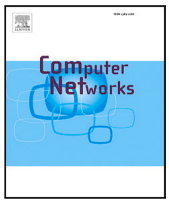
Computer Networks 256 (2025) 110900

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet



Function Placement for In-network Federated Learning

Nour-El-Houda Yellas [a,b,*], Bernardetta Addis [c], Selma Boumerdassi [a], Roberto Riggio [d], Stefano Secci [a]

[a] Cnam, Paris, France
[b] Orange, Châtillon, France
[c] Université de Lorraine, CNRS, LORIA, Nancy, France
[d] Polytechnic University of Marche, Ancona, Italy

ARTICLE INFO

Keywords:
Federated learning
Artificial intelligence functions
Placement

ABSTRACT

Federated learning (FL), particularly when data is distributed across multiple clients, helps reducing the learning time by avoiding training on a massive pile-up of data. Nonetheless, low computation capacities or poor network conditions can worsen the convergence time, therefore decreasing accuracy and learning performance. In this paper, we propose a framework to deploy FL clients in a network, while compensating end-to-end time variation due to heterogeneous network setting. We present a new distributed learning control scheme, named In-network Federated Learning Control (IFLC), to support the operations of distributed federated learning functions in geographically distributed networks, and designed to mitigate the stragglers with lower deployment costs. IFLC adapts the allocation of distributed hardware accelerators to modulate the importance of local training latency in the end-to-end delay of federated learning applications, considering both deterministic and stochastic delay scenarios. By extensive simulation on realistic instances of an in-network anomaly detection application, we show that the absence of hardware accelerators can strongly impair the learning efficiency. Additionally, we show that providing hardware accelerators at only 50% of the nodes, can reduce the number of stragglers by at least 50% and up to 100% with respect to a baseline FIRST-FIT algorithm, while also lowering the deployment cost by up to 30% with respect to the case without hardware accelerators. Finally, we explore the effect of topology changes on IFLC across both hierarchical and flat topologies.

1. Introduction

Network automation is expected to be one of the applications that could leverage on distributed AI modules for both learning and inference tasks. New network functions related to analytic tasks have already appeared in telecommunication standards, for instance the NetWork Data Analytics Function (NWDAF): it has been included in 3GPP 5G system since Release 16 [1] as a function tailored to the analysis of monitoring data from the 5G core network functions.

Indeed, the 5G core network system is being increasingly integrated in core networks, with a rapidly increasing level of geographical distribution, mostly led by the need to offer 1 ms access latency performance to 5G services. Such performance targets are therefore pushing for distribution of network functions and related monitoring, learning and inference tasks. For this re-architecturing, 5G and beyond-5G solutions are leveraging on Multi-access Edge Computing (MEC) technologies, with so-called traffic local break-out gateway to steer some traffic requiring such performance to edge application servers co-located with distributed 5G core functions [2]. On the other hand, MEC architecture hosts could, besides serving end-application needs, also allow to deploy computing functions for a set of infrastructure needs.

Few distributed learning frameworks have made their way into commercial computing systems. Among them, Federated Learning (FL) that introduces a hierarchical learning approach where edge nodes perform learning based on local data, and send the result of their local learning to a server. The server aggregates the learning parameters of multiple edge nodes and then updates the edge nodes with its global view parameters. Besides being already used for a number of mobile device usages by companies as Google and Meta [3], FL is also being considered for in-network AI functions as the NWDAF [1,4,5].

In particular, one of the challenges of FL is that the arrival of model parameters from multiple edge Artificial Intelligence Functions (AIFs) can suffer from a so strong desynchronization that these parameters can get no longer valuable, hence they could be discarded by the FL

* Corresponding author at: Orange, Châtillon, France.
E-mail addresses: nourelhouda.yellas@orange.com (N.-E.-H. Yellas), bernardetta.addis@loria.fr (B. Addis), selma.boumerdassi@cnam.fr (S. Boumerdassi), r.riggio@univpm.it (R. Riggio), stefano.secci@cnam.fr (S. Secci).

https://doi.org/10.1016/j.comnet.2024.110900
Received 19 February 2024; Received in revised form 3 November 2024; Accepted 4 November 2024
Available online 14 November 2024

server. These nodes suffering from such desynchronization are called stragglers. The main reason behind the appearance of stragglers is the heterogeneity of systems and data [6,7]; system heterogeneity is related to computing capacity (i.e., high computation delays) and network conditions (i.e., high communication delays). On the other hand, data heterogeneity refers to the case where data is not independent and not identically distributed (non-iid) among FL participants. Stragglers slow down the learning process as the aggregation task at the FL server is only triggered once all the local parameters are received, resulting in long convergence time: mitigating stragglers consists of reducing the overall training time to achieve a desired accuracy in a timely manner. To do so, different techniques can be adopted; for example, [7] classified the works in the literature to minimize the convergence time into three categories: (i) data distribution adjustment, to mitigate the non-iid data, (ii) model compression, to reduce the local training time and (iii) clients selection to determine the FL participant with high computation and communication performance. Another way to minimize the convergence time is by optimizing the local training time using hardware acceleration. In fact, the integration of AI modules in network components has been underway since less than a decade. Modern network nodes nowadays integrate Neural Processing Units (NPUs), ranging from mobile devices to core backbone equipment. The community often speculated that in the beginning of this trend, vendors did not know for which applications these units would be useful, but gambled on their future usefulness.

In this work, we present the IFLC (In-network Federated Learning Control) scheme, an adaptive scheme for the placement of anomaly detection AIFs in softwarized 5G environments. We consider a set of artificial intelligence functions at the edge of the network, deployed for instance as MEC applications in a MEC system. IFLC makes use of Hardware Accelerators (HWAs) in distributed in-network learning systems to compensate for end-to-end network and learning delays variations leading to stragglers. In a previous work, we defined a preliminary FL-AIF placement framework [8], using a mathematical programming approach. The proposed solution determines the optimal placement of FL server and clients while minimizing the number of deployed AIFs. Going beyond [8], in this paper we model the possibility of allocating hardware accelerators to reduce the training time, showing related different results and analysis. In this article, we go beyond the existing work, reformulating the model to control stragglers, and defining a refined end-to-end training latency model. Our contributions can be summarized as follows:

- We propose an algorithmic scheme, called IFLC, where we formulate a joint optimization problem for FL server and clients placement, and straggler minimization as a MILP (Mixed-Integer Linear Programming). The model aims to optimize the placement of FL clients and server based w.r.t a target end-to-end time which includes both the training and propagation time components. The MILP is modeled as a multi-objective problem where the main goal is to minimize the total cost expressed as the number of allocated CPU cores as well as the number of FL-clients stragglers resulting from high computational delays. The proposed model is able to minimize straggler occurrence in a computationally efficient way exploiting a proposed latency model;
- We present an original end-to-end learning latency model jointly considering learning and communication delays. We explore a deterministic scenario where the estimated end-to-end training time of the selected clients is within the imposed time limit, and stochastic scenarios where the selected FL clients may generate additional delays resulting in stragglers;
- We show how we can with IFLC (i) increase the local training time efficiency, (ii) minimize the occurrences of FL stragglers and (iii) reach desirable trade-offs between the number of active FL clients and the CPU utilization. An assessment of topology changes on the proposed model is also presented;

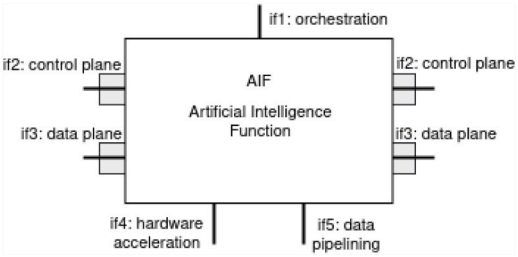


Fig. 1. AIF reference representation [8].

- Finally, we extend the proposed MILP formulation to accommodate more dynamic placements while covering single node failures; the extended formulation is added to the appendix.

The article is structured as follows. Sections 2 and 3 completes the background on AI integration in networks. Section 4 describes the IFLC scheme. We present the mathematical model in Section 5. Section 6 presents a polynomial-time resolution of the IFLC. Section 7 defines the numerical evaluation setting. The results analysis is in Section 8. Section 9 concludes the article.

2. Background

In this paper, we rely on the concept of AI Functions from the AI@EDGE H2020 European Project [9] to refer to end-to-end AI functions sub-components. Let us describe the reference AIF functional system, depicted in Fig. 1 where a set of interfaces are defined:

- if1: used by the orchestration platform for the communication with the AIF, including its configuration (e.g. for dynamic update of federated learning hyper-parameters) and the retrieval of inference results (e.g., inference running at the AIF server and/or edge AIFs);
- if2: control plane interface used for AI model parameter exchange among AIFs, e.g., the communication between edge AIFs and server AIF in federated learning;
- if3: data-plane interface used for data exchange among different AIFs, which may be used for generic distributed learning, in the case of an AIF forwarding graph;
- if4: hardWare acceleration interface used for I/O operations with HWA (e.g. GPU) for training and/or inference tasks;
- if5: for data collection and streaming, to interface with a data-pipe-lining system.

An AIF can run different types of AI applications with heterogeneous performance targets (e.g., latency requirements). Based on these requirements, the application can be centralized where one single AIF is responsible for training, or distributed where a set of AIFs collaborate to train a model (e.g., federated learning). We rely on a distributed AIF system making use of FL as depicted in Fig. 2: via if2, edge AIFs send local training results and obtain global training parameters back from the FL server AIF; if3 is unused in FL AIFs; if5 makes use of a data pipe-lining system for getting data for AIFs. Finally, the allocation of HWAs as NPU via if4 is meant to accelerate training and inference tasks, where inference could possibly be taking place at the edge AIF level besides the server one. Unlike conventional federated learning where the learning task is carried out by end devices [10], in our system we deploy the client AIFs at the edge servers (e.g., MEC hosts).

An example of in-network FL application is anomaly detection, used within network automation algorithmic loops. In this work, we use the application outlined in [11] where distributed AIFs make use of Long-Short-Term-Memory autoencoders against group of metrics related to network, storage, operating system features, to spot anomalous behavior. The goal in such application is to support automatic reconfiguration

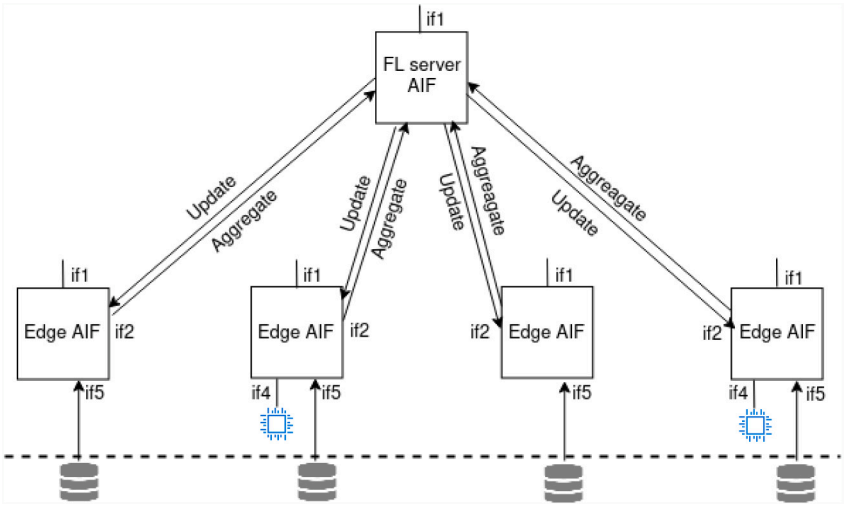


Fig. 2. FL-based AIF system [8].

of the infrastructure stack (e.g., rescaling, load-balancing, rerouting) based on the detected anomaly fingerprint.

3. Related works

Network softwarization technologies have increasingly influenced access networks, leading to a landscape where network functions are now mostly deployed as virtualized nodes. Additionally, hardware components for radio and computing systems have been re-designed to be re-programmable by external software, enabling dynamic resource allocation and sharing. This evolving environment led itself to the application of artificial intelligence, as it introduces numerous new decision making points and makes a wealth of monitoring data accessible to network and service management systems. In the following, we review recent works in the area of AI integration to networks, with a particular focus on federated learning applications.

3.1. In-network AI applications

In-network Artificial Intelligence and Machine Learning (AIML) integrates machine learning capabilities into network devices to enhance performance and efficiency [12,13], ranging from traffic engineering and performance optimization such as SLA (Service-Level Agreement) management [14,15] to network security [16,17].

In [18], authors investigate how AI and edge computing can interwork. Often, AIML is used in edge network resource allocation problems that make surface at different layers and for different resources, such as CPU, radio and link resources. In [19], the authors conduct a comprehensive survey on the usage of AIML solutions for edge computing, focusing on deep learning models. Different degrees of integration between AI and cloud/edge computing are identified, going from fully cloudified environments where AI training and inference models run on the remote cloud, to an all-on device setting where the tasks are carried out on the device.

This coupling between AIML and networking is being facilitated by edge computing and network virtualization. Standardization bodies are integrating AIML application requirements in system specifications. Namely, the Network and Data Analytics Function [20] has been proposed by 3GPP to support AIML in 5G core networks. However, various challenges are being discussed regarding different integration of training and inference sub-functions and the pipe-lining systems to get data to distributed AIFs. Another AIML application is anomaly detection and fault management, which consists in detecting abnormal network states, localizing the root cause and then proposing a remediation action to comeback to a normal working condition. In [21], the authors propose a centralized AIML framework making use of autoencoders to detect anomalies at different infrastructure levels; the ML model learns the normal state of a given system, then an anomalous state fingerprinting methodology is proposed for state qualification, and to guide a tailored remediation action.

