# From Centralized to Federated Learning: Exploring Performance and End-to-End Resource Consumption

Georgios Drainakis[a,b], Panagiotis Pantazopoulos[a], Konstantinos V. Katsaros[a], Vasilis Sourlas[a], Angelos Amditis[a], Dimitra I. Kaklamani[b]

[a]*Institute of Communications and Computer Systems (ICCS),*
[b]*National Technical University of Athens (NTUA),*
*9 Ir. Polytechneiou St., Zografou, 15773, Athens, Greece*

**Abstract**

Machine Learning (ML) is increasingly implemented in a distributed fashion to harness the data abundance generated in the mobile client devices. Contrary to cloud-based centralized learning (CL), distributed schemes like Federated Learning (FL) shift the computational load to the user-equipment. Trying to compare and characterize the performance of the two schemes, existing literature focuses solely on model accuracy, neglecting the impact of training tasks on the underlying network resources. In this work, we introduce a realistic measurement-based model to thoroughly capture the cloud-to-client bandwidth and energy footprint of ML training on the network, while assessing the achieved accuracy, in the light of different data/model ratios. Our model is implemented in an FL simulation framework that allows trace-driven mobile users to conduct image classification tasks, under a varying exposure to training data heterogeneity. Having hyper-parameters tuned inline with network resources, our experiments unveil how the data to model ratio regulates both bandwidth expenditure and the client/infrastructure energy consumption, in cellular and wireless environments. Further simulation results confirm CL's superiority over FL under data heterogeneity and capture how FL trades-off accuracy and convergence time; thus, shedding light into so-far open questions that shape (distributed) learning solutions in networked environments.

*Keywords:* Federated Learning, Network Modeling, Simulation

## 1. Introduction

The ever-increasing adoption of Artificial Intelligence (AI)/Machine Learning (ML) in a plethora of applications *e.g.*, autonomous driving, data analytics, fraud detection *etc.* has raised the demand for intelligent network services, needed to deal with the involved computationally-heavy big-data driven ML tasks [1]. Such tasks are fed with data, which is typically associated with a myriad of *distributed* end-user devices; data acquisition is then affected by user mobility, the devices' battery life and network connectivity. On the other hand, computational power required to process this data (*i.e.*, ML model training) is inherently *centralized*, residing for instance in high-performance cloud servers.

Traditional approaches employ Centralized Learning (CL) to address this chasm; clients send their sensor-acquired raw data to a central entity that thereafter performs the computationally-heavy ML training task. Driven by networking technology trends [2], distributed machine learning (DML) schemes have emerged as alternatives to CL. Relevant examples include hybrid [3] and peer-to-peer (server-less) learning [4], whereby computational tasks are offloaded to the clients' user equipment (UE). Along these lines, Federated learning (FL), a Google driven DML scheme [5] has recently attracted significant attention, being an inherently privacy-preserving scheme. In FL, the ML task is performed in a distributed manner by the (potentially) mobile clients, utilizing their own data and computing resources. Then, only the outcome of the training *i.e.*, the ML model's parameters are sent back to the central entity for aggregation; thus, not only the ML task is offloaded, but also privacy is preserved.

Whether centralized or distributed, ML tasks can inflict huge stress on the network resources; data volume required for training varies from few GBytes to hundreds of TBytes [1], while data processing needs can reach up to thousands of TeraFLOPS [1]. A fundamental problem therefore arises; which ML scheme should be employed to train highly-accurate ML models (using the generated data) subject to the resource consumption constraints? The problem of ML scheme selection is by nature multi-parameter and inherently complex. The network comprises of multiple stakeholders *e.g.*, the clients, the intermediate network operators, the cloud provider *etc.* Each stakeholder poses different requirements and limitations that affect both the training process (therefore the resulting ML model's accuracy) and the accompanying resource consumption. Relevant examples include *a*) client-related ones *e.g.*, data distribution, availability due to mobility, battery life restrictions, *b*) network-related ones *e.g.*, bandwidth availability, end-to-end delays, *c*) cloud-related ones *e.g.*, energy consumption limits. The problem's complexity is further increased by the engineering properties of the training process itself *e.g.*, ML hyperparameters, client selection *etc.*

To the best of our knowledge an in-depth analysis on the ML scheme selection problem is missing from the current literature. Existing works have attempted to address similar questions in a restrictive manner; there is an extensive theoretical [6],[7] and experimental [8],[9] research on the comparison between CL and DML in terms of convergence, but the relevance of the two approaches to the underlying network is widely neglected. In this work, we address the ML scheme selection problem from a system standpoint *i.e.*, by providing an *end-to-end* network-resource analysis, considering all stakeholders. In view of the problem complexity and the ever-increasing spectrum of centralized-to-distributed ML schemes, we restrict our analysis on the two spectrum extremes *i.e.*, the well-established CL and the recently emerging -yet popular- FL[1]. As the preferred approach is affected by a combination of parameters, we specifically focus on the following fundamental questions: 1) How does the selection of ML hyper-parameters (being essential to tune before every ML task) affects the resource consumption of each ML scheme?, 2) Given that the client data to ML model size ratio determines the network workload, how does it affect the performance of each ML scheme in terms of accuracy, convergence speed and resource consumption for each stakeholder?, 3) What is the ML schemes' behavior in the course of time (convergence speed)?, 4) How do ML schemes respond to scaling *i.e.*, including more client participants, in terms of convergence speed and resulting accuracy? and 5) What is the effect of client data heterogeneity on the ML scheme's accuracy?

*Our main contribution*: we thoroughly explore and compare the CL *vs.* FL performance by developing a solid and pragmatic cloud-to-client system model. A dedicated AI/ML software environment[2] has been realized to accurately reproduce the training process for each ML scheme. Two typical communication channels are considered, a mobile Long-Term Evolution

[1]Focusing on accuracy and network resource consumption, data privacy requirement is relaxed to ensure a meaningful CL-FL comparison.

[2]The source code is publicly available at: https://github.com/giorgosdrainakis/dml

(LTE) and a wireless local area (WLAN) network, while in parallel, client mobility is realistically captured in each setting by replaying real-world (LTE and WLAN respectively) mobility traces. Credible measurement-based models are fabricated to estimate a real-world system performance in terms of bandwidth availability, device energy consumption and processing capacity. Moreover, a set of data distributions is employed to capture client data heterogeneity, resembling the conditions of a real system. This work significantly extends our previous research [10]; our network resources analysis now incorporates all network stakeholders (from the cloud down to the client), while new radio (WLAN) and heterogeneous data distribution settings are added to capture several realistic scenarios. Lastly, the concept of evolution in time is introduced as a key parameter in the evaluation of the two ML schemes.

Our results point to a series of important outcomes. Hyperparameter tuning has a major impact on FL, as compared to CL, enabling FL to reach CL's accuracy levels, up to 85% on the Street View House Numbers (SVHN) dataset. Having secured convergence via hyperparameter tuning, the per client data to model size ratio regulates the end-to-end network utilization. At the same time, FL in WLAN suffers from a 40% more data loss compared to FL in an LTE setting. The above ratio also dictates both the clients' and the cloud's energy consumption. On the clients side, a bound is found for this ratio to avoid depletion of a typical smartphone's battery. In parallel, we show that the client energy costs are mainly dominated by the (ML) processing on the UEs. In terms of convergence, CL demonstrates an up to 13 times speedup compared to FL, at the expense of a bursty network and cloud resource utilization. On the other hand, in line with [11], we showcase that FL's convergence can be accelerated by up to 83%, if the number of participating clients per round is increased (scaling), at a negligible cost on the ML accuracy. Extensive experiments verify CL's superiority in terms of convergence over FL, under data heterogeneous conditions [7]; importantly, CL's insensitivity to data heterogeneity is not guaranteed unless shuffling techniques are employed.

The remainder of the paper is structured as follows. In Section 2 we detail the relevant literature. Our system model is introduced in Section 3. In Section 4, extensive simulation results are analysed. Lastly, our concluding remarks and pointers to future investigation appear in Section 5.

Table 1: Research directions in ML schemes comparison

| **ML schemes comparison** | | | |
|---|---|---|---|
| **CL vs FL** | | **CL/FL vs other DML schemes** | |
| *Investigation parameter* | *Research works* | *Investigation parameter* | *Research works* |
| convergence rate | [6], [9] | convergence rate | [12] |
| accuracy | [9], [8], [13] | accuracy | [3], [14], [15], [4] |
| training loss | [6] | energy cost | [3], [14] |
| ML hyperparameters | [7] | bandwidth cost | [14] |
| data distribution | [7] | privacy & security | [14] |

## 2. Related Work

The debate of employing centralized *vs.* DML schemes in a network environment has been insufficiently addressed so far. Among DML schemes, the exploration of FL as CL's alternative has a prominent role (Table 1), due to FL's inherent privacy. FL-related works are primarily focused on the resulted accuracy (Table 1), as in [8], where FL is employed for intrusion detection in an Internet of Things (IoT) environment. The proposed decentralized scheme demonstrates comparable accuracy (maximum difference is 5%) to its centralized counterpart, while preserving privacy over the sensed data. FL's reliability against state-of-the-art CL solutions for sensitive hospital data is studied in [13]. The investigation over three benchmark clinical datasets concludes that FL can reach CL's performance across a wide range of ML metrics *e.g.*, accuracy, recall *etc.* with a maximum difference of 4%.

Besides accuracy, FL's algorithms are also evaluated in terms of their convergence (Table 1) against CL, as in [6], where Distributed Stochastic Gradient Descent (D-SGD) is applied in a cluster of computing nodes, placed in a static network topology. D-SGD achieves similar training loss as CL, while exhibiting faster convergence in presence of a low-bandwidth network. Several connected nodes (distributed workers) of equal processing capacity are assumed in [9], one of which acts as the central server. Tests on the Fashion-MNIST dataset, suggest that CL achieves a 2% better accuracy compared to FL, while FL converges almost 40% faster. Since the workers and the server share the same specifications, its applicability on computationally-limited hand-held devices is limited. Moving to more practical comparisons, various FL algorithms *e.g.*, FedAvg, Cooperative *etc.* are bench-marked against CL in [7], after an exhaustive search is performed to tune the ML hyperparameters. Bayesian-based accuracy metrics employed, show that CL provides more accurate classifications at all times, with FedAvg showing promising performance, only when data is independent and identically distributed (i.i.d.).

These CL-FL comparison studies have reduced applicability in a practical system. Since they are centered towards the resulting ML model's performance, the expenditure on the underlying network's resources is largely neglected. On top, existing comparisons disregard the effect key environment parameters (data distribution, client availability, communication delays *etc.* ) can have on the ML task. On the contrary, our study explores the convergence of CL and FL with respect to their cost in terms of network resources and quantifies the trade-offs between ML performance and resource efficiency.

Besides FL, a research direction is also developed towards other DML schemes (Table 1), to be used as CL alternatives. Hybrid schemes, for example are regarded as middle-ground solutions between CL and FL. A relevant proposal in [3] demonstrates similar accuracy to CL, outperforming FL by an average of 10%; at the same time it keeps a balance between the energy efficiency of FL and the high energy cost of CL. In [14], the authors introduce a hybrid edge-FL scheme, which enhances security via a block-chain mechanism. The proposed solution outperforms both FL and CL in terms of accuracy by a maximum of 5%, exhibiting almost 50% reduction in energy consumption compared to CL, while consuming 25% more bandwidth.

