On the Optimization of Model Aggregation for Federated Learning at the Network Edge

Mengyao Li[1], Noah Ploch[2], Sebastian Troia[1], Carlo Spatocco[1], Wolfgang Kellerer[2], Guido Maier[1] [1]*Politecnico di Milano, Dipartimento di Elettronica, Informazione e Bioingegneria (DEIB), Milan, Italy*
[2]*Technical University Munich, School of Computation, Information, and Technology, Munich, Germany*

***Abstract*—The rapid increase in connected devices has significantly intensified the computational and communication demands on modern telecommunication networks. To address these challenges, integrating advanced Machine Learning (ML) techniques like Federated Learning (FL) with emerging paradigms such as Multi-access Edge Computing (MEC) and Software-Defined Wide Area Networks (SD-WANs) is crucial. This paper introduces online resource management strategies specifically designed for FL model aggregation, utilizing intermediate aggregation at edge nodes. Our analysis highlights the benefits of incorporating edge aggregators to reduce network link congestion and maximize the potential of edge computing nodes. However, the risk of network congestion persists. To mitigate this, we propose a novel aggregation approach that deploys an aggregator overlay network. We present an Integer Linear Programming (ILP) model and a heuristic algorithm to optimize the routing within this overlay network. Our solution demonstrates improved adaptability to network resource utilization, significantly reducing FL training round failure rates by up to 15% while also alleviating cloud link congestion.**

***Index Terms*—Federated Learning (FL), Model Aggregation, Multi-access-Edge Computing (MEC), Software Defined Wide Area Network (SD-WAN)**

I. Introduction

The ever-increasing volume of data generated by diverse devices and applications presents a significant challenge to existing computer networks. This highlights the urgent requirement for scalable and adaptable networking solutions that extend beyond the existing cloud-centric infrastructure [1].

Let's consider a smart city with a multitude of Internet of Things (IoT) devices continually gathering data from sensors integrated into various infrastructure elements, including but not limited to traffic lights, environmental sensors, and surveillance cameras. Concurrently, applications operating on these devices analyze this data in real time to provide insights or take actions. These actions may range from optimizing traffic flow to identifying irregularities or augmenting public safety measures. The substantial volume of data generated by these devices and applications daily warrants attention. The conventional method of consolidating all data processing activities within a centralized cloud data center is increasingly deemed impractical for several compelling reasons, such as latency and bandwidth. In the first case, some applications require low-latency responses, e.g. real-time traffic management or emergency response systems. In the second, transmitting massive volumes of data from numerous IoT devices to a centralized cloud server strains the available bandwidth, leading to congestion and potential bottlenecks.

Multi-access Edge Computing (MEC) emerges as a game-changer networking paradigm, strategically leveraging the computational and storage capabilities of end devices (clients) and edge servers. MEC aims at bringing Machine Learning (ML) model training closer to the point of data generation, facilitating data processing outside traditional cloud data centers. MEC uses computational and communication resources situated at the edge of the network, thereby establishing intermediary network resources between end devices, such as IoT devices, and the cloud [2]. As such, it enables mobile devices to run highly demanding applications while meeting strict delay requirements.

With the proliferation of MEC-based applications, users' expectations of Wide Area Network (WAN) connectivity performance have evolved dramatically. Beyond mere connectivity, also unwavering reliability, agility, and exceptional performance should be expected on WAN infrastructure. Software-Defined Wide Area Network (SD-WAN) comes into play as a transformative paradigm shift in enterprise and business networking. At its core, SD-WAN is a software-based solution that promises streamlined deployment, elevated connectivity, and centralized control. It empowers organizations to dynamically manage their WANs through virtualization technology. In short, SD-WAN stands out for its remarkable capability to swiftly establish overlay networks, often within several seconds. Through Virtual Private Network (VPN) tunnels, SD-WAN can dynamically create network overlays, enabling organizations to adapt quickly to changing network demands and configurations [3]. The agility offered by SD-WAN in creating overlay networks is particularly advantageous in environments where flexibility and scalability are important. In scenarios where new branch offices or far-edge devices (e.g. environmental sensors) need to be connected promptly or where temporary network configurations are required, SD-WAN excels in providing rapid end-to-end connectivity deployment without the need for extensive manual configuration. However, it is important to acknowledge that the speed of deployment with SD-WAN may come at the expense of certain network guarantees, such as latency and bandwidth assurances. While SD-WAN can rapidly establish connections, the reliance on VPN tunnels may introduce variability in network performance and susceptibility to fluctuations in network conditions [4].

While a majority of current ML applications remain cloud-

centric, the use of the cloud for ML introduces challenges such as constant data uploading, leading to core network congestion and unacceptable latency [5].

Federated Learning (FL) has gained increasing attention due to its ability to collaboratively train a global ML model in a distributed manner without compromising the privacy of data held by individual clients [6]. FL operates as a form of distributed ML, wherein multiple data owners collectively contribute to the training of a global model, thereby avoiding the need to share their raw data. Within the FL network architecture, entities include clients end devices, aggregators, and servers. Client devices store their original data locally and refrain from exchanging or transferring it. Instead, each local learner leverages the datasets of other learners solely through the global model, which is shared by the aggregator, without direct access to privacy-sensitive data [7]. The FL workflow involves each device's local data for training, and then transmitting the trained model to the server for aggregation. Finally, the server distributes the updated model to all clients to advance the collective learning objective [8].