Furthermore, [22] and [23] address the challenge of intrusion detection in network environments using federated learning while classifying incoming packet to the edge. [22] focuses on privacy preserving in-network traffic analysis and timely intrusion detection in IoT edge networks. In [23], authors consider programmable data plane switches to achieve high speed and scalable anomaly detection. Apart from focusing exclusively on the training task, our work differs from [22,23] by considering anomaly detection on infrastructure telemetry data, where monitoring data is split across selected FL clients.

3.2. Federated learning applications

A largely adopted strategy for geographically distributing AIFs down to network edges is Federated Learning [10]: it aims to prevent data collection aggregation at a central cloud, either for privacy issues, or for latency constraints, or even both, by collaboratively training ML models at edge nodes. Two main steps are to be considered: (i) the local training of the ML model at the FL clients and (ii) the global aggregation of the updated parameters at the FL server. The FL process, if adequately configured and designed, can grant higher efficiency in terms of network bandwidth consumption and latency, besides increasing privacy thanks to data locality. The FL process itself can be repeated with several learning rounds until the model achieves a target accuracy.

Mostly used for hand-held devices, FL is also being considered for in-network systems as well. In [4,5], the authors propose NWDAF services based on FL; each 5G core NF can have its own NWDAF instance (NWDAF leaf) collecting data from its corresponding NF, training the ML model locally and aggregate parameters at the FL-server (root) NWDAF.

In [11], the authors present how to use FL to distribute a centralized anomaly detection framework from [21]. The main goal is to cope with a set of challenges, mainly to scale with the increasing amounts of collected data and to reduce the training time to allow a near-real time re-orchestration decision.

3.3. Stragglers control

Several approaches are proposed to handle stragglers in distributed ML including careful client selection, enhancing training time efficiency and adaptive learning model update. For instance, [24] presents

a scalable solution FlexRR to mitigate stragglers in distributed ML. It relies on peer-to-peer communications among workers to detect stragglers and re-assign the work of the slowest worker to the most efficient one. Furthermore, several works consider straggler mitigation in in-network ML. For example, [25] uses Trio, a Juniper Network's programmable chipset, to mitigate stragglers in in-network distributed ML aggregation. Trio allows to process the incoming packets that contains the ML workers clients update in a non-pipelined manner using parallel threads. Trio timer threads are used to detect stragglers. Another work in [26] proposes an in-network aggregation solution to mitigate stragglers in distributed ML where the aggregation task is offloaded to programmable network switches. Efficient data structures are used to manage the synchronization barriers within the constraints of the programmable switches.

In the following, we present the existing works to mitigate stragglers in FL.

3.3.1. Client selection/placement

In federated learning, performance degradation of the learning process is highly related to client selection. Several works from the literature consider optimizing communication and computation latency. For instance, in [7] the authors consider both system and data heterogeneity to minimize the occurrence of stragglers where the client selection is done for each round. The selected clients should have near-iid data where more bandwidth is allocated for clients with low computing capacity or poor channel conditions. [27] and [28] consider dynamic client selection in hierarchical FL where both resource allocation and incentive mechanism are considered. The first step consists of an edge association task where each FL server offers rewards to FL participants to join its cluster. A second step considers choosing the model owner for each cluster. The authors consider the same processing capacity at the FL clients and allocate bandwidth resources (i.e. resource blocks) for a higher uplink bandwidth.

In [29], the authors aim to find a trade-off between energy consumption and the number of active clients by choosing less clients during the first rounds. The authors propose an estimation of both computing and propagation latency while considering the waiting time in the channel before sending the model parameters for aggregation. In [30], authors show how FL clients selection can impact the global FL model quality and reduce training time, in a strategic game-theoretic setting to select FL participants based on the computing resources they offer: the goal is to achieve a given accuracy in the global model in an edge environment. A similar work is presented in [31], where a multidimensional procurement auction for FL clients selection is used to enhance model accuracy using a lower number of rounds.

3.3.2. Training time efficiency

Another technique to minimize stragglers is to increase the local training efficiency. In [32], the authors propose a FL policy to improve the training efficiency while considering heterogeneous clients. Clients with similar computational capacities are selected for training during a given round. Moreover, HWAs can also be used to increase the learning time efficiency. More precisely, edge computing provides AI with a convenient platform for models training and inferring, with a potential solution on accelerating computations on hardware [33]; HWA can be made available pervasively in edge networks, starting from radio access and edge computing nodes. Besides reducing training and inference time depending on the type of accelerator [34], they can also decrease the energy footprint of AIML by up to 20 times [35,36].

Many works in the literature investigated on the possible usage of hardware acceleration with AIML models. For example, in [37], authors motivate the use of FPGA to accelerate deep neural network models. Indeed, they have evaluated the reduction in the computation time while comparing it to a software implementation with different numbers of threads. The authors only consider the acceleration of inference as for the considered use-case training is done off-line. Note that in some cases, the training task should also be accelerated as the model needs to be updated. For instance, if we consider real-time anomaly detection, the new state of the system should be learned after a short period of time. Another work is [38] where authors explore different acceleration designs for a neural network-based model where both GPU and FPGA were considered. The authors inspected different configurations with the aim of identifying the optimal scenario for each acceleration approach. Nevertheless, the exploitation of HWAs in distributed/federated learning client selection seems unexplored. In contrast to [37] and [38], in our work we examine the usage of HWA in a FL setting where the main goal is to compensate the variance in the end-to-end delays among FL clients.

3.3.3. Adaptive global model update

Another way to control stragglers is to adapt the global model update.

In [39], the authors propose a live gradient compensation method to avoid stragglers for distributed learning tasks: only the gradient update for the k fastest workers is used, while combining the results from the slowest worker in the next iteration: the main goal is to reduce the overall training time while producing a near convergence error close to fully synchronous gradient descent. Enlarging the view to network communications, the authors in [40] consider both communication bottleneck and straggler delays in large scale distributed learning tasks: they combine a coding approach with a bandwidth sizing strategy to avoid bottleneck hence reducing stragglers. An enhancement of the gradient coding is proposed in [41], where the data is assigned to the edge AIFs in a distributed manner so that a subset of model updates can be sufficient to compute the full gradient at the server side; then a dynamic clustering schema is associated to the set of edge AIFs to improve the completion time. A hierarchical FL mechanism that encompasses both synchronous and asynchronous training schemes is proposed in [42] to mitigate the straggling effect. [43] addresses the challenges posed by both stragglers and adversaries in FL systems. It proposes a selective approach to only aggregate updates that are trustworthy where devices are grouped based on the arrival delay of their updates. Note that in this work, we do not consider the global model update algorithm for mitigating the stragglers.

Another technique to mitigate stragglers could be to rely on asynchronous FL [44] where the updates from all the FL clients are not required to trigger the aggregation task. However, our work considers anomaly detection use case using time series data, employing models like LSTM, which require training on the most recent data, and retraining. In this context, asynchronous FL poses a challenge as the inherent nature of asynchronous updates could lead to models trained on out-of-date data, which results in biased models that do not reflect the current state of the system. As there is already a bias due to non-IID data, using asynchronous federated learning is not advisable. Nevertheless, further work could concentrate on comparison between synchronous and asynchronous federated learning to numerically assess this aspect.

3.4. Our contribution

In our work, we aim at going beyond adaptive FL client section, while including the combined control of both network and training delays. In fact, we aim at compensating large deviations in parameters arrival from edge AIFs to the FL server by allocating/releasing HWA to reduce/increase the edge AIF training delay. We propose an approach to control stragglers in in-network federated learning that combines both FL-AIF selection/placement and HWA allocation. We consider adaptive HWA usage to increase the local training efficiency at the FL clients with the goal to minimize variance in the end-to-end combined network and training latency, as defined hereafter. We do not address the combined usage of adaptive global model update, edge AIF placement and HWAs, left for future work.

In Table 1, we present the positioning of our work contribution to existing works from the literature.

Table 1
Research contribution positioning.

Work	Solution	Environment	Technique	Mitigation
[24]	reactive	datacenter	peer-to-peer communication, reassignment	out-of-network
[43]	reactive	FL/geo-distributed	staneless awareness	out-of-network
[25], [26]	reactive	distributed ML/geo-distributed	offloading to programmable switch, aggregation efficiency	in-network
This work	pro-active	FL/geo-distributed	training time and client placement efficiency	in-network

Table 2
Notations.

Sets and parameters	
N	set of edge physical servers, with $n = \|N\|$.
N_s	set of physical servers for FL server placement.
c_i	number of available CPU cores on node i.
p_{ik}	training time for $i \in N$ with k active edge AIFs.
τ	target distributed learning time (one FL round).
d_{ij}	communication latency between node i and node j.
α_{ik}	≥ 1, acceleration factor at node i with k active edge AIFs.
h_i	assumes value 1 if a HWA is available on node i.
H	maximum number of available HWAs.
S	set of delay scenarios.
q_s	probability of scenario s.
β_{ik}^s	drift of the learning time on scenario s on node i with k active edge AIFs.
η_{ij}^s	drift of the propagation delay on scenario s from node i to node j.
Δ	maximum tolerated end-to-end delay.

4. In-network federated learning control

We refer to our framework as In-network Federated Learning Control (IFLC) to express the fact that we aim at controlling the latency phenomena arising in in-network FL applications subjected to stringent training model update targets, by placing the FL-AIF clients and server to ensure the distributed learning tasks within a time threshold[1].

In in-network federated learning, we assume that data may both be generated locally in the machine hosting the FL client, and be generated elsewhere in the network and delivered to the FL clients by a data pipelining system [45].

The notations used are summarized in Table 2.

We consider a FL-based AIF system (Fig. 2) composed of a set N of physical edge servers with heterogeneous CPU resources. These edge servers are considered as possible locations for running AIFs.[2] Let c_i be the number of available CPU cores on edge node i. We denote by N_s the set of physical servers that can be used for the AIF server placement. It is worth mentioning that $N_s \equiv N$ if the FL server AIF can be installed on the edge servers and $N_s \cap N = \emptyset$ otherwise. Additionally, we consider a set of FL client AIFs that can be deployed on top of each physical node to run a given FL-based application. An AIF receives data streams from external nodes, triggering the training task. We consider that an AIF consumes all CPU resources that are made available to it on the physical node.[3]

We suppose that each physical node can be equipped with hardware acceleration to increase the local training efficiency. We denote by α_{ik} the acceleration factor at node i when k FL client AIFs are active. The total number of available HWAs is limited by a constant H. In the following, we present the modeling of the end-to-end learning time.

4.1. End-to-end training time modeling

The global training time needed to update the global AI model at the AIF server is therefore related to the training time, the propagation delays and the number of active edge AIFs.[4] Fig. 3 depicts the training time model components.

Definition 1 (*Local Training Time - p*)**.** Let p_{ik} be the local training time of an AIF on node i when k AIFs are active.

We rely on the study carried out in [8] showing that the training time increases with the data size. The scenario considered here is the one where data is evenly split across the FL clients.

This parameter depends on (i) the number of available CPU cores at the physical layer, (ii) the enabling of HWAs, and (iii) the amount of training data. More precisely, the training time is directly related to the data size, e.g., the training time in neural networks is evaluated by the number of floating point operations which depends on the data size and the architecture of the neural network [46]. Consequently, the local training time reduces with the number of active AIFs as data is distributed amongst them. We define p as an upper bound for the overall training time during a given round.

Definition 2 (*Propagation Delay - d*)**.** Let d_{ij} be the communication latency on the link that interconnects nodes i and j.

Transmission delays are negligible due to small volume of data exchanged (if2, Fig. 2), could also be incorporated in d.

Definition 3 (*End-to-End (E2E) Training Time - χ*)**.** Let χ_{ik} be the sum of the local training time at node i (while k AIFs are active) and the propagation delay between the nodes deploying this client AIF and the FL server AIF j.

As in real world scenarios stochastic delays may apply on end-to-end training times, we consider two different behaviors for an AIF: a deterministic behavior where the local training time and the propagation delay are the same as expected, and a stochastic behavior where additional delays may apply to the local training time, to the propagation delay or both.

Definition 4 (*Training Time Drift - β*)**.** Let β be the stochastic delays applied to the local training time.

This parameter is supposed to be unknown, even if it can be empirically characterized from real systems. Furthermore, we define a set S of possible *scenarios* that define the intensity of the stochastic delays. In such a way, for each scenario s we have a realization of the training time drift β: β_{ik}^s for each node i and k deployed AIF.

[1] It is worth noting that placing an AIF can mean copying a function image to a physical node or selecting a pre-fetched AIF already installed in the physical node, so that its actual instantiation can be a near-real-time operation in a similar time-scale to that of near-real-time execution algorithms.

[2] Note that in this work, only system heterogeneity is considered. We do not consider additional delays related to non-iid data and the time needed to receive data at the edge AIFs that is considered negligible as well.

[3] This corresponds to the default behavior of container-based services.

[4] FL client AIF and FL edge AIF are used interchangeably.

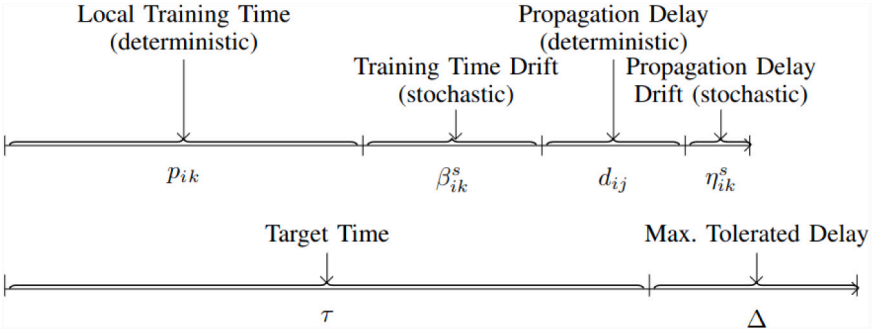


Fig. 3. E2E training time (χ) model components and threshold.

