Automatic Layer Freezing for Communication Efficiency in Cross-Device Federated Learning

Automatic Layer Freezing for Communication Efficiency in Cross-Device Federated Learning

Erich Malan *Graduate Student Member, IEEE*, Valentino Peluso *Member, IEEE*, Andrea Calimera *Member, IEEE*, Enrico Macii *Fellow, IEEE*, Paolo Montuschi *Fellow, IEEE*

***Abstract*—Federated Learning (FL) is a collaborative machine learning paradigm where network-edge clients train a global model under the orchestration of a central server. Unlike traditional distributed learning, each participating client keeps its data locally, ensuring privacy protection by default. However, state-of-the-art FL implementations suffer from massive information exchange between clients and the server. This issue prevents the adoption in constrained environments, typical of the Internet-of-Things domain, where the communication bandwidth and the energy budget are severely limited. To achieve higher efficiency at scale, the future of FL calls for additional optimizations to reach high-quality learning capability with lower communication pressure. To address this challenge, we propose *Automatic Layer Freezing* (ALF), an embedded mechanism that gradually drops a growing portion of the model out of the training and synchronization phases of the learning loop, reducing the volume of exchanged data with the central server. ALF monitors the evolution of model updates and identifies layers that have reached a stable representation, where further weight updates would have minimal impact on accuracy. By freezing these layers, ALF achieves substantial savings in communication bandwidth and energy consumption. The proposed implementation of the ALF mechanism is compatible with any FL strategy, requiring minimal effort and without interfering with existing optimizations. The extensive experiments conducted using a representative set of FL strategies applied to two image classification tasks show that ALF improves the communication efficiency of the baseline FL implementations, ensuring up to 83.91% of data volume savings with no or marginal losses of accuracy.**

***Index Terms*—Federated Learning, Internet of Things, Convolutional Neural Networks, Communication, Optimization.**

I. Introduction & Motivation

With the increasing computing power of mobile devices and sensor nodes [1], and the recent advances in training and compression of deep learning pipelines [2]–[4], complex Convolutional Neural Networks (ConvNets hereafter) are now being deployed at the edge, near the data source, enabling distributed intelligent services with a higher privacy standard, better performance, and improved scalability. Unfortunately, pushing the inference stage out of the cloud is not enough to ensure complete data protection, as third parties still need data to handle the model training. At the core of the problem is the enforcement of regulations that prevent aggregating raw data into centralized databases stored in the cloud, eventually overseas, limiting the operationalization of machine learning. This aspect slows the adoption of machine learning at scale in many application domains. A valuable solution is Federated Learning (FL) [5], a distributed learning paradigm where independent clients cooperate in training a global model without sharing their raw data, nor among themselves or with the central service's provider.

Among the many embodiments of the FL paradigm, a key difference lies in the possible relationships between the clients participating in the training. In *cross-silo* FL, the participating clients are geo-distributed data centers belonging to multiple institutions or organizations (e.g., hospitals of the national health systems [6]). In *cross-device* FL, the target of this work, the participating clients are edge devices, often tiny devices like low-power sensor nodes and mobile devices, belonging to different owners. Cross-device FL implementations can be found in mobile keyword prediction [7], emoji prediction [8], malware detection [9], and network traffic optimization [10]. Although the working principle behind cross-device FL was proven feasible, several limitations still prevent its widespread diffusion to more general and complex tasks. The first limitation concerns the functional metric, namely, the quality of training. Cross-device FL involves a large number of clients, each of them storing non-independent, identically distributed (non-IID) and imbalanced data. Moreover, some clients can be unavailable due to a limited energy budget or unreliable connection; for example, smartphones may participate in the training only if connected to a WiFi network. Those two issues affect the prediction quality achievable by FL, resulting in substantial accuracy losses compared to results attainable with conventional centralized training. The recent literature introduced many FL variants [11]–[16] that help mitigate the accuracy gap. The second limitation concerns the extra-functional metrics instead. The learning process involves frequent synchronizations between the participating clients and the central server, generating massive data volumes running above capacity in many contexts. For instance, in a typical IoT setting, the download and upload bandwidths are limited to $0.75\,\mathrm{MB/s}$ and $0.25\,\mathrm{MB/s}$, respectively [17]. Moreover, since the communication energy grows proportionally with the volume of exchanged data [18], the energy budget becomes a key aspect to consider on the client side. Finding new FL implementations that care about data volumes and not accuracy only is therefore paramount.

Erich Malan, Valentino Peluso, Andrea Calimera, and Paolo Montuschi are with the Department of Control and Computer Engineering, Politecnico di Torino, 10129 Turin, Italy (e-mail: erich.malan@polito.it; valentino.peluso@polito.it; andrea.calimera@polito.it; paolo.montuschi@polito.it).

Enrico Macii is with the Interuniversity Department of Regional and Urban Studies and Planning, Politecnico di Torino, 10129 Turin, Italy (e-mail: enrico.macii@polito.it).

With this aim, we introduce *Automatic Layer Freezing* (ALF), an optimization mechanism to alleviate the communication burden in cross-device FL for ConvNets. The working principle of ALF consists of the progressive removal of subportions of the model from the training loop, which helps reduce the parameters exchanged with the central server as the learning moves on. ALF operates on a per-layer basis, leveraging the property that layers in a ConvNet require different numbers of training iterations to reach their final representation.

Even though the layer freezing mechanism had already been explored, especially in centralized training settings, e.g., in [19], a cross-device FL scenario introduces additional challenges that are not yet fully explored. Indeed, the contribution of our work is twofold. First, to investigate the feasibility of automatic layer freezing in cross-device FL implementations. Second, to introduce a feasible, low-overhead embodiment of the freezing mechanism compatible with existing FL strategies.

The proposed ALF implementation consists of a server-side plug-in component that supervises the evolution of the models' updates gathered from the participating clients over successive learning iterations. The outcome of the monitoring procedure is a layer-wise stability index used as a proxy to understand whether a layer can be frozen permanently till the end of the learning process. The central server updates the participating clients regarding the frozen layers, enabling the elimination of these layers from the backpropagation process to save computational resources.

We conducted an extensive experimental validation using different FL strategies and a representative set of ConvNets for image classification over two datasets. The collected results yield the following findings and achievements:

- ALF offers a valuable option to reduce the communication cost of FL with no or negligible impact on the model accuracy. Depending on the benchmark, ALF enables data volume reduction up to 83.91%.
- ALF can be integrated into conventional FL strategies with minimal effort. As test cases, we considered three state-of-the-art strategies, namely, FedAvg [20], FedProx [11], and FedOpt [15]. ALF improves the quality of results in all cases and maximizes the benefits of more efficient learning strategies.
- The automatic procedure embedded in ALF can detect when and which layers can be frozen and adapt the optimization process depending on the strategy, the model, and the dataset under analysis. Moreover, ALF outperforms freezing strategies like [21] based on the manual selection of layers, showing higher flexibility in different experimental settings.

The rest of the paper is organized as follows. Section II summarizes the main components and the working flow of a generic FL strategy. Section III reviews previous works on communication-efficient FL and layer freezing in neural network training. Section IV describes the implementation details of the proposed ALF mechanism. Section V reports the experimental setup and shows the collected results. Finally, Section VI concludes the paper by discussing the main achievements of our study.

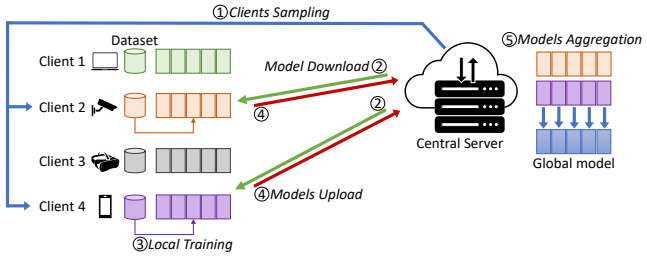


Fig. 1. Schematic view of a Federated Learning framework.

II. Federated Learning

The schematic view in Fig. 1 depicts the essential components of an FL framework. The main actors are the central server and a fleet of N clients. In a cross-device setting, which is the target of this work, the clients belong to a large pool of heterogeneous devices like smartphones, tablets, smart cameras, and, more generally, edge nodes with limited computational resources, small energy budgets, and unstable connectivity. The central server coordinates the learning process and hosts the global version of the model; the edge devices collect and hold a local dataset used for the in-situ training of the model. Each local dataset differs in size and number of classes, i.e., the training samples are *non-independently and identically distributed* (non-IID) across different devices. Moreover, for each local dataset, the available classes follow uneven distributions, i.e., the local dataset is imbalanced.

The global learning runs for a predefined number of iterations, called communication rounds. As shown in Fig. 1, each round consists of five stages summarized as follows:

1) *Clients Sampling*. The central server checks the status of all the participating clients, and a subset of online devices is selected to contribute to the training. Specifically, the server randomly picks a fraction of online devices according to a predefined hyper-parameter called the *participation ratio* (e.g., 10% of the total devices), which is a knob for resource balancing.
2) *Model Download*. The selected clients download the latest version of the global model from the central server.
3) *Local Training*. Each client trains the model with its local dataset. The training iterates over multiple epochs, whose number E is another hyper-parameter that defines the synchronization frequency with the central server. E is commonly set to low values (e.g., 5 or 10) to balance the computing energy (needed for training) and the communication cost (consumed for uploading the model). Also, smaller values mitigate the divergences from the global model.
4) *Models Upload*. The selected clients upload the updated version of their local models to the central server.
5) *Models Aggregation*. The central server updates the global model, aggregating the local models gathered from the clients. As detailed in Section III-A3, different aggregation methods exist.

