Dealing With Challenged IoT Networks in Hierarchical Federated Learning

Dealing With Challenged IoT Networks in Hierarchical Federated Learning

Alessio Sacco, *Member, IEEE,* Doriana Monaco, *Student Member, IEEE,* Guido Marchetto, *Senior Member, IEEE,* and Paolo Montuschi, *Fellow, IEEE*

***Abstract*—Federated Learning has revolutionized the way in which mobile devices and IoT can share common knowledge in data analytics. However, some challenges arise when dealing with heterogeneous and challenged networks, especially in gradient synchronization. For example, some clients (referred to as stragglers) may take much longer to report their output than other nodes. Current solutions addressing the straggling problems either propose a distributed coordination (but introduce new synchronization issues) or deadline-based approaches to discard clients after a fixed deadline (but introduce the problem of determining a suitable deadline). To this end, we propose to set a dynamic deadline in which the central server selects the best IoT nodes via an online learning approach based on predicting the response time of each client. Moreover, to further mitigate synchronization and scalability issues, we also consider a hierarchical approach in which clients send model parameters to intermediate aggregation edge servers. Our results demonstrate that this approach can lower network overhead by 78% compared to the widely adopted FedAvg and 49% to the best alternative. At the same time, the model accuracy is preserved, and the training time in challenged networks is reduced by 52% w.r.t. FedAvg and 32% w.r.t. recent solutions.**

***Index Terms*—IoT, federated learning, machine learning, inference**

I. Introduction

The rapid growth of the Internet of Things (IoT) has led to an explosion of data generated at the network edge, creating opportunities for training machine learning models across diverse scenarios, such as smart cities, precision agriculture, healthcare, smart homes, and wearable devices [1]. Traditionally, data is uploaded to centralized servers for the training of Machine Learning (ML) models, but such direct data transfer to a central server poses serious privacy risks for user data sent from edge devices. Federated Learning (FL) offers a promising solution by enabling distributed, collaborative model training directly on IoT devices, with each device acting as a client that retains its raw data locally while exchanging only some parameters of the trained model, thus enhancing privacy by keeping sensitive user-specific data on the device [2]–[9]. Although this approach is especially beneficial in the IoT context, where data often includes personal information, implementing FL on resource-constrained IoT devices reveals several challenges [7], [10]–[14]. Edge devices typically have limited computational power and memory for storage and data access, impacting their ability to process and store complex models. More in detail, *(i)* device unreliability arises due to the heterogeneity and potential disconnections of devices, especially important in mobility-driven IoT like intelligent transportation, where network disruptions affect real-time data transmission [15]; *(ii)* aggregation efficiency is reduced, as slower devices, known as (*stragglers*), can delay faster ones during training rounds; and *(iii)* low resource utilization occurs because current node selection algorithms can restrict optimal device participation.

Two main approaches have been proposed in the literature to mitigate the straggler effect in FL: asynchronous update [16]–[19] and coded computation [20]–[23]. Since devices have different computing power and resources, but also intermittent availability, each of them may conduct local training based on differing versions of the global model. Asynchronous FL approaches aim to reconcile the asynchronous nature of updates from different devices and efficiently aggregate these divergent model versions into a cohesive global model. Some approaches, e.g., [17], [18], aim at equalizing the contributions from different devices. While this approach fits well with non-independent and identically distributed (IID) datasets, it might slow down the entire process in IID contexts, where older models of delayed devices contain little useful information for the updated models of other devices.

The core idea of the coded computation approach, instead, is to introduce redundancy in the computational tasks, ensuring that the overall computation can still progress efficiently even if certain devices or nodes experience delays or slower processing speeds. For example, in recent works [21], [23], clients share a part of their coded training data with the server, which computes the missing results from stragglers. However, such solutions require extra network bandwidth to send data fragments to the server, while IoT devices often face challenging network conditions or temporary absence of signal.

To solve these limitations, we present our Hierarchical and Predictive Federated Learning (HPFL) solution, which integrates the prediction of future network conditions during model parameter exchange to proactively anticipate and mitigate the straggler effect. In addition, we design a hierarchical and centralized FL framework to further alleviate possible latencies due to the challenging network conditions. The central server of HPFL uses a predictor to determine whether intermediate nodes are affected by intolerable delays and discards these nodes in the aggregation phase. As the latency prediction is a critical step of the solution, we use two predictors that differ in nature: Vector Autoregressive Moving

Alessio Sacco, Doriana Monaco, Guido Marchetto and Paolo Montuschi are with DAUIN, Politecnico di Torino, 10129 Turin, Italy (e-mail: alessio_sacco@polito.it, doriana.monaco@polito.it, guido.marchetto@polito.it, paolo.montuschi@polito.it).

Average (VARMA), a time-series method that uses historical information to forecast future values, and Random Forest Regression (RFR), a regression model that leverages diverse input metrics to establish correlations with future latency. Moreover, based on our findings that both approaches hold merit in varied settings, we have combined and interchangeably utilized them as needed, with this decision guided by a Follow the Perturbed Leader (FPL) setting.

The tests performed over the prototypal implementation we developed show how HPFL can reduce the network overhead and, at the same time, achieve the highest accuracy among all benchmarks on two well-known datasets, MNIST and CIFAR-10. Notably, the model converges $1.5\times$ faster than other approaches on average.

We summarize our contribution as follows:

- We evaluate a range of state-of-the-art forecasting models to predict client latency and identify the most suitable experts for the task.
- We design a node selection algorithm based on the Follow the Perturbed Leader (FPL) strategy to detect straggler devices in an online fashion.
- We develop a hierarchical Federated Learning framework to alleviate the computational burden of stragglers and combine it with the FPL-based prediction mechanism.
- We provide rigorous complexity and convergence analyses for both convex and non-convex loss functions.

The rest of the paper is structured as follows. Section II studies existing approaches to the straggler problem, highlighting the differences between our work and current solutions. Section III models the FL setup and its limitations in the IoT setting. Section IV models our hierarchical federated approach, providing an overview of the system components. Later on, we model the latency prediction problem and present our regression models (Section V), which are combined in the FPL algorithm, presented in Section VI. We assess the performance of our solution and present our experimental results in Section VII. Finally, Section VIII concludes the paper.

II. Related Work

In this section, we review the current state-of-the-art approaches addressing the stragglers problem, i.e., clients that take significantly longer than others to report their output.

Asynchronous FL. One of the main solutions to the straggler problem is represented by Asynchronous FL. In AFL, the server aggregation phase is performed whenever a model is received, without the need to wait for all the clients. A possible solution is to use deadline-based approaches in which all workers compute local gradients with a variable number of samples within a fixed global cycle [24]. Another line of work proposes asynchronous decentralized SGD, where workers update their models based on the latest iterates of their neighbors [25]. Although these asynchronous methods are inherently robust to stragglers, most of the solutions [16]–[18], [24], [25] may suffer from slow convergence due to the use of outdated models.

Node selection in FL. Several algorithms have been suggested to enhance the training efficiency of Asynchronous Federated Learning (AFL) across heterogeneous devices. Unlike classic FL, which tends to select nodes based on the quantity of their training data, AFL focuses on prioritizing nodes that exhibit increased resilience and computational capability. For example, [26] introduces a heuristic greedy node selection strategy based on the nodes local computing and communication resources, while [27] considers computing power and accuracy variations of local models on each node. A recent proposal [28] devises a sketching-based client selection to cluster similar clients together and then choose the fastest client from each cluster. While these strategies effectively enhance aggregation efficiency, achieving a balance between the robustness of the global model and the risk of overfitting poses a significant challenge in AFL. Additionally, this prioritized method of node selection overlooks the inherent device unreliability commonly associated with IoT devices. To overcome this limitation, FedAR [29] introduces a concept wherein a trust score is allocated to individual nodes based on their activities. Potential candidates failing to meet the task requirements of the ML problem are filtered out before the commencement of the training round. Participants successfully fulfilling tasks are rewarded, while those failing to do so experience a reduction in their trust value. Likewise, in SAFA [10], a node exhibiting a lower probability of crashing is given a higher likelihood of being chosen in an iteration. Nodes experiencing significant delays in training models, leading to excessive staleness, are labeled as deprecated and compelled to synchronize with the server. Finally, in [30], the authors propose a channel-aware scheduling policy, along with an age-aware aggregation policy, to improve the learning performance in wireless communications.

Coded computation. This is an alternative method to AFL to overcome the limitations of stragglers. It implies algorithmic redundancy to be able to reconstruct the original computation even in the presence of delays or failures. Recent work has explored the use of codes to accelerate training for distributed machine learning, such as [20], [31]–[34]. For example, *CodedPaddedFL* [20] introduces redundancy on the devices' local data, combining one-time padding to yield privacy with gradient codes to provide straggler resilience. First, the devices share an encoded version of their data with other devices. Then, the devices and the central server iteratively and collaboratively train a global model, and the computations of straggling servers can be ignored without loss of information. Although these methods are attractive to FL, they face fundamental challenges in federated networks, where data sharing or replication across devices is often not possible due to privacy and network size constraints. Therefore, new methods that provide fault tolerance would improve the use of FL in healthcare, where IoT devices are likely to fault, face difficult network conditions, and deplete their batteries.

FL for resource-constrained devices. The authors of [35] introduced a Federated Proximal, Activity, and Resource-Aware Lightweight model for a resource-constrained IoT environment called *FedPARL*. It is a tri-layer model that first conducts sample-based model pruning on the server to reduce the model size and ease the computation on the client side. Later on, it selects trustworthy clients based on their resources and,

finally, assigns the number of local training epochs to each client based on its resource availability. On the other hand, the authors of [36] proposed a heterogeneous federated stacking model to monitor patients' activities based on classification models, called *FedStack*. This model has the ability to process a variety of client model architectures and ensemble the local models into a robust global model, which in turn can reduce the impact of the straggler clients on the classification performance. Additional approaches include Memory-augmented Impatient Federated Averaging (MIFA) [37], which leverages past device information to correct the gradients bias, and Cascade Vertical Federated Learning (CVFL) [38], in which stragglers' gradients are amplified to contribute more. New solutions, such as [39], introduce Invariant Dropout, a technique that drops the straggler neurons that contribute minimally in the training, reducing run-time model sizes.

Edge computing in FL. Multi-access Edge Computing (MEC), previously known as mobile edge computing, has garnered significant interest from both academic and industrial domains. MEC refers to a platform that harnesses the available computation power at the edge of the network to execute computing tasks for mobile devices [40]. Indeed, leveraging MEC technology in FL scenarios can provide a solution for stragglers. Stragglers can offload their computation tasks to the edge server to alleviate the computational burdens faced by these devices, consequently reducing the delay in their updates. For example, EAFL [41] facilitates stragglers in offloading a portion of their computation to the edge server, harnessing the idle computing power of the server to aid clients in their model training processes, while P4FL [42] leverages the programmability of P4-based switches to compute intermediate aggregations in a hierarchical approach. [43] leverages wireless edge networks to cluster User Equipments (UEs) with similar training time together to perform synchronous cluster aggregation while it asynchronously updates the global model.