Collaborative ML schemes on the other hand are investigated in an effort to minimize communication costs, since they do not require a dedicated central server. The concept of Swarm learning is presented in [15], where the clients (swarm nodes) share the model parameters with one another via the swarm network. Experiments on clinical data showcase an accuracy improvement of the suggested schemes over CL, for various cases of patient detection *e.g.*, leukaemia, tuberculosis, COVID-19 *etc.* Gossip learning [12] is a similar approach, where clients share a small portion of their trained ML model parameters with their neighbors. Results in a trace-based mobile network environment show similar accuracy to FL, yet with slower convergence speed. Peer-to-peer learning [4] functions like FL, without the use of a central server. Instead, the ML models are exchanged between the clients via a secret peer-to-peer sharing schema. Although the suggested scheme's convergence rate is heavily dependent on the existence of i.i.d. data, its accuracy levels are similar to those of CL.

While the majority of works on hybrid and collaborative schemes demonstrate improvements over FL, they mostly focus on the resulted accuracy, neglecting the effect on the system resources. We argue that (besides accuracy) without considering the consumed network resources, any ML scheme deployment over the network cannot be efficient. In our work, we seek to shed light into the unexplored areas of the resource consumption of ML, in an effort to accurately compare the performance of CL and FL.

## 3. System Model

Table 2: Nomenclature of involved quantities

| Symbol | Definition |
|---|---|
| $d_{sample}$ | Training dataset samples |
| $d$ | Training dataset size (bytes) |
| $z$ | Training dataset partitions (*i.e.*, total clients) |
| $r_{sample}$ | Training dataset size to samples ratio |
| $m$ | ML model size (bytes) |
| $r_{data}$ | Client data to model size ratio |
| $k$ | Per round participating clients |
| $q$ | Online clients |
| $t_{upload}$ | Total time for client data upload in each round (sec) |
| $t_{end}$ | Maximum duration of ML task (sec) |
| $s_{client}$ | Client throughput - access network (bytes/sec) |
| $c^{CH}$ | Average area throughput - access network (bytes/sec) |
| $\sigma$ | Client throughput standard deviation - access network |
| $c^{CH}_{min}$ | Minimum client throughput - access network (bytes/sec) |
| $s_{core}$ | Core network element's throughput (bytes/sec) |
| $\sigma_{ed}$ | Data skewness parameter |
| $\sigma_{iid}$ | Independently and identically distributed (i.i.d) level shape parameter |
| $v^{ML}_{client}$ | Client device computational capacity for training (samples/sec) |
| $v^{ML}_{cloud}$ | Cloud computational capacity for training (samples/sec) |
| $v^{AG}_{cloud}$ | Cloud computational capacity for model aggregation (models/sec) |
| $e_{client}$ | Total energy expenditure for all clients (J) |
| $p^{TX}_{client_i}$ | Client device power consumption for transmission (W) |
| $p^{RX}_{client_i}$ | Client device power consumption for reception (W) |
| $p^{ML}_{client_i}$ | Client device power consumption for training (W) |
| $e_{core}$ | Total energy expenditure in the core network (J) |
| $e_{cloud}$ | Total energy expenditure in the cloud (J) |
| $p^{ML}_{cloud}$ | Cloud power consumption for training (W) |
| $p^{AG}_{cloud}$ | Cloud power consumption for model aggregation (W) |
| $h_{epochs}$ | Number of epochs (ML hyperparameter) |
| $h_{batch}$ | Batch size (ML hyperparameter) |
| $h_{rate}$ | Learning rate (ML hyperparameter) |

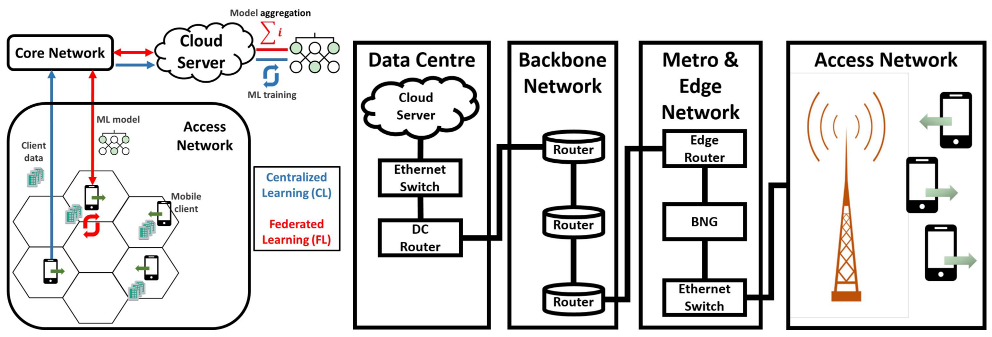
We introduce our measurement-based system model as follows: We allow for several mobile clients in a dynamic network environment, each holding an amount of training data. Clients are able to exchange data with a central entity *e.g.*, a cloud server located in a data centre (DC), via the intermediate core network. We do not consider any control messages to facilitate data exchange *e.g.*, coordination or signalling, given their negligible size compared to the actual data.

Our system model focuses on the implementation of CL and FL. The ML task is performed in several communication rounds. For CL, in each round, the clients upload their local data to the cloud server for training (see Fig. 1a). Note that although the ML task is performed in a centralized location, CL is not to be confused with the standard ML paradigm, where training is performed using the entire dataset. The latter is referred to as Standard Machine Learning (SL). Essentially, an implementation of SL would require for the server to wait for all participating client datasets to arrive in order to perform the training task. On the other hand, periodic training *i.e.*, on a per round basis occurs in CL, given that (recently generated) data arrives in every round. In our study we focus on CL, rather than SL, since the way SL functions deems it impractical for employment in a real system. Applications either cannot afford waiting for the entire dataset to reach the data centre or are agnostic to the entire client dataset. More importantly, SL lacks adaptability, since it does not produce a global ML model on a per-round basis. In an actual system, clients do not generate/acquire a single dataset, which will be uploaded once; they rather repeatedly collect data *e.g.*, user analytics, network measurements *etc.* which can be potentially streamed to the cloud in a continuous manner. Moreover, depending on the environment, this data can change in time *e.g.*, traffic scene images and therefore the output of the ML task *i.e.*, the ML model, needs to be updated on a regular basis. As such, SL is not proposed as a practical solution, but serves as a baseline to other centralized schemes. Moreover, CL is needed in our study to ensure a fair and meaningful comparison to FL, given that a global ML model is generated in every round.

In each FL round, the server communicates the (global) ML model to the selected clients, which in turn train the model locally, using their own (private) data and computing resources. Upon completion, the clients communicate the updated (local) model parameters to the central server, which aggregates the local ML models into a new global ML model (see Fig. 1a). In the following sections (3.1-3.6) we detail the elements that comprise our system model. All involved quantities are summarized in Table 2.

### *3.1. Network architecture and attributes*

The network includes the wireless (radio) and the wired part (see Fig. 1b). Two common cases are considered for the wireless part; a mobile Long-Term Evolution (LTE) and a wireless local area (WLAN) network. In the LTE case, a cellular architecture is assumed, where a basestation (BS) lies in the centre of each cell. In the WLAN case, several access points (APs) are assumed, which cover a local network area (coverage area). We refer to the wireless part (link between clients and BS/AP) as the access network. The wired part (from BS/AP up to the cloud) comprises the core network (see Fig.



(a) (b)

Figure 1: (a) In CL, mobile clients (green horizontal arrows) upload (blue vertical arrows) their data via the core network to the cloud, which performs the training centrally. In FL, the training is performed on the device (red recursive arrows) and only the ML models are shared (red vertical arrows) to the cloud, where aggregation occurs. (b) The radio part of the network (access) is connected to the cloud via the core network, which includes the metro & edge, the backbone and the DC's infrastructure part.

1b). The core network is divided to 1) the metro and edge network, where edge nodes reside, 2) the backbone network, which includes core switches and routers and 3) the infrastructure to reach the DC's cloud server.

***Access network throughput:*** The client throughput $s_{client}$ in both the uplink (UL) and the downlink (DL) (in MBytes/sec) is modelled as a Gaussian random variable (Ñ). Its mean value equals the average area throughput $c^{CH}$ (where $CH$ marks the corresponding radio technology *i.e.*, LTE or WLAN), divided by the number of online clients $q$. The inclusion of the online clients in the computation, allows to factor-in the way the locally-present number of users shapes the throughput provision in the considered area. $c^{LTE}$ refers to the average cell throughput[3], while $c^{WLAN}$ to the average coverage area throughput[4]. The standard deviation parameter $\sigma$, equal to 20% of the client's mean throughput [18] accounts for throughput variations *e.g.*, due to path loss and interference. Given that a client is found at a certain moment connected/online, we also assume that there exists a minimum throughput threshold ($c^{CH}_{min}$) to enable server-client communication, both in DL and in UL. $c^{LTE}_{min}$ is assumed equal to the 5% cell edge rate[5], which represents the

[3]That is, 5.9 (UL)/7.73 (DL) MBytes/sec for 2.5 GHz LTE according to [16]
[4]That is, 2.25 (UL)/2.38 (DL) MBytes/sec for IEEE 802.11g according to [17]
[5]That is, 0.24 (UL)/0.22 (DL) MBytes/sec according to [16]

worst radio conditions, while for WLAN, a similar value for the coverage area ($c_{min}^{WLAN}$) can be obtained[6]. Thus, $s_{client}$ for UL and DL, is given by:

$$s_{client} = max\{\tilde{N}(\frac{c^{CH}}{q}, \sigma), c_{min}^{CH}\} \tag{1}$$

***Core network throughput:*** The core network includes the following elements [20]: 1) An interface to the access network (BS for LTE or AP for WLAN); 2) The metro and edge network's elements, *i.e.*, an ethernet switch, a broadband network gateway (BNG) and the edge router; 3) the backbone network's routers and 4) the DC's elements, *i.e.*, an edge router and a data center switch. The average throughput of each element for the UL/DL ($s_{core}$) is based on Cisco routers/switches benchmarking [20] and measurements on access network interfaces (3-sector 2×2 Multiple-Input-Multiple-Output remote radio 4G/LTE) [20]. A total number of 3 backbone network's routers is considered, as a hopcount of maximum 3 in the backbone network suffices to reach the DC for the majority of popular services *e.g.*, Facebook, Google [21].

## *3.2. Client Mobility Pattern*

Unlike the synthetic or the theoretical mobility models, real-world traces promise realistic performance evaluation and credible results. However, concerns are typically expressed on their representativeness and generality of the evaluation results. Along this line, performing multiple iterations on public datasets, increases the generality and credibility of our results. For our system model, we have chosen the Shanghai Telecom Dataset [22] for LTE traces and the Wifidog [23] for WLAN traces. The Shanghai Telecom Dataset contains records of UEs accessing the Internet through a BS in a period of 15 days. The database includes timestamps (taken every minute, which is the dataset's time granularity) for connection initialization and termination; thus, the online presence can be calculated. The clients are monitored and are considered online as long as they remain in the network. When a client moves to another cell *i.e.*, serviced by another BS, a handover (HO) is assumed, to capture service continuity. We ignore the communication disruption during the HO process. However, a change of cell will affect the client throughput, which depends on the corresponding cell's congestion (see paragraph 3.1). The Wifidog dataset contains user-session traces from various

[6]That is, 0.03 (UL-DL) MBytes/sec according to [19]

free WiFi hotspots in the city of Montreal, Quebec, Canada. Similarly to the Shanghai dataset, each node denotes an AP, while timestamps are collected for user login and logout times with a granularity of 1 sec. Clients are considered online for a particular time period, if connected to a hotspot.

