Function Placement and Acceleration for In-Network Federated Learning Services

Nour-El-Houda Yellas, Bernardetta Addis*, Roberto Riggio‡, Stefano Secci
Cnam, 292 rue St. Martin, 75003, Paris, France. Email: firstname.lastname@cnam.fr
* Université de Lorraine, CNRS, LORIA, 54000 Nancy, France. Email: bernardetta.addis@loria.fr
‡ Polytechnic University of Marche, Ancona, Italy. Email: r.riggio@univpm.it

***Abstract*—Edge intelligence combined with federated learning is considered as a way to distributed learning and inference tasks in a scalable way, by analyzing data close to where it is generated, unlike traditional cloud computing where data is offloaded to remote servers. In this paper, we address the placement of Artificial Intelligence Functions (AIF) making use of federated learning and hardware acceleration. We model the behavior of federated learning and related inference point to guide the placement decision, taking into consideration the specific constraint and the empirical behavior of a virtualized infrastructure anomaly detection use-case. Besides hardware acceleration, we consider the specific training time trend when distributing training over a network, by using empirical piece-wise linear distributions. We model the placement problem as a MILP and we propose a variant of the problem. Simulation results show the impact that hardware acceleration can have in the decision of the number of AIF to enable, while dividing by a relevant factor the distributed training time. We also show how our approach exacerbates the importance of monitoring an end-to-end learning system delay budget composed of link propagation delay and distributed training time in the location of AIFs.**

***Index Terms*—Artificial intelligence function, network management, federated learning.**

I. Introduction

Closed-loop network automation systems are being defined to support autonomous reconfiguration of current and future converged connect-compute infrastructure stacks. By actively monitoring the software and hardware components of a network service infrastructure, network automation systems can support routing optimization, network functions (NF) auto-scaling, fault recovery, and anomaly detection.

Indeed, anomaly detection is a fundamental brick of a general purpose automation system. Many frameworks exist, such as the one presented in [1], taking into consideration a potentially large size (e.g. thousands) and heterogeneous set of monitoring time-series. In order to make their deployment feasible in access networks, we consider the usage of distributed learning, as presented in [2] and [3]. Here, we introduce the novel concept of Artificial Intelligence Functions (AIFs) to refer to the AI-enabled end-to-end applications subcomponents that can be deployed across edge-enabled B5G and 6G networks. In such a distributed learning setting, multiple AIFs need to be distributed close to, or at the place where monitoring data is generated, to run distributed continuous learning in support of low latency runtime inference.

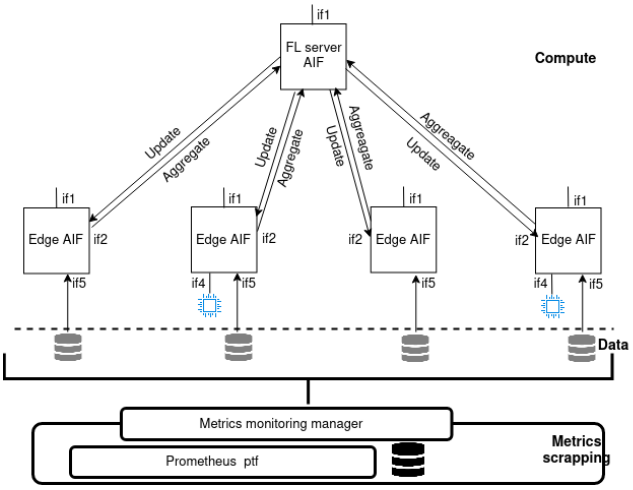


Fig. 1: FL-based anomaly detection AIF systems.

In this paper, we address the problem of placing AIFs running federated learning against connect-compute network infrastructure monitoring data, for environments where the introduction of edge computing comes with a heterogeneous and large set of computing and networking elements, requiring low latency performance. Among the multiple distributed learning techniques proposed in the literature, federated learning [4] represents a good compromise between the need to distribute the learning and to guarantee a centralized view in support of efficient inference, and receives large industry support and integration, including in the 3GPP standards [5].