In this article, we leverage MEC computing to alleviate the computational burden of IoT devices and combine it with a novel algorithm for node selection that is based on the prediction of future network conditions. In the following way, we can predict the stragglers' behavior and proactively mitigate it.

III. Problem definition

In this section, we model the FL setup and define its limitations for heterogeneous IoT devices with constrained resources. All the symbols that we use are summarized in Table I.

Consider a federated machine learning (ML) setup with a (logical) centralized server and K IoT devices (clients). Each client stores a local dataset $\mathcal{D}_k$ with a corresponding size of D_k. Each dataset comprises a data sample i, composed of an input vector x_i (*e.g.*, pixels of an image or tabular row), and an output y_i (*e.g.*, label of the image or class label). In a typical ML problem, the goal is to find the optimal model parameter $\mathbf{w}$ that characterizes the relationship between x_i and y_i by minimizing the loss function $f_i(\mathbf{w})$ between the prediction $w^T x_i$ and the observation y_i. The loss function captures the error of the model in assigning output values in the training data and depends on the ML model. For example, we can use the Mean Squared Error (MSE) $\frac{1}{2}\|y_i - \mathbf{w}^T x_i\|^2$ for linear regression tasks, or leverage the Hinge Loss in classification tasks (*e.g.*, support vector machine), but also define convex or non-convex loss functions for neural networks. For each client k, we define its loss function as:

$$F_k(\mathbf{w}_k) = \frac{1}{D_k} \sum_{i \in \mathcal{D}_k} f_i(\mathbf{w}_k), \tag{1}$$

where $\mathbf{w}_k$ denotes client k's local model parameter. Given the inherent complexity of the majority of ML models, the loss function minimization is often performed by gradient-descent techniques. In this paper, we use the mini-batch gradient descent algorithm for training, as done in other studies [44]. Given t as the index of the update step and η as the learning rate, the model parameters update is defined as:

$$\mathbf{w_k}(t) = \mathbf{w_k}(t-1) + \eta \nabla F_k(\mathbf{w_k}(t-1)), \tag{2}$$

One option to train a global model in a distributed scenario is for all clients to offload their local datasets to a central server, which performs model training on the aggregated dataset. However, transferring client data can be time-consuming and can expose them to privacy and security issues. As such, in a Federated Learning-based approach, clients perform local updates based on their local datasets, and only

TABLE I: Summary of key notation

Symbol	Description
K	Total number of clients
$\mathcal{D}_k$,D_k	Dataset of client k and number of samples
t, T	Update step and total number of local training steps
η	Learning rate
N	Total number of rounds
T_{FL},t_w	FL delay, aggregation and propagation delay
W_k	Number of CPU cycles per round of client k
e_k	Number of CPU cycles per round of client k
τ	Number of epochs in a client round
U	CPU cycles required for training 1-bit
B_s	Batch size
$L(t)$,L	Set of edge servers and number of edge servers
$\mathcal{C}^l$,C^l	Client set of edge l and size
$\mathcal{D}^l$,D^l	Aggregated dataset of edge l and size
$\mathcal{D}$,D	Global dataset and size
k_1	Number of local steps before edge aggregation
k_2	Number of edge steps before cloud aggregation
$k_1 k_2$	Number of local steps before cloud aggregation
$w_k^l(t)$	Local model parameters of client k in edge l
$w^l(t)$	Edge model parameters
$w(t)$	Global model parameters
$w*$	Optimal model parameters that minimize $F(w)$
$t_k^l(i)$	Client k propagation delay at round i
$t^l(i)$	Edge server l total delay at round i
T_{HPFL}	Total delay of the system
$F(w), F^l(w), F_k(w)$	Global, edge and client loss function
δ, Δ	Client-edge and edge-cloud gradient divergence
ρ	Lipschitz parameter
β	Smoothness parameter
σ^2	Stochastic gradient noise

their model parameters are uploaded to the central server. The server aggregates these updated parameters to generate a global model and subsequently disseminates it to clients for their next local update. Prior to parameter aggregation, each client may engage in one or more model training epochs within the local update phase, and we refer to this timeframe as a "round."

The global model parameters can be updated in different ways, but we choose the widely used FedAvg algorithm [44] to update them as follows:

$$\mathbf{w}(t) = \frac{1}{D} \sum_{k \in K} D_k \mathbf{w}_k(t), \tag{3}$$

where $\mathbf{w}_k$ evolves locally as in Eq. (2). This process repeats until the model reaches the desired accuracy or the limited resources, *e.g.*, communication or time budget runs out.

The system delay of this setting is composed of local update delay, parameter aggregation delay, and model broadcast delay. At each round of the local update, typically, a fixed time slot t_w is reserved for parameter aggregation and model broadcast. Since the clients independently perform the local update, the system delay can be formulated as follows:

$$T_{FL} = N \max_{k \in K} \frac{W_k}{e_k} + N t_w, \tag{4}$$

where N denotes the number of total rounds in the FL problem, W_k represents the computation of client k per round (*i.e.*, the number of CPU cycles), and e_k is the CPU frequency of client k. As such, $\frac{W_k}{e_k}$ stands for the update delay of client k.

The computation of client k in a round, W_k, is proportional to its dataset size and is given as:

$$W_k = \tau B_n^k U B_s = \tau \left(\frac{D_k}{B_s} \right) U B_s = \tau D_k U, \tag{5}$$

where τ represents the number of epochs per round, B_n^k denotes the batch number of client k, B_s is the mini-batch size (training data size of one iteration), and the constant U denotes the number of CPU cycles required for training 1-bit data. IoT devices are characterized by heterogeneous resources, *e.g.*, CPU and RAM, and architectures. This is the cause of different processing times at each device, as pictured in Fig. 1. In particular, devices that take exceptionally long to perform local training, as $u3$ in the picture, are referred to as *stragglers* and will slow down the overall system since the server must wait for all the devices to send their model parameters. In extreme cases, the device may also fail and never respond. It is evident that the update delay of client k, which appears in Eq. 4, is affected by the presence of stragglers and is, therefore, essential to mitigate such problem.

IV. Hierarchical Federated Learning

In the following section, we model the Hierarchical Federated Learning setup. Later on, we describe our aggregation methodology, which is based on the selection of a subset of clients.

In our architecture, we consider an FL-enabled hierarchical system composed of three layers — client, edge and cloud —

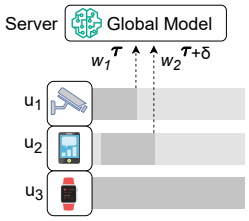


Fig. 1: Straggler effect

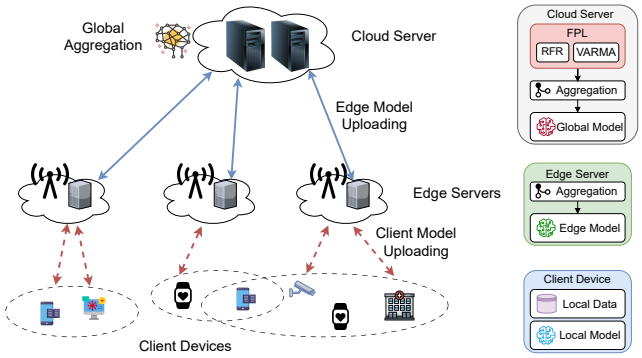


Fig. 2: HPFL architecture. Three layers constitutes the hierarchical architecture of the FL process.

as shown in Fig. 2. Our system consists of one cloud server, L edge servers indexed by l with disjoint client sets $\{C^l\}_{l=1}^{L}$ and K clients indexed by k.

Given our hierarchical system, we can consider aggregating data at the cloud server to involve numerous clients, but this comes with a substantial communication cost. Conversely, when aggregation occurs at the edge server, it only involves a limited number of clients, resulting in significantly lower communication expenses. For this reason, we employ two levels of aggregation: first, at the edge, and later on, at the cloud. Here, we extend the FedAvg algorithm to overcome the straggler's problem in our hierarchical system. Given $\mathcal{D}^l$ the aggregated dataset under edge l, our proposed algorithm works as follows: after every κ_1 steps on each client, the edge server aggregates the local clients' models. Then, after κ_2 edge model aggregations, the cloud server aggregates the edge servers' models. This means that the communication with the cloud takes place every $\kappa_1 \kappa_2$ local updates. In addition, to further mitigate straggler effects, we consider the case in which cloud aggregation would not aggregate from all local edge servers but only a subset. Therefore, the set of edge servers is time-varying and denoted as $L(t)$.

Let $\mathbf{w_k^l}(t)$ be the local model parameters of client k in edge l after the t-th local updates and $\mathbf{w^l}(t)$ the edge model parameters, the evolution of $\mathbf{w_k^l}(t)$ is given by:

$$\mathbf{w_k^l}(t) = \begin{cases} \mathbf{w_k^l}(t-1) + \eta \nabla F_k(\mathbf{w_k^l}(t-1)) & t \bmod \kappa_1 \neq 0 \\ \frac{1}{D^l} \sum_{k \in C^l} D_k \mathbf{w}_k^l(t) & t \bmod \kappa_1 = 0, \\ & t \bmod \kappa_1 \kappa_2 \neq 0 \\ \frac{1}{D} \sum_{l \in L(t)} D^l \mathbf{w}^l(t) & t \bmod \kappa_1 \kappa_2 = 0 \end{cases} \tag{6}$$

For each edge server, we predict its aggregation and broadcast delay $t_l(i)$ using the FPL module available in the server,

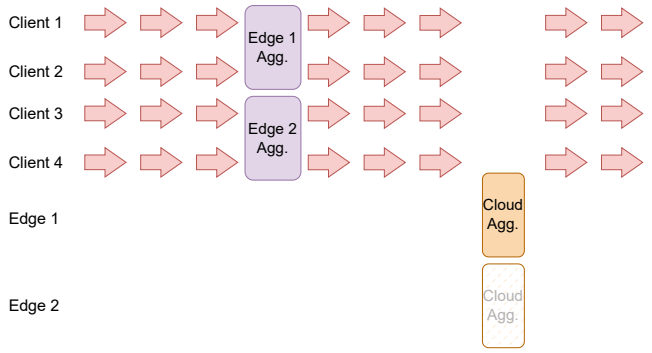


Fig. 3: Training and communication process of HPFL when $\kappa_1 = 3$ and $\kappa_2 = 2$.

described in Section V. Based on the predicted value, at time $\kappa_1\kappa_2$, we select the best edge servers to consider for the global model. It must be noted that for the edge aggregation, we consider all the local clients, as the network latencies are assumed to be limited and similar among all clients.