Recently, the embedding of FL at the network edge has attracted considerable attention both in the Industry and Academia. This interest stems from the benefits offered by both MEC and SD-WAN, such as improved efficiency in communication resource usage and enhanced user privacy [5]. However, implementing FL at the network edge entails numerous challenges due to the heterogeneity of the environment as compared to traditional data centers. These challenges encompass various aspects. Firstly, edge-to-cloud communication may suffer from slower and unstable connections, necessitating robust solutions to mitigate potential latency and reliability issues. Additionally, ensuring the trustworthiness of edge nodes becomes paramount to safeguard the integrity of FL processes, given that edge devices may be more susceptible to security vulnerabilities [5]. Furthermore, addressing the heterogeneity inherent in network and computing resource specifications at the edge requires the implementation of intelligent resource management strategies [9]. These strategies are crucial for optimizing FL performance while accommodating variations in edge device capabilities and network conditions.

In this work, we explore the deployment of FL within a MEC network enhanced by SD-WAN technology, with a particular focus on resource management during the FL training phase, as illustrated in Fig.1. Unlike conventional FL algorithms that primarily focus on aggregating local models while emphasizing privacy preservation and low latency, our proposed approach introduces a novel perspective by addressing the unique challenges posed by SD-WAN-enabled MEC environments. Existing FL algorithms typically do not consider the complexities of network connectivity, which is important in our scenario. Our approach takes into account the constraints of WAN connectivity, which is managed at the edge by SD-WAN, alongside the stringent limitations of both computational resources and network capacity. To optimize resource utilization, we perform aggregation of local models directly on strategically selected edge nodes, thereby significantly reducing tunnel capacity costs associated with cloud transmissions. To thoroughly evaluate and compare various aggregation strategies, we have developed a Discrete-Event Simulator (DES) capable of accurately modeling resource consumption behaviors in FL applications under different aggregation algorithms.

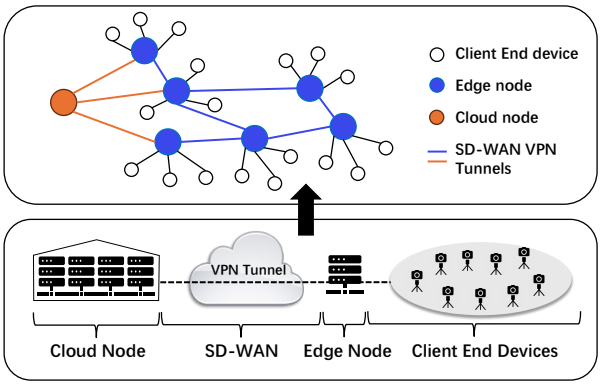


Fig. 1: Architecture of FL with MEC enhanced by SD-WAN.

This work is supported by the WatchEDGE [10] project, whose goal is to study an advanced edge computing architecture distributed over several geographically-distant sites. Each site represents an "island" equipped with edge-computing capabilities (in-network storage and processing). The islands are interconnected to each other exploiting WAN connectivity controlled at the edge by an SD-WAN. The project addresses the challenge of providing an overall orchestration of applications and resources in the WatchEDGE infrastructure in a fully distributed way. Environmental surveillance and wildlife protection serve as use case for the proposed architecture, inducing the need for AI-based image processing on heterogeneous devices, see Fig.1.

The contribution of this paper is two-folded:

- We propose an Integer Linear Programming (ILP) formulation of the edge-to-cloud FL model aggregation problem. Then, we propose a heuristic algorithm called Hierarchical Federated Edge Learning Mesh (HFEL-MESH) in order to cope with the scalability problem posed by the ILP. To validate the efficacy of our proposed methods, we conduct a thorough comparative analysis against a state-of-the-art algorithm as referenced by [11].
- We have developed a Python-based Discrete Event Simulator (DES) tailored specifically for the WatchEDGE project. This simulator is customized to emulate the infrastructure and applications inherent to the project's ecosystem, with a specific focus on AI algorithms. At its core, the simulator aims at replicating the complexity of the WatchEDGE infrastructure while facilitating the simulation of AI-based applications, particularly those related to FL for image recognition tasks at the edge.

The rest of the paper is organized as follows. Sec. II discusses the related work within FL and edge networking/computing topics. Sec. III introduces the problem statement, the ILP model, and a novel scalable heuristic algorithm. Sec. IV introduces the WatchEDGE simulator. Sec. V discusses the numerical analysis. Sec. VI concludes the paper.

II. Related work

Federated learning, introduced by McMahan et al. [6], emerged as a privacy-preserving distributed ML method. By transmitting only model weights rather than raw data, it not only preserves user privacy but also significantly reduces communication costs. In this section, we delve into the related work concerning the integration of FL within MEC networks. FL operates as an encrypted distributed ML technology, allowing clients to build their own models without disclosing their data, thus safeguarding the privacy of each client's proprietary data. Subsequently, the local model parameters of each client are communicated to a central entity, where a global common model is derived [8]. Numerous studies have explored FL, particularly focusing on Hierarchical Federated Learning (HFL). It consists of an extension of the FL paradigm that introduces a hierarchical intermediate structure among clients. In HFL, clients are organized into multiple levels, with each level representing a different layer of aggregation. The following studies encompass HFL strategies in terms of network and computing resource placement strategies.

Qi et al. [12] presented a systematic literature review on model aggregation in FL. The focus is on summarizing the proposed techniques and the ones currently applied for model fusion.

Wu et al. [13] discussed the challenges and future works for applying FL to topology-specific edge networks.

Brecko et al. [14] presented FL frameworks that are currently popular and that provide communication between clients and servers including basic models and designs of system architecture, possibilities of application in practice, privacy and security, and resource management.

Liu et al. [11] incorporated the utilization of computing resources at the network edge by exploiting HFL. As such, edge computing facilities play a crucial role in conducting intermediate aggregation of models. By strategically adjusting the frequency of partial aggregation at the edge servers and global aggregation in the cloud, they demonstrated that HFL effectively minimizes energy consumption at end devices while simultaneously reducing model training time. This hierarchical approach leverages the proximity of edge resources to devices, optimizing the efficiency of FL in distributed environments.