Definition 5 (*Propagation Delay Drift - η*)**.** Let η be an additional random value applied to the propagation delay (e.g., it can be traffic dependant following a given queuing mode or traffic independent where it is influenced by the link length).

For each scenario s, a drift is applied to the propagation delay η: η^s_{ij} for each pair of edges i and j.

Definition 6 (*Target Time - τ*)**.** Let τ be the target learning time after which aggregation is triggered at the AIF server.

Definition 7 (*Maximum Tolerated Delay - Δ*)**.** Let Δ be the tolerated elapsed training delay between the last accepted reception of the parameters from edge AIFs, and the effective start of the parameters aggregation at the FL server.

Definition 8 (*AIF Straggler or Straggling AIF*)**.** An AIF straggler or straggling AIF i is a client AIF whose end-to-end training time χ is greater than the threshold but remains within the tolerated additional time Δ (i.e., $\tau \leq\leq \tau + \Delta$).

Note: parameters sent by an AIF straggler are aggregated by the AIF server if $\chi \leq \tau + \Delta$. The values of τ and Δ depends on the use case specification requirements, with $\Delta < \tau$. Additionally, we only consider stragglers with a delay that does not exceed Δ. The stragglers with a higher delay are not considered as they cannot be chosen by IFLC model (i.e., unfeasible solution). Also, the proposed model only covers the case where straggling effect is related to training delays, this justifies the fact that we propose the allocation of HWA to mitigate the stragglers.

Additionally, we consider two possible settings for the FL AIFs:

- 'edge–edge': both server and edge AIFs are running at the edge nodes: the propagation delays in that case can be expected to be negligible as compared to the training time, considering the possible duplication of the FL server instance close to the clients.
- 'core–edge': the server is placed at a core location (beyond aggregation nodes) while edge AIFs are placed at the edges: the propagation delays for this setting can be expected to be higher than the previous ones (i.e., edge–edge setting).

4.2. Problem statement

Under the delay model, we can define the IFLC problem as follows. Given a set of physical nodes N and a defined target time for a specific application, find an optimal placement of the FL server AIF and the FL client AIFs to ensure that:

- the maximum E2E training time does not exceed the time requirements imposed by the application,
- hardware acceleration, if available, can be allocated to reduce the local training time, along with CPU resources,
- the CPU utilization is minimized,
- the average number of AIFs exceeding the target time is minimized, including the stochastic scenarios (see definitions 4 and 5).

Indirectly, the number of active AIFs is pushed down towards optimality.

5. Mathematical modeling

In the following, we present the mathematical programming model that corresponds to the IFLC scheme.

We use binary variables to represent AIF placement and HWA related decisions. x_i represents the activation state of the AIF on node i, hence it takes value 1 if node i is used to deploy an AIF. y_j provides the position of the FL server AIF, if it is equal to 1 then the FL server is placed on node j. If an edge AIF is installed on node i and the FL server is placed on node j then ξ_{ij} is equal to 1. Note that the FL server cannot be installed on the same node as a client AIF. ζ_{ik} and z_k are used to count the total number of active AIF. ψ_{ik} is equal to 1 if the hardware accelerator is present and used on node i and k AIFs are active.

Real variables are introduced to model the different components of the training time. t_i represents the local training time when an AIF is active on node i, δ_{ik} represents the amount of reduction in the local training time due to hardware acceleration, when k AIFs are active on node i and χ_i represents the E2E training time of the client AIF deployed on node i. Finally, we introduce the real variables $\tilde{t_i}^s$ and $\tilde{\delta_{ik}}^s$ which correspond to the stochastic local training time and the reduction in the stochastic local training time, respectively, for scenario s. $\tilde{\chi_i}^s$ is the E2E training time of the client AIF deployed on node i with scenario s.

The mathematical notations are summarized in Table 3.

5.1. Core model constraints

5.1.1. FL clients and FL server AIFs placement

We need to determine the location of the FL server AIF and the number and location of the client AIFs in order to guarantee that each AIF can train and send the model parameters to the FL server on time.

Constraint (1) imposes that the FL server AIF is installed on one and only one node. (2) impose that the node that hosts the AIF server cannot host an edge client AIF. We recall that variable ξ_{ij} is used to represent the fact that a FL client AIF is installed on node i and the FL server is installed on node j. Therefore, when $\xi_{ij} = 1$ the AIF installed on node i yields a communication latency of d_{ij}. Constraints (3) and (4) are consistency constraints. If the server is not installed on node j then all the variables ξ_{ij} must be equal to zero. For a given node i, one and only one variable ξ_{ij} can assume value 1 if a client AIF is installed on node i, otherwise they are all equal to zero.

$$\sum_{j \in N_s} y_j = 1 \tag{1}$$

$$y_i + x_i \leq 1 \qquad \forall i \in N \tag{2}$$

$$\xi_{ij} \leq y_j \qquad \forall i \in N, j \in N_s \tag{3}$$

$$\sum_{j \in N_s} \xi_{ij} = x_i \qquad \forall i \in N \tag{4}$$

Table 3
Mathematical notations.

Binary variables	
x_i	1, if a client AIF is active on node i.
y_j	1, if the FL server AIF is installed on node j.
w_i	1, if the hardware accelerator is allocated on node i.
ζ_{ik}	1, if an AIF is active on node i with k deployed AIFs.
ξ_{ij}	1, if a client AIF is installed on node i and the FL server AIF is installed on node j.
σ_i^s	1 if a client AIF on node i is a straggler in scenario s.
ψ_{ik}	1, if a hardware accelerator is present and used on node i and k AIFs are active.
z_k	1, if k AIFs are active.
Continuous variables	
t_i	distributed training time on node i.
χ_i	E2E training time of the client AIF deployed on node i when k AIFs are active.
δ_{ik}	time reduction at node i when k AIFs are active.
$\tilde{t}_i^s$	stochastic distributed training time on node i with scenario s.
$\tilde{\chi}_i^s$	stochastic E2E training time of the client AIF employed on node i when k AIFs are active with scenario s.
$\tilde{\delta}_{ik}^s$	time reduction at node i when k AIFs are active with scenario s.

5.1.2. Training time characterization

Constraints (5) allow to calculate the deterministic training time of node i when k AIFs are deployed.

Constraints (6) and (7) are consistency constraints, that is, when variables z_k or x_i assume value zero, the corresponding ζ_{ik}, t_i variables for node i assume value zero.

Constraints (8) together with constraints (9) allow us to count the number of deployed AIFs.

$$t_i = \sum_{k=2}^{n} p_{ik}\zeta_{ik} \qquad \forall i \in N \tag{5}$$

$$\sum_{k=2}^{n} \zeta_{ik} = x_i \qquad \forall i \in N \tag{6}$$

$$\sum_{i \in N} \zeta_{ik} \leq |N| z_k \qquad \forall k \in 2..n \tag{7}$$

$$\sum_{k=2}^{n} z_k = 1 \tag{8}$$

$$\sum_{k=2}^{n} k z_k = \sum_{i \in N} x_i \tag{9}$$

5.1.3. Hardware acceleration for deterministic training

We allow the use of hardware accelerators to reduce the training time. We introduce the necessary constraints to evaluate the impact of the hardware accelerators on the local training time. Constraints (10) impose that the hardware accelerator can be used only if available. Constraint (11) imposes that a maximum number H of hardware accelerators can be used.

$$w_i \leq h_i \qquad \forall i \in N \tag{10}$$

$$\sum_{i \in N} w_i \leq H \tag{11}$$

Constraints (12)–(15) allow to evaluate the gain in the deterministic local training time obtained using hardware acceleration while associating the two variables δ_{ik} and w_i to keep consistency. That is, if ζ_{ik} =1 and $\psi_{ik} = 1$, then $\delta_{ik} = \left(1 - \frac{1}{\alpha_{ik}}\right) p_{ik}$. Otherwise, $\delta_{ik} = 0$.

$$\psi_{ik} \leq \frac{\zeta_{ik} + w_i}{2} \qquad \forall i \in N, k \in 2..n \tag{12}$$

$$\delta_{ik} \leq \psi_{ik} \left(1 - \frac{1}{\alpha_{ik}}\right) p \qquad \forall i \in N, k \in 2..n \tag{13}$$

$$\delta_{ik} \leq \left(1 - \frac{1}{\alpha_{ik}}\right) (p_{ik}\zeta_{ik}) \qquad \forall i \in N, k \in 2..n \tag{14}$$

$$\delta_{ik} \geq \left(1 - \frac{1}{\alpha_{ik}}\right) (p_{ik}\zeta_{ik}) - \quad (1 - \psi_{ik})p \qquad \forall i \in N, k \in 2..n \tag{15}$$

Note that p represents the maximum deterministic training time and can be calculated as follows:

$$p = \max_{k=2..n, i \in N} p_{ik} \tag{16}$$

5.1.4. Target time

For each node i, constraints (17) compute the E2E training time of an active AIF on node i. Constraints (18) ensure that the maximum E2E training time that an active AIF can achieve does not exceed the accepted target time τ. These constraints are always valid even when no AIF is installed. In fact, variables t_i and δ_{ik} assume value zero when $x_i = 0$ (see, constraints (4), (5)–(6), and (14)–(15)).

$$\chi_i = t_i - \sum_{k \in 2..n} \delta_{ik} + \sum_{j \in N_s} d_{ij}\xi_{ij} \qquad \forall i \in N \tag{17}$$

$$\chi_i \leq \tau \qquad \forall i \in N \tag{18}$$

5.2. Stochastic variant

In the following, we introduce the stochastic variant of the AIF placement model. The goal is to introduce robustness against different realization scenarios. We add the following constraints to the aforementioned model.

Constraints (19) calculate the stochastic local training time for node i when k AIFs are active and a delay β_{ik}^s is applied.

$$\tilde{t}_i^s = \sum_{k=2}^{n} (p_{ik} + \beta_{ik}^s)\zeta_{ik} \qquad \forall i \in N, s \in S \tag{19}$$

In the same way as in the deterministic model, constraints (20)–(22) allow to evaluate the gain in local training time obtained using hardware acceleration while considering the additional delays applied to the training time.

$$\tilde{\delta}_{ik}^s \leq \psi_{ik} \left(1 - \frac{1}{\alpha_{ik}}\right) \tilde{p} \qquad \forall i \in N, k \in 2..n, \forall s \in S \tag{20}$$

$$\tilde{\delta}_{ik}^s \leq \left(1 - \frac{1}{\alpha_{ik}}\right) (p_{ik} + \beta_{ik}^s)\zeta_{ik} \qquad \forall i \in N, k \in 2..n, s \in S \tag{21}$$

$$\tilde{\delta}_{ik}^s \geq \left(1 - \frac{1}{\alpha_{ik}}\right) (p_{ik} + \beta_{ik}^s)\zeta_{ik} - (1 - \psi_{ik})\tilde{p} \qquad \forall i \in N, k \in 2..n, \forall s \in S \tag{22}$$

Note that $\tilde{p}$ represents the maximum stochastic training time and can be calculated as follows:

$$\tilde{p} = \max_{k=2..n, i \in N, s \in S} (p_{ik} + \beta_{ik}^s) \tag{23}$$

Finally, constraints (24) compute the E2E stochastic time of node i and constraints (25) impose that the E2E training time of an active AIF, including the additional delays applied to both the training and propagation times, are below the threshold. It is worth noticing that in

this case we accept a maximal response time $\tau + \Delta$. Each AIF with a training time exceeding τ is considered a straggler (see variable σ_i^s).

These constraints are always valid even when no AIF is installed where $\widetilde{t}_i^s$ and $\widetilde{\delta}_{ik}^s$ assume value zero when $x_i = 0$ (see, constraints (4), (5)–(6), and (21)–(22)).

$$\widetilde{\chi}_i^s = \widetilde{t}_i^s - \sum_{k=2}^{n} \widetilde{\delta}_{ik}^s + \sum_{j \in A} (d_{ij} + \eta_{ij}^s)\xi_{ij} \qquad \forall i \in N, s \in S \tag{24}$$

$$\widetilde{\chi}_i^s \leq \tau + \Delta\sigma_i^s \qquad \forall i \in N, s \in S \tag{25}$$

Additionally, we introduce the following domain constraints to complete the model:

$$t_i \geq 0 \qquad \forall i \in N \tag{26}$$

$$\widetilde{t}_i^s \geq 0 \qquad \forall i \in N, s \in S \tag{27}$$

$$\delta_{ik} \geq 0 \qquad \forall i \in N, k \in 2..n \tag{28}$$

$$\widetilde{\delta}_{ik}^s \geq 0 \qquad \forall i \in N, k \in 2..n, s \in S \tag{29}$$

$$x_i, w_i, y_j, \xi_{ij} \in \{0, 1\} \qquad \forall i \in N, j \in N_s \tag{30}$$

$$z_k \in Z \qquad \forall k \in 2..n \tag{31}$$

$$\zeta_{ik}, \in \{0, 1\} \qquad \forall i \in N, k \in 2..n \tag{32}$$

5.3. Objectives

The goal is to minimize the number of AIFs that can be in a straggling situation in order to guarantee a certain level of performance of the learning process with minimum costs. To this end, we introduce two objectives. We first minimize a measure of the stragglers in the system and then, we search for solutions with the minimal utilization of resources.