The system delay of HPFL is now determined by:

$$T_{HPFL} = N \max_{k\in K} \frac{W_k}{e_k} + \sum_{i=1}^{N}((i \bmod \kappa_1) == 0) \max_{k\in K} t_k^l(i) + \\ + \sum_{i=1}^{N}((i \bmod \kappa_1\kappa_2) == 0) \max_{l\in L(i)} t^l(i), \tag{7}$$

where $t_k^l(i)$ is the propagation delay from client k to edge server l at time i, and $t^l(i)$ is the total delay (aggregation and propagation) of edge l observed by the cloud at round i. N is the total amount of local updates to perform, which is an integer multiple of $\kappa_1\kappa_2$.

We show a visual representation of the process in Fig. 3, where an example with 4 clients, $\kappa_1 = 3$ and $\kappa_2 = 2$ is shown. Edge 2 is predicted to be a straggler and, therefore, is neglected at the global aggregation round in order to keep the aggregation phase short.

V. Regression Prediction Methods

To enhance FL convergence, the central agent forecasts future conditions to determine if favorable conditions persist. The agent receives guidance from two distinct categories of predictors and combines the most advantageous aspects of both. Specifically, we choose two algorithms from the time-series and ML-supervised regressors domains. In the following, we outline the behavior of these two methods and elaborate on how their distinctions can be combined for a more accurate predictor.

A. Predicting with Time-Series

To capture the temporal changes in data, we employ a Vector Autoregressive Moving-Average (VARMA) model. VARMA models are an extension of univariate Autoregressive-Moving Average (ARIMA) models to multivariate settings. While ARIMA models are commonly used to represent stationary time series in various domains where a single variable is measured at regular intervals, VARMA models can handle multiple parallel time series, allowing for a multivariate evolution. These models are particularly well-suited for applications in econometrics and financial markets but have also been extensively explored in other fields since the 1970s [45].

Our proposed approach utilizes a VARMA model to generate "real-time" predictions (hindcasts) within an independent dataset, employing only data up to the respective date. The equation below represents the general form of VARMA(r, q):

$$y_t = A_1 y_{t-1} + \ldots + A_r y_{t-r} + B_0\epsilon_t + \ldots + B_q\epsilon_{t-q}, \tag{8}$$

where y_t represents a vector of observed variables with dimensions $n \times 1$, while ϵ_t represents a vector of unobserved disturbances with dimensions $n \times 1$. These disturbances follow an independent and identically distributed (IID) distribution, denoted as $IID(O_{n\times 1}, I_n)$, where I_n refers to the identity matrix of size $n \times n$. The values of r and q are assumed to be non-negative integers, ensuring that at least one of them is positive.

In our approach, we employ a forecasting technique called minimization of the Mean Squared Forecast Error (MSFE) to predict future values of the series. The MSFE serves as a measure of prediction accuracy, taking into account the cumulative errors encountered so far. The current information available from the dataset, representing our current knowledge, includes both the current and past values of the series. Specifically, we concentrate on the one-step-ahead prediction, which just focuses on forecasting the next time step, denoted as y_{t+1}, based on the latest observation at time t.

B. Predicting with ML Regression

As an example of ML regression, we use a Random Forest Regression (RFR) model. RFR is an additive model that makes predictions by combining decisions made by a sequence of base models. To be more precise, this group of algorithms can be expressed as follows:

$$g(x) = f_0(x) + f_1(x) + f_2(x) + \ldots. \tag{9}$$

In this formulation, the ultimate model g is the aggregation of individual base models f_i. While each base model f_i can be implemented using any machine learning algorithm, the widely adopted approach in RFR involves using simple decision trees. In this paper, we also adopt this particular configuration. This general approach of combining multiple models to enhance predictive performance is commonly referred to as model ensembling. Furthermore, in RFR, all base models (decision trees) are built and trained independently, utilizing distinct subsets of data.

Subsequently, predictions are generated by averaging the predictions from each individual decision tree. In such a way, similar to how a forest comprises numerous trees, the random forest model is also an amalgamation of decision tree models. This characteristic endows random forests with substantial modeling capabilities, surpassing those of a single decision tree. RFR is well-suited for regression problems due to the following features it possesses: *(i)* the ability to capture non-linear or intricate relationships between inputs and outputs,

(ii) enhanced robustness compared to a solitary decision tree, as RFR employs a set of uncorrelated decision trees that diminishes sensitivity to noise within the training set, and *(iii)* the capacity to alleviate both variance and bias, thereby effectively mitigating the issue of overfitting.

C. Modeling the Latency Prediction Problem

In our system, we represent the current knowledge Y as a multi-dimensional matrix with dimensions $N \times M \times L$. Each row i contains the information of round i. For each round i, there are three features per clients collected: *(i)* the duration at round $i-1$, normalized to the highest value observed among clients, *(ii)* the RTT value sampled at round i, normalized to the highest value at the round, *(iii)* the arrival order of clients at round $i-1$. These three features are inherently correlated and essential for accurately forecasting aggregation time needed by each edge server for round $i+1$. Some additional features can include CPU and RAM usage of devices, packet loss rate, latency jitter, however, choosing a larger value for M (the number of features in our model) can result in more intricate models with a greater number of parameters, which could potentially lead to overfitting problems. On the other hand, choosing a smaller value for M can lead to simpler models, but it may not capture the full range of possible behaviors.

By opting for $M = 3$ and selecting metrics that are straightforward to gather, we are actively aiming to trade off the completeness of information and model complexity. Our design choice allows the central server to collect the needed information without the need to communicate with others, where RTT is easily obtained with little overhead in the packet header. This choice also leads to limited memory occupation yet high accuracy for this predictive component, as demonstrated by results (Section VII).

However, due to the distinctions between the two models, they also handle input data differently. In the case of the VARMA model, its input consists of the vector of metrics at timestamp t, namely $y_t, y_{t-1}, \ldots, y_1$. This implies that the input for the VARMA model encompasses all the values within the matrix. When the number of rows in Y surpasses a specified threshold ($Z = 1000$), the considered temporal window is restricted to a sub-matrix that solely includes the last Z rows. On the other hand, in the context of RFR, as it is agnostic to the chronological order of the data, it solely takes into account the last row of the matrix Y as input. Specifically, the input for the RFR model is composed of all the columns corresponding to the last row of the matrix.

Based on these inputs, the two predictors will forecast the aggregation time needed by each edge server for round $i+1$. We combine these two predictors with Follow the Perturbed Leader (FPL) method, which selects the best predictor to use at time t.

VI. Combining the Predictions with FPL

We now provide an overview of the procedure based on the Follow the Perturbed Leader (FPL) method. Subsequently, we explain how our system's central server implements our customized version of FPL in the context of Federated Learning.

A. Follow the Perturbed Leader Formulation

Online learning involves making sequential decisions based on evolving environmental conditions. This problem is viewed as one of the central challenges of machine learning. Then, it is not surprising that over the years, numerous online learning algorithms have been developed [46], [47]. In our research, we specifically explore the Follow the Perturbed Leader (FPL) algorithm, which presents an interesting property: simplicity and computational efficiency.

Our prediction with expert advice evolves according to the following formulation. During each time step t, the system generates sequential predictions $y_t \in \mathcal{Y}$. Starting from $t = 1$ and onwards, we have access to the predictions $(y_t^i)_{1 \leq i \leq n}$ made by n experts belonging to the set $\mathcal{E} = e_1, ..., e_n$. Once a prediction has been made, we obtain an observation $x_t \in \mathcal{X}$. The system then calculates the loss we incurred, denoted as $z(x_t, y_t)$, as well as the loss for each expert, represented as $z(x_t, y_t^i)$. Since our observations involve continuous values and correspond to a regression problem, the loss is computed using the formula $z(x_t, y_t) = (y_t - x_t)^2$.

Therefore, the system aims to minimize regret, which is defined as the difference between the learner's cumulative loss and the cumulative loss of the best possible prediction in hindsight. In other words, this means trying to achieve a cumulative loss "not significantly worse" than the best expert after T time steps. In more formal terms, we represent the accumulated loss of expert i as $Z_i^T = \sum_{t=1}^{T} z(x_t, y_t^i)$, and the cumulative loss of our system as $Z^T = \sum_{t=1}^{T} z(x_t, y_t)$. The regret for T iterations is defined as follows:

$$R_T = \sum_{t=1}^{T} z(x_t, y_t) - \min_{i \in 1..n} \sum_{t=1}^{T} z(x_t, y_t^i) = Z^T - \min Z_i^T, \tag{10}$$

where the term $\min Z_i^T$, is often defined as the loss of the *best expert in hindsight (BEH)*. Furthermore, if the regret is sublinear, specifically when $R_s \leq o(T)$, the learning algorithm is referred to as *Hannan-consistent*.

A possible algorithm that accomplishes Hannan-consistency is Follow the Perturbed Leader (FPL), as demonstrated by Hannan [48] and Kalai and Vempala [49]. In this approach, let γ represent an n-dimensional random variable, and η_i denote positive values. FPL involves selecting the expert exp that minimizes the cumulative loss perturbed by the following expression:

$$exp = \arg\min_i (Z_i + \eta\gamma_i). \tag{11}$$

As it can be observed, if the value of η is small, the selected expert exp will be "close" to a minimizer of the cumulative loss without perturbations. Conversely, when η is large, we expect $\mathcal{Y}$ to be "close" to a uniform distribution. In other words, the parameter η governs the degree of similarity between the algorithm and Follow the Leader, which is a variant of the algorithm that always chooses the expert with the minimum cumulative loss. However, it's worth noting that Follow the Leader, along with other deterministic learning algorithms, is not Hannan-consistent [50].

The mechanism of FPL is used to determine the *expert* predictor at each step, among VARMA and RFR, that will estimate the client's aggregation time.

Algorithm 1 Learning procedure in HPFL.

