Received 28 February 2023; revised 12 April 2023; accepted 1 May 2023. Date of publication 9 May 2023; date of current version 17 May 2023.

Digital Object Identifier 10.1109/OJCOMS.2023.3274394

The Cost of Training Machine Learning Models Over Distributed Data Sources

ELIA GUERRA [1] (Student Member, IEEE), FRANCESC WILHELMI [2] (Member, IEEE), MARCO MIOZZO [1], AND PAOLO DINI [1]

[1] Sustainable Artificial Intelligence (SAI), Centre Tecnològic de Telecomunicacions de Catalunya (CTTC/CERCA), 08860 Castelldefels, Spain

[2] Radio Systems Research, Nokia Bell-Labs, 70469 Stuttgart, Germany

CORRESPONDING AUTHOR: E. GUERRA (e-mail: eguerra@cttc.cat)

This work was supported in part by the Spanish Project PID2020-113832RB-C22(ORIGIN)/MCIN/AEI/10.13039/50110001103; in part by the European Union Horizon 2020 Research and Innovation Programme under Grant 953775 (GREENEDGE); and in part by CHIST-ERA-20-SICT-004 (SONATA) under Grant PCI2021-122043-2A/AEI/10.13039/501100011033.

ABSTRACT Federated learning is one of the most appealing alternatives to the standard centralized learning paradigm, allowing a heterogeneous set of devices to train a machine learning model without sharing their raw data. However, it requires a central server to coordinate the learning process, thus introducing potential scalability and security issues. In the literature, server-less federated learning approaches like gossip federated learning and blockchain-enabled federated learning have been proposed to mitigate these issues. In this work, we propose a complete overview of these three techniques, proposing a comparison according to an integral set of performance indicators, including model accuracy, time complexity, communication overhead, convergence time, and energy consumption. An extensive simulation campaign permits to draw a quantitative analysis considering both feedforward and convolutional neural network models. Results show that gossip federated learning and standard federated solution are able to reach a similar level of accuracy, and their energy consumption is influenced by the machine learning model adopted, the software library, and the hardware used. Differently, blockchain-enabled federated learning represents a viable solution for implementing decentralized learning with a higher level of security, at the cost of an extra energy usage and data sharing. Finally, we identify open issues on the two decentralized federated learning implementations and provide insights on potential extensions and possible research directions on this new research field.

INDEX TERMS Blockchain, decentralized learning, edge computing, energy efficiency, federated learning, machine learning.

I. INTRODUCTION

MACHINE learning (ML) models, and in particular deep neural networks, require a substantial amount of data and computational power that might not be available on a single machine. As a consequence, ML operations are normally run at cloud servers (or data centers), where batteries of powerful processing units enable short training and inference computation times. However, training ML models in a data center requires moving data from the information sources (e.g., edge devices) to the central system. This approach runs into several issues:

- *Communication overhead:* Nowadays, the huge pervasiveness of mobile services, devices, and network infrastructures makes data sources mainly distributed. As testified recently by the Ericsson Mobility Report [1], mobile network data traffic grew exponentially over the last 10 years, with a remarkable increase of 42% between Q3 2020 and Q3 2021. Mobile data traffic is projected to grow by over 4 times to reach 288 EB per month by 2027 [1]. Moving such a big amount of data from distributed sources to a central location for ML operations may create network congestion and service outage.

- *Latency:* In several real-life scenarios, transmitting data requires a stable and reliable connection to minimize latency and ensure updated models, which cannot be always guaranteed. For example, minimizing communication latency in connected vehicles is essential to guarantee road safety [2].
- *Energy consumption:* Running ML models in cloud data centers consumes a significant amount of energy and cannot be considered sustainable from an environmental perspective. As reported in [3], from 2012 to 2018, the computations required for training a deep learning (DL) model have been doubling every 3.4 months, with an estimated increase of 300000x. Estimates show that training a state-of-the-art natural language processing model produces more CO_2 than an average car in one year lifetime [4].
- *Privacy:* With the growing awareness of data privacy and security, it is often undesirable, or even unfeasible, to collect and centralize users' data [5]. For instance, a single hospital may not be able to train a high-quality model for a specific task on its own (due to the lack of data), but it cannot share raw data due to various policies or regulations on privacy [6]. Another example could be the case of a mobile user that would like to employ a good next-word predictor model without sharing his/her private historical text data.

A. EDGE AI AND FEDERATED LEARNING

To address the challenges stemming from cloud-based centralized ML, edge computing pushes cloud services to the network edge and enables distributed ML operations, i.e., the so-called edge intelligence [7]. In particular, AI on Edge [8] is the paradigm of running AI models with a device-edge-cloud synergy. It allows to relax the massive communication requirements and privacy of cloud-based ML operations [9]. Moreover, distributing ML computation over the edge has been demonstrated to save up to the 25% of the energy consumption [10]. In fact, data may be directly processed at the edge with smaller and more energy efficient devices (no need of cooling systems) and the energy cost related to communication is limited due to unnecessary data transmission.

Among the several training paradigms enabled by edge intelligence, federated learning (FL) has emerged as a popular solution by providing low communication overhead, enhanced user privacy, and security to distributed learning [11]. With FL, the ML model is trained cooperatively by edge devices without sharing local data, but only exchanging model parameters. Some prominent applications for FL can be found in [12], [13]. The usual implementation envisages an iterative procedure whereby a central server collects local updates from the clients (e.g., edge devices) and returns an aggregated global model. In the rest of the paper, we refer to centralized FL (CFL) to the traditional server-dependent FL scenario. In this setting, the server has to wait for all the clients before returning a new global update. Therefore, high network latency, unreliable links, or straggled clients may slow down the training process and even worsen the model accuracy [14]. In addition, the central server represents a single point of failure, i.e., if it becomes unreachable due to network problems or an attack, the training process cannot continue. Furthermore, it may also become a bottleneck when the number of clients is very large [15].

Decentralized and server-less solutions for federated learning have been introduced in the literature, mainly to overcome the single point of failure and the security problems [16], [17]. In [18] a decentralized FL mechanism was proposed by enabling one-hop communication among FL clients. Similarly, gossip FL (GFL) extends device-to-device (D2D) communications to compensate for the lack of an orchestrating central server [19], [20]. It guarantees a low communication overhead thanks to the reduced number of messages [21].

Beyond, we find more sophisticated proposals, like blockchain-enabled federated learning (BFL), which adopts blockchain to share FL information among devices, thus removing the figure of the orchestrating central server. In this way, blockchain removes the single point of failure for the sake of openness and decentralization and provides enhanced security via tampered-proof properties [22].

B. CONTRIBUTIONS

Despite in the literature it is possible to find papers comparing classical centralized learning in data center with CFL [23], [24], a comparison among the different federated learning approaches (centralized versus decentralized) is still missing. In this work, we aim to fill this gap and, thus, we focus on two of the most popular and widely adopted approaches for decentralizing FL: GFL and BFL. In particular, we provide a comprehensive analysis of both methods and compare them to traditional FL, i.e., CFL. Note that we combine standard performance indicators for ML models, i.e., accuracy, with indicators that quantify the efficiency of these algorithms, i.e., time complexity, communication overhead, convergence time, and energy consumption. With our comparison under fair conditions, we would like to provide the research community with a complete overview of the three approaches, so that the best model can be chosen according to the specific use cases.

The contributions of this paper may be summarized as follows:

- We overview the traditional FL setting and delve into two approaches for decentralizing it. They are selected since are two of the most popular in the literature and are kind of diverging into two completely different solutions, which are based on gossip communication and blockchain technology, respectively.
- We provide a thorough analysis to derive the running time complexity, the communication overhead and the convergence time of each overviewed mechanism for FL, including CFL, BFL, and GFL.

- We provide an energy model to measure the energy consumption of each solution, based on the associated communication and computation overheads.
- We assess the performance of each method (CFL, GFL, and BFL) through extensive simulations on widely used TensorFlow libraries [25].
- We delve into the open aspects of decentralized FL, providing insights on potential extensions, considerations, and software implementations for GFL and BFL.

The remainder of the paper is structured as follows: Section II reviews the related work. Section III describes the three studied algorithms (CFL, BFL, and GFL). Section IV analyzes their time complexity, the communication cost, introduces the communication model and the convergence time. Section V provides the energy model used in this paper. Then, Section VI compares the three mechanisms through simulation results. In Section VII, we propose solutions to address common BFL and GFL weaknesses. Section VIII provides some open issues of GFL and BFL and future research directions. Finally, Section IX concludes the paper with final remarks.

II. RELATED WORK

Distributing and decentralizing ML operations at the edge has been embraced as an appealing solution for addressing the issues of centralization (connectivity, privacy, and security) [26]. With FL, different devices collaborate to train an ML model by sharing local updates obtained from local and private data. The traditional FL algorithm (FedAvg), referred to as CFL in this paper, is introduced in [27]. In [11], the authors propose techniques to improve its communication efficiency. Nevertheless, CFL still requires a central server responsible for clients orchestration and model aggregation. The star topology is a weak aspect of CFL, since the central entity represents a single point of failure, it may limit the number of devices that can participate in the training process, augments the communication cost of the learning process, and presents privacy concerns [15], [28].

An extension of CFL, is proposed in [29] where a hierarchical aggregation scheme is adopted, i.e., a subset of edge devices aggregate the local updates shared by their neighbors. Further optimization for wireless networks is taken into account in [30], where wireless resource allocation and client selection are jointly considered to minimize CFL loss.

To address these challenges, decentralized federated learning has been proposed in [18]. The authors present a fully decentralized model, in which each device can communicate only with its one-hop neighbors. They also provide a theoretical upper bound on the mean square error. Ormándi et al. [19] introduce gossip FL, a generic approach for peer-to-peer (P2P) learning on fully distributed data, i.e., every device has only a single input sample to process at each round. The same algorithm has been tested in [20] under real-world conditions, i.e., devices have multiple input samples available (rather than only one point, as originally stated in [19]), restricted communication topology, heterogeneous communication and computation capabilities. Removing the central server brings new challenges and opportunities. On the one hand, GFL addresses scalability properly and removes the single point of failure problem of CFL. On the other hand, the lack of coordination in GFL may lead to high temporal variability and ML model inconsistencies (e.g., nodes may have different versions of the model stored in their local cache). The fact is that, in GFL, the participating nodes interact with each other in a distributed manner. This, for instance, hinders the consensus on the ML model at the beginning of each FL round, which gains in difficulty as the number of participants increases. Moreover, network topologies with sparsely connected clients may further degrade GFL performance [20].

Another prominent solution to decentralize FL is blockchain-enabled FL [31], [32], [33]. A blockchain system allows clients to submit and retrieve model updates without the central server. Additionally, the usage of blockchain guarantees security, trust, privacy, and traceability, however, it introduces delays due to the distributed ledger technology. An analysis of end-to-end latency and the effects of blockchain parameters on the training procedure of BFL is proposed in [22].