5.3.1. Minimizing the number of stragglers

The objective in (33) allow to minimize the expected number of stragglers where q_s is the probability of each scenario $s \in S$.

$$\min \sum_{s \in S} q_s \sum_{i \in N} \sigma_i^s \tag{33}$$

5.3.2. Minimizing the number of computational resources

After determining the optimal solution of the previous problem, we can further minimize the number of computational resources while limiting the increase of the expected number of stragglers. We denote by $\sigma^\star$ the minimum expected number of AIF in a straggling situation obtained by minimizing (33). Thus, the objective of the second phase optimization problem is:

$$\min \sum_{i \in N} c_i x_i \tag{34}$$

We introduce the additional constraints (35) to limit the increase of the number of stragglers.

$$\sum_{s \in S} q_s \sum_{i \in N} \sigma_i^s \leq \sigma^\star + \epsilon \tag{35}$$

Note that the objective in (33) along with constraints (35) are only used by the stochastic variant, as we impose the absence of stragglers in the deterministic scenario.

6. Polynomial-time resolution algorithm

We propose two polynomial-time resolution algorithms for solving the AIF placement problem for both the deterministic and stochastic cases, hence supporting near-real-time orchestration of FL-AIFs. They share a common structure based on the following observations:

- the number of possible locations of the FL server AIF is given by $|N_s|$ (or $|N|$ for the edge–edge setting),
- the number of installed AIFs is limited by $|N|$ (or $|N| - 1$ for the edge–edge setting),
- for each server location and given number of installed AIFs, χ can be pre-calculated.

The proposed algorithms search for the best solution for each given combination (j, k) of AIF server location (j) and number of installed edge AIFs k and keep the best one, see Algorithm 1. In the edge–edge setting, the server is chosen from the set N and its location is removed from the available AIFs location. The decision of activating an AIF is done working on the set of locations ordered by increasing c_i (i.e., number of CPU cores), breaking ties using χ (smallest first). Note that differently from the mathematical model presented in the previous section, this model considers that the end-to-end training time component χ as a parameter, computed for each combination (j, k) of AIF server location (j) and number of installed edge AIFs k. The two algorithms differ in the implementation of the function "BEST_PLACEMENT()".

Algorithm 1 General IFLC Scheme

output: $S^\star$: set of active edge AIFs, j: server AIF position

```
best_cost = ∞
S* = ∅
for j ∈ N_s do
    for k ∈ 1..n do
        Ñ ← available nodes in decreasing order of c_i
        (feasible, S, cost) = BEST_PLACEMENT(k, j, Ñ)
        if cost ≤ best_cost and feasible then
            bestcost = cost
            FL_server = j
            S* = S
```

return $(S^\star, FL\_server)$

To allow a compact representation of the two placement procedures, we report here the calculation of χ for a given couple (j, k). In the deterministic case, the E2E training time for a given node i without HWA can be calculated as:

$$\chi_i = p_{ik} + d_{ij} \tag{36}$$

and, when the hardware accelerator is allocated as:

$$\chi_i^{ha} = p_{ik} - \delta_{ik} + d_{ij} \tag{37}$$

Where δ_{ik} is the time reduction at node i when k edge AIFs are active. When HWA is used, δ_{ik} is computed by dividing the local training time by a given accelerator factor α. The value of α depends on the number of CPU cores available on the node and the number of active AIFs (i.e., data size).

For the stochastic case, for a given node i and scenario $s \in S$ it is, respectively:

$$\chi_i^s = \left\{ (p_{ik} + \beta_{ik}^s) + (d_{ij} + \eta_{ij}^s) \right\} \tag{38}$$

$$\chi_i^{s,ha} = \left\{ (p_{ik} + \beta_{ik}^s) - \delta_{ik} + (d_{ij} + \eta_{ij}^s) \right\} \tag{39}$$

Further, we can calculate the E2E training times for the worst case scenario ($\overline{\chi}$), both without and with HWA:

$$\overline{\chi_i} = \max_{s \in S} \chi_i^s \tag{40}$$

and

$$\overline{\chi_i^{ha}} = \max_{s \in S} \chi_i^{s,ha} \tag{41}$$

In the deterministic case, the BEST_PLACEMENT procedure inspects the list of ordered nodes $\tilde{N}$ and checks whether is possible to place an edge AIF without exceeding the threshold. HWAs are allocate (if available) only if necessary to reduce the training time under the threshold τ.

BEST_PLACEMENT for the deterministic case is presented in Algorithm 2. When no HWA is available, the parameter H assumes value zero. For ease of presentation, we do not explicitly add the line that save the location of the HWA (but it is saved by the implemented procedure).

Algorithm 2 Best placement - deterministic

input : j: server location, k number of active edge AIFs, $\tilde{N}$ set of available nodes for locating the AIFs
output: feasible: bool, S: set of active edge AIFs, cost: solution cost

kcount = 0, cost = 0, hwa = 0, $S = \emptyset$
Calculate deterministic E2E training times
/* try to locate k AIFs */
while $\tilde{N} \neq \emptyset$ and kcount $< k$ **do**
  i=pop($\tilde{N}$)
  if $\chi_i \leq \tau$ **then**
    kcount ++, cost += c_i, $S = S \cup \{i\}$
  else if $\chi_i^{ha} \leq \tau$ and hwa < H **then**
    kcount ++, cost += c_i, hwa ++, $S = S \cup \{i\}$
if kcount == k **then**
  return (True, S, cost)
else
  return (False, $\emptyset$, ∞)

We can observe that at each step of the function presented in Algorithm 2, the less expensive AIF location for which the E2E training time is below the threshold is selected. Thus providing an optimal solution for k, if it exists. If at the end of the process, the number of selected edge AIFs is lower than k, it means that k AIFs do not allow a viable E2E training time (with the number of available HWAs).

In the stochastic case, the objective function depends on the probability of producing stragglers for each given scenario. Let us consider again a given couple j, k of server AIF location and number of edge AIFs, and analyze the impact of placing an edge AIF on node i.

When placing an edge AIF on node i where it produces a worst case E2E time (the longest time among the scenarios) that is below the threshold τ, the contribution to the objective function is zero. When the worst E2E time is in the interval $[\tau, \tau + \Delta]$ the contribution depends on the E2E time of each scenario. The total contribution of an AIF located on node i can be calculated as follows:

$$Q_i = \sum_{s \in S : \tau < \chi_i^s \leq \tau + \Delta} q_i^s \tag{42}$$

where q_i^s is the probability of the scenario s on node i and χ_i^s is the E2E time for scenario s on node i. Similarly, we can calculate the contribution to the objective function when the node i is equipped with HWA:

$$Q_i^{ha} = \sum_{s \in S : \tau < \chi_i^{s,ha} \leq \tau + \Delta} q_i^s \tag{43}$$

where $\chi_i^{s,ha}$ is for the scenario s on node i with HWA.

In Algorithm 3, we report the structure of the function 'BEST_PLACEMENT()' in the case of stochastic scenarios. It is worth to notice that the first part of the procedure has the same structure of the deterministic case, but the worst case E2E time is used at the place of the deterministic times. A 'recovery step' is added to re-discuss the HWA positions to improve the solution in term of average number of stragglers. It is worth to notice that a HWA can be released on an already selected node i if its E2E time without HWA is below the rejection threshold $\tau + \Delta$ paying the price Q_i.

Time and space complexity

Both algorithms repeat the procedure "BEST_PLACEMENT()" for each couple (j, k) of FL server location and number of instantiated AIFs,

Algorithm 3 Best placement - stochastic

input : j: server location, k number of active edge AIFs, $\tilde{N}$ set of available nodes for locating the AIFs
output: feasible: bool, S: set of active edge AIFs, cost: solution cost

Calculate stochastic E2E training times
kcount = 0, cost = 0, hwa = 0, $S = \emptyset$ $\tilde{N}_{bck} = \tilde{N}$
while $\tilde{N} \neq \emptyset$ and kcount $< k$ **do**
  i=pop($\tilde{N}$)
  if $\chi_i \leq \tau$ **then**
    kcount ++, cost+=c_i, $S = S \cup \{i\}$
  else if $\chi_i^{ha} \leq \tau$ and hwa < H **then**
    kcount++, cost+=c_i, hwa++, $S = S \cup \{i\}$
if kcount == k **then**
  return (True, S, cost)
else
  /* add straggling AIFs */
  $\tilde{N} = \tilde{N}_{bck} \setminus S$, sort by increasing Q_i
  while $\tilde{N} \neq \emptyset$ and kcount $< k$ **do**
    i=pop($\tilde{N}$)
    if $\chi_i \leq \tau + \Delta$ **then**
      kcount++, cost+=c_i, $S = S \cup \{i\}$
  /* ''recovery step'': try moving hwa */
  if kcount $< k$ **then**
    L1 = $\left\{ i \in \tilde{N}_{bck} \setminus S : \overline{\chi_i^{ha}} \leq \tau + \Delta \right\}$
    L2 = $\left\{ i \in S : \overline{\chi_i^{ha}} \leq \tau \text{ and } \overline{\chi_i} > \tau \right\}$
    sort L1 by increasing Q_i^{ha}, sort L2 by increasing Q_i **while** $\tilde{L1} \neq \emptyset$ and $\tilde{L2} \neq \emptyset$ and kcount $< k$ **do**
      i=pop($\tilde{L1}$), l=pop($\tilde{L2}$)
      $S = S \cup \{i\}$
  if kcount == k **then**
    repeat similar "recovery step" to try improving cost
    exchanges are allowed only between already allocated edge AIFs
    return (True, S, cost)
  else
    return (False, $\emptyset$, ∞)

i.e. $|N||N_c|$ iterations. Both placement procedures (deterministic and stochastic) perform a sorting of the AIF locations and an inspection of the resulting list. The stochastic version has additional steps where it performs two sort procedures (of the residual list of nodes) and a linear comparison of two list of at size at most $|N|$. Thus, the overall time complexity for both algorithms is of the order of $O(|N|^2|N_c|\log(|N|))$, $\approx O(n^3 \log(n))$, where $n = |N|$ and under the assumption that $|N_c| = O(n)$. In terms of space complexity, IFLC algorithms store a given number of lists of size at most $|N|$, thus with a size $\approx O(n \log(n))$.

6.1. FIRST-FIT algorithm

As a lowest-complexity benchmark, we introduce a baseline placement algorithm to compare with IFLC strategies. It is a first-fit algorithm, of $\approx O(n \log(n))$ time and space complexity, that prioritizes the nodes with the highest CPU resources where it increases the number of deployed AIFs until there is no more decreasing in the E2E training time. We chose the FIRST-FIT algorithm, a commonly used heuristic for placement problems to assess the trade-off between efficiency and optimality against the MILP solution.

FIRST-FIT is given in Algorithm 4.

Algorithm 4 FIRST-FIT algorithm

output: S: set of active edge AIF, j: server AIF position

$\tilde{N} \leftarrow$ sort N in decreasing order of CPU resources
$S \leftarrow 0$
$E2E\_time = 0$
$E2E\_time\_final = \infty$
$k = 0$, *number of AIFs*
$j = random(\tilde{N})$, $\tilde{N} = \tilde{N} \setminus \{j\}$
while $\tilde{N} \neq \emptyset$ **do**
  i=pop($\tilde{N}$)
  $S = S \cup i$, $k+ = 1$
  Update($E2E\_time$, k)
  if $E2E\_time\_final \leq E2E\_time$ **then**
    $S = S - \{i\}$, $k- = 1$
    return (S)
  Update($E2E\_time\_final$, k)
return (S, j)

7. Simulated instances

In the following, we report how we set-up the numerical evaluation environment, including simulated scenarios and the dataset we used.

7.1. AIF application

In order to evaluate the IFLC strategies, we use the FL-based framework proposed in [11]. We run a set of AIFs as docker containers on a Kubernetes infrastructure where each AIF is an implementation of an LSTM autoencoder neural network system to detect anomalies in a 5G stack. The goal of this framework is to detect anomalies at different system levels, i.e., physical level, virtual/container level and access level, using thousands of time-series issued by probes from network functions, physical servers, Eth/IP and radio links. Probes are collected from a 5G testbed replaying traffic traces of a European operator, from the ANR COCO5G project (https://coco5~g.roc.cnam.fr), in the Lozere region in France for 3 months in 2019.

The data collected from the probes of the 5G3E dataset from [47] provides few dozens of feature time-series for each resource group, where groups are related to CPU, RAM, storage and link states. We use data batches of 4000 samples to train the aforementioned AIML model, assumed to be the retraining time of the system and could vary in general depending on the sampling rate. The batch size is set to the data size, hence considering all the samples. The data is then evenly load-balanced as a function of the number of edge AIFs employed.

Note that the dataset we used to evaluate the FL framework has non-iid samples because data points are (i) not independent since the servers may have potential correlations as they belong to the same Kubernetes infrastructure, (ii) non identically distributed where each server may have its own distribution and (iii) non stationary as this type of data shows seasonality when server workloads change. Having non-IID data is another motivation to avoid stragglers as much as possible. To compensate for the non-IID nature of the data, the reference AIF model groups metrics in different autoencoder groups.

We use the FL-based anomaly detection in [11] where FedAvg is used as the aggregation algorithm. The number of epochs (E) is 10 and the model is trained for one round ($R = 1$). The rest of the hyper-parameters are set as explained in [11].