1: **procedure** HPFL ▷ Overall procedure
2: Initialize all clients parameters $\mathbf{w}_k^l(0)$
3: $\eta > 0$, $L_i \leftarrow 0$
4: Initialize threshold Th
5: **for** $i = 1, 2, \cdots, N$ **do**
6: **for** each client $k = 1, 2, \cdots, K$ in parallel **do**
7: $\mathbf{w}_k^l(i) = \mathbf{w}_k^l(i-1) + \eta \nabla F_k(\mathbf{w}_k^l(i-1))$
8: **if** $i \bmod \kappa_1 = 0$ **then**
9: **for** each edge $l = 1, 2, \cdots, L$ in parallel **do**
10: $\mathbf{w}^l(i) \leftarrow$ EdgeAggregation$(\{\mathbf{w}_k^l(i)\}_{k \in C^l})$
11: **if** $i \bmod \kappa_1 \kappa_2 \neq 0$ **then**
12: **for** each client $k \in C^l$ in parallel **do**
13: $\mathbf{w}_k^l(i) \leftarrow \mathbf{w}^l(i)$ ▷ Local model downloading
14: **if** $i \bmod \kappa_1 \kappa_2 = 0$ **then**
15: $\mathbf{w}(i) \leftarrow$ CloudAggregation$\big(\{\mathbf{w}^l(i)\}_{l \in L(i)}\big)$
16: **for** each client $k = 1, 2, \cdots, K$ in parallel **do**
17: $\mathbf{w}_k^l(i) \leftarrow \mathbf{w}(i)$ ▷ Global model downloading
18: **procedure** EDGEAGGREGATION$(\{\mathbf{w}_k^l(i)\}_{k \in C^l})$ ▷ Local
19: $\mathbf{w}^l(i) \leftarrow \frac{1}{D^l} \sum_{k \in C^l} D_k \mathbf{w}_k^l(i)$
20: **return** $\mathbf{w}^l(i)$
21: **procedure** CLOUDAGGREGATION$(\{\mathbf{w}^l(i)\}_{l \in L(i)})$ ▷ Global
22: Monitor the channel with all edge servers
23: **for** each edge $l = 1, 2, \cdots, L$ in parallel **do**
24: **for** every expert e **do** ▷ FPL
25: Compute loss z of last prediction given evidence s
26: Accumulate the loss $Z_e^i \leftarrow Z_e^{i-1} + z$
27: Sample $\gamma_e \sim \mathcal{N}(0, I)$
28: $exp \leftarrow \arg\min_e (Z_e^i + \eta \gamma_e)$
29: Predict y_i asking to the expert exp given state s
30: **if** $y_i > Th$ **then**
31: Discard edge server l
32: **else**
33: Enqueue edge server l in $L(i)$
34: Store states for the future
35: $\mathbf{w}(i) \leftarrow \frac{1}{D} \sum_{l \in L(i)} D^l \mathbf{w}^l(i)$
36: **return** $\mathbf{w}(i)$

B. HPFL Process

We can now formalize the overall algorithm of HPFL (Algorithm 1). The process evolves as mentioned in Section IV in the hierarchical FL setting, where clients train the different models in parallel, and every κ_1 local updates, the `EdgeAggregation` function is called, and the clients send gradients to the edge server, that perform local aggregation, and send the result (*i.e.*, the average in FedAvg setting) back to the clients. Then, after κ_2 edge model aggregations, the cloud server aggregates the models of servers whose predicted latency is below a certain threshold T (set according to the scenario considered). This prediction is the outcome of the FPL algorithm.

C. Complexity Analysis

The overall time complexity of Algorithm 1 can be decomposed into three principal components. First, each of the K clients performs a local model update in every round, running in parallel and contributing a total cost of $O(N)$. Second, an edge-level aggregation is triggered every κ_1 rounds. Each edge server receives and aggregates the weights from its clients, adding a complexity of $O\left(\frac{N}{\kappa_1} \cdot K\right)$ overall. In addition, after each aggregation, clients update their local models with the aggregated edge model, which incurs a further cost of $O\left(\frac{N}{\kappa_1}\right)$. The third component is the global aggregation phase, executed every $\kappa_1 \cdot \kappa_2$ rounds. This phase involves (i) aggregating the models from L edge servers using the Follow the Perturbed Leader (FPL) strategy with E experts, at a cost of $O\left(\frac{N}{\kappa_1 \cdot \kappa_2} \cdot (L \cdot E)\right)$, and (ii) synchronizing the global model across all K clients (lines 16–17), which contributes an additional $O\left(\frac{N}{\kappa_1 \cdot \kappa_2}\right)$.

Combining these, the total computational complexity of the algorithm is expressed as

$$O\left(N + \frac{N(K+1)}{k_1} + \frac{N(L \cdot E + 1)}{k_1 \cdot k_2}\right).$$

In practice, since K and N are significantly larger than κ_1, κ_2, L, and E, the dominant contribution is $O(N \cdot K)$.

D. Convergence analysis

In this subsection, we prove the convergence of our hierarchical process, whether the loss function is convex or not. We first analyze the traditional scenario with only one central aggregation every τ steps, then we move to a two-layer aggregation scheme as done in HPFL.

Assumption 1. *(Convex loss function) Let* $w* := \arg\min F(\mathbf{w})$ *and holding the following conditions on the loss function for each client:*

1.a $F(\mathbf{w})$ *is convex*
1.b $F(\mathbf{w})$ *is* ρ*-Lipschitz, i.e.,* $||F(\mathbf{w}) - F(\mathbf{w}*)|| \leq \rho ||\mathbf{w} - \mathbf{w}^*||$ *for any* w, w'
1.c $F(\mathbf{w})$ *is* β*-smooth, i.e.,* $||\nabla F(\mathbf{w}) - \nabla F(\mathbf{w}^*)|| \leq \beta ||\mathbf{w} - \mathbf{w}^*||$ *for any* $\mathbf{w}, \mathbf{w}^*$

Lemma 1. *(Traditional FL) Let* F_{v_b} *be the loss function of a centralized gradient descent setting,* M *the upper bound of the deviation of distributed weights, a parameter* $\epsilon > 0$*, a constant value* $\varphi = w(1 - \frac{\beta\eta}{2})$*, where* $w = min_n \frac{1}{||v_b((n-1)\tau) - \mathbf{w}*||^2}$ *, if these conditions hold:*

- $\eta \leq \frac{1}{\beta}$
- $\eta\varphi - \frac{\rho M}{\tau \epsilon^2} > 0$
- $F_{v_b}(n\tau) - F(\mathbf{w}*) \geq \epsilon$ *for all* n
- $F(\mathbf{w}(T)) - F(\mathbf{w}*) \geq \epsilon$

then the convergence upper bound after T iterations is:

$$F(\mathbf{w}(T)) - F(\mathbf{w}*) \leq \frac{1}{T\left(\eta\varphi - \frac{\rho M}{\tau\epsilon^2}\right)}, \tag{12}$$

In particular, it is demonstrated that this is true for any $\epsilon \geq \frac{1}{2\eta\varphi T} + \sqrt{\frac{1}{4\eta^2\varphi^2T^2} + \frac{\rho M}{\eta\varphi\tau}}$ [51].

We now focus on our hierarchical scenario, introducing some terms and definitions first. The total number of local iterations T at each device is divided into J cloud intervals. We use $[p]$ to represent the edge interval from $(p-1)k_1$ to pk_1, and $\{j\}$ to represent the cloud interval from $(j-1)k_1k_2$ to jk_1k_2. Therefore, the edge intervals in a cloud interval are $p = (j-1)k_2+1, (j-1)k_2+2, ..., jk_2$. The global model parameters are computed as in Eq. 6:

$$\mathbf{w}(t) = \frac{1}{D}\sum_{l\in L(t)} D^l\mathbf{w}^l(t) \tag{13}$$

We also define a virtually centralized gradient descent update that is performed at the cloud and is updated in the following way:

$$\mathbf{u}_{[j]}(t+1) = \mathbf{u}_{[j]}(t) - \eta\nabla F(\mathbf{u}_{[j]}(t)). \tag{14}$$

After k_1k_2 steps, it is synchronized with $w(t)$ such that:

$$\mathbf{u}_{[j]}((j-1)k_1k_2) = \mathbf{w}((j-1)k_1k_2) \tag{15}$$

To prove the convergence of Algorithm 1, we demonstrate that the difference between the virtually centralized weights and the true weights is bounded.

Now, we introduce a metric to quantify the difference between the gradient of a local loss function and that of the aggregation loss function. This measure reflects the impact of data distribution variations across different nodes.

Definition 1. *(Gradient Divergence). For any weight parameter w, the gradient divergence is given by:*

$$||\nabla F_k(\mathbf{w}) - \nabla F^l(\mathbf{w})|| \leq \delta_k^l \tag{16}$$

$$||\nabla F^l(\mathbf{w}) - \nabla F(\mathbf{w})|| \leq \Delta^l \tag{17}$$

where δ_k^l is the difference between each client's gradients and its edge model, and Δ^l is the difference between each edge model's gradients and the global ones.

We define $\delta = \frac{\sum_{k=1}^{K}|D_k^l|\delta_k^l}{|D|}$ as the client-edge divergence and $\Delta = \frac{\sum_{l=1}^{L}|D^l|\Delta^l}{|D|}$ as the edge-cloud divergence. A larger gradient divergence means that the dataset distribution is more non-IID, as the local model diverges more from the global model.

We now analyze the convergence of HPFL. As in Assumption 1, we assume that our client loss function $F_k(w)$ is convex, ρ-Lipschitz and β-smooth.

Lemma 2. *(Convex) There is an upper bound on the difference between the global model parameters $w(t)$ and the virtually centralized parameters $u_{[j]}(t)$ for any t in the interval $[j]$ [52]:*

$$||\mathbf{w}(t) - \mathbf{u}_{[j]}(t)|| \leq H_c(t) \tag{18}$$

where

$$\begin{aligned} H_c(t) &= h(t-(j-1)k_1k_2, \Delta) \\ &+ h(t-((j-1)k_2+p(t)-1)k_1, \delta) \\ &+ \tfrac{k_1}{2}(p^2(t)+p(t)-2)h(k_1,\delta) \end{aligned}$$

$$h(x,\delta) = \tfrac{\delta}{\beta}((\eta\beta+1)^x - 1) - \eta\delta x$$
$$p(x) = \lceil \tfrac{x}{k_1} - (j-1)k_2 \rceil$$

Remark 1. As we have seen previously, if $\delta = \Delta = 0$, the data is IID and, in this case, $H_c(t) = 0$, while if the data is non-IID, the bound increases. $H_c(t)$ also depends on k_1 and k_2, and increases for less frequent aggregations, while if $k_1k_2 = k_1$ it is equivalent to the traditional FL mechanism and $H_c(t) = h(t-(j-1)k_1, \Delta+\delta)$. When $k_1 = k_2 = 1$, it is equivalent to traditional gradient descent with $H_c(k_1k_2) = 0$.

Remark 2. Note that $H_c(t) \leq H_c(k_1k_2)$ because $k1k2$ is the last step before the first cloud aggregation. Therefore, it represents the maximum difference between true and virtually centralized weights. In particular,

$$H_c(k_1k_2) = h(k_1k_2, \Delta) + \frac{1}{2}(k_1+1)(k_2^2+k_2+2)h(k_1,\delta), \tag{19}$$

where the first term is given by the edge-cloud divergence while the second term is caused by the client-edge divergence.