Luo et al. [15] introduced a novel Hierarchical Federated Edge Learning (HFEL) framework, where model aggregation is partially migrated to edge servers from the cloud. HFEL formulates a joint computation and communication resource allocation problem for device users to achieve global cost minimization.

Xu et al. [16] considered cost minimization associated with joint worker aggregator placement and client assignment problems in a MEC network, optimizing client clustering, aggregator placement, and server location simultaneously.

Chen et al. [17] described an FL approach over multi-hop wireless networks, and optimized FL over wireless networks by taking into account the heterogeneity in communication and computing resources at mesh routers and clients.

Various studies have proposed cluster-based FL mechanisms, focusing on hierarchical aggregation and convergence-bound analysis while considering factors such as energy cost, accuracy, latency, and time limitations. For instance, Wang et al. [18] show a cluster-based FL mechanism that emphasizes hierarchical aggregation; their analysis highlights that the convergence bound depends on the number of clusters and on the training epochs, with a primary focus on latency and compute time optimization.

Feng et al. [19] focused on minimizing the energy cost while addressing constraints such as delay, local CPU-cycle frequency, power allocation, local accuracy, and subcarrier assignment.

Meanwhile, Dinh et al. [20] provided a convergence rate analysis, illustrating the trade-off between local computation rounds and global communication rounds in FL. They employed FEDL in wireless networks as a resource allocation optimization problem that captures the trade-off between FEDL convergence time and energy consumption of clients with heterogeneous computing and power resources.

Wu et al. [21] introduced HiFlash, a framework that integrates deep reinforcement learning-based adaptive staleness control and heterogeneity-aware client-edge association strategy to boost the system efficiency and mitigate the staleness effect without compromising model accuracy.

Du et al. [22] proposed the FedMT algorithm to minimize training time through optimized client selection and routing. Their work focuses on reducing upload time per iteration within an SD-WAN topology. However, they did not explore scenarios involving intermediate computing capabilities, distinguishing our research focus.

Wang et al. [23] proposed a minimum spanning tree (MST)-based scheduling approach that considers both resource and latency costs in Federated Learning (FL) training. Their model utilizes the shortest path to determine the optimal aggregation routing operations. Additionally, they introduce scheduling strategies [24] aimed at enhancing communication efficiency for distributed AI tasks. Their MST-based strategy dynamically determines routing paths and aggregation operations to improve the overall efficiency of the FL process.

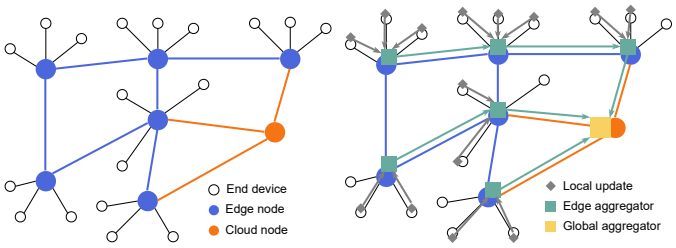
Previous research has introduced innovative FL aggregation strategies that leverage edge computing, but a common limitation is the oversight of the edge-cloud overlay network. Specifically, the aforementioned works focus more on optimizing only up to the client-edge connection, neglecting the network scenario up to the cloud. In contrast, our work stands out by enabling model aggregation directly on selected edge nodes, achieved through the optimization of resource allocation within the edge-to-cloud overlay network.

III. Federate Learning Resource Allocation at the Network Edge

A. Reference Network Architecture

The network architecture considered in this paper is derived from the WatchEDGE project [10], depicted in Fig. 2a. Within this architecture, each site comprises edge nodes. These edge sites are equipped with their own computing and networking resources, to which client end devices connect wirelessly or via physical connections. In contrast, cloud sites

symbolize public cloud infrastructure and provide extensive computational capabilities and storage capacity. Inter-site connectivity is facilitated through overlay VPN tunnels, layered atop the physical access network provided by Internet Service Providers (ISPs). These VPN tunnels are centrally managed by an SD-WAN controller. End links establish connections between client devices and edge nodes, while edge links facilitate communication between edge nodes. Cloud links, on the other hand, denote connections from edge nodes to the cloud, enabling data exchange and computational offloading. This network architecture embodies edge computing, leveraging both local edge resources and remote cloud infrastructure to support diverse applications and services.



(a) Example of a WatchEDGE functional topology with multiple clients end devices, edge nodes and a public cloud node.

(b) Aggregator overlay topology scenario. An arrow stands for an update to be routed through the link.

Fig. 2: Exemplary WatchEDGE topology and our proposed scenario.