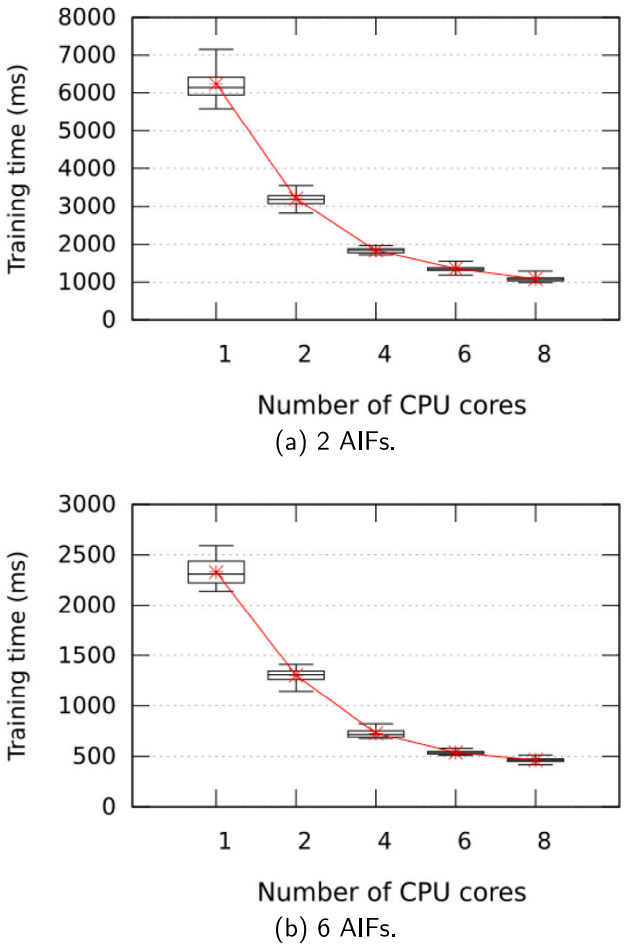


Fig. 4. Training time distribution as a function of the number of active client AIFs and available CPU cores. $R = 1$, $E = 10$.

7.2. Computation of training and propagation time samples

We generate the training time samples as a function of the number of client AIFs and the amount of computation resources using the aforementioned AIF application. Fig. 4 depicts the distribution of the maximum local training time for different numbers of edge AIFs and different numbers of available CPU cores. We can remark that the training time decreases with the number of AIFs and the available CPU cores up to a certain threshold.

In contrast to conventional user-device FL-based services, this framework considers an in-network service where the time scale at which the anomaly detection model is expected to react is on the order of few seconds, or even sub-second. Nonetheless, it is worth mentioning that our IFLC scheme can be applied on any FL application.

For the purpose of evaluating IFLC on large instances, we generate a synthetic set of pseudo-random training times that approximate a pre-specified correlation coefficient between the training time values, the number of active AIFs and the number of available CPU cores. This correlation coefficient is retrieved from the original training time samples. We make available the samples and related scripts for the simulations in [48]. Note that the generated dataset contains training times for different number of active AIFs and available CPU cores.

Fig. 5 depicts the distribution of the training times of both the original and synthetic datasets, based on the total number of CPU cores (i.e., number of CPU cores that are used by all active AIFs).

As a function of the AIF positioning setting, we configure the E2E training time components as follows:

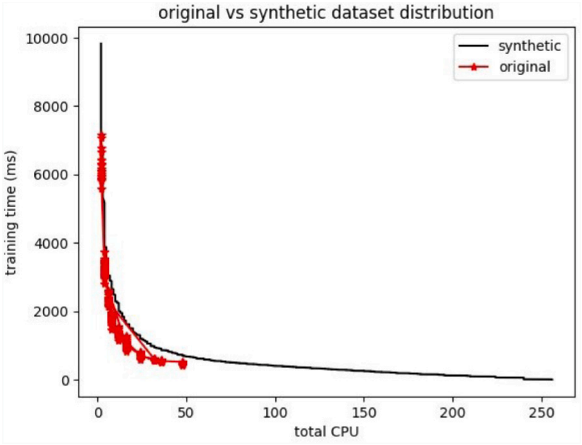


Fig. 5. Training time distribution vs the number of CPU cores.

Table 4
Training time and propagation delays options.

Cases	Training time	Propagation delay
case D–D	deterministic	deterministic
case D–S	deterministic	stochastic
case S–S	stochastic	stochastic

- 'edge–edge' setting: we define the maximum one-way latency between the furthest edge AIF and the AIF server as the 25% quantile value of the training time during one epoch divided by 10.
- 'core–edge' setting: the highest value of the one-way latency between the furthest FL client AIF and the FL server is set equal to the mean value of the training times during 10 epochs.

We consider a combination of deterministic and stochastic behaviors for both propagation and training times as shown in Table 4. Note that 'S–D' case is not cited because it generates the same solutions as 'S–S'. This can be explained by the fact that both cases apply stochastic delays on the local training time, and since the latter has the highest impact on the E2E training time compared to the propagation delay, the placement solution is the same for both cases as they have similar strictness on time constraints. We set the stochastic drifts as proportional to the nominal values of the training time and propagation delays, respectively.

We developed our mathematical model using AMPL (A Mathematical Modeling Programming Language), utilizing CPLEX as the linear solver optimizer. We run our algorithm on an Ubuntu server 14.04 LTS virtual machine with 64 GB of RAM and 8 × 2.5 GHz CPU cores.

7.3. Simulated network instances

First, as network topology we use Mandala, a hierarchical topology from [49]: it consists of connecting access nodes through three tiers, i.e., aggregation, core and application (i.e., egress) nodes. For instance, one may consider the edge servers as MEC hosts in a MEC system in a Metropolitan Area Network topology or a near edge AIF deployment (see Fig. 6). The total number of nodes is equal to 26 including 16 edge nodes. Also, we consider that each node can be randomly equipped with 1, 2, 4, 8 or 16 CPU cores. Note that IFLC can be deployed by an scheduler such as the MEC orchestrator in a MEC system.

We compare four resolution approaches:

- no-HWA: degenerate IFLC with no HWAs employed.
- IFLC-8: 50% of edge nodes (i.e. 8) equipped with HWAs.

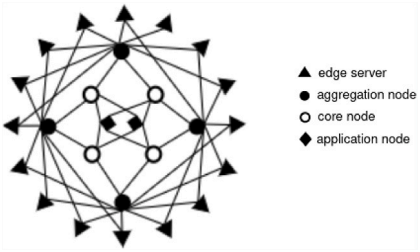


Fig. 6. Mandala network topology.
Source: [49].

- IFLC-16: all edge nodes (i.e. 16) equipped with HWAs.
- FIRST-FIT baseline Alg. 4 (not allocating HWA).

The comparison is done looking at the following features:

- the number of straggling AIFs,
- the E2E training time and its variance,
- the computational overhead related to the number of active AIFs and of CPU cores.

We rely on [37] and [38] to define the acceleration factor of HWAs: accordingly, we consider that it depends on the number of active AIFs and the number of available CPU cores [37]. More precisely, since the acceleration factor decreases with the data size [38] and that the training dataset is evenly shared among client AIFs, we assume that α_{ik} increases with the number of active AIFs.

Moreover, in order to test different levels of strictness on the training time target constraints, we use different values for the target time τ (i.e., 2 and 4 s) and the number of epochs (i.e., from 60 to 105 with a step of 5 epochs). The target time represents the retraining time to ensure that the model remains effective as new data arrives. As we are considering the use case of a near real-time anomaly detection system able to capture recent events as attacks or new vulnerabilities [50], the target time ranges from hundreds of milliseconds to seconds. In fact, the time to retrain an anomaly detection model in real-time depends on several factors such as the volume of data, the complexity of the model and the computational resources. To define the target time, we rely on the work in [51] where the end-to-end time to train a few hundred of data samples does not exceed a few seconds (i.e., ranges from a few seconds to ten seconds).

The stragglers occurrence ranges from a few milliseconds to 1/4 of the time limit delay. Extending the time limit would incur in a low quality of the model as it may not reflect the current state of the system anymore.

We consider that the maximum tolerated delay Δ is 4 times less than the target time, which roughly corresponds to the maximum lifetime network connections that are not bulk transfers. Then, we range from loose timing constraint (e.g., $\tau = 4$ s with 60 epochs) to extremely rigorous ones (e.g., $\tau = 2$ s with 100 epochs). Also, additional stochastic delays may reach nearly twice the nominal time.

We run 30 instances for each approach and each different setting where the propagation time, the stochastic delays and the placement of hardware accelerators are randomly generated for each instance. The number of available CPU cores is fixed for all instances.

8. Results analysis

In this section, we provide a detailed numerical evaluation focused on stragglers, training times and computation overhead. Overall, Table 5 presents the proportion of instances that lead to a feasible solution (with respect to the target delay bound). For the two AIF placement settings, no-HWA could produce a solution only at most for 15% of the instances. This increases to 100% when IFLC is used. On

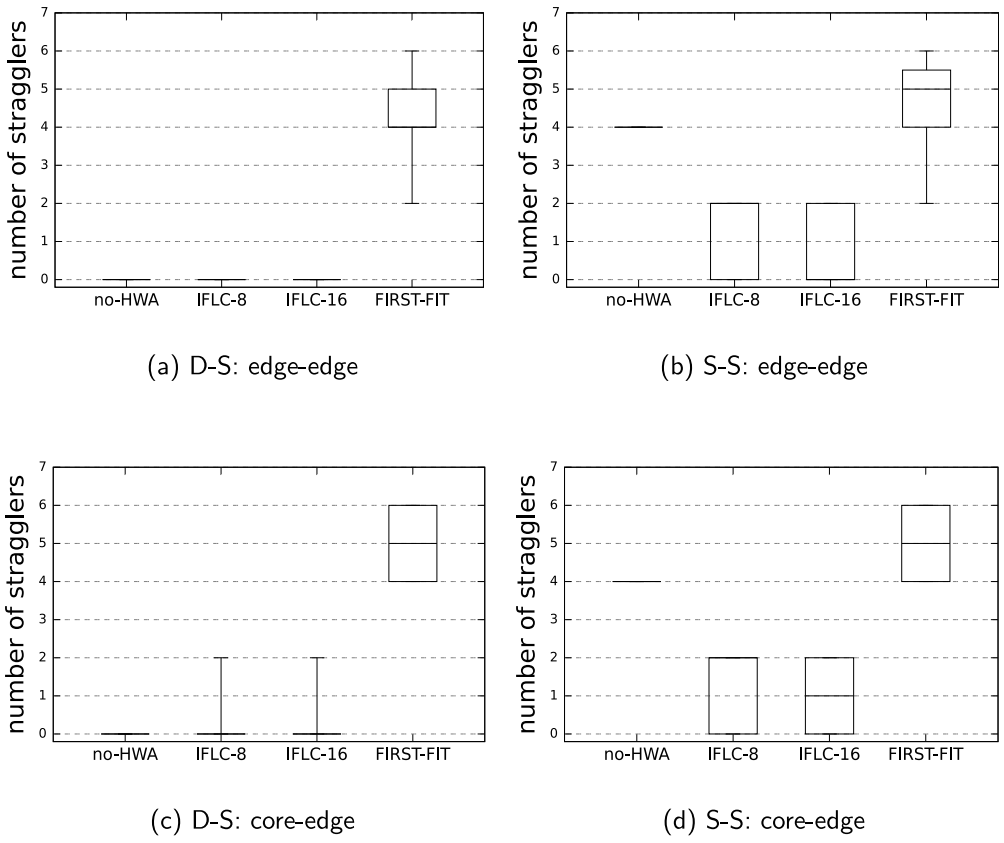


Fig. 7. Distribution of the number of straggling AIFs.

Table 5
Percentage of feasible instances per approach.

Approach	Edge–edge	Core–edge
no-HWA	15%	10%
IFLC-8	100%	100%
IFLC-16	100%	100%
FIRST-FIT	0%	0%

the other side, all the solutions produced by FIRST-FIT algorithm are unfeasible, as it has no check on the target time. It is worth noting that if the E2E training time of an AIF exceeds the time threshold, we consider that the local training parameters cannot be used by the FL aggregation task.

8.1. Number of stragglers

In Figs. 7, we present the distribution of the number of AIF stragglers for both edge–edge and core–edge settings, and excluding the 'D–D' case since it does not model the stragglers. We can notice that:

- With IFLC, the likelihood of being in a straggling situation decreases with the number of available HWAs when the training time is stochastic.
- With FIRST-FIT, the number of stragglers is the worst, and it is higher in the core–edge setting: the placement at the edge gives more flexibility thanks to lower propagation delays, hence leading to lower E2E training times. This does not happen with IFLC, showing its robustness against high propagation delays (core–edge setting).
- For all approaches and settings, the number of straggling AIFs increase when the training time is stochastic, as it can be seen from Figs. 7(b) and 7(d). Specifically, the median value increases from 4 to 5 for FIRST-FIT with the edge–edge setting, whereas in core–edge the minimum number of stragglers increases by 2. On the other side, the highest number of stragglers generated by IFLC-8 and IFLC-16 increases from 0 to 2 for both placement settings. This number increases from 0 to 4 with no-HWA for both settings.
- In the 'D–S' case (see Figs. 7(a) and 7(c)), our approach significantly outperforms FIRST-FIT, regardless of the number of HWAs. Under more stringent targets (S–S), FIRST-FIT yields lower number of stragglers than no-HWA with edge–edge, where the minimum number of stragglers achieved by no-HWA (i.e., 4 stragglers) corresponds to the first quartile value achieved by FIRST-FIT in the edge-edge setting, and the minimum value in the core-edge one. Note that the placement decision made by FIRST-FIT can not be considered as it is exceeding the threshold.