Theorem 1. *(Convex) Assuming $F_k(\mathbf{w})$ is convex, ρ-continuous, and β-smooth, if $\eta \leq \frac{1}{\beta}$, $\eta\varphi - \frac{\rho H_c(k_1k_2)}{k_1k_2\epsilon^2} > 0$, $F(\mathbf{u}_{[j]}(jk_1k_2)) - F(\mathbf{w}*) \geq \epsilon$ for all j and $F(\mathbf{w}(T)) - F(\mathbf{w}*) \geq \epsilon$, then, after T local updates, we have the following convergence upper bound of HPFL for any $\epsilon \geq \frac{1}{2\eta\varphi T} + \sqrt{\frac{1}{4\eta^2\varphi^2T^2} + \frac{\rho H_c(k_1k_2)}{\eta\varphi k_1k_2}}$:*

$$F(\mathbf{w}(T)) - F(\mathbf{w}*) \leq \frac{1}{T(\eta\varphi - \frac{\rho H_c(k_1k_2)}{k_1k_2\epsilon^2})} \tag{20}$$

Proof. The result is a derivation of *Lemma 1*, where the upper bound is given by $H_c(k_1k_2)$, as defined in Eq. 19. From the same equation, it is evident that a smaller k_1 and a larger value for k_2 produces a minor deviation $H_c(k_1, k_2)$, and, as can be seen in Eq. 20, the model converges faster. □

Then, we perform the convergence analysis of HPFL when using a non-convex loss function, e.g., when using CNNs [51]. Since $F(\cdot)$ is non-convex, the algorithm may converge to a local minimum or a saddle point. Thus, it is important to study the gradient norm in this case [51], [53].

Assumption 2. *(Non-convex loss function) Let σ^2 be the stochastic gradient noise. We assume the following conditions:*

- *2.a $\nabla F_k(\mathbf{w})$ is ρ-Lipschitz, i.e., $||\nabla F_k(\mathbf{w}) - \nabla F_k(\mathbf{w}')|| \leq \rho||\mathbf{w}-\mathbf{w}'||$, for any $k, \mathbf{w}, \mathbf{w}'$*
- *2.b $\mathbb{E}_{\xi_i\sim\mathcal{D}_k}||g(\mathbf{w}_k,\xi_i) - \nabla F_k(\mathbf{w}_k)||^2 \leq \sigma^2, \forall k, \mathbf{w}_k$, where $g(\mathbf{w}_k)$ is the stochastic gradient of $F_k(w_k)$*
- *2.c $\frac{1}{C^l}\sum_{k\in\mathcal{C}^l}||\nabla F_k(\mathbf{w}) - \nabla F^l(\mathbf{w})||^2 \leq \delta^2, \forall l, \mathbf{w}$*
- *2.d $\sum_{i=1}^{L}\frac{C^l}{K}||\nabla F^l(\mathbf{w}) - \nabla F(\mathbf{w})||^2 \leq \Delta^2, \forall \mathbf{w}$*

Theorem 2. *(Non-convex) For any client-edge server mapping in HPFL that satisfies the assumptions, if $\eta < \frac{1}{2\sqrt{6}k_1k_2\rho}$ holds, for any $t \geq 1$, the gradient norm is bounded by:*

$$\begin{aligned}\frac{\sum_{t=0}^{T-1}\mathbb{E}||\nabla F(\bar{\mathbf{w}}(t))||^2}{T} \leq \frac{2(F(\mathbf{w}_0)-F(\mathbf{w}^*))}{\eta T} + \frac{\eta\rho\sigma^2}{K} \\ + 2\alpha\eta^2\rho^2k_1k_2\frac{L-1}{K}\sigma^2 \\ + 3\alpha\eta^2\rho^2k_1^2k_2^2\Delta^2 \\ + 2\alpha\eta^2\rho^2k_1(1-\frac{L}{K})\sigma^2 \\ + 3\alpha\eta^2\rho^2k_1^2\delta^2,\end{aligned} \tag{21}$$

where α is a fixed constant.

Remark 3. In this final form, the first term is the original SGD part, and then we have a term that depends on the edge-cloud divergence and noise and another term that depends on the client-edge interaction similarly. It is clear how the divergence contributes more to the bound w.r.t. the noise. Additionally, since $k_1k_2 > k_1$, the frequency of aggregations at the cloud has a stronger influence.

Proof. We first leverage the definition of the averaged global model at $t+1$, which is bounded by [54]:

$$\begin{aligned}\mathbb{E}\left[F(\bar{\mathbf{w}}(t+1))\right] = \mathbb{E}\left[F\left[\bar{\mathbf{w}}(t) - \eta\frac{1}{K}\sum_{l=1}^{L}\sum_{k\in C^l}g(\mathbf{w}_k(t))\right]\right] \\ \leq \mathbb{E}\left[F(\bar{\mathbf{w}}(t))\right] - \eta\mathbb{E}\left[\langle\nabla F(\bar{\mathbf{w}}(t)), \frac{1}{K}\sum_{l=1}^{L}\sum_{k\in C^l}g(\mathbf{w}_k(t))\rangle\right] \\ + \frac{\eta^2\rho}{2}\mathbb{E}||\frac{1}{K}\sum_{l=1}^{L}\sum_{k\in C^l}g(\mathbf{w}_k(t))||^2\end{aligned} \tag{22}$$

For definition, we substitute $g(\mathbf{w}_k)$ with the stochastic gradient of $F_k(\mathbf{w}_k)$. Then, we compute the inner product. Then, we bound the last term thanks to the stochastic gradient noise assumption 2.b. Finally, we rearrange the terms and obtain:

$$\begin{aligned}\mathbb{E}\left[F(\bar{\mathbf{w}}(t+1))\right] \leq \mathbb{E}\left[F(\bar{\mathbf{w}}(t))\right] + \frac{\eta^2\rho}{2K}\sigma^2 - \frac{\eta}{2}\mathbb{E}||\nabla F(\bar{\mathbf{w}}(t))||^2 \\ + \frac{\eta}{2}\mathbb{E}||\nabla F(\bar{\mathbf{w}}(t)) - \frac{1}{K}\sum_{l=1}^{L}\sum_{k\in C^l}\nabla F_k(\mathbf{w}_k(t))||^2\end{aligned} \tag{23}$$

Based on the Lipschitz gradient assumption 2.a, we can bound the last term:

$$\begin{aligned}\mathbb{E}||\nabla F(\bar{\mathbf{w}}(t)) - \frac{1}{K}\sum_{l=1}^{L}\sum_{k\in C^l}\nabla F_k(\mathbf{w}_k(t))||^2 \\ \leq 2\rho^2\sum_{l=1}^{L}\frac{C^l}{K}\mathbb{E}||\bar{\mathbf{w}}(t) - \bar{\mathbf{w}}^l(t)||^2 + 2\rho^2\frac{1}{K}\sum_{l=1}^{L}\sum_{k\in C^l}\mathbb{E}||\bar{\mathbf{w}}^l(t) - \mathbf{w}_k(t)||^2\end{aligned} \tag{24}$$

We now substitute in Eq.23, divide by $\frac{\eta}{2}$, rearrange the terms and take the average over time:

$$\begin{aligned}\frac{\sum_{t=0}^{T-1}\mathbb{E}||\nabla F(\bar{\mathbf{w}}(t))||^2}{T} \leq \frac{2(F(\mathbf{w}_0)-F(\mathbf{w}^*))}{\eta T} + \frac{\eta\rho\sigma^2}{K} \\ + 2\rho^2\frac{1}{T}\sum_{t=0}^{T-1}\sum_{l=1}^{L}\frac{C^l}{K}\mathbb{E}||\bar{\mathbf{w}}(t) - \bar{\mathbf{w}}^l(t)||^2 \\ + 2\rho^2\frac{1}{T}\sum_{t=0}^{T-1}\frac{1}{K}\sum_{l=1}^{L}\sum_{k\in C^l}\mathbb{E}||\bar{\mathbf{w}}^l(t) - \bar{\mathbf{w}}_k(t)||^2\end{aligned} \tag{25}$$

As emerged from [55], it can also be observed that:

$$\begin{aligned}\frac{1}{T}\sum_{t=0}^{T-1}\sum_{l=1}^{L}\frac{C^l}{K}\mathbb{E}||\bar{\mathbf{w}}(t) - \bar{\mathbf{w}}^l(t)||^2 \leq \frac{4\eta^2k_1k_2\frac{L-1}{K}\sigma^2}{1-12\eta^2\rho^2{k_1k_2}^2} \\ + \frac{6\eta^2k_1^2k_2^2\Delta^2}{1-12\eta^2\rho^2k_1^2k_2^2} \\ + \frac{4\eta^2k_1^2k_2^2\rho^2}{1-12\eta^2\rho^2k_1^2k_2^2}\frac{1}{T}\sum_{t=0}^{T-1}\frac{1}{K}\sum_{l=1}^{L}\sum_{k\in C^l}\mathbb{E}||\mathbf{w}_k(t) - \mathbf{w}^l(t)||^2\end{aligned} \tag{26}$$

and:

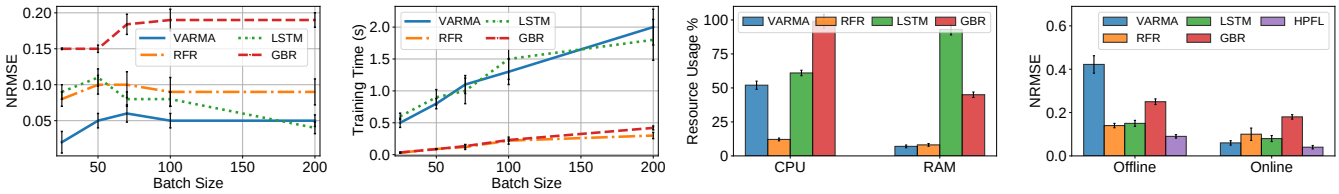
$$\begin{aligned}\frac{1}{T}\sum_{t=0}^{T-1}\frac{1}{K}\sum_{l=1}^{L}\sum_{k\in C^l}\mathbb{E}||\bar{\mathbf{w}}^l(t) - \mathbf{w}_k(t)||^2 \leq \\ \frac{2\eta^2\sigma^2\sum_{l=1}^{L}\frac{C^l-1}{K}k_1}{1-12\eta^2\rho^2k_1^2} + \frac{6\eta^2\sum_{l=1}^{L}\frac{C^l}{K}k_1^2\delta^2}{1-12\eta^2\rho^2k_1^2}\end{aligned} \tag{27}$$

By substituting Eq. 27 into Eq. 26 and the latter in Eq. 25, we can rearrange the formula and obtain Eq. 21. □

VII. Results

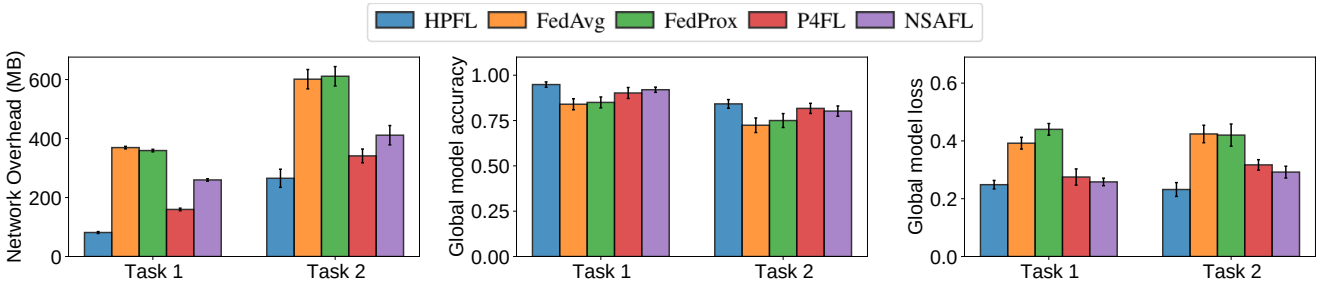
In this section, we conduct several experiments to test the advantages of HPFL. We first test the latency prediction component and demonstrate that it reaches a lower prediction error w.r.t. state-of-the-art solutions. Later on, we test the global framework and measure its accuracy on two well-known datasets, where it reaches the highest accuracy with the lowest network overhead w.r.t. the benchmarks. Finally, we demonstrate the robustness of our solution over non-IID data.