III. Related Works

In this section, we first summarize existing approaches to improve communication efficiency in cross-device FL (Section III-A). We then focus on previous related works that explored the freezing mechanism during training, both for centralized and federated setups (Section III-B).

A. Communication-efficient Federated Learning

Improving communication efficiency in cross-device FL settings is an optimization problem that can be approached at different stages of the FL workflow and leveraging different knobs. We grouped existing principal techniques using a taxonomy that reflects the four main options available. Interested readers may refer to [22] for a comprehensive review of other solutions.

1) Synchronization: The primary source of the communication overhead originates from the frequent synchronization between the central server and the clients during the download/upload stages (stages *2* and *4* described in Section II). The pioneering work introduced in [20], referred to as *FedAvg*, follows a simple yet efficient strategy that plays with the number of local training epochs E as an optimization knob. Specifically, all the clients upload their model versions after E epochs of local training, with E as a pre-defined constant. With lower E, the achievable accuracy increases as information is shared more frequently with the central server and all the participating clients. However, more frequent synchronization results in higher communication costs. The accuracy vs. communication trade-off analysis reported in [20] suggests that slightly increasing E (e.g., $E = 5$ as a rule of thumb), the communication cost gets sensibly reduced without (or with marginal) loss of accuracy. In the experimental section, we further explore the impact of E, making a parametric comparison with our ALF strategy.

More complex schemes proposed a finer control of the synchronization rate, for example, with an adaptive tuning of E at different levels of (i) spatial granularity, i.e., per-layer [23]–[25]) or per-weight [26], (ii) temporal granularity, i.e., varying E across different rounds [27], (iii) device granularity [28], i.e., adjusting E according to the clients progresses in training. ALF differs from those adaptive strategies as once a layer is frozen, it stays untouched until the end of the learning process. Although this choice might seem too coarse, our experiments validate its efficiency (see Section V-B1).

2) Training Optimization: The most common procedure to train a ConvNet relies on the cross-entropy loss, which is also the most natural choice for local training in a cross-device FL setting. However, the cross-entropy loss shows an inferior performance when applied to imbalanced datasets. This problem is well-known in centralized training and exacerbated in FL, where the label distribution is highly skewed by definition. Motivated by this observation, many researchers focused on developing training loss functions that try to reduce the negative impact of imbalanced and non-IID data on model convergence. The most representative example is *FedProx* [11], which introduces a regularization term to the loss function to penalize local updates diverging from the global model. Further studies [12], [13], [16] reported that this strategy brings benefits only for specific settings, motivating the introduction of more sophisticated loss functions and training procedures. However, there is no consensus on the best option, which may change depending on the task or context. Moreover, complex loss functions require additional computing resources, a potential issue for edge devices with limited resources.

Although conceived to improve accuracy rather than communication efficiency, the training optimization methods improve the convergence time of the learning process, indirectly reducing the number of rounds and hence the overall transmission costs required for a pre-defined accuracy level. For completeness, we investigated the interplay of ALF with *FedProx* and different training variants.

3) Aggregation Optimization: FedAvg implements the aggregation stage averaging the weights gathered from the participating clients. Despite its extensive adoption, the averaging scheme suffers from convergence issues due to (i) data heterogeneity across the clients and (ii) partial clients' participation. Several optimizations of the aggregation procedure can speed up the training convergence and, as a by-product, benefit the communication costs, as the model may reach the same accuracy in fewer rounds. This is the same principle of training optimizations.

FedNOVA [14], for instance, proposed a pre-aggregation stage where the model updates get normalized based on the client's status. *FedOpt* [15] introduced a more general formulation of the aggregation problem. Specifically, it defines three main steps in the aggregation stage, summarized as follows. First, the central server computes the differences between the current global weights and the local weights update received by the clients. Then, it aggregates these differences through averaging. The outcome, referred to as *pseudo-gradient*, is finally fed to a standard solver for training neural networks like Adam [29], Adagrad [30], or YOGI [31], which is in charge of updating the global model weights. In practice, the intuition behind *FedOpt* is to compute the gradients of the global model artificially, avoiding the back-propagation (the pseudo-gradients), thus enabling the same optimization methods deployed in classical centralized training for calculating the weight updates. The authors of [15] observed that *FedAvg* is a specific case of the *FedOpt* formulation, where the solver is SGD with a learning rate equal to 1. Experimental results show *FedOpt* (with Adam as optimizer) outperforms *FedAvg*, both in terms of accuracy and communication efficiency, demonstrating that the aggregation method plays a central role in the performance of FL. We included *FedOpt* in our assessment to study the interaction of ALF with a more efficient aggregation method.

4) Model Compression: Alternative optimization strategies play with the compression of the model to reduce the size of data exchanged. The most common examples rely on pruning [32], quantization [33], or a combination of the two [34]. The model compression can be applied on the client side to reduce the transmission cost during the uploads or on the server side [35] for reducing the cost in download. In both cases, the resource usage increases dramatically due

to the extra workload for model compression/decompression stages, both on the client and server sides. On the contrary, ALF concurrently alleviates the arithmetic workload and the communication burden (in both directions, client-to-server and server-to-client), as the number of arithmetic operations for back-propagation and the exchanged data volumes are inversely proportional to the number of frozen layers.

B. Training optimization with Parameters Freezing

This section reviews previous works that proposed the freezing principle to optimize the training of ConvNets. We first focus on freezing strategies for centralized settings, where the parameters' freezing mechanism aims to accelerate the training stage. We then focus on techniques dedicated to FL, where the parameters' freezing mechanism was applied explicitly to improve communication efficiency.

1) Parameters Freezing in Centralized Training: Most of the freezing techniques applied in centralized training settings work on a per-layer basis. Keeping a layer constant enables the training loop to skip the back-propagation for the parameters belonging to that layer, resulting in faster execution. We can classify layer freezing methods into two main categories, *adaptive* and *permanent* freezing. In the former category, layers can be frozen and unfrozen across different training iterations. The selection of the layers to train can be random [36] or guided by specific proxies, like the magnitude of the weight updates across consecutive epochs [37] or the classification accuracy measured on a calibration set [38]. In the latter category, once a layer is frozen, its parameters are kept constant until the end of the training, i.e., unfreezing is not allowed.

The most representative implementation of *permanent freezing* in centralized training was reported in [19]. The procedure monitors the similarity of the intermediate activations during training to decide whether a layer reached convergence; if so, it can be frozen for the remaining epochs. It is worth emphasizing that this method cannot be ported in a federated context because monitoring the intermediate activations requires the availability of data on the server side.

2) Parameters Freezing in Federated Learning: Only a few works investigated the potential of parameter freezing in federated contexts and for different purposes. Specifically, a series of works [39], [40] applied layers freezing to reduce the computational effort during the local training and to incentivize the clients' participation at each round. In such case, the freezing is *adaptive* because the subset of frozen layers does change along different training rounds and across the clients. Concerning permanent parameters' freezing, *FedPT* [21] proposed a method to reduce communication costs. The idea is to statically select a subset of layers to train from the beginning to the end of the learning process, leaving the remaining layers set to random values. Besides the substantial accuracy losses one may incur, selecting the trainable layers is a manual process that cannot ensure optimality. Our preliminary analysis conducted in [41] further investigated the potential of permanent freezing, proposing an incremental freezing scheme that follows the topological order of the model's layers according to a schedule manually defined at the beginning of the learning process.

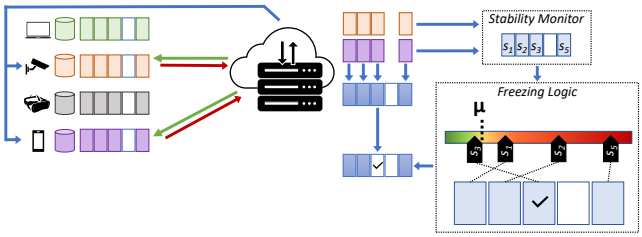


Fig. 2. Schematic view of a Federated Learning framework with ALF.

ALF addresses the layers' selection problem by relying on a fully automatic procedure that probes the evolution of the client's model updates. We adopted *FedPT* as a baseline for comparison to quantify the benefits of our automatic mechanism compared to a manual approach.

IV. FL with Automatic Layer Freezing

A. Overview

We designed the ALF mechanism as a pluggable component that can be embedded in any FL strategy with minimal effort. The schematic diagram of Fig. 2 shows the integration of ALF in a generic FL flow, illustrating a snapshot of the federated training of a ConvNet made of a set of layers $\mathcal{L}$ ($|\mathcal{L}| = 5$ in the figure). The ALF mechanism operates on the central server, relieving the edge clients of additional computations.

ALF comprises two blocks, the *stability monitor* and the *freezing logic*. The stability monitor takes the model updates from the clients as input and returns a layer-wise stability index s_l, with $l \in \mathcal{L}$. The stability index is an aggregated measure used as a proxy to assess layer convergence (more details in the following subsection); it ranges from 0 to 1, with values closer to 0 indicating minor variations of the weights across successive rounds. The freezing logic compares s_l with a user-defined stability threshold μ to check whether a layer can be frozen: if $s_l < \mu$, the layer l is deemed *stable* and can be frozen till the end of the learning process. The stability threshold μ is a hyper-parameter that serves as a control knob to trade off accuracy vs. communication cost (more details in Section V-B3).