### *3.3. Client Data Acquisition and Distribution*

Client devices acquire raw data via their sensors, cameras *etc.* which can be used for training. The acquisition is shaped by the client's acquisition rate, the UE's storage capacity and the data staleness level. To emulate such a behavior, we divide our image-classification training dataset (see paragraph 3.6) into $z$ partitions, which are assigned to the selected clients, representing the data that the clients have generated and stored on their devices. We define the dataset size to samples ratio $r_{sample} = d/d_{sample}$, where $d_{sample}$ is the total number of dataset's training samples and $d$ the dataset's size in Bytes. Marking the per client dataset size as $d_i$, the total dataset can be written as: $d = \sum_{i=1}^{z} d_i$. Assuming a fixed ML model size $m$, the per client data to model size ratio is defined as $r_{data_i} = d_i/m,\ i \in [1, z]$. $r_{data}$ is a key parameter in the ML scheme selection problem. Not only it regulates the network's resource consumption *e.g.*, large chunks of data require more bandwidth in order to be uploaded (CL) or more energy in order to be processed (FL), but can also affect the convergence of the ML task itself. Besides data acquisition, we also explore how this data is distributed across clients *i.e.*, data heterogeneity. We focus on two dimensions; variations in size are modeled by the evenly distributed level (e.d.), while variations in content are captured by the independent and identically distributed level (i.i.d.). Generally, client data can experience various levels of e.d., i.i.d. or combinations of both.

***On the e.d. level***: The e.d. level describes the dataset size distribution across clients. The size of each client's dataset is modeled as a random variable ($\tilde{\mathrm{F}}$) that follows Zipf's law, in line with related research [24]. To represent a random variable following Zipf's law, we are using Zeta distribution, which has a probability density function of $p(x) = \frac{x^{-(\sigma_{ed})}}{\zeta(\sigma_{ed})}, \sigma_{ed} \in (1, +\infty)$. $\zeta$ represents the Riemann Zeta function, while the Zeta distribution's skewed parameter $\sigma_{ed} \in (1, +\infty)$ shapes the e.d. level, moving from a uniform data distribution ($\sigma_{ed} >> 0$) towards higher asymmetry cases ($\sigma_{ed}$ close to 1); an additional restriction is also imposed that the minimum dataset size equals the size of one batch $h_{batch}$ (see paragraph 3.6), in order to ensure that training can occur at all cases. Therefore the client dataset size becomes $d_i = max\{h_{batch}, \tilde{F}(\sigma_{ed})\}, \sigma_{ed} \in (1, +\infty), \forall i \in [1, z]$. If the initial dataset is evenly distributed among the clients ($\sigma_{ed} >> 1$), the client dataset size $d_i$ and dataset to ML model size ratio $r_{data_i}$ are simplified to: $d_i$=$d/z$, $r_{data_i}$=$d/(m \cdot z), \forall i \in [1, z]$, respectively. A $\sigma_{ed}$ value close to 1 on the other hand marks a setting where few clients hold considerable amounts of data.

***On the i.i.d. level***: If a setting with independent and identically distributed (i.i.d) data is assumed, then the data samples in each client have the same probability distribution and are mutually independent. In our image-classification problem (see paragraph 3.6) that would essentially mean that each user holds samples from all classes (unbiased setting). That is represented by the i.i.d. level $\sigma_{iid}$, being equal to the total number of dataset classes. For some real-world scenarios, non-i.i.d. (biased) settings could occur, since each participating client might not be expected to possess a representative subset of all classes in the total training dataset. To study different levels of bias, we restrict the number of dataset classes a client can hold *i.e.*, the value of $\sigma_{iid}$ is smaller compared to the total number of classes.

## *3.4. Device Computational Capacity*

***User equipment***: The computational capacity of a mobile device to perform a ML task $v_{client}^{ML}$, measured in (processed) training samples/sec depends on the dataset content *e.g.*, images pose different requirements than natural language, the UE capabilities and the complexity of the ML model. We use a reference (average) value of $v_{client}^{ML}$=125 training samples/sec, as the most appropriate for our training dataset, ML model and ML hyperparameter settings (see paragraph 3.6), based on approximations for popular large-scale classification tasks [25].

***Cloud server***: Computational tasks for the cloud server include training (in CL) and ML model parameter aggregation (in FL). Regarding training, we assume that a DC is equipped with a Tensor Processing Unit (TPU), which demonstrates an average computational capacity for training $v_{cloud}^{ML} = 40K$ training samples/sec [26]. In regards to aggregation, no reference values could be found in the literature, thus we rely on an empirical approach; we measure the average capacity for training and aggregation tasks in our setup (*i.e.*, 6250 training samples/sec and 1.56 ML model aggregations/sec respectively) and compare against the training capacity reference value of 40K training samples/sec [26]. Assuming a linear relation, the average cloud aggregation capacity becomes $v_{cloud}^{AG}$=10 model aggregations/sec. Our assumptions do not cover scale-out schemes, where clusters of servers may be used to increase parallelism in DCs.

## *3.5. Device Energy Consumption*

***User equipment***: Any consumption related to the UE's standard operation *e.g.*, the device's operating system functionalities or displaying, is neglected and we focus on energy expenditure due to transmission (TX)/ reception (RX) of data and ML processing (training tasks). The energy consumption $e_{client_i}$ *i.e.*, battery discharge of the $i^{th}$ client's device is computed as: $e_{client_i}$=$e_{client_i}^{TX}$+$e_{client_i}^{RX}$+$e_{client_i}^{ML}$, where the superscript TX, RX and ML marks one of the aforementioned functions. In a given time period $t$, this can be calculated as $e_{client_i} = p_{client_i} \cdot t$, where $p_{client_i}, i \in [1, z]$ stands for the respective (average) power consumption. Average power consumption values related to transmission are reported in [27], where $p_{client_i}^{TX}$=2.2 Watts and $p_{client_i}^{RX}$=1.5 Watts, $\forall i \in [1, z]$ for LTE and $p_{client_i}^{TX}$=0.75 Watts and $p_{client_i}^{RX}$=0.25 Watts for WLAN. Likewise, for ML, based on [25], we assume $p_{client_i}^{ML}$=2 Watts, $\forall i \in [1, z]$, as the most appropriate to our ML model's hyperparameters and our choice of training task, being an image classification problem (see paragraph 3.6). The sum of all device energies $e_{client_i}$ comprises the total client energy expenditure $e_{client}$.

***Core network devices***: Energy consumption in the core network $e_{core}$ is calculated by summing the energy consumption per core component (Fig. 1b) *i.e.*, all routers, gateways and switches of the backbone, metro & edge network and the cloud's plus access network's interfaces (BS for LTE and AP for WLAN), which in turn is given by [20] in relation to the data exchanged in the UL/DL direction (as average values measured in Joules/bit ).

***Cloud server***: The computational capacity values of cloud tasks (training and aggregation) are discussed in Sec. 3.4. Energy expenditure per task can thus be calculated[7], given an average power expenditure. For the training task (in CL), being an intensive processing task, we assume an average power $p_{cloud}^{ML}$=384 Watts, based on Google's TPU benchmarking [28]. For the (less-intensive) aggregation task (FL), we assume $p_{cloud}^{AG}$=15 Watts, based on

[7]Inference *i.e.*, applying the trained ML model on new data also involves resources but may come as a stand-alone task, much later than training which is far more demanding in computations and network resources. Similarly, latency is not considered in our setup as training time requirements typically dominate over any latency considerations.

measurements for matrix multiplication tasks [29], which are similar in complexity to weighted averaging (aggregation). The cloud energy consumption $e_{cloud}$ then becomes:

$$e_{\text{cloud}} = \begin{cases} p^{ML}_{cloud} \cdot d/(v^{ML}_{cloud} \cdot r_{sample}), & \text{for CL} \\ p^{AG}_{cloud} \cdot z/(v^{AG}_{cloud}), & \text{for FL} \end{cases} \tag{2}$$

### *3.6. Machine Learning Task*

A representative ML task, namely image-classification is chosen. Specifically, the Street View House Numbers (SVHN) dataset is selected, which is widely used in prior works *e.g.*, [30],[31]. It is based on a set of real-world images, with digits taken from natural scene images (house numbers in Google Street View). It contains 531,131 32x32 colour training images (of 1.3 GB size) split in 10 classes (for digits 0-9) and 26,032 test images (of 63 Mb size). The split between training and test data is made by the providers.

To perform image-classification on SVHN, an artificial neural network was developed, employed in both CL and FL case. For the FL case, the Federated Averaging (FedAvg [5]) algorithm is used, as the default in our implementation's software (PySyft framework [32]). Note that in PySyft a uniform ML model averaging is performed (all ML models are equally weighted during aggregation), as opposed to the original FedAvg algorithm, where a weighted averaging is assumed by the number of each client's training samples. Our (linear) neural network is comprised of an input layer of 3072 neurons, which correspond to the total pixels of the input images of SVHN (32x32x3), an output layer of 10 neurons, equal to the total output classes of SVHN and an intermediate (hidden) layer of 512 neurons. Rectified Linear Unit (ReLU) activation is applied on the hidden linear layer (ReLU functions as a filter, allowing only positive values to pass through), while on the output layer LogSoftmax activation [33] is selected, being more effective for N-element classification tasks. To calculate training loss (loss layer), the negative log-likelihood loss function is preferred [33], since it couples together with LogSoftmax for classification tasks. The total size of the ML model reaches 6.1 MB. Larger (up to hundreds of MBs) and more complex ML models could be employed, like multi-layer convolutional neural networks *e.g.*, MobileNet, ResNet, DenseNet *etc.* [33], being specialized for providing accurate predictions for image-classification tasks. This, however, would not be in line with practical UE limitations *e.g.*, disk capacity, processing power, memory, battery *etc.* and most likely would not favor the user willingness to participate in the training process; thus, would limit our system's realism.

***Hyperparameter Tuning:*** Setting the ML model's hyperparameters, *i.e.*, the control variables of the ML process *e.g.*, number of epochs, batch size, learning rate *etc.* is a standard procedure, prior to the ML training task. Hyperparameters define the training process, but are not part of the resulting ML model. They rather dictate how the ML model's parameters (which we refer to as the resulting ML model) will be trained. Tuning is achieved by performing a series of training tasks (test-runs), using different combinations of hyperparameter values. The combination that yields the most accurate ML model is then used for the actual training task. Without tuning, the ML task's resulting accuracy can rapidly degrade, even fail to converge, as shown for FL in our previous CL-FL comparison [10]. Contrary to the vast majority of studies that explore hyperparameters solely on the basis of the resulting ML model's performance (*e.g.*, convergence speed, resulting accuracy *etc.* ), we also factor in the effect that their choice imposes on the underlying network's energy consumption, which is affected by processing.