In particular, we use as reference use-case the federated learning anomaly detection AIF proposed in [2]; it is a distributed variant of the framework proposed in [6], adapted for the 5G infrastructure. This federated learning framework makes use of a federated learning server AIF, and a variable number of edge AIFs: the learning task is distributed to edge AIFs by load-balancing monitoring data among them, where edge AIFs interact via the server for learning model updates. We present the reference distributed anomaly detection AIF system in Figure 1.

Our contributions are as follows:

- We propose a MILP (Mixed-integer linear programming) formulation for the placement of AIFs using a federated learning setting. The proposed model takes into consid-

eration (i) the FL server location, (ii) the latency budget covering training and communication delay components, and (iii) the respect of a target learning time;
- We assess the impact of using hardware accelerators on a subset of edge nodes in order to reduce the training time, and how the deriving latency unbalance can be compensated in the placement outcome;
- We compare the proposed approach to a variant of the AIF placement model in terms of achievable learning time and number of deployed AIFs.

The paper is organized as follows. We give an overview about existing works in Section II. We present the AIF placement model and its variant in Sections III and IV. Our experimental results are presented in Section V. We draw conclusions in Section VI.

II. Related Work

Network softwarization technologies made their way into access networks in such a way that not only nowadays network functions are already mostly deployed as virtualized nodes, but also hardware components, for radio and computing systems, are redesigned to be re-programmable by external software and dynamically allocated and shared. The derived landscape is therefore a natural application domain for artificial intelligence, because many new decision making points appear and many monitoring probes are made available 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.

A. Integration of AI in Edge Networks

Incorporating artificial intelligence and machine learning (AIML) techniques in networks is beneficial for a high number of applications, as presented in [7].

About AIML application in anomaly detection and fault management, it commonly consists in detecting abnormal network states and then localizing the root cause. When integrated in closed loop automation framework, an orchestrator can then run a remediation action to come back to a normal working condition. In [6], the authors proposed an 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, meant to guide a tailored remediation action.

Network management tasks have this potential to take advantage from AIML frameworks. On the other side, Multi-access Edge Computing (MEC) allows to bring resources for AIML computing to the edge network where data to be processed is located.

In [8], the authors comprehensively 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. On the other hand, edge computing provides AI with a convenient platform for models training and inferring, with a potential solution on accelerating computations on hardware [9]; hardware acceleration features can be made available pervasively in edge networks, start from radio access and edge computing nodes.

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 [5] (NWDAF) has been proposed by 3GPP to support AIML in 5G core networks. However, many challenges are being discussed regarding the ML training and inferring tasks, such as the input data, the placement of the training and inference models, and if these two operations should be run inside the same nodes.

B. Federated Learning Applications

A largely adopted strategy for performing the distribution of AIML models geographically at the edge network is Federated Learning (FL) [4]. Federated learning aims to prevent data collection aggregation at the central cloud, either for privacy issues or for latency constraints, or even both. FL consists of 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 result in high efficiency in terms of network bandwidth and low latency, besides increased data privacy thanks to data locality; the FL process itself can be repeated several rounds until the model achieves a given accuracy.

The breakthrough of FL paradigm gave birth to its application in many fields. For standard systems, in [10], the authors proposed a tailored structure for NWDAF function based on FL paradigm w.r.t 3GPP standards: each 5G core NF has its own NWDAF function called the NWDAF leaf, collecting data from its corresponding NF and then training the ML model locally. The root NWDAF in their architecture refers to the FL server and ensures the aggregation of the local model parameters.

In [2], the authors proposed a distributed version of the anomaly detection framework developed in [6] using FL paradigm. 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 for allowing a near-real time re-orchestration decision.

Therefore, a number of AIFs making use of FL primitives are expected to make their way into edge network architecture, and standards are already preparing to this integration. Our work considers the FL clients selection w.r.t a given target learning time while including the hardware acceleration at the training phase. We introduce the AIFs and their placement while considering jointly their processing (training) time and the communication delays with the FL server. We also consider the activation of the hardware accelerators in support of the training effort.

III. Artificial Intelligence Function Placement

We define the AIF placement as the problem of finding the optimal number of AIFs and their location on a given network graph, taking into account the inter-AIF communication patterns, the target learning loop time performance and the presence of hardware acceleration.