Overall, thanks to time modulation, IFLC always outperforms FIRST-FIT in terms of the number of stragglers, for all the cases. Adaptive HWA allocation helps reducing the local training time which allows slow AIFs to reach lower E2E training times and consequently respect the imposed target time. The number of stragglers with IFLC can be divided by more than two, and often is reduced to zero.

8.2. Training time

Fig. 8 represent the distribution of the maximum local training times. We can notice that:

- In contrast to FIRST-FIT, IFLC approaches have similar distributions of the training time for each of the placement settings. This can be explained by the fact that the latter leads to solutions that are robust against high propagation delays.

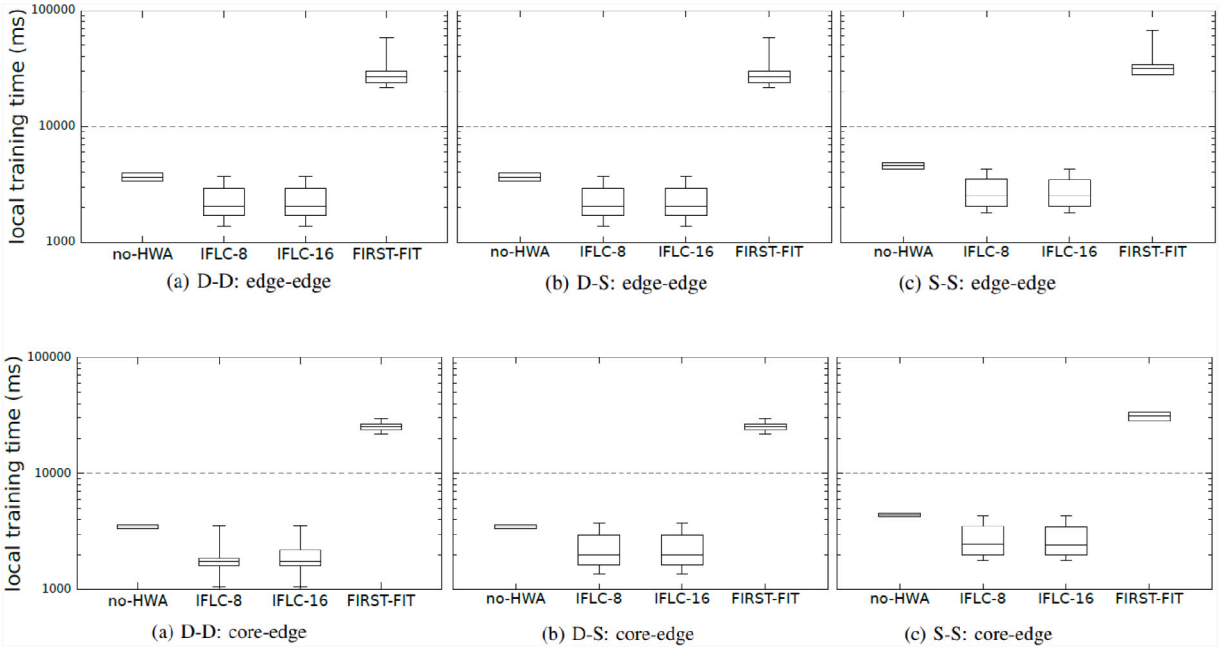


Fig. 8. Distribution of the maximum local training time.

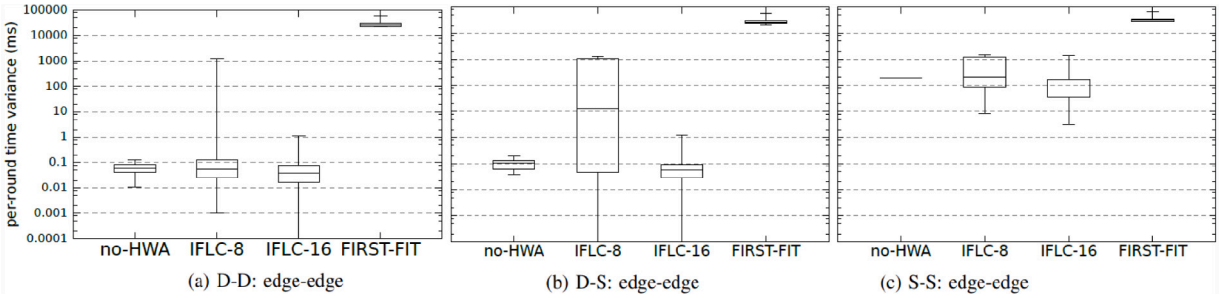


Fig. 9. Distribution of the variance in E2E training time.

- The distribution of the training time is very similar for IFLC-8 and IFLC-16 for all cases. This can be due to the fact that both IFLC-8 and IFLC-16 yield different placement solutions that generate similar end-to-end training time (i.e., same number of AIFs with higher CPU resources and a lower number of active HWAs or the opposite). This can also show that a low number of HWAs can be sufficient to respect the target time even with very strict targets.
- Both 'D–D' and 'D–S' cases have lower maximum training times compared to S–S for all cases. This happens because S–S has higher local training times due to additional applied delays.

Overall, IFLC gives lowest training times thanks to HWA, followed by no-HWA and finally FIRST-FIT which yields the highest training times (which are higher than the imposed target time, hence discarded by the FL server).

Figs. 9 report the distribution of the maximum variance in E2E training times (only for the edge–edge setting, as no major difference appears with the core–edge one). We can notice that:

- The lowest variance in E2E training time corresponds to IFLC-16 followed by no-HWA then IFLC-8. Indeed, IFLC deploys AIFs with close E2E training time with the aim of reducing the number of stragglers during each round, attempt favored by HWAs that get allocated to accelerate training for farthest AIFs from the server. The variance is further decreased in deterministic cases.
- IFLC-16 yields lower variance when compared to IFLC-8 for all cases. This can be explained by the fact that the former has more control in placing the AIFs on nodes with similar capacities as HWA is present on all nodes. On the other hand, IFLC-8 has less flexibility where the node selection depends on the HWA availability. Moreover, the variance increases with time constraints as can be seen from 'D–S' and 'S–S' cases when comparing IFLC-8 and IFLC-16. In fact, IFLC-16 allows to allocate HWA on nodes with similar processing capacities which results in equivalent local training times. On the other hand, IFLC-8 may not have available HWAs on these nodes and thus nodes with different processing capacities are used. This results in higher variance. As previously explained, this can yield similar end-to-end training times for both IFLC-8 and IFLC-16.

Globally, since (i) the weight of the local training time in the E2E training time is greater than the propagation delay one, and (ii) given that training times get higher if HWA unavailability, we can determine that finding viable solutions gets harder as it turns into finding nodes with higher CPU resources to respect the target time.

8.3. AIF computational overhead

The latter observation can be clarified by Figs. 10 and 11 that depict the distribution of the number of active AIFs and the total number of CPU cores that are used by the active AIFs. We only report the cases D–S and S–S in this section as D–D yields the similar distributions as 'D–S'.

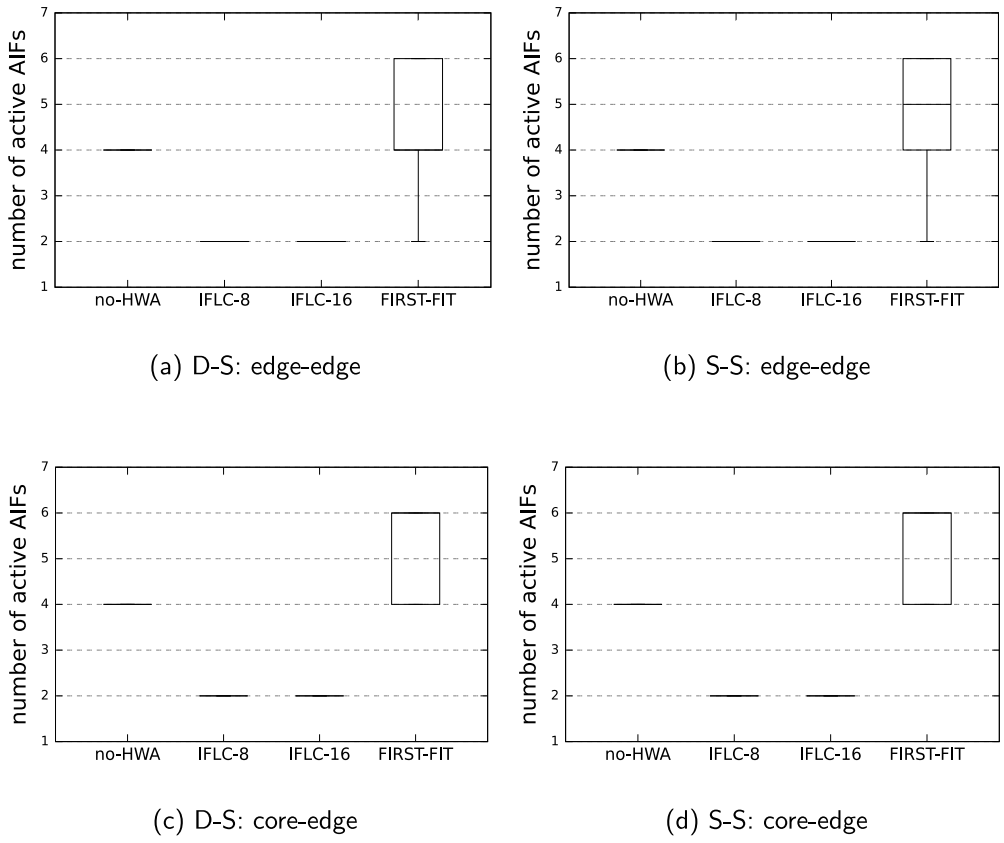


Fig. 10. Distribution of the number of active AIFs.

Besides expectable behaviors for FIRST-FIT deriving from the previous analysis, we can highlight that:

8.3.1. Total number of active AIFs

- The number of active AIFs reduces with the number of HWA. More precisely, for no-HWA feasible solutions refer to instances with less stringent time constraints. The corresponding number of active AIFs is equal to 4, that is, 4 AIFs are deployed to respect the target time. This number decreases by up to 50% thanks to HWA: both IFLC-8 and IFLC-16 yield a lower number of active AIFs (i.e, 2 AIFs) when the same time constraints apply.
- For FIRST-FIT, the minimum number of active AIFs is higher with core–edge setting,
  as it is more flexible than edge–edge in placing edge AIFs. In that case, if the stopping point is not yet achieved (i.e., possibility to decrease the local training time), FIRST-FIT will keep increasing the number of AIFs. This confirms the previous results showing that the local training time is lower with core–edge.
- IFLC-8 yields the same number of active AIFs as IFLC-16. This can be explained by the fact that IFLC may sometimes promote allocating HWA instead of increasing the number of active AIFs to reduce the local training time. As HWA may not be available on some physical nodes with IFLC-8, the latter chooses the same number of active AIFs as IFLC-16 but with higher CPU resources.

8.3.2. Total number of active CPU cores

- Higher CPU resources are needed to reduce the local training time and consequently the E2E training time for FIRST-FIT and no-HWA, which is correlated with the higher number of active AIFs even with less strict time constraints. Also, FIRST-FIT achieves the same minimum cost as IFLC for a small number of instances with edge–edge placement. For these instances, the stopping point is achieved with a low number of active AIFs w.r.t the other instances, which results in a lower number of CPU cores.
- The distribution of CPU cores is slightly different when comparing IFLC-8 and IFLC-16. As already explained, when stringent time constraints apply, IFLC-8 yield the same number of active AIFs as IFLC-16. However, the former may not have available HWA on nodes with low CPU resources which leads to solutions with slightly higher processing capacities.

Overall, the advantage of an IFLC scheme is the capability of exploiting HWA, leading to the lowest computational costs, as less CPU resources are needed.

8.4. Comparison between hierarchical and flat topology

To assess the impact of topology changes on IFLC, we propose to additionally evaluate a random flat topology with a total number of nodes and the average degree of nodes equal to the one of the previous analyzed mandala topology. The interconnection of nodes is random and the propagation delays are bounded by the minimum and maximum values used for mandala topology and are generated following the same distribution. In contrast to mandala, the random topology allows to place the AIFs allover the nodes.