For that purpose, our investigation starts from specifying the total number of epochs $h_{epochs}$, which controls the number of times the learning algorithm will work through the entire training dataset. As such, it does not only affect the achievable performance of the ML model, but is also related to the processing time and therefore consumed energy (for the clients in case of FL and for the cloud server in case of CL). Upon securing the value of $h_{epochs}$, we also investigate the effect of two other fundamental hyperparameters, namely the batch size ($h_{batch}$) and the learning rate ($h_{rate}$). $h_{batch}$ defines the number of training samples that are used in one iteration (forward/backward pass), while $h_{rate}$ refers to the step size at each iteration, essentially controlling how much the ML model changes, each time the model's weights are updated. The selection of these hyperparameters primarily affects the achievable accuracy of the ML model and to a lesser extent the processing time.

### *3.7. Emulation Environment Process*

The following steps describe how the above-mentioned modelling components, together with client selection and server-client communication are incorporated in our emulation environment, both for the CL and FL case.

Initially, the cloud server generates a (non pre-trained) ML model, which seeks to train by utilizing available client data. The server acts as a controller during the training process. Also, the SVHN dataset (see paragraph 3.6) is split into $z$ partitions (see paragraph 3.3). We then run a series of communication rounds, until all partitions are used or a predefined time deadline ($t_{end}$) is reached. The time deadline is selected, based on the respective time granularity of each mobility dataset. In each communication round:

***Step 1:*** The server identifies all online clients ($q$) determined by the corresponding mobility dataset (capturing the mobility dynamics).

***Step 2:*** A total of $k < z$ clients are randomly selected, as a subset of $q$, each of which is assigned a partition (see paragraph 3.3). If there are not enough online clients present to satisfy our selected per round participating client number ($q < k$), all the currently online clients are used. If no clients are found at all ($q = 0$), the round is terminated and a waiting period is introduced, equal to the mobility dataset's time granularity; that is 60 sec for the Shanghai Telecom and 1 sec for the Wifidog dataset (see paragraph 3.2).

***Step 3:*** The ML scheme is initiated: In case of CL, the selected clients upload their (raw) local datasets to the cloud server. The time ($t_i$) required for each client's dataset ($d_i, i \in [1, z]$) upload is equal to the time of the access network upload plus the time of the core network upload, therefore can be written as (see paragraph 3.1): $t_i = d_i/s^{UL}_{client} + \sum_j (d_i/s^{UL}_{core_j})$, for $j$ core components. A parallel communication protocol is assumed, thus the total time to upload all datasets ($t_{upload}$) equals to that of the "slowest" client: $t_{upload} = max\{t_i\}$. Upon collection, the server merges all round's data into a super-dataset, which is then randomly shuffled to mitigate the effects of variance [34]. With the collected super-dataset the server trains the ML model. Training time is calculated using the cloud server's computational capacity model (see paragraph 3.4). The completion of the ML training marks the end of the round.

For the FL case, the cloud server firstly shares the training models (of size $m$) to the selected clients ($k$). Afterwards, each client trains the model, utilizing its assigned dataset and uploads the updated model back to the server. When all models are uploaded back to the cloud server (again in a parallel manner, similar to the CL case), the server performs model aggregation (averaging). The time needed for the federated training and the server's aggregation task is calculated from the UE's and cloud servers computational capacity model respectively (see paragraph 3.4).

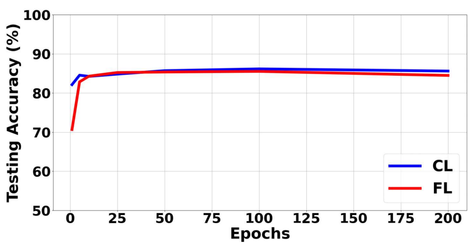
***Step 4:*** A communication failure is defined when a client goes offline, while performing a task (training or data exchange), irrespective of the task's completion percentage. Such a failure is checked by parsing the corresponding mobility dataset. In case of failure, the client's contribution is neglected by the cloud server (training for CL/aggregation for FL). However, to account for the time/resources spent for the partial communication, we consider a delay time equal to the estimated task's time, along with the respective resource consumption, assuming the worst-case scenario *i.e.*, that the connection is lost, when the task was almost completed.

## 4. Experimental Evaluation

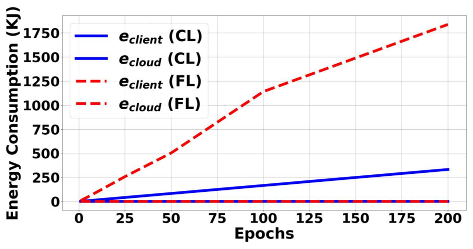
### *4.1. Simulation setup and evaluation metrics*

The simulation environment was set-up in a single desktop machine with the following characteristics: Intel Core i7-10700 (2.9 GHz), 64-bit, 16 GB RAM, Win10. To emulate the distributed learning environment the PySyft library [32] was used. The DML environment[2] is wrapped using a custom Python-based discrete event simulator, which emulates the underlying network *i.e.*, the mobile clients, the core network and the cloud server, as illustrated in Fig. 1b. The network simulator software realizes the network throughput, client mobility, data distribution, computational capacity and energy consumption models, as presented in Sec. 3. Its clock is synced to the mobility dataset's timeframe, which is used as global (time) reference. A deadline of $t_{end} = 24hrs$ is chosen across all experiments which are set to terminate when all SVHN partitions are used or $t_{end}$ is reached. We report that in 95% of the cases, termination occurred due to dataset depletion, while the average termination (simulation) time was 4 hrs, throughout all our experiments. Unlike the majority of works in literature that evaluate the performance of CL *vs.* FL with regards to their accuracy, we employ a broad set of carefully-selected metrics to capture all previously-overlooked dimensions.

***Test Accuracy:*** The effectiveness of the trained ML model is evaluated on the SVHN test data (see paragraph 3.6). The percentage of successful to total classifications provides the test accuracy metric, with an ideal value of 100%. Both in CL and in FL, we assume that the cloud server has the ability (and the capacity, given the ML model's negligible size) to save each round's ML model, so that the most accurate ML model can be extracted in the end of the experiment. Keeping track of each round's output is a standard technique in ML training [35], to avoid over-training and therefore parameter over-fitting, which results in less accurate ML models. Thus, the term Test Accuracy refers to the maximum accuracy achieved during the



(a) Testing accuracy w.r.t. epochs



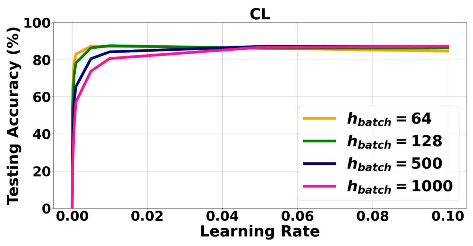
(b) Energy expenditure w.r.t. epochs

Figure 2: ML hyperparameter tuning: Number of epochs

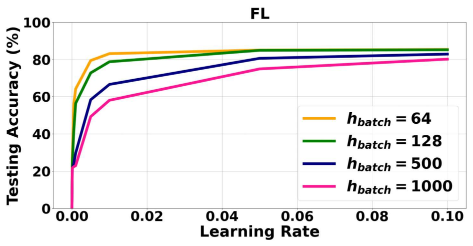
(CL/FL) experiment and not necessarily the accuracy of the chronologically latest ML model.

***Traffic Overhead:*** It is defined as the total amount of data exchanged between the cloud server and the clients for the duration of the ML process, multiplied by the total number of hops the data traverses, after leaving its origin node *i.e.*, $hops = (total\ network\ nodes) - 1$. In case of CL, the exchanged data refers to the raw data uploaded by the clients, while in FL it refers to the ML model's parameters uploaded by the clients, plus the ones distributed back to clients by the cloud server (Fig.1a). For the sake of clarity, traffic overhead is normalized to the total dataset size *i.e.*, 1.3 GB. Traffic overhead reflects the total bandwidth consumed during the ML process, accounting for both successful and unsuccessful communications. On the contrary, the total overhead generated by communication failures (see paragraph 3.7), is referred to as *Traffic Loss*.

***Energy Consumption:*** It captures the energy expenditure in all involved devices, during the ML process (paragraph 3.5), regardless the success/failure of transmission. Specifically, for the clients, we consider the total energy of all devices involved (due to ML processing and transmission/reception). For the core network it is only limited to expenditure due to data exchange. Lastly, for the cloud server, we only account for consumption due to processing (either model aggregation in FL or ML training in CL). Similarly to the definition of *Traffic Loss*, we also define the *Energy Loss*, as the total energy consumed (*e.g.*, by a client's device for training a model in FL), but the result was not utilized due to a subsequent client-server communication failure.



(a) Testing accuracy w.r.t. batch size and learning rate (CL)



(b) Testing accuracy w.r.t. batch size and learning rate (FL)

Figure 3: ML hyperparameter tuning: Batch size and Learning Rate

### *4.2. CL vs. FL: Energy-aware hyperparameter exploration*

The exploration of the ML hyperparameter values (tuning) is performed prior to the actual ML tasks (experiments), as discussed in paragraph 3.6. Starting from the number of epochs $h_{epochs}$ we consider the following values: {1,5,10,25,50,100,200}, which are commonly used in bibliography for image-classification tasks and SVHN specifically [31]. Our test-runs suggest that our ML model achieves a maximum accuracy of 85%, if $h_{epochs} \in [25, 100]$ both for CL and FL (see Fig. 2a). When looking to the underlying energy consumption in the clients side, one observes that an increase on $h_{epochs}$ causes a linear increase in energy consumption, with angles of incline equal to 0.46 and 0.78 rad for CL and FL, respectively (see Fig. 2b). To account for both accuracy and the consumed energy, a value of $h_{epochs}$=25 is selected.

Moving to the batch size $h_{batch}$ and learning rate $h_{rate}$, we perform an exhaustive search over the following values: {64,128,500,1000} for $h_{batch}$ and {0.0001,0.0005,0.001,0.005,0.01,0.05,0.1} for $h_{rate}$, based on [31], [30]. In the CL case (see Fig. 3a), the maximum accuracy is achieved for a combination of $h_{batch} \geq 128$, $h_{rate} \geq 0.05$ and for the FL case (see Fig. 3b) $h_{batch} \leq 128$, $h_{rate} \geq 0.05$. For a fair comparison of the two ML schemes (CL, FL), we assume a common set of values of $h_{batch} = 128, h_{rate} = 0.1$. The selected hyperparameter values are kept fixed throughout our experiments.