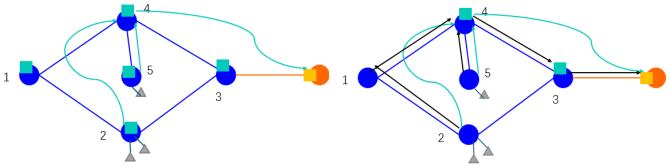
In FL model aggregation, the hierarchical two-level aggregation strategy is widely studied in the literature [15], [16], [18]. Our work extends this approach **by introducing the possibility for an aggregator to report its aggregated update to another aggregator** within the WatchEDGE network, as illustrated in Fig. 2b. This introduces a network of aggregators, improving aggregation efficiency and reducing overall transmission costs. By aggregating models on edge nodes before forwarding them to the cloud, we significantly reduce component capacity costs, particularly on links related to the cloud. If all models were transmitted directly to the cloud for aggregation, it would lead to substantial link capacity usage and potential congestion. However, our approach preserves the flexibility to directly report to a global aggregator in the cloud when necessary. As shown in Fig. 2, our model consists of a physical topology and an aggregator overlay topology. The physical topology represents the physical edge nodes and VPN links managed by an SD-WAN controller. The aggregator overlay topology (auxiliary graph) in Fig. 2b, on the other hand, is used to determine potential aggregation points and find paths to the cloud, with or without bypassing aggregation nodes. While these green nodes represent potential aggregation points, the auxiliary graph allows flexibility in deciding whether to aggregate at each node. In this auxiliary graph, if a path temporarily stops at a green node (edge aggregator) all incoming models are aggregated at that node, producing a single output model. If a path uses an auxiliary link to bypass a green node, it means aggregation does not occur at that location. To optimize this process, we first select the appropriate aggregation nodes (green nodes), which receive model updates from grey end devices and other edge nodes before performing aggregation. Once aggregation locations are determined, the next step is to establish optimal routing paths for model updates, leveraging auxiliary links in the graph (as shown in Fig. 2b) and mapping them to physical routes in the network. This procedure introduces an additional optimization challenge, which we term the *aggregator routing problem*. Our proposed aggregator overlay topology enhances flexibility beyond the traditional hierarchical two-level approach, dynamically adapting to the network's structure. To the best of our knowledge, this specific scenario has not been explored within the context of FL and MEC-based SD-WAN network. This study evaluates the cost of our aggregation method compared to conventional approaches and examines the trade-offs in FL model aggregation for both ILP and our proposed heuristic algorithm. In this study, we employ the Training Round Failure Rate (TRFR) [25] to quantify the probability that not all requests are successfully processed during an FL training phase when a set of requests is submitted to the simulator. In the FL simulator, a request refers to a training round request initiated by far-edge nodes that need to update their models to the cloud due to environmental changes, such as new data (e.g., new images) or a decline in model accuracy. The WatchEDGE infrastructure manages the initiation of these training rounds. In our simulation, requests are generated using the Application Block (AB), following a predefined inter-arrival time distribution modeled as exponential (exp). The AB is responsible for generating requests, while task management is handled by the Event Dispatcher and the Resource Allocation (RA) manager (see Section IV). The system processes all active requests and optimizes TRFR globally by strategically selecting aggregator nodes and routing paths to enhance resource efficiency. As a key performance metric, TRFR provides insights into system resource allocation and task execution effectiveness, allowing us to assess the impact of our proposed resource management strategies. Additionally, we use cumulative weighted capacity as a cost metric (Section III-C), which accounts for total resource consumption. Our objective is to minimize this metric, as it directly impacts efficiency. Together, TRFR and cumulative weighted capacity play an important role in optimizing resource utilization during model training.

B. Problem Statement

We model the network as a bidirectional graph with client nodes, denoted as physical graph $G_p = (N, E)$, where N represents the set of nodes and E represents the VPN tunnels (in order to be short, we call it links in the rest paragraph) in the MEC-based SD-WAN topology. SD-WAN can dynamically create network overlays using VPN channels, enabling organizations to adapt quickly to changing network demands and configuration. In our model, the SD-WAN overlay network is considered as physical topology, and E is considered as physical links (VPN tunnels). The nodes in N are categorized into three types: $N = R \cup L \cup U$, R (cloud servers), U (client end devices), and L (edge node aggregators). Client end devices mean the far edge in the topology, and the Edge

node aggregators mean the edge node which can be considered as the aggregator nodes. In Fig. 2, we define two types of links: Cloud Links (orange): Connecting the last edge nodes to the cloud nodes. Edge Links (blue): Connecting edge nodes to each other. To optimize aggregation and routing, we construct an auxiliary graph where selected nodes serve as edge aggregators with computational resources, while links represent potential connections between both adjacent and non-adjacent nodes. This auxiliary graph, denoted as Ga =(Na,Ea), is fully connected, where Na represents the set of nodes that can be selected as aggregators, and Ea includes all possible links, indicating the potential to bypass certain edge nodes. In the auxiliary graph, we first determine which nodes should act as aggregators and identify the corresponding auxiliary links to establish efficient paths to the cloud. Once the aggregator nodes and auxiliary links are properly selected in the auxiliary graph, the routing is mapped onto the physical topology to ensure efficient data transmission. The procedure for utilizing the auxiliary graph is detailed in Section III.A. The following formula formally describes how to determine routing in both the physical and auxiliary graphs. Our goal is to aggregate the models properly using these two graphs, ensure the low TRFR and low capacity cost.

Our proposed edge-to-cloud FL model aggregation problem can be stated as follows: **Given**: A MEC-based SD-WAN topology consisting of nodes representing clients, edge devices, and cloud servers, as well as a set of VPN channels (in order to be short, we call it links in the rest paragraph) and input requests with an inter-arrival time distribution. **decide**: The optimal aggregation locations for nodes and establish routing and capacity assignment for all the requests in the system at that time. **constrained**: The resource capacity assigned to each node and link. **objective**: Minimizing the cumulative weighted capacity cost in the ILP model (and TRFR in the heuristic algorithm, as shown in Eqn. 11 later), which together reflect the efficiency of resource utilization and the associated costs within the whole network. We address this problem by developing an ILP formulation and a heuristic algorithm called HFEL-MESH, both described in the following sections. While many FL models use TRFR as a primary metric to evaluate system performance, it is important to note that the characteristics of the ILP approach make it unsuitable for directly assessing TRFR. TRFR is tested through simulation, relying on multiple iterations of FL behavior, whereas ILP lacks this capability. Instead, we evaluate ILP performance using cumulative weighted capacity cost, which provides a measurable method within our model's constraints. Training rounds may fail due to (1) the inability to place an aggregator on an edge node due to insufficient resources or (2) the inability to transmit a model update through an edge or cloud link due to link capacity constraints. Since model updates are transmitted based on request demand, link capacity limitations can prevent updates, leading to an increase in TRFR. These training rounds fail are closely tied to overall resource consumption. Given that cloud links often act as bottlenecks, we assign them higher weights in the ILP to minimize their usage and prevent congestion. Meanwhile, the HFEL-MESH algorithm evaluates both TRFR and cumulative weighted capacity cost, ensuring an optimal balance between resource consumption and network performance. By efficiently distributing workloads across edge nodes, HFEL-MESH minimizes TRFR and optimizes overall resource utilization.