In the example of Fig. 2, the fourth layer of the ConvNet has been frozen in a previous round, as indicated by the white box. The layer freezing reduces the data volume transmitted during the download and upload phases, as the frozen layer does not participate in the local training and synchronization phases. The other layers (represented by colored boxes) still participate in the learning process. After the model uploads, the stability monitor calculates s_l for the updated layers (1, 2, 3, and 5). In the figure, $s_3 < \mu$, therefore, the third layer of the global model gets frozen. The server broadcasts this information during the following rounds, allowing the clients to skip the local training of the newly frozen layer. Note that the model aggregation and the computation of the stability index are independent processes. This makes ALF compatible with any aggregation method.

TABLE I
NOTATIONS.

Notation	Description
R	Total number of communication rounds
r	Current communication round, $r \in [1, R]$
$\mathcal{K}$	Set of participating clients
$\mathcal{K}^r$	Subset of clients selected for training at round r, $K^r \subset \mathcal{K}$
Θ^r	Global model at round r
Θ_k^r	Local model of client k at round r, $k \in \mathcal{K}^r$
$\mathcal{D}_k$	Local dataset of the client k
E	Number of local training epochs
$\mathcal{L}$	Set of layers with learnable parameters
$\mathcal{T}^r$	Set of trainable layers at round r
$\boldsymbol{w}_{l,k}^r$	Weights of the layer l ($l \in \mathcal{L}$) belonging to the local model generated by the client k.
s_l	Stability index of layer l
μ	Stability threshold

B. Implementation Details

The pseudo-code in Algorithm 1 describes the workflow of an FL strategy integrating ALF. Table I summarizes the notations in use. We considered a distributed architecture with a central server that coordinates a set of edge devices acting as clients. Each client k holds its local dataset $\mathcal{D}_k$ with private access. The server initializes the global model Θ^1 with random weights (line 1) and defines the set of trainable layers $\mathcal{T}^1$ (line 2). Initially, $\mathcal{T}^1$ comprises all the layers with learnable parameters, i.e., the set $\mathcal{L}$.

Following a synchronous FL scheme, the learning flow iterates for R communication rounds (lines 3–23). Within each round r, the server samples a subset of clients $\mathcal{K}^r$ (line 4). Each selected client k downloads the timestamps (*globalTimestep*$_l$) of the latest update for each layer l of the global model Θ^r (line 6). If the local version of the layer (*localTimestep*$_l$) is older than that of the global model (line 7), the client downloads from the server the updated weights of the layer l (line 8). Even though the timestamp is a source of additional information not needed in conventional FL schemes, it counts 8 bytes per layer, a marginal overhead if compared to the model size (from kBs to MBs). Then, the client trains the model on its local dataset for a predefined number of epochs E, producing an updated local version of the model Θ_k^r (line 9). The training involves only the layers in $\mathcal{T}^r$, which are those uploaded to the server (lines 10–11). After receiving the local updates, the server runs the aggregation, yielding a new model Θ^{r+1} for the next training round (line 12).

The server is in charge of the ALF mechanism (lines 13–23), consisting of the stability monitor (lines 15–21) and the freezing logic (lines 22–23). For each trainable layer (line 14), the stability monitor calculates an aggregated form of the layer's weights $\boldsymbol{w}_l^r$ via a weighted average of the layer updates $\boldsymbol{w}_{l,k}^r$ received from the clients (line 15). Then, it computes the difference Δ_l between the current and previous round's aggregated weights (line 16). The exponential moving average enables monitoring the updates' trend over longer observation windows. Specifically, $\boldsymbol{m}_l^r$ is the exponential moving average of the aggregated weights difference (line 18) and $\boldsymbol{p}_l^r$ is the exponential moving average of its magnitude (line 19); we set $\alpha = 0.95$. The ratio between the element-wise absolute values of the resulting quantities produces $\boldsymbol{n}_l^r$, whose values range from 0 to 1. Finally, the stability monitor calculates the stability index s_l as the element-wise average of $\boldsymbol{n}_l$, returning the layer stability index $\boldsymbol{s}_l$ (line 21). As a last step, the freezing logic applies the thresholding policy (lines 22–23): if the stability index $\boldsymbol{s}_l$ is lower than the stability threshold μ (line 22), store l in the set of frozen layers Θ^{r+1} (line 23).

Calculating the stability index $\boldsymbol{s}_l$ involves element-wise operations only, with a resulting complexity that grows linearly with the size of the involved tensors. Specifically, the most complex operation is the computation of $\boldsymbol{w}_l^r$ (line 15), where the number of elements equals the number of weights in $\boldsymbol{w}_{l,k}^r$

Algorithm 1: Workflow of an FL strategy with ALF.

1 $\Theta^1 \leftarrow$ Random model initialization
2 $\mathcal{T}^1 \leftarrow \mathcal{L}$
3 **for** *r = 1,...,R* **do**
4 Sample a subset $\mathcal{K}^r$ of clients
  /* Clients Optimization */
5 **for** $k \in \mathcal{K}^r$ **do in parallel**
6 **for** $l \in \mathcal{L}$ **do**
7 **if** *globalTimestep*$_l$ $<$ *localTimestep*$_l$ **then**
8 Download layer l of Θ^r
9 $\Theta_k^r \leftarrow$ TRAIN(Θ^r, $\mathcal{D}_k$, E, $\mathcal{T}^r$)
10 **for** $l \in \mathcal{T}^r$ **do**
11 Upload layer l of Θ_k^r
  /* Server Optimization */
12 $\Theta^{r+1} \leftarrow$ AGGREGATE(Θ_k^r, $k \in \mathcal{K}^r$)
  /* Automatic Layer Freezing */
13 $\mathcal{T}^{r+1} \leftarrow \mathcal{T}^r$
14 **for** $l \in \mathcal{T}^r$ **do**
  /* Stability Monitor */
15 $\boldsymbol{w}_l^r \leftarrow \frac{1}{\sum_{k \in \mathcal{K}^r} |\mathcal{D}_k|} \cdot \sum_{k \in \mathcal{K}^r} |\mathcal{D}_k| \cdot \boldsymbol{w}_{l,k}^r$
16 $\Delta_l \leftarrow \boldsymbol{w}_l^r - \boldsymbol{w}_l^{r-1}$
17 $\Delta_l^{\text{abs}} \leftarrow$ Element-wise absolute values of Δ_l
18 $\boldsymbol{m}_l^r \leftarrow \alpha \cdot \boldsymbol{m}_l^{r-1} + (1-\alpha) \cdot \Delta_l$
19 $\boldsymbol{p}_l^r \leftarrow \alpha \cdot \boldsymbol{p}_l^{r-1} + (1-\alpha) \cdot \Delta_l^{\text{abs}}$
20 $\boldsymbol{n}_l^r \leftarrow$ Element-wise absolute values of $\frac{\boldsymbol{m}_l^r}{\boldsymbol{p}_l^r}$
21 $s_l \leftarrow$ Average all the elements of $\boldsymbol{n}_l^r$
  /* Freezing Logic */
22 **if** $s_l < \mu$ **then**
23 $\mathcal{T}^{r+1} \leftarrow \mathcal{T}^{r+1} \setminus \{l\}$

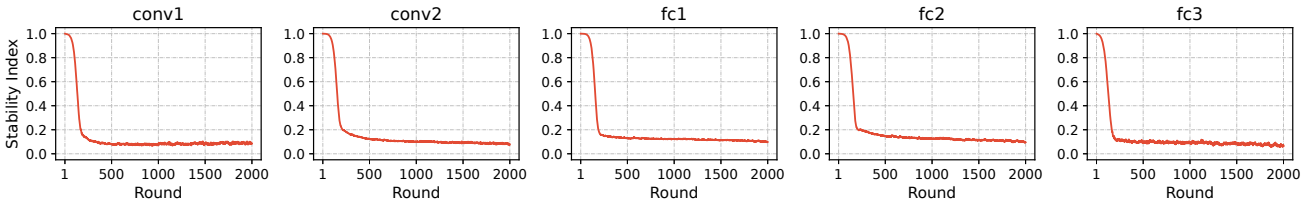


Fig. 3. Layer-wise stability index for a 5-layer ConvNet (CNN-5) trained on CIFAR-10 with FedAvg (2000 rounds).

TABLE II
LAYER-WISE DESCRIPTION OF THE CNN-5 MODEL AND ITS MEMORY FOOTPRINT. THE SIZE OF THE LAST LAYER VARIES WITH THE NUMBER OF CLASSES – 10 FOR CIFAR-10 (100 FOR CIFAR-100).

Name	**Layer Type**	**Memory** (MB)
conv1	2D Convolution	0.019
conv2	2D Convolution	0.391
fc1	Fully-connected	2.406
fc2	Fully-connected	0.289
fc3	Fully-connected	0.007 (0.074)
Total Bias		**0.003 (0.003)**
Total Weights		**3.112 (3.179)**

TABLE III
LAYER-WISE DESCRIPTION OF THE VGG-9 MODEL AND ITS MEMORY FOOTPRINT. THE SIZE OF THE LAST LAYER VARIES WITH THE NUMBER OF CLASSES – 10 FOR CIFAR-10 (100 FOR CIFAR-100).