In the following, we present the distribution of the number of stragglers, the local training time, the number of active AIFs and the number of CPU cores used, for the stochastic case ‘S–S’ and using the following approaches:

- Man-8E: 50% of edge nodes (i.e. 8) equipped with HWAs, using mandala topology with ‘edge–edge’ setting;
- Man-16E: all edge nodes (i.e. 16) equipped with HWAs, using mandala topology with ‘edge–edge’ setting;
- Man-8C: 50% of edge nodes (i.e. 8) equipped with HWAs, using mandala topology with ‘core–edge’ setting;
- Man-16C: all edge nodes (i.e. 16) equipped with HWAs, using mandala topology with ‘edge–edge’ setting;

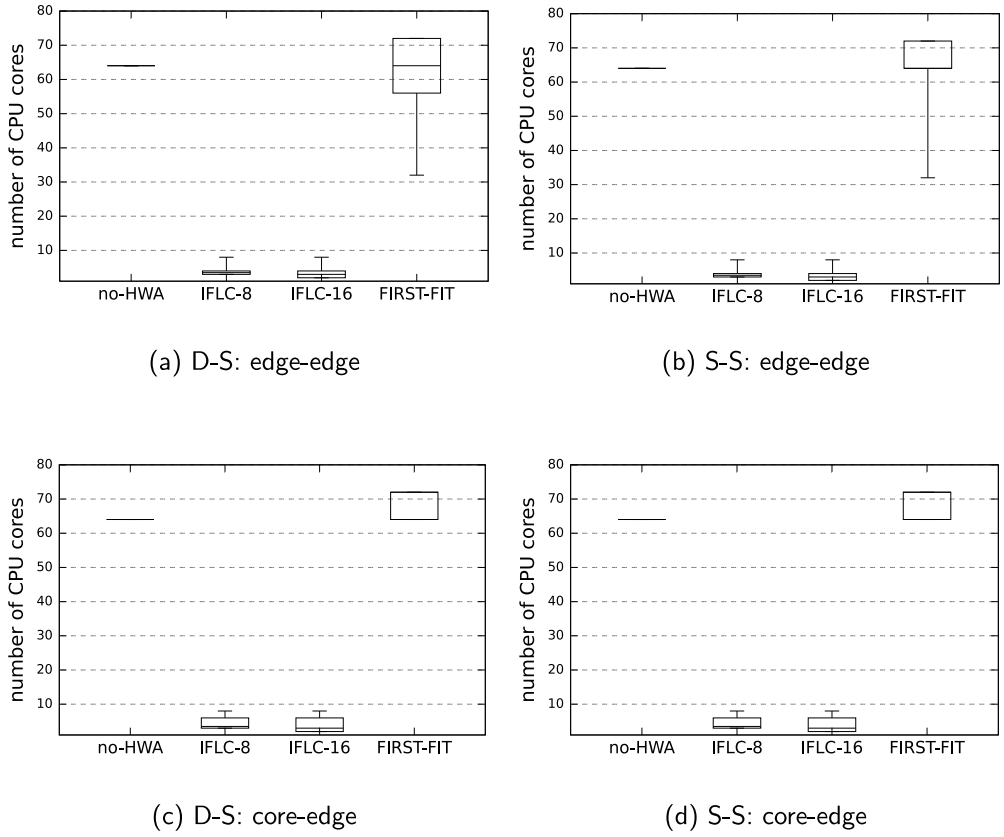


Fig. 11. Distribution of the total number of CPU cores.

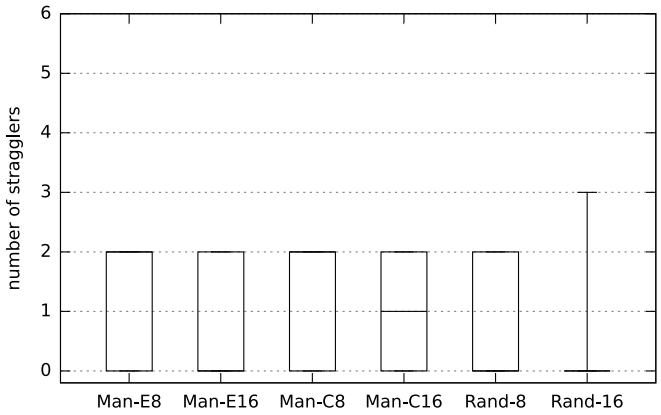


Fig. 12. Distribution of the number of straggling AIFs using IFLC: S–S case.

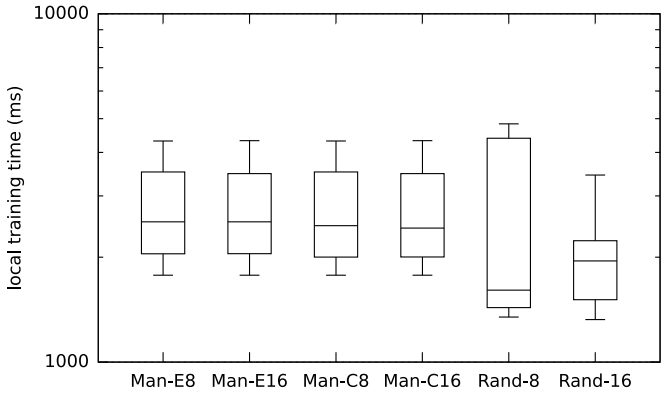


Fig. 13. Distribution of the local training time using IFLC: S–S case.

- Rand-8: 8 nodes equipped with HWAs, using the random topology;
- Rand-16: 16 nodes equipped with HWAs, using the random topology.

Differently from mandala topology, the cases with random flat topology produces feasible solutions for 95% (65%, respectively) of the instances when 16 (8, respectively) HWAs are available.

8.4.1. Number of AIF stragglers

In Fig. 12, we evaluate the number of stragglers for the three different deployments (i.e., edge and core settings with mandala, and random flat topology) for all the instances presented in Section 7.3.

- We can notice that IFLC produces the same number of stragglers when the number of hardware accelerators is equal to 8, for the three deployments.
- When increasing the number of HWAs the number of stragglers remains the same for edge scenario (i.e., Man-E16). It decreases with core scenario where the median value is reduced to 1 AIF straggler. With the random flat topology, the number of stragglers decreases w.r.t the two other scenarios except for few instances with strict time constraints where the number of stragglers achieved 3.

8.4.2. Local training time

In Fig. 13, we present the local training time for the three scenarios and for different numbers of available HWAs.

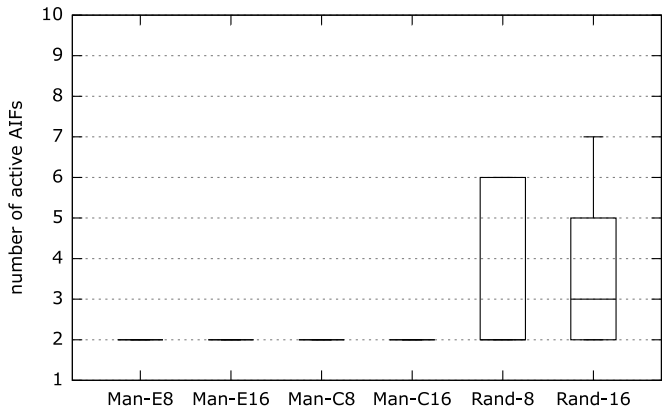


Fig. 14. Distribution of the number of active AIFs using IFLC: S–S case.

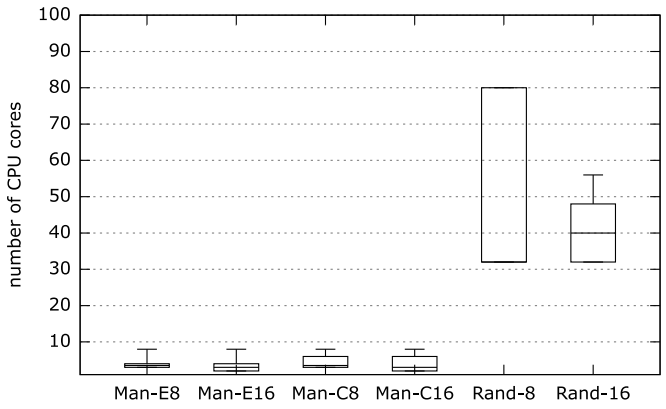


Fig. 15. Distribution of the number of CPU cores using IFLC: S–S case.

- As already mentioned, edge and core scenarios has similar local training times as for mandala topology the propagation delays are neglected compared to training time. When the number of HWAs is equal to 8, the local training time is lower when using the random flat topology for a high number of instances. The training time increases with instances having stricter time constraints when compared to edge and core scenarios. This can be explained by the fact that in the random topology, the latency on the links is more important and thus, has more impact on the end-to-end learning time. When strict time constraints applies, finding a feasible solution that respects the E2E target time with lower local training time becomes difficult since achieving the trade off between choosing nodes with high capacity to deploy AIFs and ensuring low propagation delays to the AIF server becomes harder.
- When increasing the number of HWAs to 16, the random flat topology produces lower training times w.r.t core and edge settings. This can be explained by the fact that in flat topologies we have more freedom on placing the AIFs and since the number of HWAs is higher, finding solutions with lower training times becomes easier.

8.4.3. Number of active AIFs

In Fig. 14, we present the total number of active AIFs using mandala and the random flat topology.

- We notice that the number of active clients with random topology is 3 times higher compared to mandala topology with both edge and core settings. Unlike Mandala which is a hierarchical topology, the random topology is a flat one with homogeneous links which results in higher propagation delays and thus, the contribution of propagation delays to the en-to-end delays is higher w.r.t to manadala topology. In that case, IFLC proposes a higher number of active client AIFs to reduce the local training time which allows to compensate the increased propagation delays.

8.4.4. Number of active CPU cores

In Fig. 15, we present the number of CPU cores produced by IFLC using manadala and random topology.

- As already explained, a higher number of CPU cores refers to lower training time. In fact, the random flat topology incurs in higher propagation delays and consequently higher E2E delays. Increasing the number of CPU cores allows to reduce the local training time and consequently reducing the E2E learning time.

Overall, the mandala topology simulates a mobile network with non heterogeneous links. In that case, the E2E latency on the links is lower compared to a flat topology such as the one we used, where the latency on the links are much more important. This results in totally different placement solutions produced by IFLC, as in the second case the propagation delays have higher impact on end-to-end delays. Additionally, IFLC is designed for optimizing training time efficiency thanks to the possible allocation of hardware acceleration, which explains its effectiveness with topologies where the propagation delays are of minor importance compared to the training time.

8.5. Link utilization estimation

It is worth noting that the link utilization can be influenced by the feature size of the training dataset: the higher the size of the features, the higher is the model size to be transferred. For instance, the dataset from [47] has hundreds of features, the size of weights is around 60 to 70 Megabytes. If we consider that the transmission delays are negligible (as per Definition 2) and that therefore the used links are over-provisioned as it is often the case in provider networks, we can assume that link utilization is not affected by model exchange. Certainly this assumption may not hold in other use-cases, such as private networks, which may have scarce link resources.

9. Conclusion and perspectives

In this paper, we proposed a federated learning system control scheme for dynamic placement of FL nodes for in-network applications taking jointly into consideration learning and network delays. Our scheme is designed to decrease the number of learning stragglers, while making efficient use of heterogeneous computing resources. A major highlight is that we demonstrated how adaptive hardware acceleration enabling can halve or even remove the occurrence of stragglers. We show how the proposed scheme outperforms static deployments of hardware acceleration, avoid their random or systemic use; we show that we can so avoid too high variance in the end-to-end training times on the one hand, and useless computational overhead on the other hand. Our scheme achieves this performance thanks to an original delay model we proposed to combine network delays with distributed training delays, when seeking efficient learning solutions. We also show how we can integrate stochastic variations to both network delays and local training times in the design of our in-network federated learning control scheme. Finally, we compared IFLC performance against topologies, a hierarchical one and a flat one.

Future work could cover a detailed numerical evaluation and integration of the anticipated IFLC variants to be robust against transient node failures, as well as the refining of the aggregation functions at the federated learning with server level, in order to further increase the learning efficiency. Moreover, we plan to work on scaling the resulting learning systems by means of split learning to cover multiple heterogeneous learning domains.

CRediT authorship contribution statement

Nour-El-Houda Yellas: Writing – original draft, Visualization, Validation, Software, Methodology, Investigation, Conceptualization. **Bernardetta Addis:** Writing – review & editing, Validation, Supervision, Methodology, Conceptualization. **Selma Boumerdassi:** Writing – review & editing, Validation, Supervision. **Roberto Riggio:** Writing – review & editing, Conceptualization. **Stefano Secci:** Writing – review & editing, Validation, Supervision, Resources, Methodology, Funding acquisition, Conceptualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work was funded by the H2020 AI@EDGE (https://aiatedge.eu; grant nb. 101015922), the PIA/AMI-5G INFLUENCE, the France 203 IE6 (Internet of Edges for 6G; contract nb. DOS0223931) and the ANR TREES (ANR-24-IAS1-0004; https://trees.roc.cnam.fr) projects.

Appendix. Robustness against single AIF software failure

In the following, we show how versatile the IFLC formulation is, by proposing its extension (under the stochastic IFLC model) to support transient single node failures. This extension allows to find a solution that can satisfy the target distributed learning time even in case one of the selected AIF nodes (or underlying physical node elements) fails. This makes the model robust in particular against transient failures for which it is not worth to rerun the algorithm given the limited duration. We focus on the stochastic version of our model, where the objective is to minimize the expected number of stragglers. We assume that in case of a single node failure the load balancer (in the data-pipelining system) is able to dynamically dispatch training data among the residual $k-1$ working AIFs [51].

First of all, we extend the set of scenarios $\mathcal{S}$ to introduce the single node failures scenarios. We do so also to allow us having a probability weight to different types of impairments: node failures, variability in propagation delay and training time variations due to AI-algorithm convergence. Let us call $\mathcal{S}_f$ the additional scenarios taking specifically into account node failures. We need to add the following variables and constraints to the stochastic model (see Section 5) to take into account the training time in case of single-node failure.

Local training time

We denote $\widetilde{\pi}_i^{\,s}$ the local stochastic training time for a given scenario s on active node i when k AIFs are active, but one of them is under a transient failure condition.

$$\widetilde{\pi}_i^{\,s} = \sum_{k=2}^{n} (p_{i(k-1)} + \beta^s_{i(k-1)})\zeta_{ik} \qquad \forall i \in N, k \in 2..n, s \in \mathcal{S}_f \tag{A.1}$$

we recall that the deterministic training time depends on the number of working AIFs. Therefore, if one node fails, $k-1$ AIFs share the workload and need a training time equal to $p_{(k-1)}$.