In the literature, there exist some comparisons across FL techniques. The authors of [34] compare GFL and CFL with a logistic regression model in terms of convergence time, proportion of the misclassified examples in the test set (0-1 error), and used communication resources. When nodes have a random subset of the learning samples, GFL performance is comparable with CFL; instead, CFL converges faster when a node has only labels from one class. Another comparison is proposed in [35], where the performance of FL algorithms that require a central server, e.g., FedAvg and Federated Stochastic Reduced Gradient are analyzed. Results show that FedAvg achieves the highest accuracy among the FL algorithms regardless of how data are partitioned. In addition, the comparison between FedAvg and the standard centralized algorithm shows that they are equivalent when independent and identically distributed (IID) datasets are used.

In [21], the authors compare GFL with the standard centralized data center based architecture in terms of accuracy and energy consumption for two radio access network use-cases. To achieve this goal, they use the machine learning emission calculator [36] and green algorithms [37]. In [23], the authors compare centralized data center based learning and CFL in terms of carbon footprint using different datasets. The assessment is done by sampling the CPU and GPU power consumption. In [24], the authors propose a framework to evaluate the energy consumption and the carbon footprint of distributed ML models with focus on industrial Internet of Things applications. The paper identifies specific requirements on the communication network, dataset and model size to guarantee the energy efficiency of CFL over centralized learning approaches, i.e., bounds on the local dataset or model size. Differently from our work, the

authors do not consider Blockchain-enabled FL and evaluate the algorithm performance in scenarios with small number of devices (i.e., 100). In addition, we empirically measure the energy consumption of the devices based on the real load of the computations realized during the training phase; we provide a communication model to estimate overhead and convergence time. Finally, here we introduce an analysis on the computational complexity of the three federated algorithms under study.

To sum up, in this paper, we endeavor to bridge the existing gap in the literature by providing a thorough comparison including performance analysis and cost of the different federated approaches listed above, i.e., CFL, BFL, and GFL. Differently from the other works in the literature, we combine standard metrics, i.e., accuracy, with indicators of the efficiency of these algorithms, i.e., computational complexity, communication overhead, convergence time and energy consumption. Our final aim is to contribute to the development of Green AI [38].

III. FEDERATED LEARNING IMPLEMENTATIONS

Let us consider a set of N clients (or devices) $\mathcal{N} = \{1, \ldots, N\}$ with their datasets $D_1, \ldots, D_N$. Each local dataset $D_i, \forall i \in \mathcal{N}$, contains pairs (x_i, y_i), where x_i is the feature vector, and y_i its true label. The goal of a federated setting is to train a global model (e.g., a set of weights w), that minimizes the weighted global loss function:

$$\ell = \sum_{i=1}^{N} \frac{|D_i|}{|D|} \ell_i(w, x_i, y_i), \tag{1}$$

where ℓ_i represents the local loss experienced by client i and $|D| = \sum_{i=1}^{N} D_i$. In this scenario, devices do not share raw local data with other devices. Instead, they exchange model parameter updates, computed during several iterations by training the global model on local data. In this paper, we study three different implementations to solve the federated problem stated above, namely: CFL, BFL, and GFL. The investigated solutions are depicted in Fig. 1 and we will introduce them in what follows. Though several variants are available in the literature, the three algorithms described next are baseline representations of the approaches studied and well suitable for our purposes.

A. CENTRALIZED FEDERATED LEARNING (CFL)

At the beginning of a round t, a random subset of m devices $\mathcal{S}^t \subseteq \mathcal{N}$ is selected, and the server sends the current global model to the parties. Each client makes E epochs on the local dataset with a mini-batch size of B, updates its local model w_k^{t+1} and sends it to the server. The server aggregates the received local updates and generates the new global model by computing the weighted average of the local updates, as follows:

$$w^{t+1} = \sum_{k \in \mathcal{S}^t} \frac{|D_k|}{|D|} w_k^{t+1}, \tag{2}$$

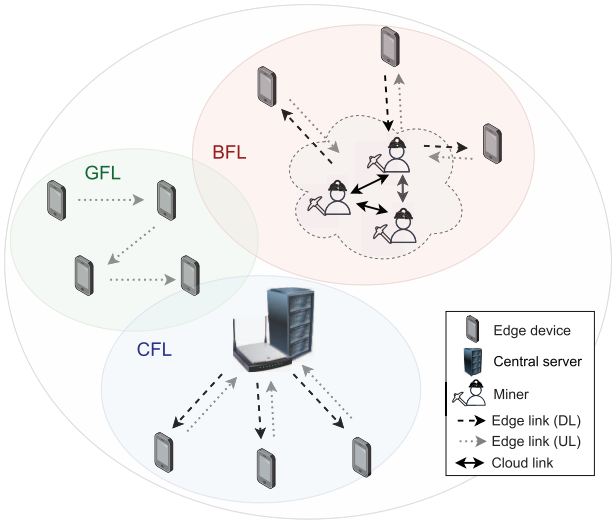


FIGURE 1. Overview of the different FL scenarios.

where $|D| = \sum_{k \in \mathcal{S}^t} |D_k|$. The process is repeated until the model reaches convergence, e.g., the loss function does not improve significantly across subsequent epochs or a specific number of training rounds have been executed. In this work, we consider the FedAvg algorithm [27] as a merging method to generate global model updates.

Algorithm 1 describes the CFL with FedAvg mechanism. The procedure MAIN is executed by the server that coordinates the whole training process. Each client executes the procedure CLIENTUPDATE and applies the stochastic gradient descent (SGD) algorithm on its local dataset with a learning rate η.

B. BLOCKCHAIN-ENABLED FEDERATED LEARNING (BFL)

BFL is based on distributed ledger technology, which collects data in the form of transactions and organizes it into blocks. Indeed, a blockchain is a sequence of blocks chained one after another through advanced cryptographic techniques. Each block contains the hash value of the previous one, leading to a tampered-proof sequence and providing properties that are essential to building trust in decentralized settings, such as transparency and immutability. In a blockchain, a set of participant nodes (miners) apply certain mining protocols and consensus mechanisms to append new blocks to the blockchain and agree on the status of the same. This procedure allows devices to write concurrently on a distributed database and guarantees that any malicious change on data would not be accepted by the majority, so that data in a blockchain is secured.

When a blockchain is applied to a federated setting, the process is going as follows [22]:

1) Each device submits its local model updates in the form of transactions to the blockchain peer-to-peer (P2P) network of miners.
2) The transactions are shared and verified by miners.

Algorithm 1 CFL

procedure MAIN
  initialize w_0
  $t \leftarrow 0$
  while convergence is not reached **do**
    $\mathcal{S}^t \leftarrow$ random set of m clients
    for each client $k \in \mathcal{S}^t$ **in parallel do**
      Download the global model w^t
      $w_k^{t+1} \leftarrow$ CLIENTUPDATE(k, w^t)
      Send w_k^{t+1} to the server
    end for
    $w^{t+1} \leftarrow \sum_{k \in \mathcal{S}^t} \frac{|D_k|}{|D|} w_k^{t+1}$
    $t \leftarrow t + 1$
  end while
end procedure

procedure CLIENTUPDATE(k, w) ▷ Run on client k
  $\mathcal{B} \leftarrow$ split the local dataset into batches of size B
  for E local epochs **do**
    for batch $b \in \mathcal{B}$ **do**
      $w \leftarrow w - \eta \nabla \ell(w, b)$
    end for
  end for
  return w
end procedure

Algorithm 2 BFL

procedure MAIN
  $t \leftarrow 0$
  initialize w_0
  while convergence is not reached **do**
    $\mathcal{S}^t \leftarrow$ random set of m clients
    for each client $k \in \mathcal{S}^t$ **in parallel do**
      Download the latest block, b^t
      $w^t \leftarrow \sum_{j \in b^t} \frac{|D_j|}{|D|} w_j^t$
      $w_k^{t+1} \leftarrow$ CLIENTUPDATE(k, w^t)
      SUBMITLOCALUPDATE(S_{tr})
    end for
    $b^{t+1} \leftarrow$MINEBLOCK(λ)
    PROPAGATEBLOCK(b^{t+1})
    if b^{t+1} **is not valid then**
      Go to line 12
    end if
    $t \leftarrow t + 1$
  end while
end procedure

3) Miners execute certain tasks to decide which node updates the chain. One of the most popular mining mechanisms, and studied in this paper, is Proof-of-Work (PoW) [39], whereby miners spend their computational power (denoted by λ) to solve computation-intensive mathematical puzzles.
4) As a result of the concurrent mining operation, a new block is created and propagated throughout the P2P blockchain network every BI seconds (on average). The block size S_B is selected such that can include a maximum of m transactions, each one representing a local model submitted by a client.
5) Clients download the latest block from its associated miner (as in [13], [31]), which would allow performing on-device global model aggregation and local training.

An important consequence of the blockchain decentralized consensus is forking. A fork occurs when two or more miners generate a valid block simultaneously (i.e., before the winning block succeeds to be propagated). The existence of forks can be seen as a waste of resources, as it may lead to extra computation and delay overhead [40].

In this work, we consider the version of BFL reported in Algorithm 2 [22], which entails the participation of multi-access edge computing (MEC) servers and edge devices. Each client downloads the updates $w_1^t \dots w_m^t \in b^t$ contained in the latest block, computes the new global w^t, and trains it on its local dataset with the CLIENTUPDATE procedure described in Section III-A. The parameters of the new updated model w_k^{t+1} are then submitted with the method SUBMITLOCALUPDATE, where S_{tr} is the transaction size. Once all the local updates are uploaded to the blockchain, a new block b^{t+1} is mined with MINEBLOCK, where the block generation rate, $\lambda = \frac{1}{BI}$, is derived from the total computational power of blockchain nodes. Finally, the new block is shared across all the blockchain nodes with the procedure PROPAGATEBLOCK, which depends on the size of block b^{t+1} (fixed to S_B). The process is repeated until convergence.

C. GOSSIP FEDERATED LEARNING (GFL)

GFL is an asynchronous protocol that trains a global model over decentralized data using a gossip communication algorithm [19], [20].

We consider the general design proposed in [20] and [21]. Overall, the participating clients start from a common initialization. The global model is then trained sequentially on local data and following a given path (e.g., a random walk) of visiting clients. Algorithm 3 describes the GFL procedure and it works as follows:

1) The N clients agree on the ML model to use and store it in their local cache, i.e., $\text{lastModel}_\text{i}$, $\forall \text{i} \in \mathcal{N}$.
2) At each round t, m clients are randomly selected and ordered in a sequence $\mathcal{S}^t = [k_1, \dots, k_m]$.
3) The ML model visits sequentially the clients in S^t. Let $w_{k_{i-1}}^t$ be the model received by $k_i \in \mathcal{S}^t$.
4) As reported in Fig. 2, first the procedure MERGE combines $w_{k_{i-1}}^t$ and $\text{lastModel}_\text{k}$, i.e., the model from the previous round in which the client has been selected. Then, the procedure CLIENTUPDATE described in Section III-A used to train the merged model on the local dataset.