Name	**Layer Type**	**Memory** (MB)
conv1	2D Convolution	0.003
conv2	2D Convolution	0.071
conv3	2D Convolution	0.282
conv4	2D Convolution	0.563
conv5	2D Convolution	1.126
conv6	2D Convolution	2.251
fc1	Fully-connected	8.002
fc2	Fully-connected	1.002
fc3	Fully-connected	0.020 (0.196)
Total Bias		**0.007 (0.008)**
Total Weights		**13.319 (13.495)**

times the number of selected clients in $\mathcal{K}_r$ (which is in general $\ll$ the total number of clients). In the following operations (lines 16-21), the number of elements equals the number of weights in the layer. Since element-wise operations can be processed in parallel by server-side CPUs/GPUs used for model aggregation, the computational overhead of the stability monitor is negligible.

As a side note, we point out that the presented flow can work with any implementation of the TRAIN and AGGREGATE procedures. The ALF procedure operates independently as a standalone process. We demonstrated this feature with extensive experiments where we applied ALF to different training and aggregation methods (see Section V).

For a preliminary validation of the stability index, we integrated the stability monitor in a conventional FL scheme based on *FedAvg* without freezing. This is equivalent to running the workflow described in Algorithm 1 dropping the lines 22–23. The results are reported in Figure 3, which shows the evolution over 2000 rounds of the layer-wise stability index of a 5-layer ConvNet trained on the CIFAR-10 dataset (more details about the experimental setup are reported in Section V-A). The plots show a clear trend: despite marginal oscillations, the stability index decreases as the learning process proceeds, demonstrating that the proposed stability monitor offers valuable information about the layer's convergence.

V. EXPERIMENTAL RESULTS

A. Experimental Setup

1) Models & Datasets: The experimental setup consists of two ConvNets with different memory footprint and number of layers. The first benchmark is a 5-layer ConvNet (CNN-5) suited for tiny IoT applications and borrowed from [20]. The network comprises two convolutional layers (conv1 and conv2) with 64 5×5 filters, two fully-connected layers (fc1 and fc2) with 394 and 192 neurons, respectively, and a final linear layer (fc3) that returns the prediction logits. The second network is a more complex 9-layer ConvNet (VGG-9) based on the VGG architecture [42], which can be ported on mobile devices with higher computational resources. Tables II and III collect the memory footprint of each layer for the two ConvNets. The total memory is a proxy of the communication cost for downloading (and uploading) the model from (to) the central server; the reported numbers refer to a standard 32-bit floating-point precision. We applied the ALF mechanism to the layers' weights only as the biases require just a few kBs of data exchange.

The ConvNets were trained using two datasets for image classification, CIFAR-10 and CIFAR-100 [43], using a standard split for training and testing. We applied a standard pre-processing pipeline for data augmentation based on a random crop followed by random horizontal flip and cutout. We split the training datasets across 100 clients with non-IID and imbalanced data partitions to emulate a cross-device setting. Similar to previous works [15], [21], the splits were obtained following a Dirichlet distribution with a concentration parameter equal to 0.3. This partitioning configuration allows each client to hold few or eventually no data samples in some classes.

2) FL Strategies, Training & Evaluation: The training is distributed across 100 clients, with a 10% participation rate, iterating over 2000 rounds. The local training executes Stochastic Gradient Descent (SGD) for $E = 5$ epochs, batch-size 50, and weight decay $1e{-}3$. The learning rate follows a polynomial decay schedule with an initial value of 0.1. Unless otherwise noted, we selected the same hyperparameters in all the experiments.

TABLE IV
TRAINING EVALUATION OF DIFFERENT FL STRATEGIES W/O AND W/ ALF. RESULTS ON CNN-5 FOR THE CIFAR-10 AND CIFAR-100 DATASETS (UNLESS OTHERWISE NOTED, $E = 5$).

	CIFAR10				CIFAR100			
	Top-1	**ΔTop-1**	**CC** (GB)	**Savings**	**Top-1**	**ΔTop-1**	**CC** (GB)	**Savings**
FedAvg	74.83±1.34	-	121.59	-	41.38±0.28	-	124.18	-
FedAvg E=10	73.91±1.96	-0.92	60.80	50.00%	37.68±0.41	-3.70	62.09	50.00%
FedAvg w/ PT-conv	53.71±2.97	-21.12	105.64	13.12%	26.12±0.38	-15.26	108.23	12.84%
FedAvg w/ PT-fc1	69.11±2.40	-5.72	27.83	77.11%	39.02±0.42	-2.36	30.42	75.50%
FedAvg w/ ALF $\mu = 0.11$	77.10±0.18	2.27	68.12	43.98%	41.93±0.03	0.55	39.73	68.01%
FedAvg w/ ALF $\mu = 0.12$	76.20±0.14	1.37	37.67	69.02%	39.87±0.04	-1.51	26.30	78.82%
FedAvg w/ ALF $\mu = 0.13$	75.87±0.20	1.04	27.88	77.07%	39.56±0.03	-1.82	20.18	83.75%
FedProx	75.05±1.38	0.22	121.59	0.00%	41.17±0.26	-0.21	124.18	0.00%
FedProx w/ ALF $\mu = 0.11$	77.35±0.15	2.52	69.89	42.52%	41.44±0.03	0.06	38.40	69.08%
FedOpt	76.25±0.08	1.42	121.59	0.00%	42.03±0.07	0.65	124.18	0.00%
FedOpt w/ ALF $\mu = 0.11$	76.52±0.01	1.69	36.17	70.25%	41.10±0.00	-0.28	26.20	78.90%
FedOpt w/ ALF $\mu = 0.12$	76.25±0.01	1.42	30.84	74.64%	40.96±0.01	-0.42	22.78	81.66%
FedOpt w/ ALF $\mu = 0.13$	75.85±0.00	1.02	27.81	77.13%	40.37±0.01	-1.01	19.98	83.91%

TABLE V
TRAINING EVALUATION OF DIFFERENT FL STRATEGIES W/O AND W/ ALF. RESULTS ON VGG-9 FOR THE CIFAR-10 AND CIFAR-100 DATASETS (UNLESS OTHERWISE NOTED, $E = 5$).

	CIFAR-10				CIFAR-100			
	Top-1	**ΔTop-1**	**CC**	**Savings**	**Top-1**	**ΔTop-1**	**CC**	**Savings**
FedAvg	81.12±1.09	-	520.29	-	49.18±0.23	-	527.17	-
FedAvg E=10	81.23±1.63	0.11	260.15	50.00%	48.78±0.31	-0.40	263.59	50.00%
FedAvg w/ PT-conv	43.05±2.41	-38.07	352.92	32.17%	18.43±0.51	-30.75	359.80	31.75%
FedAvg w/ PT-fc1	79.15±1.61	-1.97	208.50	59.93%	43.70±0.20	-5.48	215.38	59.14%
FedAvg w/ ALF $\mu = 0.11$	83.41±0.07	2.29	365.15	29.82%	49.52±0.05	0.34	392.71	25.51%
FedAvg w/ ALF $\mu = 0.12$	83.10±0.06	1.98	242.11	53.47%	48.56±0.04	-0.62	140.50	73.35%
FedAvg w/ ALF $\mu = 0.13$	82.66±0.04	1.54	151.24	70.93%	47.99±0.03	-1.19	87.46	83.41%
FedProx	81.13±1.24	0.01	520.29	0.00%	49.86±0.22	0.68	527.17	0.00%
FedProx w/ ALF $\mu = 0.11$	83.04±0.05	1.92	347.92	33.13%	49.77±0.06	0.59	392.44	25.56%
FedOpt	84.14±0.12	3.02	520.29	0.00%	53.81±0.11	4.63	527.17	0.00%
FedOpt w/ ALF $\mu = 0.11$	84.20±0.01	3.08	158.04	69.62%	52.99±0.00	3.81	132.79	74.81%
FedOpt w/ ALF $\mu = 0.12$	84.16±0.00	3.04	145.35	72.06%	52.30±0.00	3.12	115.44	78.10%
FedOpt w/ ALF $\mu = 0.13$	83.72±0.00	2.60	128.88	75.23%	51.55±0.00	2.37	104.40	80.20%

For an exhaustive assessment, we conducted a comparative analysis considering different underlying FL strategies, each of them corresponding to a specific optimization (please refer to Section III) as listed below:

- *Synchronization*. As a baseline, we considered the standard FedAvg [20] strategy. We studied the impact of synchronization frequency by running experiments with two different numbers of epochs: $E = 5$, $E = 10$. For $E = 10$, we halved the number of rounds to 1000 so that the number of local training steps keeps the same, yet with a halved communication cost.
- *Training Optimization*. To estimate the potential benefits of more efficient local training, we considered the training procedure proposed by FedProx [11], where a regularization term (referred to as the proximal term) was added to the client's loss function. We set the scaling factor of the proximal term to $1e{-4}$.
- *Aggregation Optimization*. We considered the FedOpt [15] strategy with the Adam solver. Specifically, we set the Adam learning rate to $5e{-3}$ with a polynomial decay schedule, first momentum 0.9, second momentum 0.99, and adaptivity $1e{-3}$.
- *Parameters Freezing*. To assess the efficiency of the proposed automatic freezing mechanism, we considered FedPT [21] as an alternative strategy in which the selection of frozen layers is manually defined and fixed at the beginning of the training. Specifically, we explored two implementations denoted with *PT-conv* and *PT-fc1*. In the former, all the convolutional layers are frozen; in the latter, only the first fully-connected layer is frozen, which is the layer with the largest memory footprint for the two ConvNets under analysis. In both implementations, the training was conducted within the FedAvg strategy.