The functional architecture of an AIF is given in Figure 2. To support its operation, we identify five interfaces:

- 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 server AIF and/or edge AIFs);
- if2: used for AI model parameter exchange among AIFs (AIF control plane interface), e.g., the communication between edge AIFs and server AIF in federated learning;
- if3: used for data exchange among different AIFs (AIF data-plane interface) - which may be used for generic distributed learning, in the case of an AIF forwarding graph. if3 is not used in the case of FL;
- if4: hardware acceleration interface with components as GPU and Smart-NICs, for training and inference tasks;
- if5: for data collection and streaming, to interface with a data-pipe-lining system. In this work, data pipelining delay is negligible.

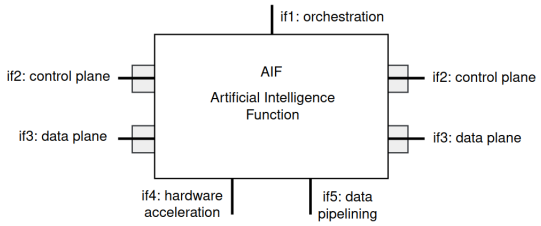


Fig. 2: AIF reference representation.

Moreover, the propagation delays over the link between candidate edge AIF locations and server AIF are not negligible with respect to the training time, and the servers hosting AIFs offer them the same computing capacity.

A. Empirical federated learning time distributions

In order to build a purpose-built model of the training time as a function of the number of federated learning nodes that are used, we run the FL-based anomaly detection framework proposed in [2] which consists of training LSTM (Long-Short Term Memory) autoencoders in a distributed manner, to learn and reconstruct the normal working conditions of a given system. To do so, we run a set of AIFs on a Kubernetes infrastructure [11] where the placement is done automatically using a kubernetes scheduler. In fact, each AIF is an implementation of a LSTM model. The model is composed of a set of autoencoders that allow to detect anomalies at different system levels, i.e., physical level, virtual level and access level, using different groups of metrics, e.g. CPU, memory and radio metrics.

We use the 5G3E dataset from [12], providing few dozens of feature time-series for each resource group, where groups are CPU, RAM, network link state, storage resources, and RAN and UE nodes. We train the ML model using data batches of 800 samples each, corresponding to 2 minutes of collected data for each data batch: this is the assumed retraining time of the system, which could vary in general depending on the sampling rate. The batch size is set to the data size, hence considering all the samples. The number of epochs is 10 and the model is trained for one round. The rest of the FL-based anomaly detection AIF hyper-parameters are the same as in [2].

In Figure 3, we present the corresponding training time distribution as a function of the number of active AIFs based on CPU resources (collected at the container level).

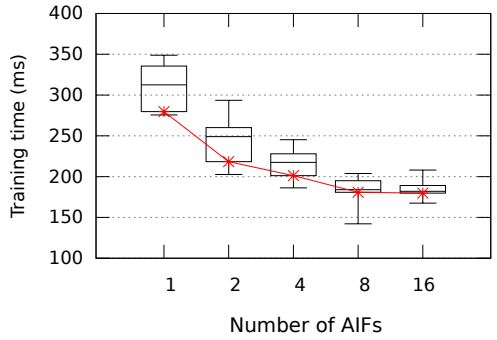


Fig. 3: Training time vs number of AIFs. Number of rounds $R = 1$, number of epochs $E = 10$.

We can remark that the training time decreases with the increase of the number of AIFs up to a certain threshold value. A certain variance exists, due to CPU scheduling and storage system systemic variations at the operating system level. We employ in the AIF placement model the piece-wise linear function obtained using the first quartile values, indicated in red in Figure 3. In the following, we show how we rely on the obtained results to build the AIF placement model.

B. Problem formulation

We consider a set N of physical servers with the same computing capacity, each server can host at most one AIF. The communication latency between edge AIFs and the FL-server depends on the placement decision. Whereas, the distributed training time t is a decreasing function of the number of AIFs, and does not depend on the AIF location. The main goal is to minimize the number of active AIFs used for training, while guarantying that the overall learning loop time is at most T, or equivalently, that the time of a single FL round is not longer than $\frac{T}{R}$, where R is the total number of rounds.