(a) Route in auxiliary graph (b) Route in physical graph

Fig. 3: How to route the request.

Let's consider an example to illustrate the routing process. Fig. 3a depicts a physical topology consisting of five nodes and five links, following the same legend as Fig. 2. In this scenario, a request requires aggregating the model from far-edge nodes. Specifically, the far-edge nodes are associated with edge node 2 and edge node 5. The aggregated model must then be transmitted to the cloud. To determine the routing, we first analyze the auxiliary graph, which contains the same five nodes but features a fully connected topology. The chosen routing path is highlighted in green in Fig. 3a. One auxiliary link starts from node 2, bypasses node 1, and reaches node 4. This implies that the model from node 2 will be sent via node 1 to node 4, where it will be aggregated. Similarly, the model from node 5 is directly transmitted to node 4 for aggregation, as indicated by the green link. At node 4, both models are aggregated before being sent to the cloud. Additionally, another green link in the auxiliary graph connects node 4 to the cloud while bypassing node 3, meaning no aggregation occurs at node 3. Next, we determine the physical routing in the physical graph, as shown in Figure 3b. Black arrows indicate the paths taken by the physical links. First, we map the auxiliary link (2,4) onto the physical graph using the path (2,1) $\rightarrow$ (1,4). The auxiliary link (5,4) directly corresponds to the physical link (5,4). Finally, we define the physical route for the auxiliary link (4, cloud) using the path (4,3) $\rightarrow$ (3, cloud).

C. ILP Model Formulation

In this section, we delve into the ILP model formulation. The sets, parameters and variables are shown in Table I, Table II and Table III respectively. In the following we introduce the optimization function and constraints.

The objective function minimizes the cumulative weighted capacity utilized by nodes and links in the physical topology. This is achieved by calculating the total capacity cost of each component, weighted according to its importance, for every served request. We assign specific weights to all components, including both links and nodes, with the highest weight assigned to the cloud link to prevent congestion. Aggregation procedure cost the node capacity, and the model update cost the link capacity. Our goal is to decrease the cumulative weighted capacity cost to prevent the congestion so that to have more capacity to serve the requests and lower the TRFR

as much as possible. The formal definition of the objective function is presented in Eq. 1.

$$\min \quad \alpha_n \cdot \eta_n + \beta_e \cdot \eta_e \quad \forall n \in N \tag{1}$$

where α_n and β_e denote the weights of node n and link e; η_n and η_e measure the capacity occupied for node n and link e. η_n and η_e are restricted by Eq. 8 and Eq. 10. These two equations calculate how much capacity is occupied for each node and each link in the whole system. In order to decrease the congestion on cloud links, their weights are much higher than that of edge links.

To optimize this process, we first use the flow constraint Eqn. 2 to select the appropriate aggregation nodes and the proper routing in the auxiliary graph. Once aggregation locations are determined, the next step is to map the auxiliary route to physical routes in the network. We use Eqn. 3 to map the auxiliary link back to the physical route in the physical topology. An auxiliary link in the auxiliary graph could be consist of many physical link in the physical topology. The constraints in our model define the routing and resource allocation in both the auxiliary and physical graphs. We denote $S^+(i)$ and $S^-(i)$ as the sets of outgoing and incoming links from node i, respectively. Eqn. 2 enforces the flow constraint for routing in the auxiliary graph between each node pair $p \in P$, where p represents the request's source and destination nodes, denoted as $a(p)$ and $b(p)$. The source node typically represents a far-edge node, which needs to update and aggregate its model to the cloud. The destination node refers to the cloud node. The path should start at $a(p)$ and end at $b(p)$. For each auxiliary link $e \in Ea$, $i(e)$ and $j(e)$ refer to the source and destination nodes, respectively, with $j(e)$ being an aggregator. Each auxiliary link in the auxiliary graph may map to one or more physical links in the physical topology. When an auxiliary link is selected for use, its endpoint $j(e)$ is designated as an aggregator. Eqn. 3 enforces a constraint that if an auxiliary link is used, as indicated in Eqn. 2, then aggregation must occur at its destination node $j(e)$. And this node $j(e)$ will be assigned the role of an aggregator node, and it will consume computational resources accordingly. Eqn. 4 determines the physical route for each auxiliary link, mapping it onto the physical topology. Eqn. 5 enforces link capacity constraints to prevent congestion. Eqn. 6 checks whether additional links should aggregate on node j in the physical topology. Eqn. 7 and 8 compute node capacity costs and ensure compliance with node capacity limitations. Similarly, Eqn. 9 and 10 calculate link capacity costs and enforce link capacity constraints, ensuring efficient routing and aggregation within the network.

$$\sum_{e \in S^+(i)} q^k_{e,p} - \sum_{e \in S^-(i)} q^k_{e,p} = \begin{cases} z^k_{a(p)} \ if \ i(e) = a(p) \wedge j(e) \notin R \\ -z^k_{a(p)} \ if \ i(e) = b(p) \\ 0 \quad others \end{cases}$$
$$\forall p \in P, i \in N, j \in N - R, k \in K \tag{2}$$

$$q^k_{e,p} \leq \mu^k_{j(e)} \leq \sum_{k \in K} q^k_{e,p} \quad \forall e \in E_a, k \in K, j(e) \in L \cup R, p \in P \tag{3}$$

$$\sum_{e' \in S^+(i)} x^k_{e',e} - \sum_{e' \in S^-(i)} x^k_{e',e} = \begin{cases} \mu^k_{i(e')} \ if \ i(e') = a(e) \\ -\mu^k_{i(e')} \ if \ i(e') = b(e) \\ 0 \quad others \end{cases} \tag{4}$$
$$\forall e \in E_a, i \in N, k \in K$$

TABLE I: Sets description for ILP model.