For the evaluation, we probed the evolution of the classification accuracy of the global model on the test set. Specifically, we computed the moving average over a fixed window of 30 communication rounds, following the procedure in [15]. This filtering suppresses the oscillations due to the intrinsic instability of the training process, enabling a more reliable assessment. Moreover, we measured the client's communication cost in the upload and download phases for synchronization with the central server. All the experiments were run within an emulation strategy developed with PyTorch, version 1.9.

TABLE VI
NUMBER OF COMMUNICATION ROUNDS BEFORE FREEZING (AND ACCURACY GAP TO THE END OF TRAINING). VALUES REPORTED FOR EACH LAYER OF THE CNN-5 MODEL TRAINED WITH DIFFERENT VALUES OF μ (0.11, 0.12, AND 0.13). RESULTS ON CIFAR-10.

		0.11	0.12	0.13
FedAvg w/ ALF	**conv1**	269 (-7.34)	254 (-6.81)	238 (-7.15)
	conv2	617 (-4.75)	478 (-3.81)	423 (-4.15)
	fc1	1194 (-2.90)	635 (-3.42)	456 (-4.14)
	fc2	1212 (-1.80)	658 (-1.88)	489 (-2.02)
	fc3	214 (-9.81)	200 (-10.63)	189 (-10.58)
FedOpt w/ ALF	**conv1**	246 (-5.22)	234 (-5.73)	209 (-6.97)
	conv2	407 (-2.99)	356 (-3.72)	318 (-4.03)
	fc1	616 (-2.68)	521 (-3.27)	469 (-2.03)
	fc2	646 (-1.18)	562 (-1.02)	511 (-2.20)
	fc3	212 (-7.06)	200 (-8.68)	187 (-8.82)

TABLE VII
NUMBER OF COMMUNICATION ROUNDS BEFORE FREEZING (AND ACCURACY GAP TO THE END OF TRAINING). VALUES REPORTED FOR EACH LAYER OF THE CNN-5 MODEL TRAINED WITH DIFFERENT VALUES OF μ (0.11, 0.12, AND 0.13). RESULTS ON CIFAR-100.

		0.11	0.12	0.13
FedAvg w/ ALF	**conv1**	683 (-1.11)	447 (-0.86)	348 (-1.28)
	conv2	716 (-0.60)	490 (-0.20)	381 (-0.59)
	fc1	641 (-2.20)	420 (-1.56)	319 (-2.28)
	fc2	512 (-3.55)	336 (-3.08)	259 (-3.93)
	fc3	451 (-4.33)	279 (-4.54)	223 (-5.39)
FedOpt w/ ALF	**conv1**	381 (-2.08)	372 (-1.92)	325 (-1.84)
	conv2	499 (-0.42)	439 (-0.60)	383 (-0.61)
	fc1	430 (-1.44)	371 (-1.93)	323 (-1.87)
	fc2	254 (-4.06)	222 (-5.02)	205 (-4.63)
	fc3	168 (-7.16)	162 (-7.31)	158 (-6.93)

B. Results

1) Training Evaluation: Tables IV and V report the average accuracy, the relative standard deviation over the last 30 rounds, and the communication cost at the end of training for the selected networks and datasets. The tables consist of three sections, one for each FL strategy under analysis (FedAvg, FedProx, and FedOpt), collecting the results with and without the ALF mechanism. We also considered different values of the stability threshold μ, showcasing the impact of this parameter on the training efficiency.

Our first analysis focused on the FedAvg strategy. We validated alternative optimization strategies for communication reduction, i.e., synchronization frequency tuning (FedAvg $E = 10$) and parameter freezing with FedPT (FedAvg w/ PT-conv1 and FedAvg w/ PT-fc1). Looking at the results on CNN-5 (Table IV), these strategies ensure communication savings at the cost of a substantial accuracy drop. Specifically, with $E = 10$, the accuracy losses range from -0.92% for CIFAR-10 to -3.70% for CIFAR-100. Similarly, FedAvg w/ PT-fc1 cuts the communication cost by 77.11% (75.50%) for CIFAR-10 (CIFAR-100) with -5.72% (-2.36%) lower accuracy. This demonstrates that freezing from the beginning of the training causes excessive accuracy degradation. Moreover, the results highlight another important limitation of manual freezing: the wrong selection of the frozen layers leads to sub-optimal solutions. For instance, PT-conv brings larger accuracy losses (down to -21.12% for CIFAR-10) with lower savings (13.12%). This finding motivates the need for an automatic mechanism to detect when and which layers to freeze for concurrent accuracy and communication cost improvement.

The proposed ALF mechanism, as demonstrated by the reported results, targets this specific need instead. On CNN-5 for CIFAR-10, FedAvg w/ ALF outperforms its competitors in all evaluation metrics and can reach even higher accuracy levels than the baseline implementation. The stability threshold μ works as a knob to balance the transmission cost, as the layers get frozen sooner with higher values of μ. With $\mu = 0.13$, FedAvg w/ ALF requires a communication cost close to that of FedAvg w/ PT-conv (27.88 GB vs. 27.83 GB) reaching much higher accuracy (75.87% vs. 69.11%). On CIFAR-100, with larger values of μ (0.12 and 0.13), we achieved lower accuracy than the baseline FedAvg (-1.82% in the worst case), but still far better than the other optimization strategies.

The experiments with FedProx and FedOpt further highlight the efficiency of ALF and its flexibility under different settings. Note that the adopted FL strategies require the same communication cost because the only differences compared to FedAvg lie in the local training (FedProx) or the server aggregation (FedOpt). Still, no modification is operated during the transmission phases. In all strategies, the learning encompasses the same number of rounds, and the communication cost of each round keeps constant during the process, regardless of the impact of the model updates on the final prediction quality. Instead, ALF modulates the communication volume depending on the evolution of the clients' updates, gradually lowering the amount of data exchanged as the learning moves on. Therefore, its integration guarantees a substantial reduction in the communication cost (up to 83.91%), still retaining competitive accuracy levels.

The results on the VGG-9 model (Table V) reinforce the above observations, showing that ALF works even with a more complex network. Moreover, the comparative assessment reveals an important interplay between ALF and the optimization brought by more advanced FL schemes. This can be inferred by comparing the results of FedAvg and FedOpt with ALF with $\mu = 0.11$. In both cases, applying ALF improves efficiency, yet with different results. Combining FedOpt and ALF maximizes accuracy and communication costs for all datasets. The benefits can be observed mainly on CIFAR-100, where FedAvg w/ ALF ($\mu = 0.11$) shows 0.34% accuracy improvements with 25.51% of savings, while FedOpt w/ ALF ensures 3.81% higher accuracy with 74.81% lower communication cost. This trend suggests that the more advanced model aggregation proposed by FedOpt accelerates the layers' stability, activating the ALF freezing logic sooner, which leads to higher accuracy and communication savings.

To better understand the impact of the model aggregation method on the layers' stability, we recorded the number of rounds after which each network layer got frozen. For the sake of simplicity, we only report the analysis on CNN-5 (Tables VI and VII). Higher values of μ accelerate the freezing of all layers in both strategies. However, the most

TABLE VIII
COMMUNICATION COST IN GB OF DIFFERENT FL STRATEGIES W/O AND W/ ALF ($\mu = 0.11$). RESULTS ON CNN-5 FOR THE CIFAR-10 DATASET.

	FedAvg		FedProx		FedOpt	
A_{th}	w/o ALF	w/ ALF	w/o ALF	w/ ALF	w/o ALF	w/ ALF
74.0	58.55	50.42	52.59	50.13	28.70	27.91
74.5	64.81	60.01	63.59	59.87	32.71	28.12
75.0	100.31	67.97	97.70	69.61	33.13	31.91
75.5	-	68.03	-	69.70	55.02	36.09
76.0	-	68.05	-	69.84	58.85	36.09
76.5	-	68.06	-	69.84	71.56	36.10
77.0	-	68.09	-	69.85	-	-

TABLE IX
COMMUNICATION COST IN GB OF DIFFERENT FL STRATEGIES W/O AND W/ ALF ($\mu = 0.11$). RESULTS ON CNN-5 FOR THE CIFAR-100 DATASET.

	FedAvg		FedProx		FedOpt	
A_{th}	w/o ALF	w/ ALF	w/o ALF	w/ ALF	w/o ALF	w/ ALF
39.0	39.86	32.94	39.55	32.16	24.96	21.95
39.5	44.46	36.19	49.49	37.70	27.69	25.02
40.0	58.18	39.10	58.55	37.90	30.36	25.66
40.5	79.35	39.24	80.35	38.13	32.97	25.87
41.0	103.88	39.43	111.20	38.31	40.42	26.10
41.5	-	39.64	-	-	50.04	-
42.0	-	39.72	-	-	88.35	-

TABLE X
COMMUNICATION COST IN GB OF DIFFERENT FL STRATEGIES W/O AND W/ ALF ($\mu = 0.11$). RESULTS ON VGG-9 FOR THE CIFAR-10 DATASET.