Algorithm 3 GFL

procedure MAIN
  initialize $lastModel_k$ for each client k
  $t \leftarrow 0$
  while convergence is not reached **do**
    $\mathcal{S}^t \leftarrow$ random set of m clients
    $[k_1, ..., k_m] \leftarrow$ GETSEQUENCE($\mathcal{S}^t$)
    for $i = 1, ..., m$ **do**
      $w^t_{k_{i-1}}$ ▷ Model trained by the previous node in the sequence
      $w^t_{k_i} \leftarrow$ MERGE($w^t_{k_{i-1}}$, lastModel_{k_i})
      $w^t_{k_i} \leftarrow$ CLIENTUPDATE(k_i, $w^t_{k_i}$)
      $\text{lastModel}_{k_i} \leftarrow w^t_{k_{i-1}}$
      Send model to the next client
    end for
    $t \leftarrow t + 1$
  end while
end procedure

procedure MERGE(w, w')
  $w \leftarrow \frac{w+w'}{2}$
  return w
end procedure
procedure GETSEQUENCE(S^t)
  $[k_1, \ldots, k_m] \overset{\text{i.i.d.}}{\sim} U(S^t)$
  return $[k_1, \ldots, k_m]$
end procedure

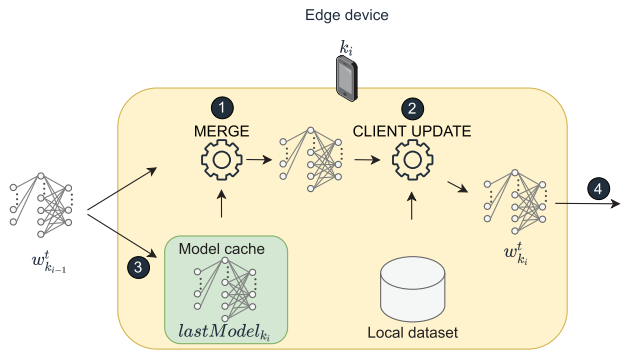


FIGURE 2. Overview of the operations executed by a node in the GFL algorithm.

5) The local cache of k_i is updated with the model $w^t_{k_{i-1}}$, i.e., lastModel_{k_i} and the model trained on the local dataset $w^t_{k_i}$ is shared with the next node in the sequence $\mathcal{S}^t$.
6) If k_i is the last client of the sequence the model $w^t_{k_i}$ is sent to the first node of the next FL round, i.e., $k_1 \in S^{t+1}$.

A round is completed when the model has visited all the clients in the sequence. The algorithm stops when convergence is reached (after a given number of rounds). As an example, we consider a scenario with N clients. After the local cache initialization, m clients are randomly selected and ordered in a sequence $\mathcal{S}^0 = [k_1, \ldots, k_m]$. The first client $k_1 \in \mathcal{S}^0$ trains the model stored in its local cache, with the procedure CLIENTUPDATE on the local dataset and shares the model $w^0_{k_1}$ with the following node in the sequence, i.e., $k_2 \in S^0$. When k_2 receives $w^0_{k_1}$, first applies the MERGE procedure to combine it with the model stored in lastModel_{k_2} and then trains it on its local dataset with the procedure CLIENTUPDATE. After the local training, the model $w^0_{k_2}$ is shared with the next node in the sequence, and $w^0_{k_1}$ is stored in the local cache lastModel_{k_2}. The same process is repeated at each visited client till the end of the sequence. The last client $k_m \in S^0$, after the execution of the procedures MERGE and CLIENTUPDATE, sends $w^0_{k_m}$ to the first node of the sequence for the next round S^1. These operations are executed iteratively till convergence is reached, e.g., a predefined number of rounds are executed.

IV. COMPUTATIONAL AND COMMUNICATION COSTS

In this section, we introduce the mathematical statements for the calculation of the time complexity of the three federated algorithms discussed in Section III. We also elaborate on the data overhead due to the communication of the different model updates during the several rounds of the process for each implementation. Finally, we derive the equations for the calculation of the time to reach the convergence of the three analyzed federated approaches. The results proposed hereafter are derived using the following assumptions:

1) Scalar operations (sums and products) cost $O(1)$.
2) The time complexity of the matrix multiplication is linear with the matrix size, i.e., $A \in \mathbb{R}^{i \times j}$ and $B \in \mathbb{R}^{j \times k}$, the cost of the product is $O(i \cdot j \cdot k)$.
3) For a single input pair (x_i, y_i), the time complexity required to compute $\nabla \ell$ is linear with the number of model's weights, $O(|w|)$.
4) During the mining process, with the PoW, a miner computes the nonce of a block using brute force until finding a hash value lower or equal to a certain threshold [41], referred to as the mining difficulty. Assume that the hash value has b bits, and that its solution should be smaller than 2^{b-l} bits (being l a value determined by the mining difficulty), if the miner samples the nonce values at random, the probability of a valid value is 2^{-l}. Henceforth, 2^l sampling operations are required for mining a block. The time complexity is $O(2^l)$.
5) The set of nodes that have a local copy of the blockchain is $\mathcal{N}_B = \{1, \ldots, N_B\}$, without loss of generality, is assumed to be $\mathcal{N} \cap \mathcal{N}_B = \varnothing$.
6) We assume that convergence of the FL training procedure is reached after R rounds.

Theorem 1: The time complexity of CFL is:

$$O(RmE|D_{\max}||w|), \tag{3}$$

TABLE 1. Computational complexity and communication overhead for CFL, BFL and GFL.

Algorithm	Time complexity	Communication Overhead
CFL	$O(RmE\|D_{\max}\|\|w\|)$	$2Rm\|w\|$
BFL	$O(R(\|w\|m^2 + E\|D_{\max}\|\|w\|m + 2^l + m\|w\|N_B))$	$R(\|w\|m^2 + \|w\|m + m\|w\|N_B)$
GFL	$O(RmE\|D_{\max}\|\|w\|)$	$Rm\|w\|$

where $D_{\max} = \max_{k\in\mathcal{N}} |D_k|$. The communication overhead is given by

$$2Rm|w| \tag{4}$$

Proof: See Appendix A-A. ■

Theorem 2: The time complexity of BFL is:

$$O\Big(R\Big(|w|m^2 + E|D_{\max}||w|m + 2^l + m|w|N_B\Big)\Big), \tag{5}$$

where N_B is the number of nodes that have a local copy of the blockchain and l is related to the PoW difficulty (see Assumption 4). Its communication overhead is

$$R\Big(|w|m^2 + |w|m + m|w|N_B\Big) \tag{6}$$

Proof: See Appendix A-B. ■

Theorem 3: The time complexity of GFL is

$$O(RmE|D_{\max}||w|) \tag{7}$$

and its communication overhead is

$$Rm|w| \tag{8}$$

Proof: See Appendix A-C. ■

The three algorithms have a time complexity that depends on the dataset size $|D_{\max}|$. Additionally, the time complexity of BFL (5) is also a function of the blockchain parameters N_B and l. In particular, the dominant term in (5) is $R2^l$. Hence, the time complexity of BFL is exponential in the PoW difficulty l, while for CFL and GFL is polynomial in $RmE|D_{\max}||w|$. Table 1 summarizes the different results obtained for time complexity and communication overhead.

Lastly, we conclude our analysis by computing the total execution time of each algorithm until convergence (convergence time) as a function of the delay introduced by the communication rounds and the computational operations. To do that, we characterize the links whereby the different types of nodes exchange information (e.g., local model updates, blocks), keeping the topology introduced in Fig. 1 in mind. We classify two different types of connections: cloud (solid arrows) and edge (dotted and dashed arrows). Cloud connections (assumed to be wired) are used by miners in the blockchain; instead, edge connections (assumed to be wireless) are used by edge nodes to upload/download models. Given its popularity and easiness of deployment, we adopt IEEE 802.11ax links for edge connections [42]. Since edge devices are often energy-constrained, we consider different values of transmission power for the edge connections. The central server and blockchain node use a transmission power of P^c_{TX}, instead, edge devices use P^e_{TX}, with $P^e_{\text{TX}} \leq P^c_{\text{TX}}$. The wired connection has a capacity C_{P2P}. Additionally, we identify three main types of computational operations during the federated learning processes: local model training, model parameters exchange, and blockchain data sharing. Based on this, we can compute the convergence time of CFL, BFL, and GFL as follows:

$$T_{\text{CFL}} = T_{\text{train}} + T^e_{\text{Tx}} + T^c_{\text{Tx}}, \tag{9}$$

$$T_{\text{BFL}} = T_{\text{BC}} + T_{\text{train}} + T^e_{\text{Tx}} + T^c_{\text{Tx}}, \tag{10}$$

$$T_{\text{GFL}} = T_{\text{train}} + T^e_{\text{Tx}}, \tag{11}$$

where T_{train} is the total amount of time spent for training the ML model locally, $T^{c/e}_{\text{Tx}}$ is the total transmission time of the central server/blockchain nodes (c) or the edge devices (e), and computed according to the model detailed in Appendix B. T_{BC} is the total delay introduced by blockchain and described in steps 2-4 of the process in Section III-B.

V. ENERGY FOOTPRINT

In this section, we define the models used to characterize the energy consumption that results from the FL operations. Driven by (9), (10) and (11), the total amount of energy consumed in each scenario is:

$$\mathcal{E}_{\text{CFL}} = \mathcal{E}_{\text{train}} + \mathcal{E}^e_{\text{Tx}} + \mathcal{E}^c_{\text{Tx}}, \tag{12}$$

$$\mathcal{E}_{\text{BFL}} = \mathcal{E}_{\text{BC}} + \mathcal{E}_{\text{train}} + \mathcal{E}^e_{\text{Tx}} + \mathcal{E}^c_{\text{Tx}}, \tag{13}$$

$$\mathcal{E}_{\text{GFL}} = \mathcal{E}_{\text{train}} + \mathcal{E}^e_{\text{Tx}}, \tag{14}$$

where $\mathcal{E}_{\text{train}}$ is the energy consumed by all the nodes during the local training, and $\mathcal{E}^{c/e}_{\text{Tx}}$ is the energy required to transmit the model via IEEE 802.11ax wireless links during the whole algorithm execution, from either a central server/blockchain node (c) or an edge device (e). $\mathcal{E}_{train}$ is calculated as:

$$\mathcal{E}_{\text{train}} = \sum_{r=0}^{R-1} \sum_{i\in\mathcal{S}^r} P^r_{\text{CPU}_i} \Delta^r_i, \tag{15}$$

where $P^r_{\text{CPU}_i}$ is the average power consumed by the CPU and DRAM during a round r, and Δ^r_i is the duration of the operation for the i - th client. The model proposed in (15) is general enough to capture the characteristics of the three algorithm implementations. Moreover, it opens the possibility of using energy measurement libraries, which represent a good trade-off between the complexity of using real sensors and the high abstraction level of mathematical models.