The time of a single round depends on the distributed training time and the transmission latency of the ”slowest” AIF and can be calculated as follows (if no hardware acceleration is considered):

$$p_k + \max_{i \in N}\{d_{ij}\}$$

where d_{ij} is the transmission latency of an AIF hosted by the node $i \in N$ when the FL server is hosted by node j, and p_k represents the distributed processing time when k AIFs are installed. The parameter p_k is defined using the linear approximation on the values given by Figure 3.

Furthermore, we introduce the possibility of using a limited number of hardware accelerators; each one can reduce the distributed training time of an AIF of a factor α.

1) Mathematical model: We need to determine the AIFs location (edge and FL server) and the placement of hardware accelerators to calculate the distributed learning time and the latency associated to each AIF. We introduce these elements gradually in the model presentation.

For the placement, we introduce binary variables $y_j, j \in N$, that take value 1 if the FL server is installed on node j and 0 otherwise, and binary variables x_i that take value 1 if an AIF is activated on node i and 0 otherwise. The objective function (1) minimizes the number of activated AIFs. Constraints (2) impose that one FL server is installed, and Constraints (3) that the same node does not host both the FL server and an edge AIF. Constraints (4) impose that at least two AIFs are installed to ensure the training task in a federated manner.

Constraints (5) guarantee that the time of a single round for each AIF is below $\frac{T}{R}$. The constraint involves the following variables and parameters:

- variable t that represents the training time before hardware acceleration
- variable δt_i that represents the gain in processing time due to the hardware accelerator (δt is zero if no hardware accelerator is used on node i)
- binary variable $\xi_{ij}, i \in N, j \in N$ that takes value 1 if an AIF is installed on node i and the FL server is installed on node j, and 0 otherwise; it allows to calculate the transmission latency;
- parameter D that represent the maximum accepted latency between the FL server and any AIF.

If the AIF is not activated, i.e. $x_i = 0$, the constraint is always valid, as the right-hand side of the constraint reduces to $\frac{T}{R} + p_1 + D$[1]

Constraints (7) allow to calculate the distributed learning time without acceleration[2] using:

- an auxiliary parameter $\tilde{p}_k$ representing the gain in processing time passing from $k-1$ to k AIFs: $\tilde{p}_k = p_k - p_{k-1}$. If, by convention, we set $\tilde{p}_1 = p_1$, the processing time when k AIFs are active can be calculated as follows: $p_k = \sum_{i=1}^{k} \tilde{p}_i$.
- the counting variable z_k, with $k = 1..n$. Variable z_k assumes value 1 if at least k AIFs are activated (or equivalently if the k-th AIF is activated). Note that there is no correspondence with the indexing of the variables z and x.

Constraint (8) allows to count the number of active AIFs, linking z and x variables; constraints (9) impose consistency on the values of z_k.

$$\min \sum_{i \in N} x_i \tag{1}$$

$$\sum_{j \in N} y_j = 1 \tag{2}$$

$$y_i + x_i \leq 1 \quad \forall i \in N \tag{3}$$

$$\sum_{i \in N} x_i \geq 2 \tag{4}$$

$$t - \delta t_i + \sum_{j \in N} d_{ij}\xi_{ij} \leq \frac{\mathrm{T}}{R} + (1 - x_i)(p_1 + D) \quad \forall i \in N \tag{5}$$

$$\delta t_i \leq (1 - \alpha)t \quad \forall i \in N \tag{6}$$

$$t = \sum_{k=1}^{n} \tilde{p}_k z_k \tag{7}$$

$$\sum_{k=1}^{n} z_k = \sum_{i \in N} x_i \tag{8}$$

$$z_k \geq z_{k+1} \quad \forall k = 1..n-1 \tag{9}$$

$$\xi_{ij} \leq y_j \quad \forall i \in N, j \in N \tag{10}$$

$$\sum_{j \in N} \xi_{ij} = x_i \quad \forall i \in N \tag{11}$$

$$\delta t_i \geq (1 - \alpha)t - (1 - w_i)(1 - \alpha)p_1 \quad \forall i \in N \tag{12}$$