Sets	Description
E	Set of links in the physical network
E_a	Set of links in the auxiliary graph
K	Set of requests
L	Set of edge nodes in the network
N	Set of nodes in the network
N_a	Set of nodes in the auxiliary graph
P	Set of node pairs (client - cloud couples) for paths in topology
R	Set of cloud servers in the network
U	Set of user clients in topology

TABLE II: Parameters description for ILP model.

Para	Description
C_n	Integer, capacity available on node n
C_e	Integer, capacity available on link e
α_n	Float, the weight of the nodes capacity
β_e	Float, the weight of the links capacity
λ	Float, input request arrival rate
ξ	Integer, Tunable hyper-parameter
σ^k_n	Integer, the required node capacity of request $k \in K$
σ^k_e	Integer, the required link capacity of request $k \in K$

$$x^k_{e',e} \leq \mu^k_{e'} \leq \sum_{k \in K} x^k_{e',e} \quad \forall e' \in E, e \in E_a, k \in K \tag{5}$$

$$\mu^k_{e'(i,j)} \cup \mu^k_{e''(m,j)} \leq \zeta^k_j \leq \sum_{j \in L \cup R} \mu^k_{e'(i,j)} \cup \mu^k_{e''(m,j))}$$
$$\forall e', e'' \in E, k \in K, i \neq m \tag{6}$$

$$\sum_{k \in K} (\mu^k_n \cup \zeta^k_n) \cdot \sigma^k_n = \eta_n \quad \forall n \in N \tag{7}$$

$$\eta_n \leq C_n \quad \forall n \in N \tag{8}$$

$$\sum_{k \in K} \mu^k_e \cdot \sigma^k_e = \eta_e \quad \forall e \in E \tag{9}$$

$$\eta_e \leq C_e \quad \forall e \in E \tag{10}$$

D. Hierarchical Federated Edge Learning Mesh (HFEL-MESH) Algorithm

After building and evaluating the ILP formulation, we developed an heuristic algorithm called HFEL-MESH to solve the edge-to-cloud FL model aggregation problem. This type of resource allocation problems in MEC-based networks have been proved to be *NP-hard* [26]. As such, due to the scalability problem of ILP, we propose a heuristic algorithm that significantly saves computing time.

Our proposed HFEL-MESH heuristic algorithm is shown in Alg. 1. To ensure a comprehensive comparison, we adopt the HFEL algorithm proposed by Luo et al. [15] as a baseline for the hierarchical two-level scenario. Our HFEL-MESH

TABLE III: Variables description for ILP model.

Var	Description
$q^k_{e,p}$	Binary, equals to 1 if link $e \in E_a$ is allocated for path between node pair $p \in P$ for request k, in the auxiliary graph
$x^k_{e',e}$	Binary, equals to 1 if auxiliary link $e \in E_a$ selected link $e' \in E$ for connection in physical graph for request k
t_{failed}	Integer, number of failed requests
t_{total}	Integer, number of total requests
z^k_n	Binary, equals to 1 if request k used client $n \in U$ can be accepted by the model
η_n	Integer, the capacity occupied on node $n \in N$
η_e	Integer, the capacity occupied on link $e \in E$
μ^k_n	Binary, if request $k \in K$ uses node $n \in N$
μ^k_e	Binary, if request $k \in K$ uses link $e \in E$
Ψ^{cloud}	Integer, cloud cost
ζ^k_n	Binary, if there is node n used as aggregator in the physical graph

algorithm builds upon HFEL, using its aggregator placement and client clustering as input to enhance comparability. In the second phase, HFEL-MESH optimizes the routing of edge aggregators by jointly selecting nodes and determining link paths through the SD-WAN.

Alg. 1 consists of two main components: 1) Optimization of resource allocation for client-to-edge association (Lines 1–12). 2) Optimization of edge aggregators overlay routing for FL model aggregation (Lines 13–31). For the first component in the algorithm (for both HFEL and HFEL-MESH), we try to associate the clients and edges. For all the possible client-edge pairs, it computes the cost for the proposed route and all involved components. The edge association procedure assigns clients to an edge node using two functions: device transfer and device exchange operations. Assuming an edge node pair (v1, v2) where a client c1 is initially associated to v1, device transfer refers to c1 being associated with v2: c1 $\rightarrow$ v2. For a pair (v1, v2) where c1 is associated to v1 and c2 to v2, device exchange operation denotes c2 $\rightarrow$ v1 and c1 $\rightarrow$ v2. To find an optimum, the algorithm iterates over all edge node pairs (v1, v2) and their respective clients until no further operation yields a lower cost per edge node pair.

Next, we address the second component, which solves the aggregator overlay routing problem. This part is our new proposed method for HFEL-MESH. As outlined in Algorithm 1, this approach iteratively connects aggregators with the lowest routing cost. When a new request arrives, HFEL-MESH generates an auxiliary graph where each aggregator initially appears as an isolated node, corresponding to an edge node in the physical graph. After computing routing costs between aggregators and to the cloud for each pair (a_source, a_target), the algorithm links the source aggregator to its potential target through a directed edge, assigning the routing cost as the edge weight.