As described in Section IV, we may have two types of communication links: cloud and edge. Considering that cloud links are wired, we assume that their energy consumption is

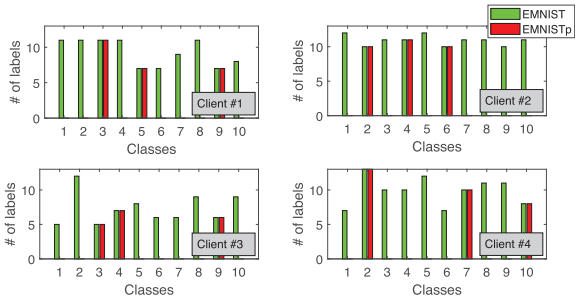


FIGURE 3. Distribution of samples across the four first clients for both EMNIST and EMNISTp federated datasets.

negligible. Instead, we compute the energy consumption of the edge connections according to the following equation:

$$\mathcal{E}_{\text{Tx}}^{c/e} = T_{\text{Tx}}^{c/e} P_{\text{Tx}}^{c/e}, \tag{16}$$

where $T_{\text{Tx}}^{c/e}$ and $P_{\text{Tx}}^{c/e}$ are the transmission time and power of a central server/miner (c) or an edge device (e). The additional term $\mathcal{E}_{\text{BC}}$ for BFL is associated to mining operations of the PoW. We measure that consumed energy based on the model proposed in [43] and according to the following equation:

$$\mathcal{E}_{\text{BC}} = P_h \frac{1}{\lambda} N_{\text{chain}}, \tag{17}$$

where P_h is the total hashing power of the network, λ is the block generation rate, and N_{chain} is the number of blocks on the main chain.

VI. PERFORMANCE EVALUATION

In this section, we first describe the experimental settings adopted to compare the three federated approaches and then, we present numerical results.

A. SIMULATION SETUP

We use the *Extended MNIST (EMNIST)* dataset available on *Tensorflow Federated (TFF)* library [44]. The input features are black and white images that represent handwritten digits in $\{0, 1, \ldots, 9\}$, coded in a matrix of 28×28 pixels. Considering only digits, it contains 341 873 training examples and 40 832 test samples, both divided across 3 383 clients. The training and the test sets share the same clients list so that each client has at least one sample. Each local dataset groups all the samples of the same writer and does not change over time. In the EMNIST dataset, all clients have a rich number of samples for all the classes, thus data distribution across them can be considered as IID. To evaluate the targeted federated mechanisms in more challenging settings, we create a new version of the EMNIST dataset, called EMNISTp, by randomly restricting each client dataset to 3 classes only. EMNISTp contains 102 418 training samples. Fig. 3 shows the available samples of the first 4 clients, for both versions of the dataset.

To correctly classify these samples we choose two models proposed in [27]. The first one is a feed-forward neural network (FFNN) with an input layer of 784 neurons, two hidden layers of 200 neurons activated with the rectified linear unit (ReLU) function, and an output layer of 10 neurons with *Softmax* activation function. The number of trainable parameters for the FFNN ($|w'|$) is 199 210. Assuming that each parameter requires 4 bytes, i.e., size of a *float32* variable, the total amount of space required ($S_{w'}$) is 796.84 KB. The second one is a convolutional neural network (CNN) with the following structure:

1) A 5×5 convolutional layer with 32 channels and ReLU activation function.
2) A 2×2 max pooling layer.
3) A 5×5 convolutional layer with 64 channels and ReLU activation function.
4) A 2×2 max pooling layer.
5) A fully connected layer with 512 units and ReLu activation.
6) An output layer with 10 neurons and Softmax activation function.

In total, the number of trainable parameters for the CNN ($|w''|$) is 582 026 and the size in memory ($S_{w''}$) is 2.33 MB. We opted for a FFNN to reproduce a realistic scenario where edge devices might not have enough computational power to train more sophisticated deep learning mechanisms, like NNs based on convolutional architectures. Despite its simplicity, the selected FFNN model accurately classifies the digits of the EMNIST dataset, as shown next. Then, we use also a CNN to evaluate the multiple algorithms' performance using a more complex model (details in Section VI-C).

The three FL algorithms are implemented with *Tensorflow (TF)* [25], *Tensorflow Federated (TFF)* [45] and *Keras* [46] libraries. We have extended the Bitcoin model provided by BlockSim [47] to simulate the blockchain behavior. BlockSim is an event-based simulator that characterizes the operations carried out to store data in a blockchain, from the submission of transactions to mining blocks and reaching consensus in a decentralized manner. Accordingly, BlockSim allows simulating the delays added by the blockchain in a BFL application, i.e., the T_{BC} parameter defined in Section IV.

We create a validation set by choosing a subset of 200 clients from the test set. Following the TFF documentation [48], the training accuracy is computed at the beginning of each round; instead the validation accuracy is calculated at the end. For this reason, may happen that the validation accuracy is higher than the training one. At the end of each simulation, we evaluate the performance on the test set.

As for $\mathcal{E}_{\text{train}}$ and T_{train}, we have used Carbontracker [49], a Python library that periodically samples the hardware energy consumption and measures the execution time. Moreover, P_{Tx}^{c} is set to 20 dBm and $P_{\text{Tx}}^{e} = 9$ dBm. Table 2 reports all the other parameters used in our simulations. We note here that we have used the same FL parameters for a fair comparison, being the number of rounds R of CFL and GFL equivalent to the main chain length (N_{chain}) in BFL. In this way, we guarantee that the number of global rounds of each learning algorithm is the same. We run the experiments with

TABLE 2. Simulation parameters.

	Parameter	Description	Value
Fed. Learning	$\|w'\|$	Number of FFNN model parameters	199 210
	$\|w''\|$	Number of CNN model parameters	582 026
	$S_{w'}$	FFNN model parameters size	796.84 KB
	$S_{w''}$	CNN model parameters size	2.33 MB
	η	Learning rate	0.2
	N	Number of total clients	3382
	E	Local epochs number	5
	R	Number of rounds	200
	m	Number of clients for each round	200
	B	Batch size	20
	ℓ_i	Local loss function	Sparse Cat. Crossentropy
Blockchain	N_{chain}	Number of blocks in the main chain	200
	BI	Block interval	15 s
	N_B	Number of blockchain nodes	200
	N_m	Number of miners	10
	C_{P2P}	Capacity of P2P links	100 Mbps
	S_H	Block header size	25 KB
	S'_B	Block size with FFNN	160.368 MB
	S''_B	Block size with CNN	467 MB
	S'_{tr}	Transaction size with FFNN	796.84 KB
	S''_{tr}	Transaction size with CNN	2.33 MB
	P_h	Total hashing power	1350 W
Communication (IEEE 802.11ax)	P^e_{Tx}	Tx power for edge devices	9 dBm
	P^c_{tx}	Tx power for a central server	20 dBm
	σ_{leg}	Legacy OFDM symbol duration	4 µs
	N_{sc}	Number of subcarriers (20 MHz)	234
	N_{ss}	Number of spatial streams	1
	T_{e}	Empty slot duration	9 µs
	T_{SIFS}	SIFS duration	16 µs
	T_{DIFS}	DIFS duration	34 µs
	T_{PHY}	Preamble duration	20 µs
	$T_{\text{HE-SU}}$	HE single-user field duration	100 µs
	L_s	Size OFDM symbol	24 bits
	L_{RTS}	Length of an RTS packet	160 bits
	L_{CTS}	Length of a CTS packet	112 bits
	L_{ACK}	Length of an ACK packet	240 bits
	L_{SF}	Length of service field	16 bits
	L_{MAC}	Length of MAC header	320 bits
	CW	Contention window (fixed)	15

the following hardware configurations: Intel i5-6600 with 8GB of RAM (HW1) and two Intel Xeon 6230 with 188GB of RAM (HW2).

B. RESULT ANALYSIS

Table 3 reports the accuracy of each algorithm implementation with the FFNN model on the two considered datasets executed on HW1. CFL and BFL achieve the best accuracy (both close to 0.9), instead, GFL presents lower values. This result validates the claim that, under similar setups as in our simulations (i.e., each block contains m local updates organized in transactions), the central parameter server of CFL can be replaced by a blockchain network, properly dimensioned, without compromising the learning accuracy. On the other hand, GFL achieves a validation accuracy around 0.5 after 200 rounds. We justify this behavior by noticing that, before the CLIENTUPDATE procedure, the model received from the previous node in the sequence is merged with that in the previous round. For the earliest training rounds, there is a high probability that the merging procedure works with a fresh model that has never been trained, hence disrupting the knowledge from the previous clients. We analyze this phenomenon more in depth in Section VII-A.

Table 3 also details the convergence time of each algorithm, the percentage of energy consumed in the computations (as a percentage of the total energy consumed), the total amount of energy needed, and the communication overhead (i.e., data to be shared during the rounds of the algorithms). Computational energy is, generally speaking, the energy spent for performing computations needed for the specific algorithm implementation. For CFL and GFL, it takes into account the energy spent in the local training at the clients, $\mathcal{E}_{\text{train}}$. Instead, in BFL, within computational energy, we consider both local training at clients and the energy spent for the mining process in the blockchain network, i.e., $\mathcal{E}_{\text{train}}$ and $\mathcal{E}_{\text{BC}}$ respectively.

The fastest and the most energy-efficient algorithm is GFL: it is able to save the 18% of training time, the 18% of energy, and the 51% of data to be shared with respect to the CFL solution. However, GFL main drawback resides in the poor accuracy achieved, as stated above. BFL is the slowest and the most energy-hungry federated implementation, mainly due to the overhead introduced by the blockchain network to secure data in a decentralized way. Additionally, it is to be noted that computation is the most energy-consuming task for CFL, BFL, and GFL. For BFL, the mining process drains 1125 Wh, i.e., 98% of the total energy. We highlight here that our comparison may be unfair in this respect, since both CFL and GFL are not including any security mechanism. However, we believe that it is worth including BFL in our analysis on distributed versus centralized federated learning since our results show that the secure and decentralized method introduced by the blockchain network, despite increasing algorithm costs, does not jeopardize its accuracy compared to its centralized counterpart CFL. Finally, GFL is the implementation that requires the lowest communication overhead. More precisely, in this case, we need to include an extra cost to share the global model across the nodes at the end of the last round (not considered in the table), which is approximately 0.16 GB (the cost of one extra round).

C. MODEL AND IMPLEMENTATION DEPENDENCIES

Table 4 and Table 5 report the performance of the CNN model on EMNIST (EMNISTp) datasets executed on two different platforms HW1 and HW2, respectively. Similar to the previous FFNN case, BFL is the slowest and the most energy demanding algorithm. Instead in this case, GFL reaches higher validation accuracy on EMNIST, i.e., 0.8, but is still not able to get the performance of the other two algorithms. Moreover, using CNN, GFL is the fastest algorithm and saves up to 16% of the execution time, with respect to CFL on HW1. Hence, model selection plays a key role for the algorithm performance and may facilitate the training process, as in the case of GFL. Finally, it is confirmed that the communication overhead of GFL is the lowest.