$$\delta t_i \leq w_i(1 - \alpha)p_1 \quad \forall i \in N \tag{13}$$

$$w_i \leq h_i \quad \forall i \in N \tag{14}$$

$$\sum_{i \in N} w_i \leq H \tag{15}$$

$$t \geq 0 \tag{16}$$

$$\delta t_i \geq 0 \quad \forall i \in N \tag{17}$$

$$x_i, y_i, w_i \in \{0, 1\} \quad \forall i \in N \tag{18}$$

$$z_k \in \{0, 1\} \quad \forall \quad k = 1..n \tag{19}$$

$$\xi_{ij} \in \{0, 1\} \quad \forall i \in N, j \in N \tag{20}$$

Constraints (10) ensures that for each node i, the only ξ that can be activated is the one corresponding to the node of the installed server. Finally, coherence between ξ and x variables is enforced by Constraints (11)[3].

The possibility of using a hardware accelerator is represented by binary variable w_i that assumes value 1 if the hardware accelerator on node i is used, 0 otherwise. The time reduction δt_i is determined by Constraints (12) and (13) as a function of the learning time t and the accelerator factor $0 < \alpha < 1$. If

[1] by definition $t \leq p1$ and $d_{ij} \leq D$.

[2] t can be removed by substitution, but we kept it for the sake of clarity

[3] using these constraints, variables x can be removed, but we kept for the sake of clarity

$w_i = 1$ the time reduction $\delta t_i = (1 - \alpha)t$, otherwise is equal to 0.

We assume that only a set of physical nodes are provided with hardware accelerator, represented by the indicator parameter h_i, its value is 1 if a hardware accelerator is available on node i, and 0 otherwise: constraints(14) enforce this condition. Furthermore, a maximum number of hardware accelerator H can be installed, as imposed by Constraint (15).

IV. Variant of the AIF placement problem

We propose a variant of the placement model that considers the FL update arrival time variance in the objective. We calculate the difference between the highest and the lowest learning loop times[4], where a learning loop encompasses edge AIF training and the transmission of the new learning metrics to the FL server. The main objective of this model is to reduce the straggler effects [13], where the aggregation after each round of training depends on the slowest AIF: data coming too late at the FL server due to an excessive propagation delay, a too long edge AIF training time, or the combination of the two, is not counted at a given round, hence decreasing the training quality.

We add two variables $maxT$ and $minT$, representing respectively the highest and the lowest learning time produced during the training taking into account the communication delay with the FL server. This relationship is enforced by the the following constraints:

$$maxT \geq t - \delta t_i + \sum_{j \in N} d_{ij}\xi_{ij} \quad \forall i \in N \tag{21}$$

$$minT \leq t - \delta t_i + \sum_{j \in N} d_{ij}\xi_{ij} \quad \forall i \in N \tag{22}$$

Then, we introduce a new term in the objective function which represents the difference between the maximum and the minimum training time. The new objective function is presented by equation (23), where parameters A and B are non-negative reals.

$$\min\ A \cdot \sum_{i \in N} x_i + B \cdot (maxT - minT) \tag{23}$$

V. Experimental results

In the following, we detail the experimental setting, then we provide a preliminary evaluation of our placement model along with a comparison with the model variant.

A. Simulation setting

We use the Mandala topology proposed in [14] to simulate a real access network topology: it consists of connecting access nodes through three tiers, i.e., aggregation, core and application nodes. In our setting, the edge servers represents the MEC hosts in a MEC system and it corresponds to a MAN (Metropolitan Area Networks) topology or a near edge AIF deployment (see figure 4).

[4]here the learning time refers to the duration needed before inference and it includes both the training time and communication delays.

We solve the AIF placement problem using the following formulations:

- Baseline: simplified case where no hardware acceleration is considered;
- AIF-P: placement MILP formulation presented in Equations (1) (20);
- AIF-P-var: MILP variant presented in Section IV. After several preliminary tests with different values of A and B, we opted for $A = 1$ and $B = 10^3$. These values allows the variance to have an impact on the solutions and, thus, to differentiate the behaviour of the two formulations.