We develop our solution using Python and PyTorch to construct the ML model. We perform simulations using three datasets denoting three tasks: the MNIST (*Task 1*), CIFAR-10 (*Task 2*), Fashion-MNIST (*Task 3*) datasets. The MNIST dataset is a well-known database used for solving the hand-written digit classification problem by adopting a convolutional neural network [56]. It consists of black and white images with dimension 28×28, split into $60,000$ training images and $10,000$ testing images. The model used for this dataset is a CNN with 5 layers with the structure of $784 - [32C3{-}32C3{-}32C5S2]{-}[64C3{-}64C3{-}64C5S2]{-}128{-}10$ and dropout to 40%, as proposed in [57], where $32C3$ means a convolution layer with 32 feature maps using a 3×3 filter and stride 1, and $32C5S2$ means a convolution layer with 32 feature maps using a 5×5 filter and stride 2. The CIFAR-10 dataset consists of color images (32×32 RGB with three channels) classified into 10 classes, e.g., ships, cats, and dogs. It is split into $50,000$ training samples and $10,000$ test samples [58] and organized into five training



(a) NRMSE (b) Training time (c) CPU and RAM consumption (d) Offline vs. online

Fig. 4: Latency prediction



(a) Network traffic overhead (b) Model accuracy (c) Model loss

Fig. 5: Results analysis in terms of (a) communication overhead (b) accuracy (c) model loss. HPFL is the most accurate approach with the lowest network overhead.

batches and one test batch, each containing $1,000$ images, while the test batch includes $1,000$ randomly selected images from each class. In this paper, we employ the ResNet-56 CNN architecture, a deep learning model with 56 layers that has been pre-trained [59]. To prevent overfitting, the identity mapping in ResNet enables the model to skip a CNN weight layer if the current layer is deemed unnecessary. Specifically, the convolutional layers encompass a total of 56 weighted layers, primarily consisting of 3×3 filters, ending with a global average pooling layer followed by a 1000-way fully-connected layer with softmax activation. The Fashion-MNIST dataset was created to overcome some limitations of the MNIST dataset, which is too easy to solve and does not represent modern tasks [60]. This dataset is composed of fashion item images from Zalando, organized into a training set of $60,000$ images and a test set of $10,000$. Each image is grayscale with a 28×28 resolution and is labeled according to one of 10 possible classes. To solve this task, we employ a CNN structured as follows: $1-[16C5P2]-[32C5P2]-128-10$. Here, $16C5P2$ denotes a convolutional layer with 16 feature maps and a 5×5 kernel, followed by max pooling with a 2×2 window.

We compare our HPFL with the following algorithms:

- FedAvg [44]: the conventional FL algorithm and framework that performs the average of the collected neural network weights.
- FedProx [61]: this method addresses the challenge of heterogeneity, introducing a proximal term that constrains local updates to remain closer to the initial global model.
- P4FL [42]: this approach leverages P4 switches as an intermediate aggregation point to mitigate the bottleneck at the central FL server.
- NSAFL [26]: a reference asynchronous FL solution that selects client nodes according to greedy heuristics considering local computation and communication resources.

A. *Latency Prediction Accuracy*

We first evaluate our prediction component against state-of-the-art predictors in terms of training time, prediction error, and required computational resources. The link delay is replicated using traces collected in [62], where the captures were made through WebRTC, under different emulated 4G network conditions, and measures are taken every 1ms. We test VARMA and RFR used as stand-alone predictors but also combined together as in HPFL. We compare these models against LSTM [63], a special Recurrent Neural Network (RNN) equipped with a memory cell to learn long-term dependencies in the data, and Gaussian Process Regression (GBR) [64], a probabilistic kernel-based model. We measure the error via Normalized Root Mean Squared Error (NRMSE), a metric that measures the distance between predictions and real values, which is later normalized to the dataset used.

We first test the predictors' performance (Figure 4) and show the prediction error during testing (a) and the training time (b), and the resource consumption of CPU and RAM during training (c). Later, we compare offline to online learning (d).

VARMA stands out as the most accurate model, even though its training time highly increases with the size of the batch. A similar behavior is shown by LSTM, which, however, is less accurate. On the contrary, GBR and RFR maintain a very low training time, and RFR also shows a constant low error. Additionally, RFR is the less demanding method in terms of computational resources. Therefore, we can exploit the advantages of VARMA and RFR by combining them with the FPL algorithm in HPFL.

Figure 4d shows the NRMSE of the benchmarks in both offline and online learning scenarios. In the offline modality,

the training is performed once over the training set, and the model is later used only in inference mode. In the online modality, the training is performed periodically over a sliding batch of data comprising the data observed during testing. All the predictors show an improvement in the online version, especially VARMA with a mean $NRMSE = 0.06$. However, HPFL outperforms all the benchmarks, as it computes the best model to use at each step among VARMA and RFR, showcasing a mean $NRMSE = 0.04$.

B. Communication Overhead

The next experiment assesses the network overhead introduced by each federated solution. In fact, transmitting model parameters over the network comes with a cost. For this experiment, we consider 10 different sites connected to 3 diverse edges, where every link has 100 Mbps bandwidth, but the link between edges and the central server is of 50 Mbps bandwidth. The mean delay is 20ms. We set the edge aggregation every 3 ($\kappa_1 = 3$) and the global aggregation with a period of 2 ($\kappa_2 = 2$).

We show the network traffic originated by HPFL and benchmarks in Fig. 5a. All the solutions show an increased overhead for Task 2 because the size of the model to exchange at each aggregation round is way bigger than Task 1. In both tasks, HPFL clearly exhibits the lowest burden and, specifically, reduces the network overhead for Task 1 by a factor of 78% compared to FedAvg and 49% to P4FL. Compared to a traditional approach such as FedAvg, the hierarchical approach allows reducing the traffic by a factor of L/K where L is the number of edge devices, and K is the number of IoT nodes of the process. Additionally, another hierarchical approach, P4FL, based on the idea of partially aggregating model gradients at the edge and cloud, results in additional bytes in the network. This result is extremely important in cross-silo scenarios, where the central server may be far from sites, e.g., hospitals, and network latency represents an issue originating synchronization problems [7], [65], [66].

C. FL Model Performance

We now measure the improvements in the learning process by quantifying the accuracy (Fig. 5b) and the loss (Fig. 5c) of the global model trained via the FL process. While traditional approaches are obstructed by the straggler problem, our solution can converge to a high global accuracy and a minimal loss. Moreover, the asynchronous approach of NSAFL can efficiently mitigate the straggler issue, but the older model considered in the aggregation phase leads to lower accuracy. Specifically, NSAFL reaches an average accuracy of 92% for Task 1 and 80% for Task 2, while HPFL reaches an accuracy of 95% in Task 1 and 84% on Task 2 on average, with significantly lower network overhead. This result is also confirmed when evaluating the evolution of the model's accuracy over time for Task 1 (Fig. 6a) and Task 2 (Fig. 6b). A lower network overhead decreases the chances of congestion and, therefore, results in a faster and more stable accuracy trend.

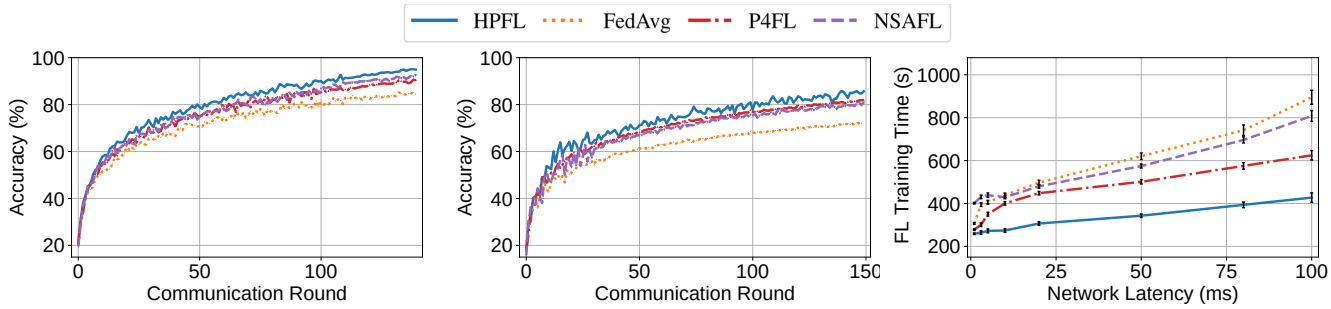
Finally, we evaluate the impact of the latency between the IoT devices and the central server on the training time in Fig. 6c. Our findings reveal that when the server is located near the devices (network latency close to 0) the training processes for the approaches are quite similar. In this case, NSAFL requires the highest training time, as the asynchronous mechanism slows down the aggregation process. However, as latency increases, we can observe significant improvements. Thanks to the edge servers, clients can receive a first response, and the evaluation phase can begin without waiting for final aggregation on the server. Notably, HPFL offers predictive capabilities that reduce training time, even in a more traditional hierarchical environment. This is especially important for cloud-centric solutions where the server is located in a different city or infrastructure, like a hospital. In IoT scenarios, as explained in [25], [67], we expect even greater benefits in the required time, and HPFL can positively tolerate the presence of stragglers, i.e., clients that take much longer to report their output.

D. Non-IID data

We now assess the impact of non-IID data on the performance of our model and state-of-the-art solution NSAFL. For this purpose, we leverage the Fashion-MNIST dataset (Task 3), as in other works [26], [68]. Specifically, we perform multiple experiments varying the partition of data among clients from more balanced partitions to very imbalanced class distributions. To achieve this, we employ a Dirichlet partitioner, as in [69], [70], and adjust the parameter α to change the skewness of the dataset. High values of α, e.g., up to 100, create a balanced distribution close to IID. Low $alpha$ values close to 0, create a heterogeneous distribution of client data. Figure 7 shows the distribution of classes among each client in the balanced case and the most imbalanced case that we test, e.g., a) $alpha = 50$ and b) $alpha = 0.3$. The resulting variation of accuracy in the classification Task 3 is shown in Figure 7c. Clearly, imbalanced partitions reduce the accuracy of the global model. However, it is evident how robust HPFL is, showing a performance decrement of 7% in the worst case and demonstrating an overall higher accuracy w.r.t. NSAFL.