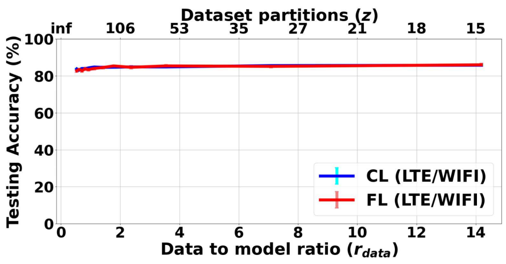
### *4.3. CL vs. FL: The effect of client data to model size ratio $r_{data}$*

In this section, we investigate the way the per client data to model ratio $r_{data}$ shapes the performance of the ML schemes under evaluation. Essentially, $r_{data}$ indicates the amount of data a client holds (in relation to the fixed ML model size $m$). For this set of experiments, we assume data is symmetric across all clients (*i.e.*, i.i.d. and e.d. setting) and vary the number of dataset partitions *i.e.*, total clients $z = \{15, 30, 60, 90, 120, 150, 190, 230, 260, 300, 330, 360, 400\}$. Therefore the data to model ratio becomes $r_{data_i}=d/(m \cdot z), \forall i \in [1, z]$ (see paragraph 3.3). We also, for now, assume a constant value for the per round participants $k$=5. Keeping in mind that the total dataset (SVHN) size $d$ is fixed to 1.3 GB (see paragraph 3.6), larger values of $r_{data}$ represent setups, where fewer clients with larger dataset partitions ($z$) participate in total, in the ML scheme. Vice versa, smaller $r_{data}$ values represent a situation, where more clients with fewer data participate in total. Nevertheless, the aggregated (total) training data (SVHN) in each experiment remains the same, ensuring a fair comparison for all training tasks. Each of the 13 experiments (representing different $r_{data}$ or equivalently $z$ values) runs with both LTE and WLAN settings, for 10 different sample time periods taken randomly from the respective mobility dataset, yielding a total of 260 pairs of CL-FL experiments. All measurements of this section are taken at the end of each experiment and mean values out of the 10 samples are depicted together with the corresponding 95% confidence intervals.[8]

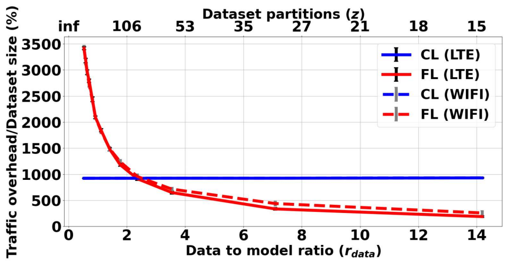
***Impact of*** $r_{data}$ ***on the achievable testing accuracy:*** Both CL and FL manage to converge, achieving an average testing accuracy level of 85%, in both the LTE and the WLAN case (see Fig. 4a), for all values of $r_{data} \in [0.5, 15]$ (or $z \in [14, 426]$ - for the sake of clarity we use a dual-valued x-axis *i.e.*, showing the datasets partitions $z$ on the top and the $r_{data}$ parameter at the bottom). The reported accuracy value equals to the maximum accuracy achieved during the ML hyperparameter tuning process (see Fig. 3). Our results match previous findings, where hyperparameter tuning and increasing the number of epochs has been identified as a promising technique to enhance the convergence of FL's default algorithm *i.e.*, FedAvg [7]. It is now demonstrated that determining the ML hyperparameters prior to training, enables FL to converge for all values of $r_{data} \in [0.5, 15]$, achieving in fact the same accuracy as CL.

***Impact of*** $r_{data}$ ***on the bandwidth consumption:*** The impact of $r_{data}$ on the ML schemes' network resource utilization (reflected by the normalized traffic overhead) is depicted in Fig. 4b. We observe that for low ratios ($r_{data}$<2), FL is bandwidth-demanding, consuming exponentially more data

[8]For the sake of clarity in the Appendix A, we present indicative performance results employing a larger model (CNN). Even with that size *i.e.*, tens of MBs, its achievable applicability over hand-held devices is questionable, thus we maintain our focus on the originally considered, light-weight one.

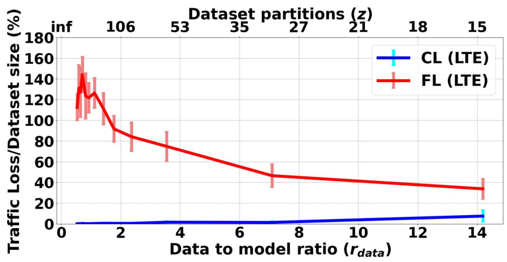


(a) Achieved testing accuracy

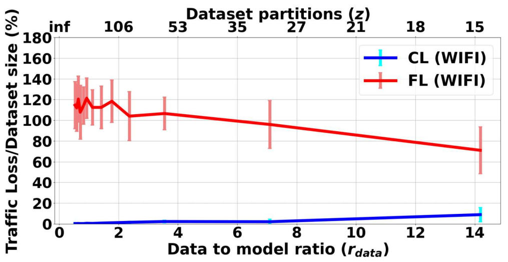


(b) Normalized traffic overhead

Figure 4: Testing accuracy and normalized traffic overhead w.r.t. data to model ratio



(a) LTE



(b) WIFI

Figure 5: Normalized traffic loss w.r.t. data to model ratio

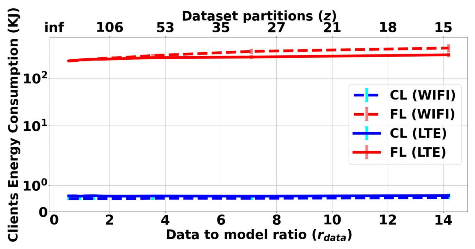
in comparison to CL. This stems from the fact that more clients participate in the training process, therefore more ML models are uploaded/downloaded to/from the server. In contrast, for large ratios ($r_{data}$>3), where the total number of clients decreases, FL naturally becomes more bandwidth efficient. In fact FL demonstrates an exponential decrease of (exchanged) data compared to CL. This holds for both studied network settings, namely in LTE and WLAN. Note that this behavior is not related to the number of training rounds, since in our simulation training stops upon dataset depletion. Interestingly, a relative-equilibrium area exists (2<$r_{data}$<3), where CL and FL share similar network resource utilization profiles.

An increased $r_{data}$ *i.e.*, a setup where few clients hold more data, would also benefit FL in terms of traffic loss, which basically represents unnecessary bandwidth usage from communication failures due to client mobility. Note that our setup does not focus on fairness or resilience aspects *e.g.*, replacing a failed node or investigation of asynchronous schemes [36]; it rather studies the effect of mobility on the default CL/FL setup, where failed nodes are excluded from the training/aggregation phases, respectively. As a result, performing FL with $r_{data} >> 3$ results in a traffic loss reduction up to 67% for LTE (see Fig. 5a) and up to 41% for WLAN (see Fig. 5b), as opposed to FL with $r_{data} < 2$. This reduction relates to the fact that in FL the client only exchanges ML models (of fixed size $m$) with the cloud server, regardless the actual amount of (raw) data he holds. Thus, a lower number of participating clients (for $r_{data} >> 3$) results in a lower number of exchanged ML models (*i.e.*, exchanged MBytes) and therefore decreased data loss. In FL (unlike CL) the traffic overhead and traffic loss accordingly is not dictated by the size of each client dataset, but by the number of the total participants. Another point to mention is that FL on a WLAN channel exhibits on average 40% more data loss compared to FL with LTE; at the same time, data loss in WLAN has larger confidence intervals compared to LTE. These observations reflect the average time a client is likely to remain (connected) in a service area; for WLAN that is by nature limited and erratic, as opposed to LTE, where users can potentially be always-on.

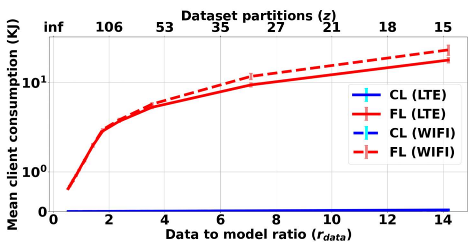
Contrary to the FL case, traffic loss is close to zero in CL, unless $r_{data} > 10$, where it reaches a maximum of 10%. In CL, no local processing (which is a time-consuming task in a resource-limited UE device) is required from the mobile clients; therefore delays may only occur due to transmission, which in turn minimizes the probability of a device drop. On a more general remark for both ML schemes in LTE and WLAN, the traffic loss reaches a maximum of 16% of the total traffic overhead; this low percentage of losses is also reflected in the consistently almost-negligible confidence interval size of all graphs in Fig. 4b. Thus, the overall bandwidth expenditure (traffic overhead) appears (under the considered conditions) significantly more sensitive to the portion of data ($r_{data}$) each user holds, rather than his own mobility.

***Impact of $r_{data}$ on the energy consumption:***

Investigating the overall energy expenditure from the client's perspective, as analyzed in paragraph 3.5, our experimentation suggests that CL outperforms FL for all values of $r_{data}$. Specifically, if FL is employed the clients collectively consume 300 times more energy in the LTE case (that is 350 for WIFI), as opposed to CL (Fig. 6a). The CL energy-efficiency (compared to FL) in the clients stems from the fact that it is the processing (*i.e.*, on-device ML training in FL) rather than data transmission which constitutes the prime factor for a UE's energy expenditure and thus its battery depletion. In fact, the total energy consumption in FL due to processing (proc) is 100 times higher (Fig. 7a) compared to the total energy consumption due to

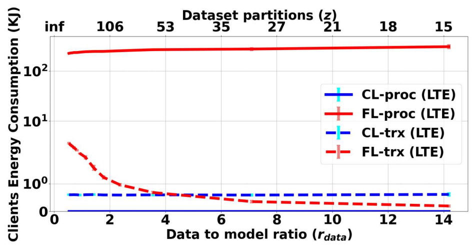


(a) Total client energy consumption

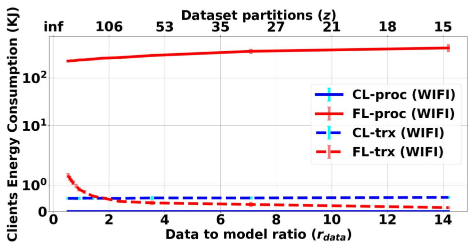


(b) Mean client energy consumption

Figure 6: Total and mean client energy consumption w.r.t. data to model ratio



(a) LTE

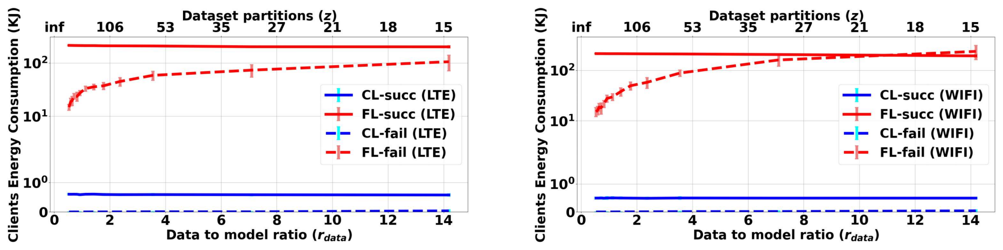


(b) WIFI

Figure 7: Client energy consumption w.r.t. data to model ratio, broken down to processing (proc) and transmission/reception (trx)

transmission-reception (trx) in the LTE case (that is 280 times in WLAN - Fig. 7b). Since CL does not require local processing in the clients side, any minor gains in energy consumption due to transmission in FL (FL-trx) for larger $r_{data}$ values (Fig. 7) do not affect the overall ML schemes expenditure comparison. These minor gains stem from the fact that less ML models are exchanged in FL for higher $r_{data}$ values.