This iterative procedure tries to locate the edge with the minimum cost, determining the first pair of aggregators to be connected. Upon connection, the overlay topology graph is checked for cycles. Then, the auxiliary graph is updated by removing the node corresponding to the source aggregator. This process is repeated for the total number of aggregators, optimizing routing for each aggregator based on cost. The computation of routing costs between aggregators takes into account the current utilization and load of both nodes and links. Eq. 11 computes the TRFR, representing the training round failure rate in a simulation run, which will be used in the HFEL-MESH algorithm. Additionally, the cost of reporting to the cloud in the algorithm is defined by Eq. 12.

$$TRFR = \frac{t_{\text{failed}}}{t_{\text{total}}} \tag{11}$$

$$\Psi^{cloud} = \frac{\mathbb{V}}{\xi \cdot v} \tag{12}$$

Additionally, a parameter used to calculate the cost of transmitting a model through the cloud link to the cloud in the algorithm is defined by Eqn. 12. Ψ^{cloud} represents the capacity cost of the cloud link and is used in Alg. 1, line 31, to compute both TRFR and cumulative weighted capacity cost. The term $\mathbb{V}$ denotes the total number of algorithm iterations, equivalent to the number of aggregators, meaning that as the number of aggregators increases, so do the algorithm's complexity and overall cost. The parameter ξ, a fixed integer, acts as a tunable hyper-parameter, introducing flexibility into the system. Instead of assigning a fixed cost, adjusting ξ allows us to analyze the impact of varying cloud link costs [1]. Increasing ξ artificially lowers the cloud link reporting cost, encouraging the algorithm to prioritize cloud-based aggregation. The variable v represents the current iteration number, meaning that as iterations progress, the cost associated with cloud link usage increases. To ensure consistency, we maintain the same Ψ^{cloud} for both the ILP (set a proper σ_e^k for the link cost for only one iteration test in ILP) and heuristic approaches. This process results in a graph encapsulating all feasible routing paths between aggregators and to the cloud, allowing the algorithm to facilitate local updates while selecting low-cost, low-TRFR routes.

Algorithm 1: HFEL-MESH

Input: Set of clients U, set of aggregators L, and set of cloud R.
Output: Placed request with aggregator number and placement, client association, and aggregation topology.

1 Get current network state;
2 Create an auxiliary graph G';
3 $G' \leftarrow (N_a, E_a)$ with $N_a = \emptyset, E_a = \emptyset$;
4 **for** *each possible node* $a_{source} \in L$ **do**
5 Add node a_{source} in G' ;
6 **for** *each possible* $a_{target} \in L, a_{source} \neq a_{target}$ **do**
7 $G' \leftarrow$ add node a_{target};
8 $G' \leftarrow$ add edge $(a_{\text{source}}, a_{\text{target}})$ with weight $= \text{cost}_{a_{\text{source}} \rightarrow a_{\text{target}}}$;
9 **end**
10 $G' \leftarrow$ add cloud node c_{source};
11 $G' \leftarrow$ add auxiliary link $(a_{\text{source}}, c_{\text{source}})$ with weight $= \text{cost}_{a_{\text{source}} \rightarrow c}$;
12 **end**
13 Create overlay graph G'' by connecting aggregator pairs;
14 **repeat**
15 **repeat**
16 Create a set of spanning tree auxiliary edge E_a;
17 Try to find the smallest weight edge $e_{\text{min}} \leftarrow \min_{\text{weights}} E_a$;
18 Extract source and target aggregator/cloud from e_{min};
19 Connect source aggregator to its target in the overlay topology;
20 **if** *cycle in overlay topology graph* G'' **then**
21 temporarily remove selected e_{min} edge from set E_a;
22 remove connection in overlay topology graph;
23 **end**
24 **until** *overlay topology graph* G'' *has no cycle*;
25 remove edge e_{min};
26 remove edge $(e_{\text{min,target}}, e_{\text{min,source}})$;
27 remove node $e_{\text{min,source}}$
28 **until** *there has only aggregator nodes in* G';
29 Find each physical route for E_a in the physical graph G
30 **return** overlay topology graph and physical graph
31 Calculate TRFR and cumulative weighted capacity cost

IV. WatchEDGE simulator

To comprehend the dynamic behavior of computation and communication resources of a MEC-based SD-WAN in which FL applications are deployed, we implemented a Python-based DES. This section introduces the WatchEDGE simulator, which aims at modeling the dynamic usage of network resources where applications comprising multiple inter-dependent sub-tasks are deployed across the network. The simulator comprises five main components, as shown in Fig. 4: AB, event dispatcher, RA manager, results collector, and network.

[1]In the HFEL-MESH algorithm (Section III-D), we use a tunable hyperparameter to adjust the cloud cost. The impact of changing this parameter on cloud link utilization and system performance is demonstrated in the results.

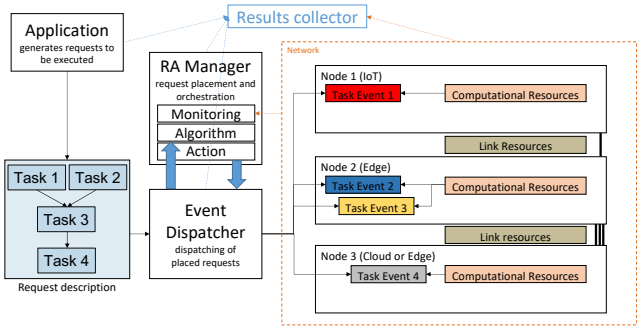


Fig. 4: WatchEDGE simulator architecture.