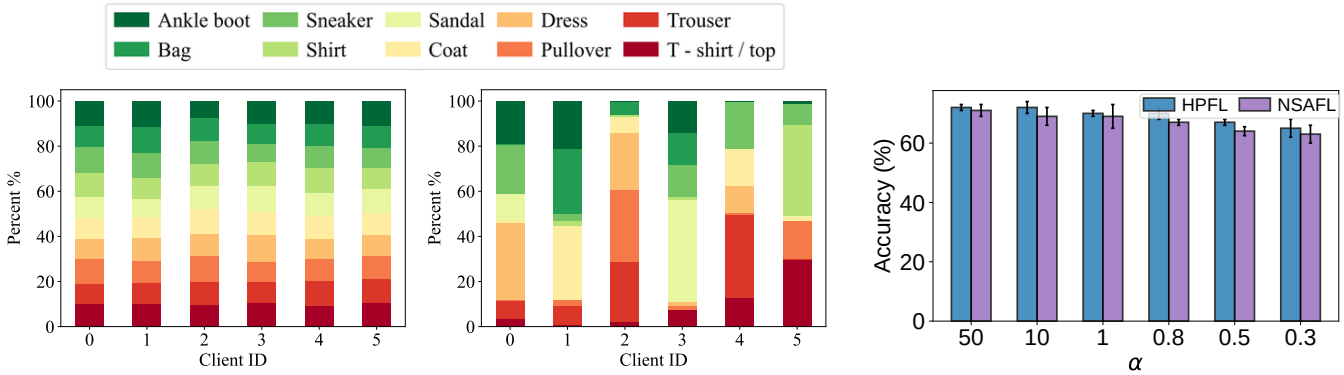
VIII. Conclusion

In this paper, we presented HPFL, a Federated Learning (FL) approach aiming to alleviate synchronization issues in challenged IoT networks. Our proposed dynamic deadline and hierarchical approach offer a promising solution to the synchronization challenges inherent to FL, paving the way for more efficient and effective collaboration among mobile and IoT devices. Results showed that HPFL can reduce the number of bytes required in the communication while improving the model accuracy. While results focused on a simple aggregation strategy, our solution can be extended to even more sophisticated aggregation schemas as the field continues to evolve. In addition, by leveraging normalized features and relative values, HPFL enables the model to focus on behavioral patterns rather than absolute metrics tied to a specific network size. This abstraction allows HPFL to adapt its predictions across deployments of different scales, making it particularly well-suited for applications where device heterogeneity is common



(a) Accuracy Task 1 (b) Accuracy Task 2 (c) Training time

Fig. 6: Accuracy evolution for (a) Task 1 (b) Task 2. (c) Training time when incrementing the average network latency perceived by IoT devices. HPFL converges faster than other solutions to a better policy.



(a) α = 50 (b) α = 0.3 (c) Accuracy Task 3

Fig. 7: Dataset distribution among clients for (a) α = 50 (b) α = 0.3. (c) Accuracy of the model over non-IID data created varying the α parameter of the Dirichlet distribution.

(e.g., smart homes). In healthcare, where privacy and real-time processing are critical, HPFL minimizes delays caused by unreliable devices while preserving data confidentiality by keeping sensitive patient data on-device. Its ability to predict and mitigate stragglers ensures consistent model updates, making it ideal for applications like remote patient monitoring, activity recognition, and personalized health analytics.

References

[1] O. Aouedi, T.-H. Vu, A. Sacco, D. C. Nguyen, K. Piamrat, G. Marchetto, and Q.-V. Pham, "A Survey on Intelligent Internet of Things: Applications, Security, Privacy, and Future Directions," *IEEE Communications Surveys & Tutorials*, vol. 27, no. 2, pp. 1238–1292, 2025.

[2] Q. Guo, F. Tang, and N. Kato, "Federated reinforcement learning-based resource allocation for d2d-aided digital twin edge networks in 6g industrial iot," *IEEE Transactions on Industrial Informatics*, vol. 19, no. 5, pp. 7228–7236, 2022.

[3] G. Gad, E. Gad, Z. M. Fadlullah, M. M. Fouda, and N. Kato, "Communication-Efficient and Privacy-Preserving Federated Learning via Joint Knowledge Distillation and Differential Privacy in Bandwidth-Constrained Networks," *IEEE Transactions on Vehicular Technology*, 2024.

[4] K. Bonawitz, H. Eichner, W. Grieskamp, D. Huba *et al.*, "Towards Federated Learning at Scale: System Design," *Proceedings of Machine Learning and Systems*, vol. 1, pp. 374–388, 2019.

[5] A. Sacco, F. Esposito, and G. Marchetto, "A Federated Learning Approach to Routing in Challenged SDN-Enabled Edge Networks," in *6th IEEE Conference on Network Softwarization (NetSoft '20)*. IEEE, 2020, pp. 150–154.

[6] J. Konečnỳ, H. B. McMahan, D. Ramage, and P. Richtárik, "Federated Optimization: Distributed Machine Learning for On-Device Intelligence," *arXiv preprint arXiv:1610.02527*, 2016.

[7] O. Aouedi, A. Sacco, K. Piamrat, and G. Marchetto, "Handling Privacy-Sensitive Medical Data With Federated Learning: Challenges and Future Directions," *IEEE Journal of Biomedical and Health Informatics*, vol. 27, no. 2, pp. 790–803, 2022.

[8] A. E. Giannopoulos, S. T. Spantideas, M. Zetas, N. Nomikos, and P. Trakadas, "Fedship: Federated over-the-air learning for communication-efficient and privacy-aware smart shipping in 6g communications," *IEEE Transactions on Intelligent Transportation Systems*, 2024.

[9] L. Pappone, A. Sacco, and F. Esposito, "On Traffic Matrix Estimation via Super-Resolution and Federated Learning," *IEEE Transactions on Network and Service Management*, 2024.

[10] W. Wu, L. He, W. Lin, R. Mao, C. Maple, and S. Jarvis, "SAFA: A Semi-Asynchronous Protocol for Fast Federated Learning With Low Overhead," *IEEE Transactions on Computers*, vol. 70, no. 5, pp. 655–668, 2020.

[11] C. Xu, Y. Qu, Y. Xiang, and L. Gao, "Asynchronous Federated Learning on Heterogeneous Devices: A Survey," *Computer Science Review*, vol. 50, p. 100595, 2023.

[12] T. Zhang, L. Gao, C. He, M. Zhang, B. Krishnamachari, and A. S. Avestimehr, "Federated learning for the internet of things: Applications, challenges, and opportunities," *IEEE Internet of Things Magazine*, vol. 5, no. 1, pp. 24–29, 2022.

[13] O. Aouedi, A. Sacco, L. U. Khan, D. C. Nguyen, and M. Guizani, "Federated Learning for Human Activity Recognition: Overview, Advances, and Challenges," *IEEE Open Journal of the Communications Society*, vol. 5, pp. 7341–7367, 2024.

[14] M. M. Fouda, Z. M. Fadlullah, M. I. Ibrahem, and N. Kato, "Privacy-Preserving Data-Driven Learning Models for Emerging Communication Networks: A Comprehensive Survey," *IEEE Communications Surveys & Tutorials*, 2024.

[15] Y. Zhu, B. Mao, and N. Kato, "On a novel high accuracy positioning with intelligent reflecting surface and unscented kalman filter for intelligent transportation systems in b5g," *IEEE Journal on Selected Areas in Communications*, vol. 42, no. 1, pp. 68–77, 2023.

[16] L. You, S. Liu, Y. Chang, and C. Yuen, "A Triple-Step Asynchronous Federated Learning Mechanism for Client Activation, Interaction Optimization, and Aggregation Enhancement," *IEEE Internet of Things Journal*, vol. 9, no. 23, pp. 24 199–24 211, 2022.

[17] H.-S. Lee and J.-W. Lee, "Adaptive Transmission Scheduling in Wireless Networks for Asynchronous Federated Learning," *IEEE Journal on Selected Areas in Communications*, vol. 39, no. 12, pp. 3673–3687, 2021.

[18] Q. Wang, Q. Yang, S. He, Z. Shi, and J. Chen, "Asyncfeded: Asynchronous federated learning with euclidean distance based adaptive weight aggregation," *arXiv preprint arXiv:2205.13797*, 2022.

[19] R. Lu, W. Zhang, Q. Li, H. He, X. Zhong, H. Yang, D. Wang, Z. Xu, and M. Alazab, "Adaptive asynchronous federated learning," *Future Generation Computer Systems*, vol. 152, pp. 193–206, 2024.

[20] R. Schlegel, S. Kumar, E. Rosnes, and A. G. i Amat, "CodedPaddedFL and CodedSecAgg: Straggler Mitigation and Secure Aggregation in Federated Learning," *IEEE Transactions on Communications*, vol. 71, no. 4, pp. 2013–2027, 2023.

[21] C. Li, M. Xiao, and M. Skoglund, "Adaptive coded federated learning: Privacy preservation and straggler mitigation," *arXiv preprint arXiv:2403.14905*, 2024.

[22] J. S. Ng, W. Y. B. Lim, Z. Xiong, X. Cao, D. Niyato, C. Leung, and D. I. Kim, "A Hierarchical Incentive Design Toward Motivating Participation in Coded Federated Learning," *IEEE Journal on Selected Areas in Communications*, vol. 40, no. 1, pp. 359–375, 2021.

[23] Y. Sun, J. Shao, Y. Mao, S. Li, and J. Zhang, "Stochastic coded federated learning: Theoretical analysis and incentive mechanism design," *IEEE Transactions on Wireless Communications*, 2023.

[24] N. Ferdinand, H. Al-Lawati, S. C. Draper, and M. Nokleby, "Anytime Minibatch: Exploiting Stragglers in Online Distributed Optimization," *arXiv preprint arXiv:2006.05752*, 2020.

[25] S. Dutta, G. Joshi, S. Ghosh, P. Dube, and P. Nagpurkar, "Slow and Stale Gradients Can Win the Race: Error-Runtime Trade-Offs in Distributed SGD," in *International Conference on Artificial Intelligence and Statistics*. PMLR, 2018, pp. 803–812.

[26] Z. Chen, W. Liao, K. Hua, C. Lu, and W. Yu, "Towards Asynchronous Federated Learning for Heterogeneous Edge-Powered Internet of Things," *Digital Communications and Networks*, vol. 7, no. 3, pp. 317–326, 2021.

[27] J. Hao, Y. Zhao, and J. Zhang, "Time Efficient Federated Learning with Semi-asynchronous Communication," in *IEEE 26th International Conference on Parallel and Distributed Systems (ICPADS)*. IEEE, 2020, pp. 156–163.