TABLE 3. FFNN simulation results on EMNIST (EMNISTp) datasets.

	Acc. Training	Acc. Validation	Acc. Test	Conv. Time (s)	Comp. Energy (%)	Tot. Energy (Wh)	Comm. Overhead (GB)
CFL	0.9 (0.76)	0.87 (0.77)	0.86 (0.76)	46458.56 (45571.86)	98.91 (98.61)	21.84 (17.2)	63.75
BFL	0.88 (0.74)	0.87 (0.78)	0.86 (0.77)	51036.87 (50077.75)	99.98 (99.98)	1147.21 (1142.65)	12781.31
GFL	0.44 (0.36)	0.42 (0.12)	0.41 (0.11)	38201.67 (36821.67)	99.57 (99.14)	17.83 (8.98)	31.87

TABLE 4. CNN simulation results on HW1 and EMNIST (EMNISTp) datasets.

	Acc. Training	Acc. Validation	Acc. Test	Time (s)	Comp. Energy (%)	Tot. Energy (Wh)	Comm. overhead (GB)
CFL	0.99 (0.97)	0.97 (0.9)	0.96 (0.91)	132761.16 (126691.59)	99.04 (98.1)	72.35 (36.72)	186.4
BFL	0.99 (0.97)	0.97 (0.9)	0.96 (0.91)	138452.61 (132284.94)	99.94 (99.94)	1198.11 (1161.99)	37373.2
GFL	0.99 (0.67)	0.81 (0.22)	0.8 (0.22)	113573.11 (106616.64)	99.73 (99.28)	83.95 (31.22)	93.2

TABLE 5. CNN simulation results on HW2 and EMNIST (EMNISTp) datasets.

	Acc. Training	Acc. Validation	Acc. Test	Conv. Time (s)	Comp. Energy (%)	Tot. Energy (Wh)	Comm. overhead (GB)
CFL	0.99 (0.97)	0.97 (0.9)	0.96 (0.91)	125883.47 (124869.07)	99.65 (99.51)	201.68 (141.69)	186.4
BFL	0.99 (0.97)	0.97 (0.9)	0.96 (0.91)	131488.87 (130555.3)	99.95 (99.95)	1329.65 (1273.95)	37373.2
GFL	0.99 (0.67)	0.8 (0.22)	0.8 (0.22)	114217.66 (107854.86)	99.93 (99.84)	319.77 (143.22)	93.2

TABLE 6. Mapping between services and federated learning implementations.

Service	FL scenario	Preferred KPIs	Suggested algorithm
Healthcare	Cross-silo	Privacy and Accuracy	BFL
Mobile traffic prediction	Cross-silo	Accuracy and Latency	CFL
Urban traffic forecasting	Cross-device	Latency and Energy	GFL
Autonomous driving	Cross-device	Latency and Accuracy	GFL
Next word prediction	Cross-device	Energy	CFL

However, we report here some inconsistencies in performing energy measurements. In fact on HW1, differently from the FFNN case, CFL is the most energy efficient on EMNIST and saves 14% of energy with respect to GFL. On EMNISTp, instead, the situation is different since GFL saves 15% of the energy. Moreover, when using HW2 (Table 5), CFL results to be the most energy efficient for both EMNIST and EMNISTp. Such inconsistencies are mainly due to the fact that the average computational power consumption in CFL implementation is higher than GFL (around 103W and 93W on EMNIST, respectively); however GFL requires longer training time (T_{train}). Instead in the FFNN model implementation, the average computational power consumption is higher for GFL (around 19W for CFL and 13W for GFL), but GFL requires lower training time. The reason lays mainly in the software implementations.[1] In fact, CFL and BFL are based on TFF, which executes the training process for all the participating clients in parallel. Differently, GFL is based on the standard TF libraries and the training process is executed sequentially one client after the other.

In view of the above, we state here that hardware and software implementation play a key role in the energy assessment. Therefore, it is essential that future research directions will focus on: i) joint optimization of federated algorithms and their software implementations, ii) definition of standard libraries for the three categories of algorithms studied in this paper, and iii) design of effective and open test platforms for experiment comparison.

1. https://github.com/eliaguerra/Federated_comparison_cttc

D. MAPPING BETWEEN FL IMPLEMENTATION AND SERVICES

In state of the results achieved by our analysis, we map the most suitable FL implementation given a specific application scenario. Among the several services listed in [12], [13], in Table 6 we focus on some emerging examples such as Healthcare and Mobile Traffic Prediction for *cross-silo* scenarios and Urban Traffic Forecasting, Connected Vehicles, Next-word Prediction for *cross-device* settings. In the Healthcare scenario, accurate predictions that guarantee the confidentiality of users' data are required [50]. For these reasons, we suggest the usage of BFL. In the context of mobile networks, multiple base stations may be interested in training an ML model for mobile traffic prediction without sharing raw data and save network resources. In this case, accuracy and latency are the most important KPIs [51]. Our suggestion here is to use CFL. Nevertheless, we may also find BFL as an appealing solution when multiple operators cooperate on the mobile traffic forecasting task. In urban traffic forecasting, a network of sensors is in charge of creating a model to predict vehicular traffic flow. Considering the problem description and the limited amount of energy available at each sensor, the two most important KPIs are latency and energy [52], the suggested approach is a GFL (or GFL-NM) algorithm. For connected vehicles, latency and accuracy are the two most important KPIs [2]. Considering that the connection with a central server may be unavailable, we suggest the usage of GFL (or GFL-NM) to take advantage of nearby vehicles to share model updates. In next word prediction for mobile keyboards, minimizing energy is of paramount importance to guarantee a flawless user

TABLE 7. FFNN simulation results of GFL on EMNIST (EMNISTp) datasets with higher number of rounds and local computations.

R	E	Acc. Training	Acc. Validation	Acc. Test	Conv. Time (s)	Comp. Energy (%)	Tot. Energy (Wh)
200	5	0.44 (0.36)	0.42 (0.12)	0.41 (0.11)	36401.67 (35021.67)	99.59 (99.19)	17.83 (8.98)
400	5	0.55 (0.41)	0.51 (0.16)	0.5 (0.15)	72791.81 (70029.31)	99.58 (99.18)	35.08 (17.91)
800	5	0.85 (0.64)	0.67 (0.19)	0.66 (0.17)	145651.71 (140069.23)	99.59 (99.18)	70.95 (35.8)
200	10	0.57 (0.58)	0.4 (0.09)	0.41 (0.1)	37377.47 (35448.58)	99.7 (99.38)	24.55 (11.77)
400	10	0.66 (0.37)	0.47 (0.17)	0.48 (0.16)	74989.72 (70857.46)	99.7 (99.37)	49.17 (23.23)
800	10	0.94 (0.58)	0.67 (0.3)	0.67 (0.29)	149448.14 (141730.77)	99.7 (99.38)	98.37 (46.87)

experience so the most important KPI is energy [53], the suggested approach is CFL.

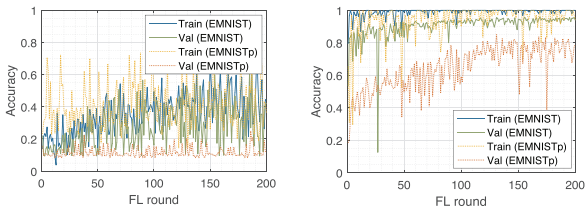
VII. PROPOSED IMPROVEMENTS

A. GFL ACCURACY

As described in Section VI-B, GFL does not achieve the same accuracy level as CFL and BFL. We identify two possible reasons for this:

1) The number of rounds is not sufficient for it to converge: the number of visited nodes might not be sufficient to achieve an acceptable accuracy.
2) The merge step negatively impacts the overall performance of the learning algorithm: the model received in the previous round and stored in the local cache slows down the learning process.

To verify the first hypothesis, we execute GFL algorithm, with the FFNN model on HW1, changing the number of rounds ($R = \{200, 400, 800\}$) and varying the number of local computations ($E = \{5, 10\}$). Table 7 shows the results obtained. Considering the EMNIST dataset, the best results are achieved with $R = 800$ and $E = \{5, 10\}$, i.e., a higher test accuracy of 0.66, but still lower than CFL and BFL. Moreover, the model is overfitting with $R = 800$ rounds; hence, when increasing the number of rounds, a regularization method would be needed. On the EMNISTp dataset, the accuracy is even lower for each combination of the hyperparameters tested. Increasing the number of rounds, R, and the local epochs, E, increases GFL validation accuracy to 0.94 but it quadruplicates the convergence time and requires 5 times more the energy of the baseline configuration ($R = 200$ and $E = 5$) on the EMNIST dataset. To verify the second hypothesis, we run GFL algorithm without the merge step (*GFL-NM*). The pseudocode of this algorithm is the same in Algorithm 3 except for having replaced the old Line 9 with the new command $w^t_{k_i} \leftarrow w^t_{k_{i-1}}$. Thus, in GFL-NM, given a sequence of clients $\mathcal{S}_t$ the model is trained incrementally on the client datasets. GFL-NM achieves a training accuracy of ~ 1.0 (0.94), a validation accuracy of 0.94 (0.78) and a test accuracy of 0.93 (0.78) on the EMNIST (EMNISTp) dataset (see Fig. 4(b)), higher than both CFL and BFL. These results suggest that the MERGE step compromises the training performance. In fact, at the beginning of the learning process, there is a high probability that a model visits a node that has never been visited before and with lastModel storing initialization values. In this case, the received model is merged with a model that has never been trained before, as shown in Algorithm 3, which negatively impacts the resulting merged weights. Figure 4 shows the comparison between the learning curves of GFL and GFL-NM.



(a) GFL (b) GFL-NM

FIGURE 4. Training and validation accuracy on EMNIST and EMNISTp.

In conclusion, we proved that both 1) and 2) influence the achieved accuracy. Moreover, GFL-NM solves the accuracy problem of standard GFL and achieves the best performance from the point of view of all the metrics.

B. DELAY ANALYSIS OF BFL

Blockchain technology, while enabling a reliable and secure FL operation, entails very high overheads in terms of time and energy for the sake of keeping decentralization. The performance of a blockchain, typically measured in transactions per second (TPS), together with the granted degree of security, strongly depends on the nature of the blockchain (e.g., degree of visibility, type of consensus, mining protocol), its configurable parameters (e.g., block interval, difficulty), and the size of the P2P network maintaining it. Furthermore, as discussed in Section VIII-D, the necessary energy to maintain a blockchain is correlated to its performance in TPS and security, thus leading to the well-known performance, security, and energy trilemma.