Hardware acceleration

Let us denote by $\tilde{\pi}$ the maximum stochastic time in case of single node failure among all possible scenarios.

$$\widetilde{\pi} = \max_{k=2..n, i\in N, s\in\mathcal{S}_f} (p_{ik-1} + \beta^s_{ik-1}) \tag{A.2}$$

and ω^s_{ik} the reduction in time due to hardware acceleration in the case of single-node failure. The following constraints allow to set correctly the value of the time reduction:

$$\widetilde{\omega}^s_{ik} \le \psi_{ik}\left(1 - \frac{1}{\alpha_{ik-1}}\right)\widetilde{\tau} \qquad \forall i \in N, k \in 2..n \quad \forall s \in \mathcal{S}_f \tag{A.3}$$

$$\widetilde{\omega}^s_{ik} \le \left(1 - \frac{1}{\alpha_{ik-1}}\right)(p_{ik-1} + \beta^s_{ik-1})\zeta_{ik} \qquad \forall i \in N, k \in 2..n \quad s \in \mathcal{S}_f \tag{A.4}$$

$$\widetilde{\omega}^s_{ik} \ge \left(1 - \frac{1}{\alpha_{ik-1}}\right)(p_{ik-1} + \beta^s_{ik-1})\zeta_{ik} - (1-\psi_{ik})\widetilde{\tau} \qquad \forall i \in N, k \in 2..n \quad \forall s \in \mathcal{S}_f \tag{A.5}$$

E2E learning time

The E2E learning time when a single node fails, represented by variable $\widetilde{\Pi}$, is determined by the following constraints:

$$\widetilde{\Pi}_i^{\,s} = \widetilde{\pi}_i^{\,s} - \sum_{k=2}^{n} \widetilde{\omega}^s_{ik} + \sum_{j\in A}(d_{ij} + \eta^s_{ij})\xi_{ij} \qquad \forall i \in N, s \in \mathcal{S}_f \tag{A.6}$$

and the active stragglers for single-node scenarios $\mathcal{S}_f$ can be determined by:

$$\widetilde{\Pi}_i^{\,s} \le \tau + \Delta\sigma_i^s \qquad \forall i \in N, s \in \mathcal{S}_f \tag{A.7}$$

where variables $\Delta\sigma_i^s$ are extended to set $\mathcal{S}_f$.

Objective

The objective in (33) is modified to take into account all failure scenarios as follows:

$$\min \sum_{s\in S} q_s \sum_{i\in N} \sigma_i^s + \sum_{s\in\mathcal{S}_f} q_s^f \sum_{i\in N} \sigma_i^s \tag{A.8}$$

where q_s^f is the probability of single-node failure scenario $s \in \mathcal{S}_f$.

Data availability

Data will be made available on request.

References

[1] S. Schwarzmann, C.C. Marquezan, R. Trivisonno, S. Nakajima, V. Barriac, T. Zinner, ML-based QoE estimation in 5G networks using different regression techniques, IEEE Trans. Netw. Serv. Manag. 19 (3) (2022).
[2] Y. Mao, et al., A survey on mobile edge computing: The communication perspective, IEEE Commun. Surv. Tutor. 19 (4) (2017).
[3] Timothy Yang, et al., Applied federated learning: Improving google keyboard query suggestions, 2018, arXiv:1812.02903.
[4] Y. Jeon, et al., A distributed NWDAF architecture for federated learning in 5G, in: ICCE, 2022.
[5] P. Rajabzadeh, A. Outtagarts, Federated learning for distributed NWDAF architecture, in: ICIN, 2023.
[6] S. Hosseinalipour, et al., From federated to fog learning: Distributed machine learning over heterogeneous wireless networks, IEEE Commun. Mag. 58 (2020).
[7] H. Ko, et al., Joint client selection and bandwidth allocation algorithm for federated learning, IEEE Trans. Mob. Comput. 22 (2023).
[8] N.-E.-H. Yellas, B. Addis, R. Riggio, S. Secci, Function placement and acceleration for in-network federated learning services, in: CNSM, 2022.
[9] D2.1: Use cases, requirements, and preliminary system architecture, AI@EDGE H2020 Project.
[10] B. McMahan, et al., Communication-efficient learning of deep networks from decentralized data, in: Artificial intelligence and statistics, PMLR, 2017.
[11] S. Bin Ruba, N.-E.-H. Yellas, S. Secci, Anomaly detection for 5G softwarized infrastructures with federated learning, in: 6GNet, 2022.
[12] Zheng. Changgang, et al., In-network machine learning using programmable network devices: A survey, IEEE Commun. Surv. Tutor. (2023).
[13] R. Boutaba, et al., A comprehensive survey on machine learning for networking: evolution, applications and research opportunities, J. Internet Serv. Appl. 9 (1) (2018).

[14] J. Bendriss, et al., AI for SLA management in programmable networks, in: DRCN, 2017.
[15] V.C. Emeakaroha, et al., Low level metrics to high level SLAs-LoM2HiS framework: Bridging the gap between monitored metrics and SLA parameters in cloud environments, in: International Conference on High Performance Computing & Simulation, 2010.
[16] L. Boero, M. Marchese, S. Zappatore, Support vector machine meets software defined networking in IDS domain, in: ITC, 2017.
[17] S.T. Miller, C. Busby-Earle, Multi-perspective machine learning a classifier ensemble method for intrusion detection, in: Proceedings of the International Conference on Machine Learning and Soft Computing, 2017.
[18] S. Deng, et al., Edge intelligence: The confluence of edge computing and artificial intelligence, IEEE Internet Things J. 7 (8) (2020).
[19] Z. Zhou, et al., Edge intelligence: Paving the last mile of artificial intelligence with edge computing, Proc. IEEE 107 (8) (2019).
[20] 3GPP TS 23.288, Architecture enhancements for 5G system to support network data analytics services, v. 17.1.0, 2021.
[21] A. Diamanti, J.M. Vilchez-Sanchez, S. Secci, An AI-empowered framework for cross-layer softwarized infrastructure state assessment, IEEE Trans. Netw. Serv. Manag. 19 (4) (2022).
[22] Mingyuan Zang, et al., Federated learning-based in-network traffic analysis on IoT edge, in: 2023 IFIP Networking Conference, IFIP Networking, IEEE, 2023.
[23] Qiaofeng Qin, et al., Line-speed and scalable intrusion detection at the network edge via federated learning, in: 2020 IFIP Networking Conference, Networking, IEEE, 2020.
[24] A. Harlap, et al., Addressing the straggler problem for iterative convergent parallel ML, in: Proceedings of the seventh ACM symposium on cloud computing, 2016.
[25] Yang, et al., Using trio: Juniper networks' programmable chipset for emerging in-network applications, in: Proceedings of the ACM SIGCOMM 2022 Conference.
[26] D. Cardoso, et al., Serene: Handling the effects of stragglers in in-network machine learning aggregation, NOMS (2023).
[27] W.Y.B. Lim, et al., Decentralized edge intelligence: A dynamic resource allocation framework for hierarchical federated learning, IEEE Trans. Parallel Distrib. Syst. 33 (2022).
[28] W.Y.B. Lim, et al., Dynamic edge association and resource allocation in self-organizing hierarchical federated learning networks, IEEE J. Sel. Areas Commun. 39 (2021).
[29] L. Yu, et al., Jointly optimizing client selection and resource management in wireless federated learning for Internet of Things, IEEE Internet Things J. 9 (2022).
[30] L.U. Khan, et al., Federated learning for edge networks: Resource optimization and incentive mechanism, IEEE Commun. Mag. 58 (10) (2020).
[31] R. Zeng, et al., FMore: An incentive scheme of multi-dimensional auction for federated learning in MEC, in: ICDCS, 2020.
[32] Y. Cui, et al., Client scheduling and resource management for efficient training in heterogeneous IoT-edge federated learning, IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. 41 (2022).
[33] J. Lee, et al., Adaptive precision CNN accelerator using radix-x parallel connected memristor crossbars, 2019, arXiv preprint.
[34] H. Giefers, et al., Analyzing the energy-efficiency of sparse matrix multiplication on heterogeneous systems: A comparative study of GPU, Xeon Phi and FPGA, in: IEEE ISPASS, 2016.
[35] B. Betkaoui, D.B. Thomas, W. Luk, Comparing performance and energy efficiency of FPGAs and GPUs for high productivity computing, in: International Conference on Field-Programmable Technology, 2010.
[36] M. Qasaimeh, et al., Comparing energy efficiency of CPU, GPU and FPGA implementations for vision kernels, in: IEEE ICESS, 2019.
[37] Y. Guan, et al., FPGA-based accelerator for long short-term memory recurrent neural networks, in: ASP-DAC, 2017.
[38] Dimitrios Danopoulos, et al., LSTM acceleration with FPGA and GPU devices for edge computing applications in B5G MEC in SAMOS, 2022.
[39] J. Xu, S.-L. Huang, L. Song, T. Lan, Live gradient compensation for evading stragglers in distributed learning, in: IEEE INFOCOM, 2021.
[40] A. Reisizadeh, S. Prakash, R. Pedarsani, A.S. Avestimehr, CodedReduce: A fast and robust framework for gradient aggregation in distributed learning, IEEE/ACM Trans. Netw. 30 (1) (2021).
[41] B. Buyukates, et al., Gradient coding with dynamic clustering for straggler-tolerant distributed learning, IEEE Trans. Commun. 71 (6) (2023).
[42] Z. Chai, et al., FedAT: A high-performance and communication-efficient federated learning system with asynchronous tiers, in: SC21: International Conference for High Performance Computing, Networking, Storage and Analysis, 2021.
[43] J. Park, et al., Sageflow: Robust federated learning against both stragglers and adversaries, in: Advances in Neural Information Processing Systems, vol. 34, 2021.
[44] Chenhao Xu, et al., Asynchronous federated learning on heterogeneous devices: A survey, Comp. Sci. Rev. 50 (2023) 100595.
[45] E. Zeydan, J. Mangues-Bafalluy, Recent advances in data engineering for networking, IEEE Access 10 (2022).
[46] D. Justus, et al., Predicting the computational cost of deep learning models, in: 2018 IEEE International Conference on Big Data (Big Data).
[47] D. Chi Phung, et al., An open dataset for beyond-5G data-driven network automation experiments, in: 6GNet, 2022.
[48] IFLC dataset and scripts, 2023, (url): https://github.com/nehyellas/IFLC. (Accessed 25 June 2023).
[49] W. da Silva Coelho, et al., Function splitting, isolation, and placement trade-offs in network slicing, IEEE Trans. Netw. Serv. Manag. 19 (2) (2021).
[50] Albert Bifet, Ricard Gavalda, Learning from time-changing data with adaptive windowing, in: Proceedings of the 2007 SIAM International Conference on Data Mining, Society for Industrial and Applied Mathematics, 2007.
[51] P. Ntumba, N.-E.-H. Yellas, S. Bin-Ruba, F. Ben-Abdesslem, S. Secci, Data pipeline system designs for in-network learning, in: CNSM 2024, https://hal.science/hal-03883727.

Nour-El-Houda Yellas is currently a postdoc researcher at Orange Innovation. She received the master's degree in network and engineering from Paris Saclay University in 2019 and the Ph.D. degree from Cnam, Paris in 2023. Her research activities concern mainly the automation and optimization of 5G mobile networks orchestration in multiaccess edge computing infrastructures using data analytics.

Bernardetta Addis received the M.S. and Ph.D. degrees in computer science engineering from the University of Florence, Florence, Italy, in 2001 and 2005, respectively. She worked as a Research Associate (to Operations Research Methods for Health Care Problems) with the Computer Science Department (Dipartimento di Informatica), University of Turin (Università degli Studi di Torino), Turin, Italy. She is currently professor at University of Lorraine, France. Her expertise is on optimization with particular reference to nonlinear global optimization and recently to heuristics and exact methods for integer optimization.

Selma Boumerdassi received the M.Sc. degree in computer engineering from ESI, Algeria, and the M.Sc. and Ph.D. degrees in computer science from UVSQ, France. She was an Assistant Professor with UVSQ. She is currently an Associate Professor with CNAM and a Research Associate with INRIA, Paris. Her research interests include wireless and mobile networks, with a special focus on the impact and use of social networks, trajectory prediction, information dissemination, and lightweight security.

Roberto Riggio is an Associate Professor (RTD-B) in the Information Engineering Department at the Polytechnic University of Marche in Ancona, Italy. He received his Ph.D. from the University of Trento (Italy), then he was postdoc at the University of Florida, Researcher/Chief Scientist at CREATE-NET in Trento (Italy), Head of Unit at FBK in Trento (Italy), Senior 5G Researcher at the i2CAT Foundation in Barcelona (Spain), and Senior Researcher in the Connected Intelligence Group at RISE AB in Stockholm, Sweden. He serves in the TPC/OC of leading conferences in the networking field, including IEEE IM/NOMS, IEEE NFV-SDN, IEEE NetSoft, and IEEE CNSM. He is an associate editor for the Wiley International Journal of Network Management, the Springer Wireless Networks journal, and the IEEE Transactions on Network and Service Management. He is a member of the ACM and a Senior Member of the IEEE.

Stefano Secci is professor of communication networks at Cnam, Paris, France; lead of the Networks and IoT Systems research group, and vice-chair of the computer science department. In 2010-2018, he was associate professor at LIP6, Sorbonne University (UPMC - Paris VI), France. He received the M.Sc. degree in communications engineering from Politecnico di Milano, Italy, in 2005, and a dual Ph.D. degree in information and communication technologies from Politecnico di Milano and Télécom ParisTech, France, in 2009. In 2010, he worked as Post-Doctoral Fellow with NTNU, Norway, and George Mason University, USA. Before the Ph.D., in 2005-2006, he worked as a research associate with Ecole Polytechnique de Montréal, Canada, and with Politecnico di Milano, and as a network engineer with Fastweb Italia, Italy.