[28] N. Yan, Y. Li, J. Chen, X. Wang, J. Hong, K. He, and W. Wang, "Efficient and straggler-resistant homomorphic encryption for heterogeneous federated learning," in *IEEE INFOCOM 2024-IEEE Conference on Computer Communications*. IEEE, 2024, pp. 791–800.

[29] A. Imteaj and M. H. Amini, "FedAR: Activity and Resource-Aware Federated Learning Model for Distributed Mobile Robots," in *19th IEEE International Conference on Machine Learning and Applications (ICMLA)*. IEEE, 2020, pp. 1153–1160.

[30] C.-H. Hu, Z. Chen, and E. G. Larsson, "Scheduling and Aggregation Design for Asynchronous Federated Learning Over Wireless Networks," *IEEE Journal on Selected Areas in Communications*, vol. 41, no. 4, pp. 874–886, 2023.

[31] Z. Charles and D. Papailiopoulos, "Gradient Coding Using the Stochastic Block Model," in *IEEE International Symposium on Information Theory (ISIT '18')*. IEEE, 2018, pp. 1998–2002.

[32] Z. Charles, D. Papailiopoulos, and J. Ellenberg, "Approximate Gradient Coding via Sparse Random Graphs," *arXiv preprint arXiv:1711.06771*, 2017.

[33] T. Tang, R. E. Ali, H. Hashemi, T. Gangwani, S. Avestimehr, and M. Annavaram, "Adaptive verifiable coded computing: Towards fast, secure and private distributed machine learning," in *2022 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 2022, pp. 628–638.

[34] A. Reisizadeh, S. Prakash, R. Pedarsani, and A. S. Avestimehr, "Coded Computation Over Heterogeneous Clusters," *IEEE Transactions on Information Theory*, vol. 65, no. 7, pp. 4227–4242, 2019.

[35] A. Imteaj and M. H. Amini, "FedPARL: Client Activity and Resource-oriented Lightweight Federated Learning Model for Resource-constrained Heterogeneous IoT Environment," *Frontiers in Communications and Networks*, vol. 2, 2021.

[36] T. Shaik, X. Tao, N. Higgins, R. Gururajan, Y. Li, X. Zhou, and U. R. Acharya, "FedStack: Personalized Activity Monitoring Using Stacked Federated Learning," *Knowledge-Based Systems*, vol. 257, p. 109929, 2022.

[37] X. Gu, K. Huang, J. Zhang, and L. Huang, "Fast Federated Learning in the Presence of Arbitrary Device Unavailability," *Advances in Neural Information Processing Systems (NeurIPS '21)*, vol. 34, pp. 12 052–12 064, 2021.

[38] W. Xia, Y. Li, L. Zhang, Z. Wu, and X. Yuan, "Cascade Vertical Federated Learning Towards Straggler Mitigation and Label Privacy over Distributed Labels," *IEEE Transactions on Big Data*, 2023.

[39] I. Wang, P. Nair, and D. Mahajan, "Fluid: Mitigating stragglers in federated learning using invariant dropout," *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[40] N. Abbas, Y. Zhang, A. Taherkordi, and T. Skeie, "Mobile Edge Computing: A Survey," *IEEE Internet of Things Journal*, vol. 5, no. 1, pp. 450–465, 2017.

[41] Z. Ji, L. Chen, N. Zhao, Y. Chen, G. Wei, and F. R. Yu, "Computation Offloading for Edge-Assisted Federated Learning," *IEEE Transactions on Vehicular Technology*, vol. 70, no. 9, pp. 9330–9344, 2021.

[42] A. Sacco, A. Angi, G. Marchetto, and F. Esposito, "P4FL: An Architecture for Federating Learning with In-Network Processing," *IEEE Access*, pp. 103 650–103 658, 2023.

[43] Y.-J. Liu, G. Feng, H. Du, Z. Qin, Y. Sun, J. Kang, X. Li, and D. Niyato, "Adaptive clustering based straggler-aware federated learning in wireless edge networks," *IEEE Transactions on Communications*, 2024.

[44] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-Efficient Learning of Deep Networks From Decentralized Data," in *Artificial intelligence and statistics*. PMLR, 2017, pp. 1273–1282.

[45] E. Hannan, "The Identification of Vector Mixed Autoregressive-Moving Average System," *Biometrika*, vol. 56, no. 1, pp. 223–225, 1969.

[46] N. Littlestone and M. K. Warmuth, "The Weighted Majority Algorithm," *Information and computation*, vol. 108, no. 2, pp. 212–261, 1994.

[47] M. Zinkevich, "Online Convex Programming and Generalized Infinitesimal Gradient Ascent," in *Proceedings of the 20th international conference on machine learning (ICML '03)*, 2003, pp. 928–936.

[48] J. Hannan, "Approximation to Bayes Risk in Repeated Play," *Contributions to the Theory of Games*, vol. 3, pp. 97–139, 1957.

[49] A. Kalai and S. Vempala, "Efficient Algorithms for Online Decision Problems," *Journal of Computer and System Sciences*, vol. 71, no. 3, pp. 291–307, 2005.

[50] N. Cesa-Bianchi and G. Lugosi, *Prediction, Learning, and Games*. Cambridge university press, 2006.

[51] S. Wang, T. Tuor, T. Salonidis, K. K. Leung, C. Makaya, T. He, and K. Chan, "Adaptive federated learning in resource constrained edge computing systems," *IEEE journal on selected areas in communications*, vol. 37, no. 6, pp. 1205–1221, 2019.

[52] L. Liu, J. Zhang, S. Song, and K. B. Letaief, "Client-edge-cloud hierarchical federated learning," in *ICC 2020-2020 IEEE international conference on communications (ICC)*. IEEE, 2020, pp. 1–6.

[53] ——, "Hierarchical federated learning with quantization: Convergence analysis and system design," *IEEE Transactions on Wireless Communications*, vol. 22, no. 1, pp. 2–18, 2022.

[54] L. Bottou, F. E. Curtis, and J. Nocedal, "Optimization methods for large-scale machine learning," *SIAM review*, vol. 60, no. 2, pp. 223–311, 2018.

[55] J. Wang, S. Wang, R.-R. Chen, and M. Ji, "Demystifying why local aggregation helps: Convergence analysis of hierarchical sgd," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 36, no. 8, 2022, pp. 8548–8556.

[56] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-Based Learning Applied to Document Recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[57] "What is the best cnn architecture for mnist?" https://www.kaggle.com/code/cdeotte/how-to-choose-cnn-architecture-mnist/notebook. Accessed: 2024-1-10.

[58] "The cifar-10 dataset," https://www.cs.toronto.edu/ kriz/cifar.html. Accessed: 2024-1-10.

[59] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR '16)*. IEEE, 2016, pp. 770–778.

[60] H. Xiao, K. Rasul, and R. Vollgraf. (2017) Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms.

[61] T. Li, A. K. Sahu, M. Zaheer, M. Sanjabi, A. Talwalkar, and V. Smith, "Federated Optimization in Heterogeneous Networks," in *Proceedings of Machine Learning and Systems 2 (MLSys 2020)*, vol. 2, 2020, pp. 429–450.

[62] P. Graff, X. Marchal, T. Cholez, S. Tuffin, B. Mathieu, and O. Festor, "An Analysis of Cloud Gaming Platforms Behavior Under Different

Network Constraints," in *17th International Conference on Network and Service Management (CNSM)*. IEEE, 2021, pp. 551–557.
[63] S. Hochreiter, "Long short-term memory," *Neural Computation MIT-Press*, 1997.
[64] A. G. Wilson, D. A. Knowles, and Z. Ghahramani, "Gaussian process regression networks," *arXiv preprint arXiv:1110.4411*, 2011.
[65] P. Kairouz, H. B. McMahan, B. Avent *et al.*, "Advances and Open Problems in Federated Learning," *Foundations and Trends in Machine Learning*, vol. 14, no. 1–2, pp. 1–210, 2021.
[66] S. Silva, B. A. Gutman *et al.*, "Federated Learning in Distributed Medical Databases: Meta-Analysis of Large-Scale Subcortical Brain Data," in *16th international symposium on biomedical imaging (ISBI 2019)*. IEEE, 2019, pp. 270–274.
[67] T. Li, A. K. Sahu, M. Zaheer, M. Sanjabi, A. Talwalkar, and V. Smith, "Federated Optimization in Heterogeneous Networks," *Proceedings of Machine Learning and Systems*, vol. 2, pp. 429–450, 2020.
[68] H. Wen, Y. Wu, J. Hu, Z. Wang, H. Duan, and G. Min, "Communication-efficient federated learning on non-iid data using two-step knowledge distillation," *IEEE Internet of Things Journal*, vol. 10, no. 19, pp. 17 307–17 322, 2023.
[69] T. Lin, L. Kong, S. U. Stich, and M. Jaggi, "Ensemble distillation for robust model fusion in federated learning," *Advances in neural information processing systems*, vol. 33, pp. 2351–2363, 2020.
[70] T.-M. H. Hsu, H. Qi, and M. Brown, "Measuring the effects of non-identical data distribution for federated visual classification," *arXiv preprint arXiv:1909.06335*, 2019.

Alessio Sacco is an Assistant Professor at Politecnico di Torino, Italy. He received the Ph.D. degree in computer engineering from the same university in 2022. His research interests include architecture and protocols for network management; implementation and design of cloud computing applications; algorithms and protocols for service-based architecture, such as Software Defined Networks (SDN), used in conjunction with Machine Learning algorithms.

Doriana Monaco received the M.Sc degree in Computer Engineering from Politecnico di Torino, in 2022, where she is currently pursuing a Ph.D degree. Her research interests cover computer networks monitoring and management; distributed learning applications for Software-Defined Networks (SDN); cloud-computing solutions.

Guido Marchetto received a Ph.D. degree in computer engineering from the Politecnico di Torino, in 2008, where he is currently a Full Professor with the Department of Control and Computer Engineering. His research topics cover distributed systems and formal verification of systems and protocols. His interests also include network protocols and network architectures.

Paolo Montuschi (M'90-SM'07-F'14) is a full professor with the Department of Control and Computer Engineering, Rector's Delegate for Information Systems, and a past member of the Board of Governors at Politecnico di Torino, Italy. His research interests include computer arithmetic, computer graphics, and intelligent systems. He is an IEEE Fellow, a life member of the International Academy of Sciences in Turin, and of HKN, the Honor Society of IEEE. He serves as the Editor-in-Chief of the IEEE Transactions on Emerging Topics in Computing, the 2020-21 Chair of the IEEE TAB/ARC and the co-Chair of the 2021 TAB/PSPB Ad Hoc Committee on Publications Strategy. Previously, he served in a number of positions, including the Editor-in-Chief of the IEEE Transactions on Computers (2015-18), the IEEE Computer Society Awards Committee Chair (2017-20), a Member-at-Large of IEEE PSPB (2018-20), and as the Chair of its Strategic Planning Committee (2019-20). More information at http://staff.polito.it/paolo.montuschi.