To showcase the effect of using different types of blockchain networks, Fig. 5 shows the total delay incurred by the blockchain to the FL operation to generate up to 200 blocks under different blockchain configurations. Notice that, in the proposed setting, each block is equivalent to an FL round. In particular, we vary the total number of miners ($N_m = \{1, 10, 100\}$) and the block interval ($BI = \{5, 15, 600\}$ s), which affect the time required to achieve consensus.

First, a higher number of miners leads to a higher fork probability, provided that more nodes need to agree on the same status of the ledger. Note that with $N_m = 1$ the fork probability, i.e., $p_f(N_m)$, is 0 since there is only one miner. By contrast, a higher block interval allows mitigating the

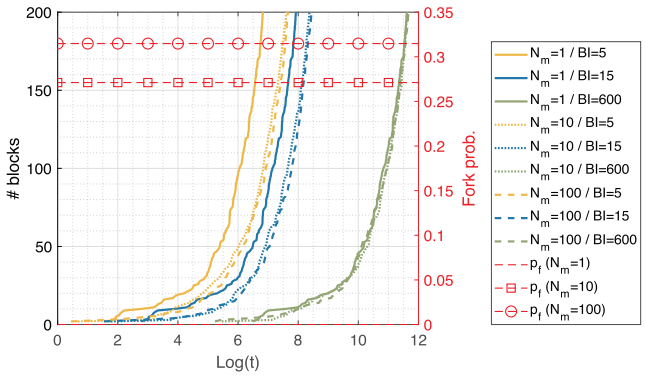


FIGURE 5. Blockchain delay as a function of the number of miners (N_m) and the block interval (BI). The fork probability associated with each N_m is shown in red.

effect of forks, since the probability that two miners mine a block simultaneously is lower [54].

As shown in Fig. 5, the blockchain delay increases with the block interval (BI), which indicates the average time for mining a block. Notice that, in a PoW-based blockchain, the block interval is fixed by tuning the mining difficulty according to the total computational power of miners. As for the impact of N_m on the delay, its effects on the delay are more noticeable for low BI values. In particular, a higher fork probability is observed as N_m increases, thus incurring additional delays to the FL application operating on top of the blockchain.

VIII. OPEN ISSUES AND FUTURE RESEARCH DIRECTIONS

A. GFL

In our opinion, and encouraged by our results, the investigation of new methods for merging the model updates from the distributed sources to achieve faster and higher accuracy is an interesting and open research line. To the best of our knowledge, there are still very few works that go in this direction in the literature. In [21], the authors implement an incremental version of GFL with a single round on the edge devices using the entire local dataset (hence, without requiring any merge step). Similarly, [55] proposes an iterative continual learning algorithm, where a model is trained incrementally on the local datasets without applying any merge operation.

B. BFL

To optimize the performance of a blockchain, a widely adopted approach consists of finding the best block generation rate [31], which is controlled by tuning the mining difficulty. Other approaches consider optimizing the block size [56], which better fits scenarios where the intensity of transaction arrivals depends on the nature of the application running on top of the blockchain (e.g., FL updates provided by clients).

Regarding the communication cost of BFL, it can be improved by leveraging the computational capacity of blockchain miners to speed up the FL operation. In particular, instead of including individual local models in a block, each block can bring a global model, aggregated by the miner responsible for building the block. This approach has been widely adopted in the literature (see, e.g., [57]), and would lead to reduced time complexity and communication cost (see Appendix A-B).

Finally, another important open aspect regarding blockchain-enabled FL lies in the implications of decentralization on the learning procedure. In this paper, we have assumed that the blockchain is perfectly shared and accessed by FL devices to carry out training, thus acting as a central orchestrating server. However, the decentralized data sharing in blockchain naturally leads to model inconsistencies, provided that different FL devices can use the information from different blocks to compute local model updates.

C. ROLE OF MEC IN DECENTRALIZED FL

Due to its low-latency and computation-acceleration capabilities, the MEC paradigm is of particular relevance to uplifting the performance of FL applications. The interplay between FL and MEC has been largely studied from the data offloading perspective, but when it comes to decentralized FL approaches, MEC can play different roles. First, FL devices in GFL can leverage MEC to perform model training offloading by sharing their local datasets to a nearby MEC server. As shown in [58], ML model training offloading is an appealing solution for mitigating the straggler nodes problem in FL. And this is particularly useful in settings with computationally constrained devices such as in IoT. Nevertheless, offloading a dataset to an edge server entails a trade-off between the data transfer time and the local computation savings. For that, it is critical to properly optimize the amount of data to be offloaded with respect to the total training delay. In addition, the MEC approach can be useful for speeding up the training procedure thanks to a proper client selection scheduling [59].

When it comes to BFL, MEC can be leveraged not only for ML model training, but also to support the computation and storage requirements of the blockchain, which are typically unfeasible for small devices. In the literature, MEC servers have been considered for performing either model training or mining tasks (or both) [13]. In this work, and closer in spirit to what was proposed in [32], we have assumed that the blockchain operations are handled by MEC servers.

D. DISCUSSION ON SECURITY ASPECTS

The decentralization of FL through either GFL or BFL is important for mitigating single-point-of-failure issues and boosting scalability and democratization. As seen in Section VI-B, GFL provides a high degree of decentralization thanks to the P2P interactions between FL clients. However, its performance is significantly lower than the centralized counterpart. As for BFL, it provides outstanding accuracy values, but at a high cost in terms of energy consumption and communication. The fact is that, to ensure

security, robustness, and reliability, a blockchain typically requires to perform computation-intensive validations and exhaustive information replication. For instance, to ensure transparency, the blockchain network needs to trace all the events.

Unlike CFL and GFL, the BFL setting treasures security properties by design, which makes it robust against external threats. Thanks to blockchain's tamper-proof and traceability properties, attacks on the training operation such as poisoning [60] or Sybil attacks [61] can be mitigated [62], [63], [64]. Of course, in BFL, the cost of security in terms of energy consumption and communication overheads strongly depends on the FL application scenario and its requirements. In particular, different types of blockchains can better suit different FL applications. Blockchain types are mainly categorized as follows:

- *Public blockchains:* Are open and allow everyone to contribute by submitting transactions and participating in the consensus through mining blocks. The most prominent public blockchains are Bitcoin and Ethereum, which use methods like PoW and PoS for securing the data in such a type of open scenario.
- *Private blockchains:* The governance of private blockchains is restricted, so only a designed authority can update the blockchain with new transactions. An example of a private blockchain is Corda, where trusted nodes can quickly confirm the validity of transactions, thus boosting efficiency.
- *Consortium blockchains:* The blockchain governance is ruled by a limited set of trusted participants (instead of a single organization). This setting suits well the cooperative enterprise setting, where participants can be trusted to a certain extent. The most popular consortium blockchain platform is Hyperledger Fabric, whereby the applied consensus method (RAFT) does not require all the nodes in the blockchain to participate in the validation of data, thus speeding up performance.

Depending on the blockchain setting, the required computation energy and communication overheads may vary for sustaining the desired level of security. Throughout this paper, we have considered the most decentralized type of blockchain, i.e., public blockchains, but the other settings may be more appropriate for specific cases. The analysis of the cost for other blockchain types is left as future work, but we refer the interested reader to the survey in [65].

IX. CONCLUSION

Decentralized server-less federated learning is an appealing solution to overcome CFL limitations. However, finding the best approach for each scenario is non trivial due to the lack of comprehensive comparisons. In this work, we have proposed a complete overview of these techniques and evaluated them through several key performance indicators: accuracy, computational complexity, communication overhead, convergence time, and energy consumption. To do so, we have proposed a comprehensive theoretical analysis and the physical implementation of these algorithms.

An extensive simulation campaign underlines our analysis. From numerical results, it emerges that GFL is the algorithm that requires less communication overhead to reach convergence. Then, CFL and GFL have similar behavior in terms of energy consumption and accuracy, but slightly differ based on the DL model adopted and the hardware used. BFL represents a viable solution for implementing decentralized learning with a high accuracy and level of security at the cost of an extra energy usage and data sharing.

Moreover, we have discussed some open issues and future research directions for the two decentralized federated methods, like the poor accuracy achieved by GFL and the blockchain overhead in BFL. Regarding GFL, we have argued that the main drawback lies in the method used to merge model updates across the algorithm steps. We have demonstrated that with an incremental approach, the modified version of GFL is able to outperform CFL and BFL in terms of accuracy. As for BFL, we have indicated that possible optimizations go in the direction of finding the best block generation rate and block size. Moreover, we have reasoned on the possibility of reducing the time complexity by including the global model in a block, which is aggregated by the same miner building the block. In addition, we have pointed out the importance of further studies on the implication of model inconsistencies due to the fact that the blockchain cannot be perfectly shared and accessed by (all) the FL devices.

Finally, we have argued on the key role played by the libraries used for the implementation and their influence on the energy consumption on different hardware platforms. We call for the definition of standard libraries and open test platforms to be used for research purposes.

APPENDIX A
PROOFS

A. PROOF OF THEOREM 1

Let us consider the procedure ClientUpdate, whose time complexity is $E(|D_{\max}||w| + 2\frac{|D_{\max}|}{B}|w|)$. In fact, a single client k performs the training phase on its local dataset D_k along E local epochs and updates the model parameters. The first operation has a time complexity of $|D_k||w|$ and the second $2\frac{|D_k|}{B}|w|$. The update it is executed a number of times equal to $\frac{|D_k|}{B}$, and requires a product and a sum. Each client performs E local epochs, so the total cost is:

$$\sum_{k\in S_t} E\left(|D_k||w| + 2\frac{|D_k|}{B}|w|\right) \tag{18}$$

To obtain an upper bound that does not depend on k, we can use $|D_{\max}|$ as an upper bound of $|D_k|$:

$$\sum_{k\in S_t} E\left(|D_k||w| + 2\frac{|D_k|}{B}|w|\right) \leq mE\left(|D_{\max}||w| + 2\frac{|D_{\max}|}{B}|w|\right). \tag{19}$$

We can divide the MAIN procedure in Algorithm 1 into two blocks. The first, up to Line 10, has a cost upper bounded by

$$mE\left(|D_{\max}||w| + 2\frac{|D_{\max}|}{B}|w|\right) + 2|w|m. \tag{20}$$

In parallel every client downloads the global model, executes CLIENTUPDATE, and sends the updated parameters back to the server. The download and upload operations have a time complexity proportional to $|w|$. Considering that the same procedure is repeated by m clients, the upper bound in (20) easily follows. The second block starts from Line 10, where the server aggregates the local updates and computes the new global model. The number of arithmetical operations performed is:

$$2|w|m. \tag{21}$$

Combining (20) and (21), and considering the number of total rounds R required to reach convergence, the total cost of CFL is given by:

$$R\left[mE\left(|D_{\max}||w| + 2\frac{|D_k|}{B}|w|\right) + 4m|w|\right] = RmE|D_{\max}||w| + 2RmE\frac{|D_k|}{B}|w| + 4Rm|w|. \tag{22}$$

The first addend in (22) is the dominant term for the asymptotic time analysis, so this completes the proof to obtain (3).