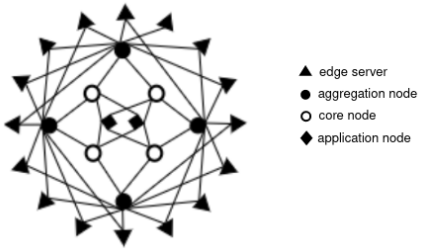


Fig. 4: Mandala topology [14].

We analyze the impact of the following parameters on the behavior of the three aforementioned formulations:

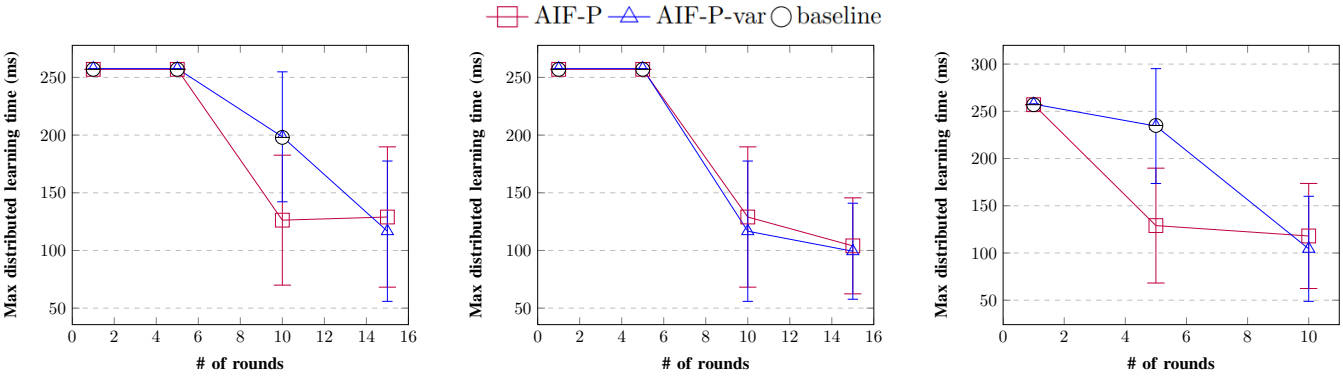
- Number of rounds: we test different rounds of FL training, i.e., 1, 5, 10 and 15 before the inference step. In fact, increasing the number of rounds helps increase the quality of the learning.
- Target learning time: we evaluate the proposed algorithm for different target times, i.e., 2 s, 1.6 s and 1.2 s. This parameter specifies the time during which we train the model, before its exploitation for inference.

In order to evaluate the proposed solutions, we generate 15 different configurations for both the placement and the number of hardware accelerators. The number of available accelerators is randomly generated within the interval [10%, 60%] of the total number of nodes, in order to analyze the impact of their availability on the proposed solution.
The latency on the links is randomly chosen such that the highest round-trip time for the shortest path between the most distant nodes is equal to 7% of the lowest training time, with a small part of communication delay with respect to the training time. The acceleration factor α is set to 0.5, so that the training time reduction is not too large nor too small.

B. Results analysis

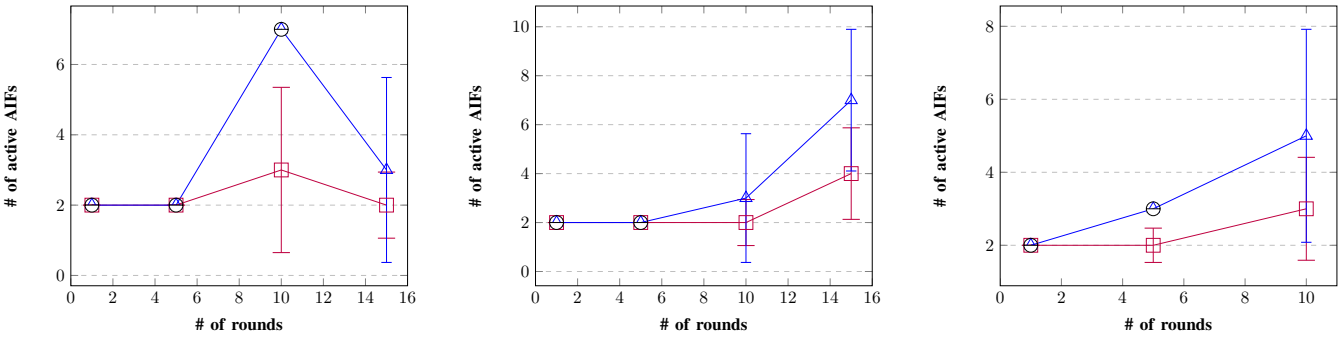
In the following, we present the numerical results.