A secondary point to note is that the average per client energy expenditure (*i.e.*, the total energy expenditure divided by the number of clients) increases linearly as $r_{data}$ increases (Fig. 6b - note the logarithmic scale in y-axis). For our configuration, the mean client consumption varies in the interval $[0.64, 20.7]KJ$ for LTE and $[0.81, 28.8]KJ$ for WLAN. The fact that client consumption in WIFI is consistently increased compared to LTE stems from WLAN's lower data rates (see paragraph 3.1), combined with its higher energy transmission costs (see paragraph 3.5). Assuming a modern smartphone's typical battery [37] with 2400 mAh/3.8 V, resulting in 32.8 KJ battery capacity, we deduce that in the worst-case scenario of our configuration ($r_{data} \approx 14$), 63% of each UE's battery energy is depleted in the LTE



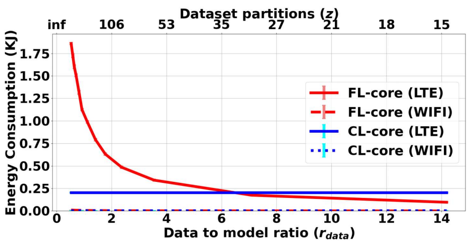
(a) LTE (b) WIFI

Figure 8: Client energy consumption w.r.t. data to model ratio, broken down to successful (succ) and failed (fail) communication

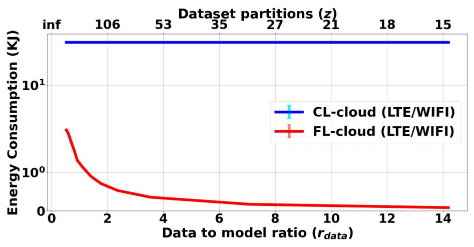
case (that is 88% for WLAN).

The results so far suggest that for $r_{data} > 4$, FL is on average 24% more energy efficient in an LTE setting, as opposed to WLAN (Fig. 6). However, LTE's energy-efficiency does not appear in the CL case, although transmissions of large size datasets (instead of ML models) occur. As portrayed in Fig. 8, LTE's energy-efficiency is related to processing and specific to energy loss (see paragraph 4.1) *i.e.*, energy consumed for local processing, but followed by a communication failure. Since more communication failures occur in a WLAN setting compared to LTE, as discussed earlier in paragraph 4.3, the amount of Energy Loss (and thus total energy expenditure) is higher. In fact, energy loss is largely affected by $r_{data}$; larger $r_{data}$ values represent larger per client dataset sizes implying higher energy consumption to perform training over these datasets. Thus, a potential communication failure of a client with a large $r_{data}$ essentially means larger energy loss compared to a client with smaller $r_{data}$. In FL, energy loss amounts to less than 16% of the total expenditure for $r_{data}$ <2, while reaches 30% for $r_{data}$ >10 in the LTE case (Fig. 8a). The respective WLAN values are 18% and 53% (Fig. 8b).

Unlike the client energy expenditure, both the core network's and the cloud's energy consumption exhibit an exponential reduction as $r_{data}$ increases (Fig. 9). For the core network, this reduction is only seen in the LTE case; the difference between the LTE-WLAN is mainly shaped by the energy expenditure in the network elements between the access and the metro & edge network, namely the BS and the AP respectively (see paragraph 3.1). Interestingly for the LTE case, a value exists for $r_{data} \approx 7$, where an equilibrium occurs between FL and CL (Fig. 9a). In any case, core expenditure in LTE is at least two orders of magnitude greater compared to WLAN, regardless the

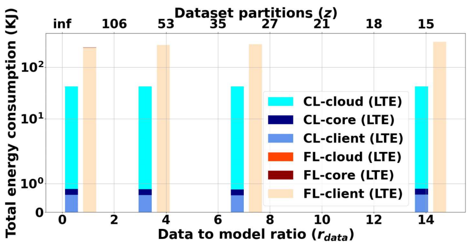


(a) Core network

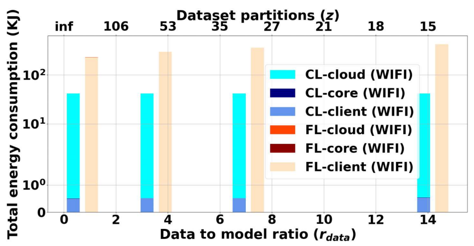


(b) Cloud processing

Figure 9: Core and cloud energy consumption w.r.t. data to model ratio



(a) LTE



(b) WIFI

Figure 10: End-to-end energy consumption for all network stakeholders

ML scheme. As expected, the cloud's consumption is reduced, when the ML task is offloaded to the clients (Fig. 9b); that reduction varies between 95% and 99%, as $r_{data}$ increases from 0.5 up to 15 and relates to the total clients *i.e.*, the total (produced) ML models aggregated in the cloud. The overall (system-wise) energy footprint for each ML scheme in the LTE and the WLAN case is depicted in Fig.10. Overall, CL demonstrates a higher energy efficiency compared to FL, both in LTE (Fig.10a) and in WLAN (Fig.10b). For $r_{data} \approx 1$, CL outperforms FL by 83% in the LTE case and 82% in WIFI. As $r_{data} \rightarrow 15$ these values are increased to 86% and 90% respectively. CL's energy efficiency is mainly dictated by the lack of processing (as opposed to FL) in clients' devices.

***Discussion on the effect of*** $r_{data}$***:*** The overall performance of CL and FL as $r_{data}$ increases is summarized in Table 3. A setting with reduced $r_{data}$ values ($r_{data} \rightarrow 0$) *i.e.*, with many clients holding few data, greatly favors CL over FL. When $r_{data}$ increases ($r_{data} \rightarrow \infty$), several benefits emerge for FL. FL becomes bandwidth-efficient, outperforming CL. As a result, the clients (end-users) minimize their data consumption, which can be costly especially

Table 3: Effect of $r_{data}$ on CL/FL performance

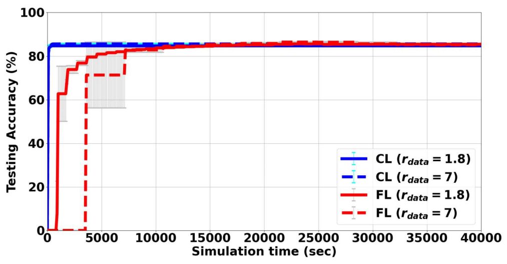
| Metric w.r.t. $r_{data}$ | CL | FL | Winner ($r_{data} \to 0$) | Winner ($r_{data} \to \infty$) |
|---|---|---|---|---|
| Achieved accuracy | Constant | Constant (via tuning) | None | None |
| Bandwidth consumption | Constant | Exponential decrease | CL | FL |
| Traffic loss | Constant | Linear decrease | CL | None |
| Total client energy consumption | Constant | Linear increase | CL | CL |
| Per client energy consumption | Constant | Linear increase | CL | CL |
| Energy loss | Constant | Linear increase | CL | CL |
| Core energy consumption | Constant | Exponential decrease | CL | FL |
| Cloud energy consumption | Constant | Exponential decrease | FL | FL |

in the LTE environment, given the limitations in the users' monthly data allowance. Furthermore, the network infrastructure is benefited, since bandwidth reservation and congestion are reduced. An increased $r_{data}$ also benefits both the network and the cloud infrastructure energy-wise. However, energy costs are distributed to the client devices, which can lead to client dropping, due to battery unavailability in a resource (battery)-constrained environment *e.g.*, in smartphones or IoT devices. Regardless, in presence of clients holding different volumes of data ($r_{data}$ varying across clients), elaborate client selection schemes can be employed to jointly optimize resource consumption for all network stakeholders (clients, network infrastructure, cloud infrastructure); that is subject to future exploration.

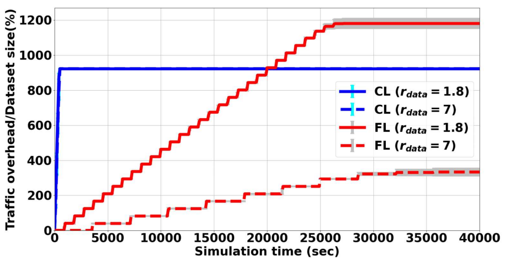
### *4.4. CL vs. FL: Convergence speed*

In this section, we will analyse how ML schemes evolve across time. We make use of the samples (experiments) described in paragraph 4.3, zooming on the time dimension per experiment, instead of the final resulted values; for example Testing Accuracy refers to the achieved accuracy per time instance. Capitalizing on the results from the aforementioned paragraph, we limit our investigation to two values for $r_{data}$, namely 1.8 and 7, representing a setting with more clients which hold few data and a setting with few clients holding considerably more data, respectively.

As depicted in Fig. 11a, CL reaches its maximum accuracy levels in the very first rounds (before 1K secs), utilizing the full dataset (Fig. 11b). At that time it outperforms FL in terms of accuracy by an average of 22%. CL's faster convergence however, comes at a cost. In CL, data exchange is performed in a bursty manner, which results in a consumption of 100% of CL's required bandwidth before 1K secs. At the same time, the cloud's energy is consumed early compared to FL (42 KJ in less than 1000 secs), due to ML processing in the cloud server (Fig. 12b). FL, on the other hand, proceeds

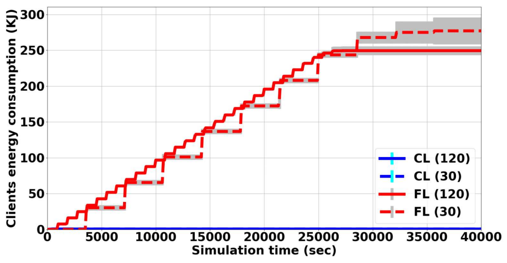


(a) Accuracy w.r.t. time

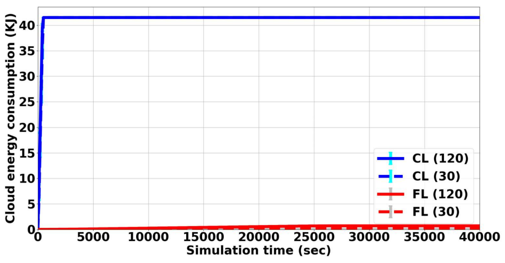


(b) Bandwidth consumption w.r.t. time

Figure 11: Evolution of accuracy and traffic overhead across simulation time



(a) Clients energy consumption w.r.t. time

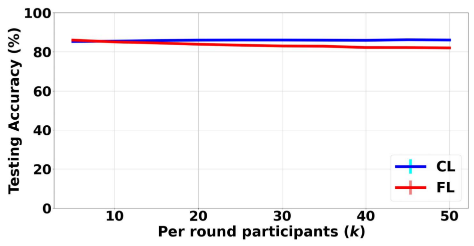


(b) Cloud energy consumption w.r.t. time

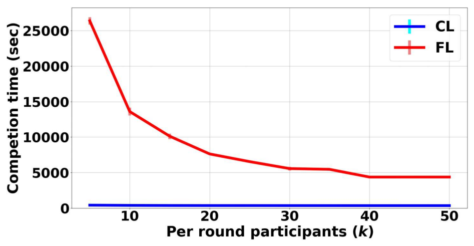
Figure 12: Evolution of energy consumption across simulation time

gradually (linear evolution) towards its completion (at 13K secs benchmark for $r_{data}$ =1.8 and 15K for $r_{data}$ =7) and so does the network's data expenditure (Fig. 11b) as well as the client energy consumption (Fig. 12a). Since FL's accuracy reaches its peak before the end of the experiments (Fig. 11a), any bandwidth or energy spent after the above-mentioned benchmarks has no obvious benefit. It is also observed that an increased $r_{data}$ does not affect CL's behavior; in FL however, the ML task is performed in a total of less client devices, therefore parallelism is decreased. As a result, the ML convergence, as well as the respective resource consumption footprint expands across time.