	FedAvg		FedProx		FedOpt	
A_{th}	w/o ALF	w/ ALF	w/o ALF	w/ ALF	w/o ALF	w/ ALF
81.0	255.98	271.23	272.37	271.32	85.59	94.91
81.5	430.28	354.10	-	346.98	98.60	99.84
82.0	-	363.97	-	347.16	101.20	101.40
82.5	-	364.21	-	347.32	122.01	121.44
83.0	-	364.38	-	347.88	140.74	138.01
83.5	-	-	-	-	202.13	157.82
84.0	-	-	-	-	283.56	157.85

TABLE XI
COMMUNICATION COST IN GB OF DIFFERENT FL STRATEGIES W/O AND W/ ALF ($\mu = 0.11$). RESULTS ON VGG-9 FOR THE CIFAR-100 DATASET.

	FedAvg		FedProx		FedOpt	
A_{th}	w/o ALF	w/ ALF	w/o ALF	w/ ALF	w/o ALF	w/ ALF
49.0	489.74	392.34	297.59	283.57	78.55	77.85
49.5	-	392.61	430.44	386.08	83.56	86.17
50.0	-	-	-	-	89.36	90.06
50.5	-	-	-	-	105.17	105.50
51.0	-	-	-	-	110.71	110.33
51.5	-	-	-	-	133.11	131.90
52.0	-	-	-	-	140.76	132.29
52.5	-	-	-	-	153.14	132.55
53.0	-	-	-	-	181.35	132.59
53.5	-	-	-	-	294.43	-

interesting finding is that, for a fixed μ, every layer gets frozen sooner in FedOpt than in FedAvg. Considering $\mu = 0.11$ on the CIFAR-10 dataset (Table VI), fc1 is frozen after 1194 rounds in FedAvg and 616 rounds in FedOpt. ALF was able to adapt to the underlying learning algorithm. If the network layers converge quicker to their final representation due to a more efficient FL scheme, then ALF can detect the improved stability and control the layer freezing accordingly. Moreover, a comparison between CIFAR-10 (Tables VI) and CIFAR-100 (Table VII) reveals the freezing round and the freezing order do change depending on the training dataset, suggesting that the layers' convergence depends on data, and not just the network topology or the learning strategy. On CIFAR-10, conv1 is frozen before fc2, whereas on CIFAR-100, it is the opposite. This consideration further justifies the need for an automatic mechanism to drive the layers freezing.

Tables VI and VII also report the difference between the accuracy reached when the layers are frozen and the final accuracy achieved at the end of training. When ALF freezes the first layer (8% to 22% of the training progress, depending on the dataset and the FL setting), the model accuracy is far from the final value (from -7.06% to -10.63% for CIFAR-10, from -4.33% to -7.31% for CIFAR-100), which means ALF activates the freezing mechanism before the model converges to a stable accuracy. The final accuracy is achieved by updating only a progressively smaller subset of the layers, indicating that training and synchronizing all layers is unnecessary for model convergence.

2) Accuracy vs. Communication Cost Trade-offs: The communication volume can be shrunk by reducing the number of rounds, still preserving accuracy. This solution is straightforward and orthogonal to any strategy and optimization method. However, the kind of FL strategy adopted affects the achievable performance. To evaluate the potential benefits, we monitored the evolution of the global model's test accuracy over time, annotating the communication cost needed to reach predefined accuracy thresholds A_{th} for the three FL strategies under analysis. Specifically, we considered the conventional implementation and the integration with ALF for each FL strategy.

Table VIII summarizes the results for the CNN-5 trained on the CIFAR-10 dataset. Dashes in the table indicate that the corresponding FL strategy failed to reach the target accuracy within 2000 rounds (i.e., the total number of rounds). For ALF, the results refer to a setting with $\mu = 0.11$. As already demonstrated by previous studies [11], [35], more advanced FL schemes, like FedProx and FedOpt, show faster convergence than FedAvg, reducing the data exchange needed to achieve a predefined level of accuracy. Specifically, FedProx brings slight reductions of few GBs (e.g., for $A_{th} \geq 75.0\%$ the cost reduces from $100.31\,\text{GB}$ down to $97.70\,\text{GB}$). In contrast, FedOpt ensures substantial improvements in both accuracy and performance. The integration of ALF further increases the achievable savings for all strategies and accuracy levels. The most impressive case is FedOpt, where ALF enables substantial communication cost savings, from $71.56\,\text{GB}$ to $36.10\,\text{GB}$ for 76.5% of accuracy.

Similar considerations hold for the CIFAR-100 dataset (Table IX). Under their standard implementation, the adopted FL strategies need substantial overhead to improve the accuracy by a few percentage points. For instance, to increase accuracy from 39.0% to 41.0%, FedAvg consumes $64.02\,\text{GB}$ more data (from $39.86\,\text{GB}$ to $103.88\,\text{GB}$), FedOpt $15.46\,\text{GB}$ (from

TABLE XII
COMMUNICATION COST IN GB OF FEDOPT W/O AND W/ ALF FOR DIFFERENT VALUES OF THE STABILITY THRESHOLD μ. RESULTS ON CNN-5 FOR THE CIFAR-10 DATASET.

		FedOpt w/ ALF										
A_{th}	**FedOpt**	**0.05**	**0.06**	**0.07**	**0.08**	**0.09**	**0.10**	**0.11**	**0.12**	**0.13**	**0.14**	**0.15**
74.0	28.70	28.70	28.63	28.49	28.45	23.79	28.34	27.91	27.51	27.29	24.50	22.46
74.5	32.71	32.71	32.37	32.41	32.37	32.11	31.80	28.12	30.52	27.71	24.51	22.46
75.0	33.13	33.13	32.92	32.96	32.91	32.65	32.32	31.91	30.60	27.71	24.51	22.48
75.5	55.02	52.65	47.24	46.40	45.78	45.02	40.77	36.09	30.63	27.72	24.52	-
76.0	58.85	58.63	54.43	53.30	51.97	45.72	40.84	36.09	30.75	-	-	-
76.5	71.56	65.89	62.23	60.96	52.02	45.95	40.93	36.10	-	-	-	-
77.0	-	87.11	74.94	61.24	52.36	-	-	-	-	-	-	-
77.5	-	96.87	-	-	-	-	-	-	-	-	-	-

TABLE XIII
COMMUNICATION COST IN GB OF FEDOPT W/O AND W/ ALF FOR DIFFERENT VALUES OF THE STABILITY THRESHOLD μ. RESULTS ON CNN-5 FOR THE CIFAR-100 DATASET.

		FedOpt w/ ALF										
A_{th}	**FedOpt**	**0.05**	**0.06**	**0.07**	**0.08**	**0.09**	**0.10**	**0.11**	**0.12**	**0.13**	**0.14**	**0.15**
39.0	24.96	24.96	24.96	24.80	21.94	21.83	21.58	21.95	21.82	19.51	17.25	15.81
39.5	27.69	27.69	27.69	26.56	24.97	24.88	24.06	25.02	22.18	19.70	17.43	-
40.0	30.36	30.36	30.36	28.62	27.58	27.20	26.37	25.66	22.34	19.87	17.47	-
40.5	32.97	32.97	32.97	33.05	31.61	31.99	30.06	25.87	22.66	-	-	-
41.0	40.42	40.42	39.47	38.63	38.01	36.82	30.23	26.10	22.68	-	-	-
41.5	50.04	49.84	47.05	42.76	41.05	-	30.30	-	-	-	-	-
42.0	88.35	63.19	60.00	53.32	44.66	-	-	-	-	-	-	-
42.5	-	82.36	66.90	-	-	-	-	-	-	-	-	-

$24.96\,\text{GB}$ to $40.42\,\text{GB}$). In contrast, FedOpt w/ ALF calls for $4.15\,\text{GB}$ more only (from $21.95\,\text{GB}$ to $26.10\,\text{GB}$). This observation proves that, as the learning process proceeds, only small portions of the local models should be trained and synchronized with the global model.

As already stated, layer freezing can reduce the maximum achievable accuracy. FedOpt w/ ALF is about 1% less accurate than FedOpt w/o ALF. However, ALF guarantees 41% of accuracy with only $26.10\,\text{GB}$, while, with a comparable cost ($27.69\,\text{GB}$), the conventional FedOpt achieves 39.5% of accuracy. Notice that ALF offers the possibility to trade off accuracy and communication cost with a fine-tuning of the stability threshold μ, as it will be detailed in Section V-B3.

The analysis of the VGG-9 experiments confirms what was observed for CNN-5 (Tables X and XI). At lower accuracy levels, ALF might require higher communication cost, yet with a small difference of few GBs. Indeed, in the early phases of learning, just a few layers are frozen. Hence, the impact of freezing on the data volume is initially low. However, the savings increase at higher accuracy levels as the transmission cost per round gradually reduces. This effect is evident in CIFAR-100, where FedOpt w/ ALF requires less than $1\,\text{GB}$ to increase accuracy from 51.5% to 53%.

3) Sensitivity Analysis: As discussed in Section V-B1, the stability threshold μ acts as a control knob for tuning the freezing mechanism. A sweeping analysis, with μ ranging from 0.05 to 0.15, step 0.01, on the CNN-5 model for both CIFAR-10 and CIFAR-100, is reported in Tables XII and XIII.

Although there is no closed-form solution to identify the most suitable value of μ for a given accuracy or communication budget, it is clear that multiple values of μ guarantee high accuracy and cost reductions simultaneously. For CIFAR-10, eight configurations of ALF reach 76.5% accuracy with savings ranging from 7.92% ($\mu = 0.05$) to 49.54% ($\mu = 0.11$) compared to the conventional implementation of FedOpt. Similarly, for CIFAR-100, four configurations guarantee accuracy higher than 42.0% ($\mu \leq 0.08$). In general, lower values of μ ensure better accuracy than FedOpt w/o ALF, with competitive or lower data volume (as for CIFAR-100 with $\mu \leq 0.06$).