1) Edge AIF learning time: In Figure 5, we present the average of the maximum edge AIF distributed training time (maximum among all the feasible solutions) for the 15 different hardware accelerator configurations, for different target times (see Figures 5a, 5b and 5c) and using different numbers of rounds. The error bars represent the standard deviation. We observe that:



(a) Target learning time=2 s (b) Target learning time=1.6 s (c) Target learning time=1.2 s

Fig. 5: Max learning time vs number of rounds.



(a) Target learning time=2 s (b) Target learning time=1.6 s (c) Target learning time=1.2 s

Fig. 6: Number of active AIFs vs number of rounds.

- The distributed processing time decreases with the increase of the number of rounds for all target times. In fact, increasing the number of rounds requires a higher number of active AIFs at each round since the increase of this latter decreases the training time (as already discussed in Section III-A).
- In Figure 5a, we can notice that with 15 rounds, and unlike all the other cases, AIF-P suffers a slight increase in the training time when compared to 10 rounds. This apparent contradiction is due to the fact that we consider only the maximum distributed processing time over all the feasible solutions. Some instances that yield solutions with high learning times using 10 rounds are not accounted in the case of 15 rounds since no feasible solution exists for them.
- We can also observe that AIF-P and AIF-P-var have similar behaviors: they can produce a placement solution for all the proposed numbers of rounds for a target learning time equal to 2 s. On the other side, both models are not able to find any feasible solution for some stringent time constraint cases; these unfeasible solutions correspond to a number of rounds higher than 10 for a target learning time equal to 1.2 s. Moreover, AIF-P produces a lower distributed training time than AIF-P-var when $R = 10$ for $T = 2s$ (R = 5 for T = 1.2 s respectively). However, this is not the case anymore when R increases. Differently than AIF-P-var, AIF-P tends to use hardware accelerators instead of increasing the number of deployed AIFs in order to respect the learning time threshold.
- The baseline solution shows a worst performance when compared to the other methods: any feasible solution is found for a number of rounds that surpasses 10 and 5 when the target time is equal to 2 s (5a), and 1.6 s and 1.2 s respectively (5b and 5c).

These results confirm the usefulness of hardware accelerators during the training phase. In fact, the reduction in the distributed training time is between 50% and 59% for the two approaches using hardware acceleration.

2) Number of active AIFs: In Figure 6, we present the average number of active AIFs chosen by the different formulations, while varying both the number of rounds and the target learning time. The length of the error bars represents the variation in the number of AIFs produced by the different 15 instances for the same setting, i.e, same number of rounds and the same target learning time.

- As expected, the plots show an inverse relationship between the distributed processing time and the number of deployed AIFs (Figures 5 and 6). All the strategies show

an increase in the number of AIFs with the increase of the time constraints, except for the case of 15 rounds with a target time equal to 2 s using AIF-P and AIF-P-var, as shown in Figure 6a. As mentioned before, we consider the average of the obtained values only for the feasible solutions. For some settings, no feasible solution was found with 15 rounds. When T =2 s and $\mathrm{T} = 1.6$ s (respectively $\mathrm{T} = 1.2$ s) for a number of rounds equal to 10 (respectively 5), AIF-P deploys a lower number of AIFs than AIF-P-var, however the distributed training time is lower with AIF-P. This can be explained by the fact that this latter uses (more) hardware accelerators to reduce the distributed learning time instead of increasing the number of AIFs.

- Figures 6b and 6c show that the baseline algorithm cannot find any feasible placement solution when the number of rounds exceed 5. This can be explained by the increased number of exchanges between the AIFs and the FL server which has a direct impact on the overall learning time. In this case, finding a feasible solution with respect to the imposed target times is not possible. On the other hand, AIF-P and AIF-P-var show higher performance in placing the AIFs with stringent time constraints thanks to the use of hardware accelerators.
- It is worth mentioning that there exist some settings where AIF-P and AIF-P-var do not provide feasible solutions. This happens when we have strict time constraints. For instance, through the 15 instances, both algorithms provide 30% of unfeasible solution with $\mathrm{T} = 2$ s and $R = 15$. This is related to the placement and the number of hardware accelerators on network nodes provided for each instance.