Overall, unless bandwidth limitations are posed in the system, CL is the preferable choice over FL, to accelerate the training process. As a result, CL could be utilized in ML applications whose data varies in time *e.g.*, time-series forecasting, to constantly produce up-to-date global ML models. Also, since CL has faster completion times, network resources are occupied for less time, leaving room for other applications and potentially avoid bottlenecks. FL,



(a) Resulted accuracy w.r.t. per round participants



(b) Task completion time w.r.t. per round participants

Figure 13: Impact of increasing the number of participating clients per round

on the other hand could be employed for a more balanced transition towards the maximum achievable ML accuracy, when network resource economy is of greater importance compared to time *e.g.*, in data analytics. On top, FL could be further optimized via a mechanism that monitors the FL's accuracy improvement, avoiding the resource waste that occurs, when convergence is reached. Finally, a smart dynamic setting, similar to Hybrid learning [3] could be utilized, to adapt between CL-FL, according to the ML application's requirements and the system's available resources.

## *4.5. CL vs. FL: Varying the number of participating clients*

We now fix the number of clients/partitions to $z = 100$, or $r_{data} \approx 2$, an area where CL and FL demonstrate a similar bandwidth consumption profile; moreover, FL exhibits certain gains in terms of the client energy consumption (see paragraph 4.3). Our goal is to investigate how each ML scheme's accuracy is affected, when the number of participating clients per round $k$ varies. Essentially, a small value of $k$ represents a setting where few clients are selected in each round *e.g.*, due to low availability or client scarcity, therefore more communication rounds are required to complete the ML task. Vice versa, an increased $k$ suggests a client-rich setting, where more clients participate in each round. We vary $k$ from 5 clients per round (which was our initial setting) up to 50, with a step of 5. Each experiment is repeated for 10 different time periods/samples, taken from the LTE mobility dataset, resulting in a total of 100 pairs of CL-FL experiments.

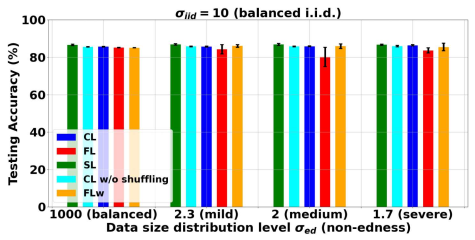
The results suggest that scaling the participants has practically no effect on CL's accuracy (Fig. 13a) nor on its completion time (Fig. 13b). FL's accuracy on the other hand is reduced up to 4%, when $k$ increases.

This result is in line with [5], [11], where it is shown that increasing parallelism (*i.e.*, allowing for more per round participants and decreasing the total rounds) degrades the overall accuracy, due to the smaller number of aggregation (FedAvg) steps. In fact, it is suggested that an optimal $k$ value exists in FL's client selection process, depending on the total number of online clients, the selected ML hyperparameters and the nature of the ML task. For image classification tasks like SVHN with a batch size of 128, the ratio of $k/z$ needs to be close to 0.1.

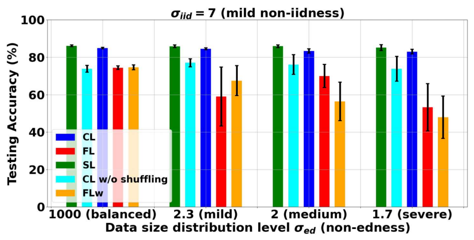
Interestingly, FL enjoys a nearly-exponential reduction in completion time, as $k$ (linearly) increases *i.e.*, an increase of $k$ from 5 to 50 reduces the completion time by 83% (Fig. 13b). The relatively small number of extra participating clients *i.e.*, no more than 50, should be easy to realise in a real-world system of hundreds of users *e.g.*, mobile devices in an urban cellular network. If $k \to 50$, the (completion) time gap between CL and FL is drastically decreased; at the same time FL is getting somewhat restricted, due to the reduction of the resulted accuracy. Such a trade-off is not present in the case of CL, whereby scaling the number of clients has no actual impact. A noteworthy closing remark involves the impact of scaling on the network resources consumption. The relevant quantities (see paragraph 4.3) increase analogously without any irregularities and therefore, we omit the corresponding plots.

## *4.6. CL vs. FL: Effect of data heterogeneity*

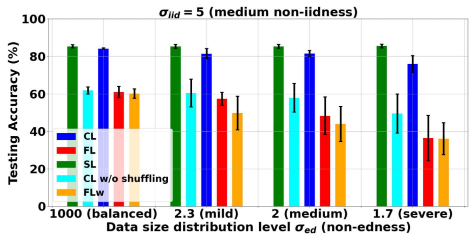
Thus far the SVHN dataset was assumed to be evenly distributed (in size) across the clients (e.d.) and that each client's subdataset is representative compared to SVHN (i.i.d.). We now investigate the ML schemes' response (in terms of achieved accuracy), when the above-mentioned assumptions are relaxed (see paragraph 3.3). For the non-e.d. setting, we assume four cases for the Zipf parameter $\sigma_{ed} \in \{1.7, 2, 2.3, 1000\}$, representing various degrees of data distribution skewness; as $\sigma_{ed} \to 1$, the skewness degree becomes severe. Similarly for the non-i.i.d. setting, we select four cases for the i.i.d. parameter $\sigma_{iid} \in \{3, 5, 7, 10\}$, which represent the total number of classes contained in a client's subdataset. SVHN contains samples from all 10 classes, thus a $\sigma_{iid} = 10$ is essentially an i.i.d. setting, while smaller $\sigma_{iid}$ values mark increasing non-i.i.d. conditions. We also assume a total number of clients $z = 100$ and fix the per round participants $k = 10$, as suggested by the results of paragraph 4.5. For comparison purposes, we include two extra centralized ML schemes besides CL *i.e.*, Standard Machine Learning (SL)



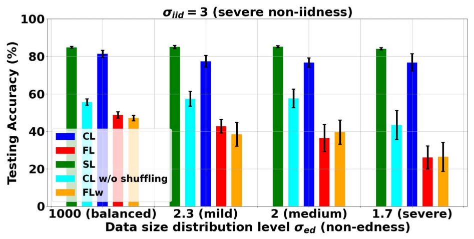
(a) Balanced i.i.d.



(b) Mild non-iidness



(c) Medium non-iidness



(d) Severe non-iidness

Figure 14: Effect of client data size (e.d.) and content (i.i.d.) variations

and CL without (w/o) shuffling and one additional federated *i.e.*, Federated Learning with weighted averaging (FLw). For SL, the server waits for the majority (at least 80%) of SVHN data to arrive and then performs training. CL without shuffling is identical to CL, however no data shuffling occurs prior to training. FLw uses a weighted averaging scheme during aggregation, as opposed to FL's uniform aggregation (see Sec. 3.6). Each experiment is repeated 10 times, using LTE traces, resulting in a total of 160 sextuples of {SL, CL w/o shuffling, CL, FL, FLw} experiments.

From all centralized ML schemes (SL, CL and CL w/o shuffling), SL exhibits the highest performance, reaching an accuracy of 85% (Fig. 14). SL's performance is not affected by the various degrees of non-edness or non-iidness, essentially reflecting the advantages of centralized ML under heterogeneous environments [7]. In SL, the cloud server waits for almost all subdatasets to arrive, before merging them into a super-dataset and initiating the ML training. Therefore, any stochasticity introduced in the (distributed) client subdatasets, either in size or content, is eliminated when the merging occurs. As discussed in paragraph 3, SL is impractical in a real system, so it is merely used for comparison purposes.

CL w/o shuffling can be regarded as a dynamic alternative to SL, since training is performed in each communication round, using the collected round's client data. Under an i.i.d. setting (Fig. 14a), CL w/o shuffling achieves similar performance to SL, which is not affected by the data size distribution (non-edness). When non-iidness appears and as it increases (Fig. 14b-Fig. 14d), CL w/o shuffling exhibits lower accuracy levels (down to 56%). The introduction of non-edness further diminishes its performance, which drops down to 44% for severe non-i.i.d. and non-e.d. conditions (Fig. 14d). This degradation is related to the distributed manner of client data and the fact that ML training is performed in each round. Under high data heterogeneity, consecutive series of non-representative data samples can be uploaded in the cloud server *e.g.*, several batches containing only one SVHN class out of ten. Such bias is likely to have a detrimental effect on the ML training task. Moreover, given that each round's ML model is used for the next round training, any abnormalities will be cascaded, resulting in less accurate ML models. To mitigate those negative effects, we apply and experimentally evaluate a popular technique prior to ML training *i.e.*, shuffling [34]. CL with shuffling is referred to as CL, for simplicity. As portrayed in Fig. 14, CL achieves the same or higher accuracy levels compared to CL w/o shuffling. In fact, SL only outclasses CL by a maximum of 2% for mild (Fig. 14a-14b) and 7% for major (Fig. 14c-14d) data heterogeneity respectively.

FL on the other hand, being a distributed learning scheme is affected both from non-iidness and non-edness; even the introduction of small levels of non-iidness can lead to accuracy drops down to 53% (Fig. 14b). When in fact non-iidness is combined with non-edness, the training task cannot converge (Fig. 14d), therefore accuracy drops below 40%. Such performance is related to FL's training algorithm (uniform FedAvg). As such, a ML model trained with high diversity data will be equally weighted with another model (from another client) trained with low diversity (biased) data. FLw (FedAvg with weighted averaging) introduces a simple bias mitigation technique by rewarding ML models trained with larger datasets with larger weights. When non-iidness is low, this technique enables FLw to achieve an up to 11% better accuracy compared to FL for various levels of non-edness (Fig. 14a-14b). However, under the presence of higher levels of non-iidness (Fig. 14c-14d) FL achieves similar or (up to 8%) better accuracy compared to FLw. This is because weighted averaging (FLw) rewards ML models according to their training data quantity (larger datasets have larger weights) and not quality *i.e.*, data diversity, entropy, number of classes in the dataset *etc.* As such, not only FLw is prone to non-iidness (similarly to FL), but can potentially demonstrate erratic behavior [38]. Overall, it becomes evident that applying

FL over highly heterogeneous data environments requires advanced statistical methods that deserve a dedicated exploration.

