STEEL project). Experiments presented in this paper were carried out using the Chameleon Cloud, CHI@Edge and Grid'5000 testbeds, in the framework of a collaboration within the JLESC international lab.

References

[1] H. B. McMahan, E. Moore, D. Ramage, and B. A. y Arcas, "Federated learning of deep networks using model averaging," *CoRR*, vol. abs/1602.05629, 2016.

[2] D. C. Nguyen, Q.-V. Pham, P. N. Pathirana *et al.*, "Federated learning for smart healthcare: A survey," *ACM Comput. Surv.*, vol. 55, no. 3, Feb. 2022.

[3] A. Hard, C. M. Kiddon, D. Ramage *et al.*, "Federated learning for mobile keyboard prediction," 2018.

[4] D. Guliani, F. Beaufays, and G. Motta, "Training speech recognition models with federated learning: A quality/cost framework," in *ICASSP 2021 - 2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2021, pp. 3080–3084.

[5] D. Rosendo, A. Costan, P. Valduriez, and G. Antoniu, "Distributed intelligence on the edge-to-cloud continuum: A systematic literature review," *J. Parallel Distrib. Comput.*, vol. 166, p. 71–94, Aug. 2022.

[6] C. Prigent, A. Costan, G. Antoniu, and L. Cudennec, "Enabling federated learning across the computing continuum: Systems, challenges and future directions," *Future Generation Computer Systems*, vol. 160, pp. 767–783, 2024.

[7] S. Svorobej, P. Takako Endo, M. Bendechache *et al.*, "Simulating fog and edge computing scenarios: An overview and research challenges," *Future Internet*, vol. 11, no. 3, 2019.

[8] I. M. Elshair, T. J. S. Khanzada, M. F. Shahid, and S. Siddiqui, "Evaluating federated learning simulators: A comparative analysis of horizontal and vertical approaches," *Sensors*, vol. 24, no. 16, 2024.

[9] S. Sebbio, G. Morabito, A. Catalfamo *et al.*, "Federated learning on raspberry pi 4: A comprehensive power consumption analysis," in *Proceedings of the IEEE/ACM 16th International Conference on Utility and Cloud Computing*, ser. UCC '23, 2024.

[10] Y. Gao, M. Kim, S. Abuadbba *et al.*, "End-to-end evaluation of federated learning and split learning for internet of things," in *2020 International Symposium on Reliable Distributed Systems (SRDS)*, 2020, pp. 91–100.

[11] T. Zhang, C. He, T. Ma *et al.*, "Federated learning for internet of things," in *Proceedings of the 19th ACM Conference on Embedded Networked Sensor Systems*, ser. SenSys '21, 2021, p. 413–419.

[12] L. Ridolfi, D. Naseh, S. S. Shinde, and D. Tarchi, "Implementation and evaluation of a federated learning framework on raspberry pi platforms for iot 6g applications," *Future Internet*, vol. 15, no. 11, 2023.

[13] J. Nguyen, J. Wang, K. Malik *et al.*, "Where to begin? on the impact of pre-training and initialization in federated learning," in *ICLR*, 2023.

[14] M. Tahir and M. I. Ali, "On the performance of federated learning algorithms for iot," *IoT*, vol. 3, no. 2, pp. 273–284, 2022.

[15] D. J. Beutel, T. Topal, A. Mathur *et al.*, "Flower: A friendly federated learning research framework," *arXiv preprint arXiv:2007.14390*, 2020.

[16] F. Lai, Y. Dai, S. S. Singapuram *et al.*, "FedScale: Benchmarking model and system performance of federated learning at scale," in *International Conference on Machine Learning (ICML)*, 2022.

[17] J. H. Ro, A. T. Suresh, and K. Wu, "FedJAX: Federated learning simulation with JAX," *arXiv preprint arXiv:2108.02117*, 2021.

[18] A. M. Abdelmoniem, C.-Y. Ho, P. Papageorgiou, and M. Canini, "A comprehensive empirical study of heterogeneity in federated learning," *IEEE Internet of Things Journal*, vol. 10, no. 16, 2023.

[19] D. Rosendo, P. Silva, M. Simonin *et al.*, "E2clab: Exploring the computing continuum through repeatable, replicable and reproducible edge-to-cloud experiments," in *2020 IEEE International Conference on Cluster Computing (CLUSTER)*, 2020, pp. 176–186.

[20] N. Ferro and D. Kelly, "Sigir initiative to implement acm artifact review and badging," *SIGIR Forum*, vol. 52, no. 1, p. 4–10, Aug. 2018.

[21] M. B. A. McDermott, S. Wang, N. Marinsek *et al.*, "Reproducibility in machine learning for health research: Still a ways to go," *Sci Transl Med*, vol. 13, no. 586, Mar. 2021.

[22] B. Chen, M. Wen, Y. Shi *et al.*, "Towards training reproducible deep learning models," in *Proceedings of the 44th International Conference on Software Engineering*, ser. ICSE '22, 2022.
[23] "Ambiant (outdoor) air pollution," https://www.who.int/news-room/fact-sheets/detail/ambient-(outdoor)-air-quality-and-health, accessed: 2024-12-10.
[24] "Real-time air quality monitoring by purpleair," https://www2.purpleair.com/, accessed: 2024-11-27.
[25] K. Keahey, J. Anderson, M. Sherman *et al.*, " Chameleon@Edge Community Workshop Report," University of Chicago, Tech. Rep., 12 2021.
[26] K. Keahey, J. Anderson, Z. Zhen *et al.*, "Lessons learned from the chameleon testbed," in *2020 USENIX Annual Technical Conference (USENIX ATC 20)*, Jul. 2020, pp. 219–233.
[27] F. Cappello, E. Caron, M. Dayde *et al.*, "Grid'5000: a large scale and highly reconfigurable grid experimental testbed," in *The 6th IEEE/ACM International Workshop on Grid Computing, 2005.*, 2005, pp. 8 pp.–.
[28] C. Prigent, "FL Simulation Performance Evaluation," https://github.com/cedricprigent/fl-simulation-performance-evaluation.