The possibility to use hardware accelerators reduces the per-round learning time, hence offering more flexibility in placing and activating AIFs and consequently training the model for a higher number of rounds. Also, the use of advanced formulations may have less benefit if the baseline strategy achieves a feasible solution. Their benefit can be shown when the constraints on target time are more stringent or the number of rounds is increased.

VI. Conclusion and Perspectives

In this paper, we tackled the problem of artificial intelligence function placement in a federated learning environment where hardware accelerators can be used.

We proposed a MILP formulation that takes into consideration several challenges, mainly the location of FL nodes and the communication and processing delays. We then proposed a variant of the model and presented preliminary results on the performance evaluation of the proposed solutions while comparing them to a baseline placement solution.

We have shown that the baseline solution has inferior performance in finding feasible placement solutions when compared to the two more flexible models. On the other hand, our strategies show similar performance in finding feasible solutions but with different behaviors, i.e., one reduces the learning time and increases the number of active AIFs while the other behaves in the opposite.

The choice between these two strategies can be based on time constraints and the overall quality of learning, which decreases if data is highly distributed, preventing the local model from having a sufficient view of the system.

The obtained results shows that the proposed models allow to increase the number of rounds from 150% up to 300% when compared to the baseline placement solution thanks to hardware acceleration.

Future works may investigate including multiple objectives in a Pareto efficient way. We also plan to apply other distributed learning behavior than federated learning for artificial intelligence functions.

VII. Acknowledgment

This work was funded by the H2020 AI@EDGE (https://aiatedge.eu; grant nb. 101015922) and the PIA/AMI-5G INFLUENCE projects.

References

[1] A. Diamanti, J. M. S. Vilchez and S. Secci, "The SYRROCA AI-empowered network automation platform", ICIN, 2021.
[2] S. Bin Ruba, N-E-H. Yellas and S. Secci, "Anomaly Detection for 5G Softwarized Infrastructures with Federated Learning", 6GNet 2022.
[3] Y. Liu et al., "Deep Anomaly Detection for Time-Series Data in Industrial IoT: A Communication-Efficient On-Device Federated Learning Approach", IEEE Internet of Things Journal, 2021.
[4] Mcmahan et al. Communication-efficient learning of deep networks from decentralized data. In : Artificial intelligence and statistics. PMLR, 2017.
[5] 3GPP TS 23.288, Architecture enhancements for 5G System to support network data analytics services, v. 17.1.0, Jun. 2021.
[6] Diamanti Alessio, Vilechez Jos Manuel Snchez, et SECCI, Stefano. "An AI-empowered framework for cross-layer softwarized infrastructure state assessment" IEEE TNSM, 2022.
[7] R. Boutaba et al., A comprehensive survey on machine learning for networking: evolution, applications and research opportunities. Journal of Internet Services and Applications, 2018.
[8] S. Deng et al., "Edge Intelligence: The Confluence of Edge Computing and Artificial Intelligence", in IEEE Internet of Things Journal, 2020.
[9] J. Lee et al., Adaptive precision CNN accelerator using radix-X parallel connected memristor crossbars, 2019. [Online]. Available: arXiv.abs/1906.09395.
[10] Y. Jeon et al., "A Distributed NWDAF Architecture for Federated Learning in 5G", ICCE, 2022.
[11] Kubernetes. URL: http://kubernetes.io.
[12] D. Chi Phung et al., "An Open Dataset for Beyond-5G Data-driven Network Automation Experiments", *in proc.* 6GNet 2022.
[13] S. Michael et al., "Asynchronous Federated Learning for Geospatial Applications", in Joint European Conference on Machine Learning and Knowledge Discovery in Databases, 2019.
[14] W. da Silva Coelho et al., "On the impact of novel function mappings, sharing policies, and split settings in network slice design", CNSM, 2020.