## 5. Conclusions

We have identified the need for deeper understanding of the underlying network resources' relevance with the ML-schemes running on-top; in response, we have contributed an end-to-end systematic analysis. A novel measurement-based, pragmatic model has been introduced to account for the end-to-end network resources and energy consumption involved when CL and FL compete on image-classification tasks. User mobility patterns are incorporated into the model via real-world traces to capture the environment's dynamicity while different channels (LTE and WLAN) facilitate the involved communications. The main results of our experimentation with the SVHN dataset, using simulation code that we release, are summarized in the following points:

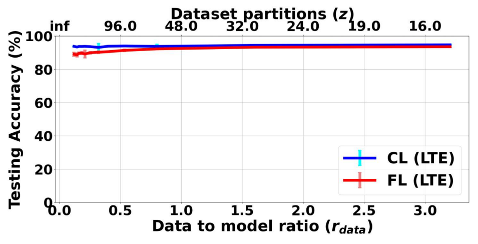
- Hyperparameter tuning prior to ML training and specifically, configuring the number of training epochs is needed to ensure FL convergence, especially when clients local datasets are less than 2 times the ML model's size. Incrementing the number of local FL epochs yields comparable accuracy to CL, (linearly) increasing the client energy consumption.
- When clients hold large datasets compared to the ML model, several benefits emerge for the infrastructure, if FL is used; 1) bandwidth expenditure is exponentially decreased and so do the cloud energy needs, as well as the core network's energy expenditure (only in LTE), 2) data loss due to mobility reduces for at least 40%.
- FL clients' energy consumption is dominated by ML processing which appears multiple times more energy-hungry than transmission, regardless the communication channel (LTE/WLAN). Large local datasets (more than 10 times the size of the ML model) deplete at least 63% of an average UE's battery, potentially discouraging client participation in FL.
- CL, which demonstrates a 13 times greater convergence speed compared to FL, emerges as a candidate to minimize the allocation time of system devices by ML tasks. This comes at the expense of energy (in the cloud) and bandwidth consumption spikes. FL, on the other hand achieves similar accuracy in a more progressive and bandwidth-efficient manner.
- Confirming previous studies, a speed-up in FL's convergence time is observed when more clients are involved in the training process (per round)

at the expense of a slight downgrade on the resulted accuracy.

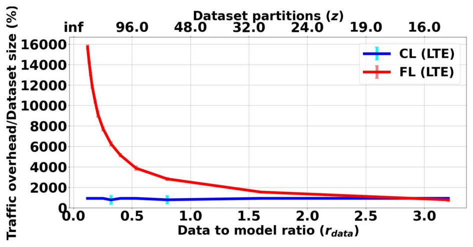
- FL exhibits a large accuracy degradation under data asymmetry (in size or content); for CL, however, this can be mitigated, if shuffling techniques are introduced prior to the centralized ML training.

Our work also reveals some interesting future research directions that can be explored using our released simulation software; the role of training data-to-model ratio deserves an exhaustive study over more complex models both in terms of size and neural-network architecture. Data acquisition rate and client storage constraints could increase the level of realism, if considered. Bias (non-iidness) mitigation algorithms for FL constitute another research direction. Finally, putting more dynamic ML schemes, such as Hybrid Learning under the microscope could provide further insights for those parameters shaping the ML scheme selection in resource-constrained network environments.

## Appendix A. Impact of $r_{data}$: From ANN to CNN model



(a) Achieved testing accuracy for CNN model



(b) Normalized traffic overhead for CNN model

Figure A.15: Testing accuracy and normalized traffic overhead w.r.t. data to model ratio for CNN model

In this section we run a number of of experiments as described in the setup of Sec. 4.3, using a more complex ML model and specifically a custom (untrained) Convolutional Neural Network (CNN). The model comprises of two convolutional layers of sizes (3,32,5) and (32,256,5), followed by three dense layers of sizes (6400,1024), (1024,128) and (128,10), reaching a total size of 27 Mbytes. Note that our initial model had a size of 6 Mbytes. Our aim is essentially to explore the performance of a larger model and verify that the relevant behavior resembles the trends observed in our main experimentation.

As depicted in Fig. A.15a the CNN model achieves an average of 93% accuracy for FL and 94% for CL. That is an improvement of 8% and 9% respectively over the original linear (ANN) model. At the same time, traffic overhead is increased for FL (up to 4.5 times more compared to our initial setup). As expected, bandwidth consumption for FL follows an exponential decrease, similar (but shifted to different $r_{data}$ values) to what was observed for the original model (Fig. A.15b). Such behavior provides further evidence for the validity of our prior results.

On a final, more practical note, the above performance figures require considerable resources to be achieved (translated into over 300% more time to perform a single training epoch compared to the original model). This implies that any deployment of an FL solution (over hand-held devices of limited resources) would call-for the usage of small-sized models at the expense of slightly reduced accuracy.

## References

[1] J. Hestness, et al., Beyond human-level accuracy: Computational challenges in deep learning, in: Proceedings of the 24th Symposium on Principles and Practice of Parallel Programming, 2019, pp. 1–14.

[2] W. Shi, et al., Edge computing: Vision and challenges, IEEE Internet of Things Journal 3 (5) (2016) 637–646. doi:10.1109/JIOT.2016.2579198.

[3] W. Hong, et al., Optimal design of hybrid federated and centralized learning in the mobile edge computing systems, in: IEEE International Conference on Communications (ICC) Workshops, 2021, pp. 1–6.

[4] T. Wink, Z. Nochta, An approach for peer-to-peer federated learning, in: 2021 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W), IEEE, 2021, pp. 150–157.

[5] B. McMahan, et al., Communication-efficient learning of deep networks from decentralized data, in: Artificial Intelligence and Statistics, PMLR, 2017, pp. 1273–1282.

[6] X. Lian, et al., Can decentralized algorithms outperform centralized algorithms? A case study for decentralized parallel stochastic gradient descent, in: Advances in Neural Information Processing Systems, 2017, pp. 5330–5340.

[7] A. Nilsson, et al., A performance evaluation of federated learning algorithms, in: Proceedings of the Second Workshop on Distributed Infrastructures for Deep Learning, 2018, pp. 1–8.

[8] S. A. Rahman, et al., Internet of things intrusion detection: Centralized, on-device, or federated learning?, IEEE Network 34 (6) (2020) 310–317.

[9] K. Chandiramani, et al., Performance analysis of distributed and federated learning models on private data, Procedia Computer Science 165 (2019) 349–355.

[10] G. Drainakis, et al., Federated vs. centralized machine learning under privacy-elastic users: A comparative analysis, in: 2020 IEEE 19th International Symposium on Network Computing and Applications (NCA), pp. 1–8.

[11] T. Nishio, R. Yonetani, Client selection for federated learning with heterogeneous resources in mobile edge, in: ICC 2019-2019 IEEE International Conference on Communications (ICC), IEEE, 2019, pp. 1–7.

[12] I. Hegedűs, et al., Decentralized learning works: An empirical comparison of gossip learning and federated learning, Journal of Parallel and Distributed Computing 148 (2021) 109–124.

[13] G. H. Lee, S.-Y. Shin, Federated learning on clinical benchmark data: Performance assessment, Journal of medical Internet research 22 (10) (2020) e20891.

[14] S. Otoum, et al., Blockchain-supported federated learning for trustworthy vehicular networks, in: IEEE GLOBECOM, 2020, pp. 1–6.

[15] Warnat-Herresthal, et al., Swarm learning for decentralized and confidential clinical machine learning, Nature 594 (7862) (2021) 265–270.

[16] M. R. o. Akdeniz, Millimeter wave channel modeling and cellular capacity evaluation, IEEE journal on selected areas in communications 32 (6) (2014) 1164–1179.

[17] K. Gomez, et al., Achilles and the tortoise: Power consumption in IEEE 802.11 n and IEEE 802.11 g networks, in: IEEE Online Conference on Green Communications (OnlineGreenComm), 2013, pp. 20–26.

[18] M. Rizwan, S. A. Abbas, Median path loss, fading and coverage comparison at 3.5 ghz and 700mhz for mobile wimax, in: 2008 IEEE International Multitopic Conference, IEEE, 2008, pp. 266–271.

[19] M. J. Crisp, et al., Uplink and downlink coverage improvements of 802.11 g signals using a distributed antenna network, Journal of Lightwave Technology 25 (11) (2007) 3388–3395.

[20] A. Vishwanath, et al., Energy consumption comparison of interactive cloud-based and local applications, IEEE Journal on selected areas in communications 33 (4) (2015) 616–626.

[21] Y.-C. Chiu, et al., Are we one hop away from a better internet?, in: Procs. of 2015 Internet Measurement Conference (IMC), pp. 523–529.

[22] S. Wang, et al., Edge server placement in mobile edge computing, Journal of Parallel and Distributed Computing 127 (2019) 160–168.

[23] M. Lenczner, A. G. Hoen, CRAWDAD dataset ilesansfil/wifidog (ver. 2015-11-06), Downloaded from: https://crawdad.org/ilesansfil/wifidog/20151106/session.

[24] Y. Yan, et al., Risk minimization against transmission failures of federated learning in mobile edge networks, IEEE Access 8 (2020) 98205–98217.

[25] J. Liu, et al., Performance analysis and characterization of training deep learning models on mobile device, in: 2019 25th IEEE International Conference on Parallel and Distributed Systems, pp. 506–515.

[26] Y. Kochura, et al., Batch size influence on performance of graphic and tensor processing units during training and inference phases, in: Intern'l Conf. on Computer Science, Engineering and Education Applications, Springer, 2019, pp. 658–668.

[27] A. Nika, et al., Energy and performance of smartphone radio bundling in outdoor environments, in: Proceedings of the 24th International Conference on World Wide Web, 2015, pp. 809–819.

[28] N. P. Jouppi, et al., In-datacenter performance analysis of a tensor processing unit, in: Proceedings of the 44th Annual International Symposium on Computer Architecture, 2017, pp. 1–12.

[29] T. Jakobs, et al., Reducing the power consumption of matrix multiplications by vectorization, in: International Conference on Computational Science and Engineering (CSE), IEEE, 2016, pp. 213–220.

[30] J. Luketina, et al., Scalable gradient-based tuning of continuous regularization hyperparameters, in: International conference on machine learning, PMLR, 2016, pp. 2952–2960.

[31] M. Courbariaux, et al., Binaryconnect: Training deep neural networks with binary weights during propagations, in: Advances in neural information processing systems, 2015, pp. 3123–3131.

[32] A. Ziller, et al., Pysyft: A library for easy federated learning, in: Federated Learning Systems, Springer, 2021, pp. 111–139.

[33] PyTorch Neural Network API.
URL `https://pytorch.org/docs/stable/nn.html`

[34] Q. Meng, et al., Convergence analysis of distributed stochastic gradient descent with shuffling, Neurocomputing 337 (2019) 46–57.

[35] M. M. Bejani, M. Ghatee, A systematic review on overfitting control in shallow and deep neural networks, Artificial Intelligence Review (2021) 1–48.

[36] Y. Chen, et al., Asynchronous online federated learning for edge devices with non-iid data, in: 2020 IEEE International Conference on Big Data (Big Data), IEEE, 2020, pp. 15–24.

[37] Samsung specifications: Eb-bg531bbe battery - 2400mah li-ion.
URL `https://batteriesglobal.com/products/samsung-eb-bg531bbe-battery`

[38] X. Zhang, M. Hong, S. Dhople, W. Yin, Y. Liu, Fedpd: A federated learning framework with adaptivity to non-iid data, IEEE Transactions on Signal Processing 69 (2021) 6055–6070.