When it comes to the communications overhead of CFL, the result easily follows considering that, for each round, each clients downloads and uploads the model parameters.

B. PROOF OF THEOREM 2

In each algorithm's round, every client in $\mathcal{S}^t$ has to download the latest block from the closest edge server (miner) to obtain the current global model. These operations, as described before, have a cost of $|w|m$ and $2|w|m$, respectively. Then, after running the CLIENTUPDATE procedure in Algorithm 2, clients submit the new model weights with a cost of $|w|$. These steps are done by each node in $\mathcal{S}^t$ (in total, m nodes), so the total cost is:

$$m\left(2|w|m + |w|m + E|D_{\max}||w| + 2E\frac{|D_{\max}|}{B}|w| + |w|\right). \tag{23}$$

When all the local updates have been computed, it is necessary to create a block, reach consensus throughout the mining operation, and propagate the block across all the blockchain nodes. The cost of these operations is given by:

$$2^l + m|w|N_B. \tag{24}$$

If we combine together (23) and (24), we obtain the total time complexity of the algorithm

$$R\left(3|w|m^2 + E|D_{\max}||w|m + 2E\frac{|D_{\max}|}{B}|w|m + |w|m + 2^l + m|w|N_B\right). \tag{25}$$

The dominant addends are reported in (5).

The communication overhead of BFL can be easily derived from the algorithm description.

In this analysis, we considered the less efficient implementation, whereby each client has to perform the computation of the new global model given the updates in the latest block. To improve this, we can move the instruction in Line 8 outside the *for* loop and execute it before the MINEBLOCK procedure. In this way, the new block has size $|w|$, since it contains only the parameters of the new model. Following the same analysis described before, the computational complexity is:

$$O\left(R\left(mE|D_{\max}||w| + 2^l + N_B|w|\right)\right). \tag{26}$$

And the communication overhead is:

$$R(2|w|m + N_B|w|). \tag{27}$$

C. PROOF OF THEOREM 3

Let k_i be a client in the sequence $[k_1, \ldots, k_m]$. Following the steps of Algorithm 3, three main operations are performed: 1) MERGE, 2) CLIENTUPDATE and 3) send of the model parameters to the next client of the sequence. The first one is the average of two model parameters, so its cost is $2|w|$. The cost of the second operation has already been computed in (19) and the cost of parameter sharing is $|w|$. By summing up these contributions we obtain:

$$m\left[E\left(|D_{\max}||w| + 2\frac{|D_{\max}|}{B}|w|\right) + 3|w|\right]. \tag{28}$$

This process is repeated for R rounds, so the time complexity is:

$$Rm\left[E\left(|D_{\max}||w| + 2\frac{|D_{\max}|}{B}|w|\right) + 3|w|\right], \tag{29}$$

where the first addend is the dominant one.

Given that each client shares its local model only with the following node in the sequence, the communication overhead is given by (8).

APPENDIX B
EDGE CONNECTION MODEL

To compute the total duration for transmitting model weights, we assume IEEE 802.11ax channel access procedures [42], which also include the overheads to carry out the distributed coordination function (DCF) operation. In particular, the duration of a packet transmission is defined as:

$$T_{\text{Tx}} = Rm(T_{\text{RTS}} + T_{\text{SIFS}} + T_{\text{CTS}} + T_{\text{DATA}} + T_{\text{SIFS}} + T_{\text{ACK}} + T_{\text{DIFS}} + T_{\text{e}}), \tag{30}$$

where T_{RTS} is the duration of the ready-to-send (RTS) control frame, T_{SIFS} is the short interframe space (SIFS) duration,

T_{CTS} is the duration of the clear-to-send (CTS) control frame, T_{DATA} is the duration of the data payload, T_{ACK} is the duration of the acknowledgement (ACK) frame, T_{e} is the duration of an empty slot, R is the number of FL rounds, and m the number of participating clients.

To compute the duration of each type of IEEE 802.11ax control frame, i.e., RTS, CTS, and ACK, we compute them as:

$$T_{\text{RTS/CTS/ACK}} = T_{\text{PHY}} + \left\lceil \frac{L_{\text{SF}} + L_{\text{RTS/CTS/ACK}}}{L_{\text{s}}} \right\rceil \sigma_{\text{leg}}, \quad (31)$$

where T_{PHY} is the duration of the PHY preamble, L_{SF} is the length of the service field (SF), $L_{\text{RTS/CTS/ACK}}$ is the length of the control frame, L_{s} is the length of an orthogonal frequency division multiplexing (OFDM) symbol, and σ_{leg} is the duration of a legacy OFDM symbol.

As for the duration of the data payload, it is computed as:

$$T_{\text{DATA}} = T_{\text{HE-SU}} + \left\lceil \frac{L_{\text{SF}} + L_{\text{MAC}} + L_{\text{DATA}}}{L_{\text{s}}} \right\rceil \sigma, \quad (32)$$

where $T_{\text{HE-SU}}$ is the duration of the high-efficiency (HE) single-user field, L_{MAC} is the length of the MAC header, L_{DATA} is the length of a single data packet (in our case, it matches with the model size, S_w), and σ is the duration of an OFDM symbol. The number of bits per OFDM symbol will vary, so as the effective data rate, based on the employed modulation and coding scheme (MCS), which depends on the transmission power used.

REFERENCES

[1] *Ericsson Mobility Report November 2021*, Ericsson, Stockholm, Sweden, 2021.

[2] J. Wang, J. Liu, and N. Kato, "Networking and communications in autonomous driving: A survey," *IEEE Commun. Surveys Tuts.*, vol. 21, no. 2, pp. 1243–1274, 2nd Quart., 2019.

[3] "AI and compute." May 2018. [Online]. Available: https://openai.com/blog/ai-and-compute/

[4] E. Strubell, A. Ganesh, and A. McCallum, "Energy and policy considerations for deep learning in NLP," in *Proc. 57th Annu. Meeting Assoc. Comput. Linguist.*, 2019, pp. 3645–3650.

[5] T. Zhang and S. Mao, "An introduction to the federated learning standard," *Mobile Comput. Commun.*, vol. 25, no. 3, pp. 18–22, 2022.

[6] Q. Li et al., "A survey on federated learning systems: Vision, hype and reality for data privacy and protection," *IEEE Trans. Knowl. Data Eng.*, vol. 35, no. 4, pp. 3347–3366, Apr. 2023.

[7] M. Chen et al., "Distributed learning in wireless networks: Recent progress and future challenges," *IEEE J. Sel. Areas Commun.*, vol. 39, no. 12, pp. 3579–3605, Dec. 2021.

[8] S. Deng, H. Zhao, W. Fang, J. Yin, S. Dustdar, and A. Y. Zomaya, "Edge intelligence: The confluence of edge computing and artificial intelligence," *IEEE Internet Things J.*, vol. 7, no. 8, pp. 7457–7469, Aug. 2020.

[9] Z. Zhou, X. Chen, E. Li, L. Zeng, K. Luo, and J. Zhang, "Edge intelligence: Paving the last mile of artificial intelligence with edge computing," *Proc. IEEE*, vol. 107, no. 8, pp. 1738–1762, Aug. 2019.

[10] E. Ahvar, A.-C. Orgerie, and A. Lebre, "Estimating energy consumption of cloud, fog and edge computing infrastructures," *IEEE Trans. Sustain. Comput.*, vol. 7, no. 2, pp. 277–288, Apr.–Jun. 2022.

[11] J. Konečnỳ, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon, "Federated learning: Strategies for improving communication efficiency," 2016, *arXiv:1610.05492*.

[12] M. Aledhari, R. Razzak, R. M. Parizi, and F. Saeed, "Federated learning: A survey on enabling technologies, protocols, and applications," *IEEE Access*, vol. 8, pp. 140699–140725, 2020.

[13] D. C. Nguyen et al., "Federated learning meets blockchain in edge computing: Opportunities and challenges," *IEEE Internet Things J.*, vol. 8, no. 16, pp. 12806–12825, Aug. 2021.

[14] C. Xie, S. Koyejo, and I. Gupta, "Asynchronous federated optimization," 2019, *arXiv:1903.03934*.

[15] P. Kairouz et al., "Advances and open problems in federated learning," *Found. Trends® Mach. Learn.*, vol. 14, nos. 1–2, pp. 1–210, 2021.

[16] V. Mothukuri, R. M. Parizi, S. Pouriyeh, Y. Huang, A. Dehghantanha, and G. Srivastava, "A survey on security and privacy of federated learning," *Future Gener. Comput. Syst.*, vol. 115, pp. 619–640, Feb. 2021.

[17] L. Barbieri, S. Savazzi, M. Brambilla, and M. Nicoli, "Decentralized federated learning for extended sensing in 6G connected vehicles," *Veh. Commun.*, vol. 33, Jan. 2022, Art. no. 100396.

[18] A. Lalitha, S. Shekhar, T. Javidi, and F. Koushanfar, "Fully decentralized federated learning," in *Proc. 3rd Workshop Bayesian Deep Learn. (NeurIPS)*, 2018, pp. 1–9.

[19] R. Ormándi, I. Hegedüs, and M. Jelasity, "Gossip learning with linear models on fully distributed data," *Concurrency Comput. Pract. Exp.*, vol. 25, no. 4, pp. 556–571, 2013.

[20] L. Giaretta and Š. Girdzijauskas, "Gossip learning: Off the beaten path," in *Proc. IEEE Int. Conf. Big Data (Big Data)*, 2019, pp. 1117–1124.

[21] M. Miozzo, Z. Ali, L. Giupponi, and P. Dini, "Distributed and multi-task learning at the edge for energy efficient radio access networks," *IEEE Access*, vol. 9, pp. 12491–12505, 2021.

[22] F. Wilhelmi, L. Giupponi, and P. Dini, "Blockchain-enabled server-less federated learning," 2021, *arXiv:2112.07938*.

[23] X. Qiu, T. Parcollet, D. Beutel, T. Topal, A. Mathur, and N. Lane, "Can federated learning save the planet?" in *Proc. NeurIPS Tackling Climate Change Mach. Learn.*, 2020, pp. 1–9.

[24] S. Savazzi, V. Rampa, S. Kianoush, and M. Bennis, "An energy and carbon footprint analysis of distributed and federated learning," 2022, *arXiv:2206.10380*.

[25] M. Abadi et al. "TensorFlow: Large-scale machine learning on heterogeneous systems." 2015. [Online]. Available: https://www.tensorflow.org/

[26] J. Verbraeken, M. Wolting, J. Katzy, J. Kloppenburg, T. Verbelen, and J. S. Rellermeyer, "A survey on distributed machine learning," *ACM Comput. Surveys*, vol. 53, no. 2, pp. 1–33, 2020.

[27] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. Y. Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Proc. Artif. Intell. Stat.*, 2017, pp. 1273–1282.