4) Convergence Time: We investigated the effects of the ALF mechanism on the convergence time, ensuring the layers freezing does not affect the training time. To provide a proxy of the training duration, we measured the number of rounds needed to reach a predefined set of accuracy thresholds for all the FL strategies under analysis (Tables XIV-XVII). The results refer to an ALF setting with $\mu = 0.11$. In most cases, the ALF mechanism reduces the number of rounds if compared with the baseline implementations. The best case is for CNN-5 trained on CIFAR-100 (Table XV), where ALF achieves 41.0% of accuracy with 981 fewer rounds than the standard FedAvg (1672 vs. 691). Therefore, ALF maximizes communication savings thanks to the combined effects of smaller transmitted payloads and fewer rounds. In some settings, we observed a slight increase in rounds due to ALF. This happens for VGG-9 trained with FedProx and FedOpt in particular (Tables XVI and XVII). Nevertheless, the overhead is rather quite limited, just a few tens of rounds, with a worst-case of 70 rounds for $A_{\text{th}} \geq 53.0$ in VGG-9 on CIFAR-100 trained with FedOpt (Table XVII). Regarding this overhead, one should consider that the total training time depends on the round duration and not just the number of rounds. That opens to some additional considerations. The effect of ALF is to shorten the round duration by alleviating the workload of local training

TABLE XIV
ROUND-TO-ACCURACY OF DIFFERENT FL STRATEGIES W/O AND W/ ALF ($\mu = 0.11$). RESULTS ON CNN-5 FOR THE CIFAR-10 DATASET.

	FedAvg		FedProx		FedOpt	
Ath	**w/o ALF**	**w/ ALF**	**w/o ALF**	**w/ ALF**	**w/o ALF**	**w/ ALF**
74.0	962	864	864	860	471	467
74.5	1065	1046	1045	1045	537	471
75.0	1649	1207	1606	1242	544	543
75.5	-	1215	-	1254	904	669
76.0	-	1221	-	1285	967	684
76.5	-	1234	-	1296	1176	873
77.0	-	1591	-	1307	-	-

TABLE XV
ROUND-TO-ACCURACY OF DIFFERENT FL STRATEGIES W/O AND W/ ALF ($\mu = 0.11$). RESULTS ON CNN-5 FOR THE CIFAR-100 DATASET.

	FedAvg		FedProx		FedOpt	
Ath	**w/o ALF**	**w/ ALF**	**w/o ALF**	**w/ ALF**	**w/o ALF**	**w/ ALF**
39.0	641	533	636	525	401	367
39.5	715	592	796	632	445	423
40.0	936	655	942	650	488	449
40.5	1277	670	1293	677	530	473
41.0	1672	691	1790	722	650	585
41.5	-	756	-	-	805	-
42.0	-	1799	-	-	1422	-

and data transmission phases. Indeed, the training duration is proportional to the number of operations needed for updating the model weights, which inversely correlates with the number of frozen layers increasing over time. Furthermore, ALF shortens the communication time needed for synchronization (in both the download and upload phases) by reducing the size of the transmitted payloads. As a result, the increased number of rounds does not necessarily lead to longer training time or worse performance.

VI. CONCLUSION

This work introduced ALF, an automatic mechanism for layer freezing aimed at reducing communication costs in cross-device FL. ALF monitors the evolution of the models' updates generated by the clients to detect stable layers, i.e., layers whose updates bring marginal or no improvement to the prediction capability of the global model. Also, ALF integrates a freezing logic that keeps the weights of the stable layers constant in the local and global models, avoiding the need for their synchronization and, hence, the corresponding transmission costs. Extensive validation on different experimental settings demonstrated the benefits of ALF, showing that the proposed mechanism brings concurrent improvements in accuracy and communication efficiency. Moreover, we proved that ALF is embeddable into existing FL strategies with minimal modifications of the standard flow. All the operations of ALF are performed on the central server, avoiding additional overhead on the resource-constrained clients. Most importantly, when integrated into more advanced FL strategies, ALF can leverage complementary strategies operating at different levels, e.g., local training or central aggregation, to accelerate the layers freezing and maximize the achievable accuracy and savings through joint optimization.

TABLE XVI
ROUND-TO-ACCURACY OF DIFFERENT FL STRATEGIES W/O AND W/ ALF ($\mu = 0.11$). RESULTS ON VGG-9 FOR THE CIFAR-10 DATASET.

	FedAvg		FedProx		FedOpt	
Ath	**w/o ALF**	**w/ ALF**	**w/o ALF**	**w/ ALF**	**w/o ALF**	**w/ ALF**
81.0	983	1044	1046	1047	328	365
81.5	1653	1473	-	1599	378	384
82.0	-	1632	-	1606	388	390
82.5	-	1642	-	1613	468	469
83.0	-	1650	-	1708	540	540
83.5	-	-	-	-	776	691
84.0	-	-	-	-	1089	726

TABLE XVII
ROUND-TO-ACCURACY OF DIFFERENT FL STRATEGIES W/O AND W/ ALF ($\mu = 0.11$). RESULTS ON VGG-9 FOR THE CIFAR-100 DATASET.

	FedAvg		FedProx		FedOpt	
Ath	**w/o ALF**	**w/ ALF**	**w/o ALF**	**w/ ALF**	**w/o ALF**	**w/ ALF**
49.0	1857	1787	1128	1163	297	297
49.5	-	1932	1632	1686	316	329
50.0	-	-	-	-	338	344
50.5	-	-	-	-	398	404
51.0	-	-	-	-	419	424
51.5	-	-	-	-	504	542
52.0	-	-	-	-	533	558
52.5	-	-	-	-	580	591
53.0	-	-	-	-	687	757
53.5	-	-	-	-	1116	-

REFERENCES

[1] D. Yates and M. Z. Islam, "Data mining on smartphones: An introduction and survey," *ACM Comput. Surv.*, vol. 55, no. 5, pp. 101:1–101:38, May 2023.

[2] V. Peluso and A. Calimera, "Weak-mac: Arithmetic relaxation for dynamic energy-accuracy scaling in convnets," in *Int. Symp. on Circuits and Systems (ISCAS)*, 2018, pp. 1–5.

[3] V. Peluso, R. G. Rizzo, A. Cipolletta, and A. Calimera, "Inference on the edge: Performance analysis of an image classification task using off-the-shelf cpus and open-source convnets," in *6th Int. Conf. on Social Networks Analysis, Management and Security (SNAMS)*, 2019, pp. 454–459.

[4] M. Grimaldi, V. Peluso, and A. Calimera, "Optimality assessment of memory-bounded convnets deployed on resource-constrained RISC cores," *IEEE Access*, vol. 7, pp. 152 599–152 611, 2019.

[5] Q. Yang, Y. Liu, T. Chen, and Y. Tong, "Federated machine learning: Concept and applications," *ACM Trans. Intell. Syst. Technol.*, vol. 10, no. 2, pp. 12:1–12:19, Mar. 2019.

[6] J. Xu, B. S. Glicksberg, C. Su, P. B. Walker, J. Bian, and F. Wang, "Federated learning for healthcare informatics," *J. Heal. Informatics Res.*, vol. 5, no. 1, pp. 1–19, Mar. 2021.

[7] A. Hard, K. Rao, R. Mathews, F. Beaufays, S. Augenstein, H. Eichner, C. Kiddon, and D. Ramage, "Federated learning for mobile keyboard prediction," *CoRR*, vol. abs/1811.03604, 2018.

[8] S. Ramaswamy, R. Mathews, K. Rao, and F. Beaufays, "Federated learning for emoji prediction in a mobile keyboard," *CoRR*, vol. abs/1906.04329, 2019.

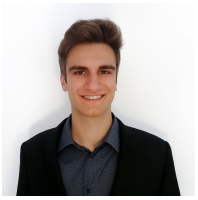
[9] V. Rey, P. M. S. Sánchez, A. H. Celdrán, and G. Bovet, "Federated learning for malware detection in iot devices," *Comput. Networks*, vol. 204, p. 108693, Feb. 2022.

[10] J. Pei, K. Zhong, M. A. Jan, and J. Li, "Personalized federated learning framework for network traffic anomaly detection," *Comput. Networks*, p. 108906, May 2022.

[11] T. Li, A. K. Sahu, M. Zaheer, M. Sanjabi, A. Talwalkar, and V. Smith, "Federated optimization in heterogeneous networks," in *Proc. of Machine Learning and Systems (MLSys), 2020*, 2020.

[12] D. A. E. Acar, Y. Zhao, R. Matas, M. Mattina, P. Whatmough, and V. Saligrama, "Federated learning based on dynamic regularization," in *International Conference on Learning Representations (ICLR)*, 2020.