[28] T. Li, A. K. Sahu, A. Talwalkar, and V. Smith, "Federated learning: Challenges, methods, and future directions," *IEEE Signal Process. Mag.*, vol. 37, no. 3, pp. 50–60, May 2020.

[29] M. Chen, H. V. Poor, W. Saad, and S. Cui, "Wireless communications for collaborative federated learning," *IEEE Commun. Mag.*, vol. 58, no. 12, pp. 48–54, Dec. 2020.

[30] M. Chen, Z. Yang, W. Saad, C. Yin, H. V. Poor, and S. Cui, "A joint learning and communications framework for federated learning over wireless networks," *IEEE Trans. Wireless Commun.*, vol. 20, no. 1, pp. 269–283, Jan. 2021.

[31] H. Kim, J. Park, M. Bennis, and S.-L. Kim, "Blockchained on-device federated learning," *IEEE Commun. Lett.*, vol. 24, no. 6, pp. 1279–1283, Jun. 2019.

[32] U. Majeed and C. S. Hong, "FLchain: Federated learning via MEC-enabled blockchain network," in *Proc. IEEE 20th Asia–Pac. Netw. Oper. Manag. Symp. (APNOMS)*, 2019, pp. 1–4.

[33] X. Bao, C. Su, Y. Xiong, W. Huang, and Y. Hu, "FLChain: A blockchain for auditable federated learning with trust and incentive," in *Proc. 5th Int. Conf. Big Data Comput. Commun. (BIGCOM)*, 2019, pp. 151–159.

[34] I. Hegedüs, G. Danner, and M. Jelasity, "Decentralized learning works: An empirical comparison of Gossip learning and federated learning," *J. Parallel Distrib. Comput.*, vol. 148, pp. 109–124, Feb. 2021.

[35] A. Nilsson, S. Smith, G. Ulm, E. Gustavsson, and M. Jirstrand, "A performance evaluation of federated learning algorithms," in *Proc. 2nd Workshop Distrib. Infrast. Deep Learn.*, 2018, pp. 1–8.

[36] A. Lacoste, A. Luccioni, V. Schmidt, and T. Dandres, "Quantifying the carbon emissions of machine learning," 2019, *arXiv:1910.09700*.

[37] L. Lannelongue, J. Grealey, and M. Inouye, "Green algorithms: Quantifying the carbon footprint of computation," *Adv. Sci.*, vol. 8, no. 12, 2021, Art. no. 2100707.

[38] R. Schwartz, J. Dodge, N. A. Smith, and O. Etzioni, "Green AI," *Commun. ACM*, vol. 63, no. 12, pp. 54–63, 2020.
[39] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," *Decentralized Bus. Rev.*, vol. 25, no. 7, 2008, Art. no. 21260.
[40] F. Wilhelmi and L. Giupponi, "Discrete-time analysis of wireless blockchain networks," in *Proc. IEEE 32nd Annu. Int. Symp. Pers. Indoor Mobile Radio Commun. (PIMRC)*, 2021, pp. 1011–1017.
[41] Z. Zheng, S. Xie, H. Dai, X. Chen, and H. Wang, "An overview of blockchain technology: Architecture, consensus, and future trends," in *Proc. IEEE Int. Congr. Big Data (BigData Congr.)*, 2017, pp. 557–564.
[42] B. Bellalta, "IEEE 802.11 AX: High-efficiency WLANs," *IEEE Wireless Commun.*, vol. 23, no. 1, pp. 38–46, Feb. 2016.
[43] N. Lasla, L. Al-Sahan, M. Abdallah, and M. Younis, "Green-PoW: An energy-efficient blockchain proof-of-work consensus algorithm," *Comput. Netw.*, vol. 214, Sep. 2022, Art. no. 109118.
[44] "EMNIST—TensorFlow federated." Accessed: Dec. 2022. [Online]. Available: https://www.tensorflow.org/federated/api_docs/python/tff/simulation/datasets/emnist
[45] TensorFlow. "TensorFlow federated." Accessed: Dec. 2022. [Online]. Available: https://www.tensorflow.org/federated
[46] "Keras: The python deep learning API." Accessed: Dec. 2022. [Online]. Available: https://keras.io/
[47] M. Alharby and A. van Moorsel, "BlockSim: An extensible simulation tool for blockchain systems," *Front. Blockchain*, vol. 3, p. 28, Jun. 2020. [Online]. Available: https://www.frontiersin.org/article/10.3389/fbloc.2020.00028
[48] TensorFlow. "Federated learning for image classification." Accessed: Dec. 2022. [Online]. Available: https://github.com/tensorflow/federated/blob/v0.17.0/docs/tutorials/federated_learning_for_image_classification.ipynb
[49] L. F. W. Anthony, B. Kanding, and R. Selvan, "Carbontracker: Tracking and predicting the carbon footprint of training deep learning models," 2020, *arXiv:2007.03051*.
[50] J. Xu, B. S. Glicksberg, C. Su, P. Walker, J. Bian, and F. Wang, "Federated learning for healthcare informatics," *J. Healthcare Inf. Res.*, vol. 5, pp. 1–19, Mar. 2021.
[51] A. Rago, G. Piro, G. Boggia, and P. Dini, "Multi-task learning at the mobile edge: An effective way to combine traffic classification and prediction," *IEEE Trans. Veh. Technol.*, vol. 69, no. 9, pp. 10362–10374, Sep. 2020.
[52] C. Lanza, E. Angelats, M. Miozzo, and P. Dini, "Urban traffic forecasting using federated and continual learning," in *Proc. 6th Conf. Cloud Internet Things (CIoT)*, 2023, pp. 1–8.
[53] A. Hard et al., "Federated learning for mobile keyboard prediction," 2018, *arXiv:1811.03604*.
[54] Y. Shahsavari, K. Zhang, and C. Talhi, "A theoretical model for fork analysis in the bitcoin network," in *Proc. IEEE Int. Conf. Blockchain (Blockchain)*, 2019, pp. 237–244.
[55] Y. Huang et al., "Continual learning for peer-to-peer federated learning: A study on automated brain metastasis identification," 2022, *arXiv:2204.13591*.
[56] F. Wilhelmi, S. Barrachina-Muñoz, and P. Dini, "End-to-end latency analysis and optimal block size of proof-of-work blockchain applications," *IEEE Commun. Lett.*, vol. 26, no. 10, pp. 2332–2335, Oct. 2022.
[57] S. R. Pokhrel and J. Choi, "Federated learning with blockchain for autonomous vehicles: Analysis and design challenges," *IEEE Trans. Commun.*, vol. 68, no. 8, pp. 4734–4746, Aug. 2020.
[58] Z. Ji, L. Chen, N. Zhao, Y. Chen, G. Wei, and F. R. Yu, "Computation offloading for edge-assisted federated learning," *IEEE Trans. Veh. Technol.*, vol. 70, no. 9, pp. 9330–9344, Sep. 2021.
[59] T. Nishio and R. Yonetani, "Client selection for federated learning with heterogeneous resources in mobile edge," in *Proc. IEEE Int. Conf. Commun. (ICC)*, 2019, pp. 1–7.
[60] Y. Song, T. Liu, T. Wei, X. Wang, Z. Tao, and M. Chen, "FDA^3: Federated defense against adversarial attacks for cloud-based IIoT applications," *IEEE Trans. Ind. Informat.*, vol. 17, no. 11, pp. 7830–7838, Nov. 2021.
[61] C. Fung, C. J. Yoon, and I. Beschastnikh, "Mitigating sybils in federated learning poisoning," 2018, *arXiv:1808.04866*.
[62] Y. Lu, X. Huang, K. Zhang, S. Maharjan, and Y. Zhang, "Blockchain empowered asynchronous federated learning for secure data sharing in Internet of Vehicles," *IEEE Trans. Veh. Technol.*, vol. 69, no. 4, pp. 4298–4311, Apr. 2020.
[63] M. Shayan, C. Fung, C. J. M. Yoon, and I. Beschastnikh, "Biscotti: A blockchain system for private and secure federated learning," *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 7, pp. 1513–1525, Jul. 2020.
[64] J. Kang, Z. Xiong, D. Niyato, Y. Zou, Y. Zhang, and M. Guizani, "Reliable federated learning for mobile networks," *IEEE Wireless Commun.*, vol. 27, no. 2, pp. 72–80, Apr. 2020.
[65] D. Li et al., "Blockchain for federated learning toward secure distributed machine learning systems: A systemic survey," *Soft Comput.*, vol. 26, no. 9, pp. 4423–4440, 2022.

ELIA GUERRA (Student Member, IEEE) received the bachelor's degree in information engineering and the master's degree in computer engineering from the University of Padova, Italy, in 2019 and 2021, respectively. He is currently pursuing the Ph.D. degree with the Technical University of Catalonia. He is currently working with CTTC for the GREENEDGE (MSCA ETN) Project. During his studies, he developed a passion for machine learning and algorithms. His main research lines are distributed/decentralized and sustainable machine learning algorithms.

FRANCESC WILHELMI (Member, IEEE) received the B.Sc. degree in telematics engineering, the M.Sc. degree in intelligent and interactive systems, and the Ph.D. degree in information and communication technologies from Universitat Pompeu Fabra in 2015, 2016, and 2020, respectively. He is currently working as a Researcher with Nokia Bell-Labs.

MARCO MIOZZO received the M.Sc. degree in telecommunication engineering from the University of Ferrara, Italy, in 2005, and the Ph.D. degree from the Technical University of Catalonia (UPC) in 2018. In June 2008, he joined the Centre Tecnologic de Telecomunicacions de Catalunya (CTTC). In CTTC, he has been involved in several EU funded projects. He participated in several research and development projects, among them SCAVENGE, 5G-Crosshaul, Flex5Gware, and SANSA, working on environmental sustainable mobile networks with energy harvesting capabilities through learning techniques. He is currently collaborating with the EU funded H2020 GREENEDGE (MSCA ETN) and SONATA (CHIST-ERA). His main research interests are sustainable mobile networks, green wireless networking, energy harvesting, multiagent systems, machine learning, green AI, and explainable AI.

PAOLO DINI received the M.Sc. and Ph.D. degrees from the Università di Roma La Sapienza in 2001 and 2005, respectively. He is currently a Senior Researcher with the Centre Tecnologic de Telecomunicacions de Catalunya. He has been involved in more than 25 research projects. He is currently the Coordinator of the CHIST-ERA SONATA Project on sustainable computing and communication at the edge and the Scientific Coordinator of the EU H2020 MSCA Greenedge European Training Network on edge intelligence and sustainable computing. His research activity is documented in almost 90 peer-reviewed scientific journals and international conference papers. His current research interests include sustainable networking and computing, distributed optimization and optimal control, machine learning, multiagent systems, and data analytics. He received two awards from the Cisco Silicon Valley Foundation for his research on heterogeneous mobile networks in 2008 and 2011, respectively. He serves as a TPC in many international conferences and workshops and a reviewer for several scientific journals of the IEEE, Elsevier, ACM, Springer, and Wiley.