[13] Q. Li, B. He, and D. Song, "Model-contrastive federated learning," in *Conf. on Computer Vision and Pattern Recognition (CVPR), 2021*, 2021, pp. 10 713–10 722.
[14] J. Wang, Q. Liu, H. Liang, G. Joshi, and H. V. Poor, "A novel framework for the analysis and design of heterogeneous federated learning," *IEEE Trans. Signal Process.*, vol. 69, pp. 5234–5249, 2021.
[15] S. J. Reddi, Z. Charles, M. Zaheer, Z. Garrett, K. Rush, J. Konečný, S. Kumar, and H. B. McMahan, "Adaptive federated optimization," in *International Conference on Learning Representations (ICLR)*, 2021.
[16] X. Mu, Y. Shen, K. Cheng, X. Geng, J. Fu, T. Zhang, and Z. Zhang, "Fedproc: Prototypical contrastive federated learning on non-iid data," *Future Gener. Comput. Syst.*, vol. 143, pp. 93–104, Jun. 2023.
[17] J. Wang, Z. Charles, Z. Xu, G. Joshi, H. B. McMahan, B. A. y Arcas, M. Al-Shedivat, G. Andrew, S. Avestimehr, K. Daly, D. Data, S. N. Diggavi, H. Eichner, A. Gadhikar, Z. Garrett, A. M. Girgis, F. Hanzely, A. Hard, C. He, S. Horváth, Z. Huo, A. Ingerman, M. Jaggi, T. Javidi, P. Kairouz, S. Kale, S. P. Karimireddy, J. Konečný, S. Koyejo, T. Li, L. Liu, M. Mohri, H. Qi, S. J. Reddi, P. Richtárik, K. Singhal, V. Smith, M. Soltanolkotabi, W. Song, A. T. Suresh, S. U. Stich, A. Talwalkar, H. Wang, B. E. Woodworth, S. Wu, F. X. Yu, H. Yuan, M. Zaheer, M. Zhang, T. Zhang, C. Zheng, C. Zhu, and W. Zhu, "A field guide to federated optimization," *CoRR*, vol. abs/2107.06917, 2021.
[18] J. Huang, F. Qian, A. Gerber, Z. M. Mao, S. Sen, and O. Spatscheck, "A close examination of performance and power characteristics of 4g LTE networks," in *10th Int. Conf. on Mobile Systems (MobiSys)*, 2012, pp. 225–238.
[19] M. Raghu, J. Gilmer, J. Yosinski, and J. Sohl-Dickstein, "SVCCA: singular vector canonical correlation analysis for deep learning dynamics and interpretability," in *30th Annual Conference on Neural Information Processing Systems (NeurIPS)*, 2017, pp. 6076–6085.
[20] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Proceedings of the 20th Int. Conf. on Artificial Intelligence and Statistics (AISTATS)*, vol. 54, 2017, pp. 1273–1282.
[21] H. Sidahmed, Z. Xu, A. Garg, Y. Cao, and M. Chen, "Efficient and private federated learning with partially trainable networks," *CoRR*, vol. abs/2110.03450, 2021.
[22] W. Y. B. Lim, N. C. Luong, D. T. Hoang, Y. Jiao, Y. Liang, Q. Yang, D. Niyato, and C. Miao, "Federated learning in mobile edge networks: A comprehensive survey," *IEEE Commun. Surveys Tuts.*, vol. 22, no. 3, pp. 2031–2063, 2020.
[23] Y. Chen, X. Sun, and Y. Jin, "Communication-efficient federated deep learning with layerwise asynchronous model update and temporally weighted aggregation," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 31, no. 10, pp. 4229–4238, Oct. 2020.
[24] S. Lee, T. Zhang, C. He, and S. Avestimehr, "Layer-wise adaptive model aggregation for scalable federated learning," *CoRR*, vol. abs/2110.10302, 2021.
[25] Z. Zhu, Y. Shi, J. Luo, F. Wang, C. Peng, P. Fan, and K. B. Letaief, "Fedlp: Layer-wise pruning mechanism for communication-computation efficient federated learning," *CoRR*, vol. abs/2303.06360, 2023.
[26] C. Chen, H. Xu, W. Wang, B. Li, B. Li, L. Chen, and G. Zhang, "Synchronize only the immature parameters: Communication-efficient federated learning by freezing parameters adaptively," *IEEE Trans. Parallel Distrib. Syst.*, pp. 1–18, 2023.
[27] 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 J. Sel. Areas Commun.*, vol. 37, no. 6, pp. 1205–1221, Jun. 2019.
[28] L. Wang, W. Wang, and B. Li, "CMFL: mitigating communication overhead for federated learning," in *39th Int. Conf. on Distributed Computing Systems (ICDCS)*, 2019, pp. 954–964.
[29] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *3rd Int. Conf. on Learning Representations (ICLR)*, 2015.
[30] J. C. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *J. Mach. Learn. Res.*, vol. 12, pp. 2121–2159, 2011.
[31] M. Zaheer, S. J. Reddi, D. S. Sachan, S. Kale, and S. Kumar, "Adaptive methods for nonconvex optimization," in *31st Annual Conference on Neural Information Processing Systems (NeurIPS)*, 2018, pp. 9815–9825.
[32] F. Sattler, S. Wiedemann, K. Müller, and W. Samek, "Robust and communication-efficient federated learning from non-iid data," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 31, no. 9, pp. 3400–3413, Sep. 2020.
[33] J. Xu, W. Du, Y. Jin, W. He, and R. Cheng, "Ternary compression for communication-efficient federated learning," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 33, no. 3, pp. 1162–1176, Mar. 2022.
[34] J. Mills, J. Hu, and G. Min, "Communication-efficient federated learning for wireless edge intelligence in iot," *IEEE Internet Things J.*, vol. 7, no. 7, pp. 5986–5994, Jul. 2020.
[35] A. Muhammad, A. Moustafa, and T. Ito, "Fedopt: towards communication efficiency and privacy preservation in federated learning," *Applied Sciences*, vol. 10, no. 8, p. 2864, Apr. 2020.
[36] K. Goutam, S. Balasubramanian, D. Gera, and R. R. Sarma, "Layerout: Freezing layers in deep neural networks," *SN Comput. Sci.*, vol. 1, no. 5, p. 295, Sep. 2020.
[37] Y. Miyauchi, H. Mori, T. Youkawa, K. Yamada, S. Izumi, M. Yoshimoto, H. Kawaguchi, and A. Inoue, "Layer skip learning using LARS variables for 39% faster conversion time and lower bandwidth," in *25th Int. Conf. on Electronics, Circuits and Systems (ICECS)*, 2018, pp. 673–676.
[38] P. Safayenikoo and I. Akturk, "Weight update skipping: Reducing training time for artificial neural networks," *IEEE Trans. Emerg. Sel. Topics Circuits Syst.*, vol. 11, no. 4, pp. 563–574, Dec. 2021.
[39] T. Yang, D. Guliani, F. Beaufays, and G. Motta, "Partial variable training for efficient on-device federated learning," in *Int. Conf. on Acoustics, Speech and Signal Processing (ICASSP) 2022*, 2022, pp. 4348–4352.
[40] K. Pfeiffer, M. Rapp, R. Khalili, and J. Henkel, "Coco-fl: Communication- and computation-aware federated learning via partial NN freezing and quantization," *CoRR*, vol. abs/2203.05468, 2022.
[41] E. Malan, V. Peluso, A. Calimera, and E. Macii, "Communication-efficient federated learning with gradual layer freezing," *IEEE Embedded Syst. Lett.*, vol. 15, no. 1, pp. 25–28, Mar. 2023.
[42] H. Wang, M. Yurochkin, Y. Sun, D. S. Papailiopoulos, and Y. Khazaeni, "Federated learning with matched averaging," in *8th Int. Conf. on Learning Representations (ICLR)*, 2020.
[43] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," University of Toronto, Tech. Rep., 2009.



Erich Malan (Graduate Student Member, IEEE) received the M.Sc. degree in Data Science and Engineering from the Politecnico di Torino in 2021. He is currently pursuing the Ph.D. degree with the Department of Control and Computer Engineering, Politecnico di Torino. His main research interest focuses on distributed artificial intelligence.

Valentino Peluso (Member, IEEE) received the M.Sc. degree in electronic engineering and the Ph.D. degree in computer engineering from the Politecnico di Torino. He is currently an Assistant Professor with the Department of Control and Computer Engineering, Politecnico di Torino. His main research interest focuses on the optimization and compression of deep learning models for their deployment on low-power embedded systems.

Andrea Calimera (Member, IEEE) received the M.Sc. degree in electronic engineering and a Ph.D. in computer engineering from the Politecnico di Torino. He is an Associate Professor of computer engineering with the Politecnico di Torino. His research interests cover the design automation of digital circuits and embedded systems, including optimization techniques for low-power and reliability, energy/quality management, logic synthesis, and emerging computing paradigms.

Enrico Macii (Fellow, IEEE) received the Laurea degree in electrical engineering from the Politecnico di Torino, Turin, Italy, the Laurea degree in computer science from the Università di Torino, Turin, and the Ph.D. degree in computer engineering from the Politecnico di Torino received respectively in 1990, 1991, and 1995. He is currently a Full Professor of Computer Engineering with the Politecnico di Torino. His research interests include the design of electronic digital circuits and systems, with a particular emphasis on low-power consumption aspects.

Paolo Montuschi (Fellow, IEEE) 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 architectures, 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 2020-23 Chair of the IEEE TAB/ARC, and a member of the IEEE Awards Board. Previously, he served in a number of positions, including the Editor-in-Chief of the IEEE Transactions on Computers (2015-18), the 2017-20 IEEE Computer Society Awards Committee Chair, a Member-at-Large of IEEE PSPB (2015-20), and as the Chair of its Strategic Planning Committee (2019-20).