The AB generates requests based on the type of application being modeled. Specifically, our implementation simulates FL applications that aim at training a global model from the local updates of clients. These requests are generated following a predefined inter-arrival time distribution; they include information about the tasks to be deployed, their interdependence represented as an event graph, required resources, constraints, and other properties. These descriptions are then passed to the event dispatcher which collects and then forwards them to the RA manager. Upon the request arrival, the RA manager examines the request description and utilizes both the network monitoring module and the implemented algorithm, in our case this refers to the HFEL-MESH heuristic algorithm detailed in Sec.III-D, in charge of making placement decisions. After completing the placement task, the RA manager communicates the details of the placement decision back to the event dispatcher. The dispatcher then proceeds to deploy the requests throughout the simulated network. This deployment involves constructing an event graph, which is based on the descriptions provided in the requests. By processing the simulation events outlined in the event graph, the request is executed, and the designated computational task interacts with the allocated resources. The network component abstracts a real network to enable the simulation of network resource usage. For this purpose network edges and nodes host resource components that can be consumed by simulation events following an underlying resource consumption model.

In this work, nodes contain computational resources while links (assumed to be VPN tunnels of an SD-WAN) host bandwidth resources. The discussed simulator architecture shows several advantages.

To begin with, its modular structure allows quick adaptation of the simulated resource allocation problem, e.g. adding complexity through more resource types per node or investigating a different allocation problem. For instance, the application component can be repurposed to generate other types of resource-consuming requests and the network's behaviour solely depends on the resources and resource consumption models attached to nodes and links. Furthermore, the detached RA manager ensures an isolated implementation of each allocation method, thus guaranteeing equal conditions for different strategies. The simulator[2] is written in Python language (v3.9.16) and programmed to run in a Docker environment. All presented components of the simulator are implemented as Python classes following best practices of object-oriented programming whenever possible.

[2]Apart from the descriptive information provided in this paper, we will release our source code upon acceptance of the paper in a GitLab repository, and will provide instructions on how to reproduce our numerical results.

V. Experiments and Results

In this section, we outline the experiments conducted to evaluate our proposed algorithms. Initially, we introduce the simulation settings encompassing the used network topologies, computing and network resources, and the specific use case under consideration for this study. Subsequently, we present a comparative analysis between our proposed ILP and heuristic algorithm implementations, comparing them with a state-of-the-art algorithm proposed by Luo et al. [15].

A. Simulation settings

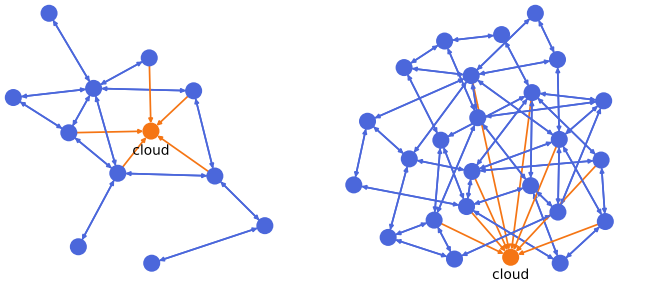
To evaluate the performance of the proposed HFEL-MESH algorithms, we integrated them into the WatchEDGE simulator as discussed in Section IV. This simulator facilitates in-depth simulations of edge networks and computing infrastructure leveraging SD-WAN and MEC technologies. Experimental assessments were carried out on a machine powered by an Intel(R) Core(TM) i7-9700 CPU.

As stated in the Introduction section, our use case revolves around environmental surveillance of wildlife. Specifically, we consider a set of end devices, acting as clients, capable of capturing images from the environment, akin to stationary smart cameras. Each client possesses the capability to gather images, constructing a local dataset, and run ML training aimed at identifying specific events, such as the presence of certain wild animals or the initiation of a wildfire. Within our context, we utilize this use case to configure the input settings of our simulator and conduct experiments. It is important to note that our focus lies on network resource allocation for FL applications rather than solely optimizing the accuracy of FL-based image recognition models.

We conducted evaluations on two network topologies, labeled as medium and large, as shown in Fig. 5, which were adapted from the work of Xiang et al. [27]. The medium topology comprises 11 edge nodes, each supporting 20 end devices per edge node, while the large topology comprises 24 edge nodes, each with 10 end devices. Additionally, each topology includes a cloud node to facilitate model aggregation.

We simulate an FL application in the simulator to generate training rounds for requests that trigger training processes on client end devices and subsequent model aggregation to both edge nodes and the cloud. The implemented algorithms are responsible for optimizing the distribution of model aggregation across the network, potentially leveraging techniques such as intermediate model aggregation through edge aggregators. Notably, the FL application generates independent training round requests, simulating multiple model owners seeking to train their respective models.

Our primary focus is not on comparing the latency of our approach with standard Federated Learning (FL) procedures. Instead, we aim to simulate the specialized HFEL-MESH aggregation, routing, and resource allocation mechanisms. To



(a) Medium topology with 11 edge nodes.

(b) Large topology with 24 edge nodes.

Fig. 5: Network topologies used for experiments. Blue nodes are edge nodes, and orange nodes represent a public cloud.

accurately simulate the training process, we deviate from conventional approaches of expressing training time in terms of CPU cycles. Instead, we rely on experimental results reported by Rajagopal et al. [28]. For simulating model aggregation, we adopt a CPU-cycle-based computation latency model proposed by Liu et al. [11]. The communication latency to the cloud is calculated as the number of CPU cycles required to execute one request, multiplied by the dataset size and divided by the CPU cycle frequency allocated to the training task. Moreover, we specify the capacities assigned to links and nodes, as outlined in Table IV.This approach allows us to assess HFEL-MESH's performance in edge computing environments, with a particular emphasis on resource management and aggregation. By concentrating on these elements, we provide a more focused analysis, rather than emphasizing the broader FL training process. In our model, the edge aggregation process consumes resources at the edge aggregator, corresponding to specific nodes within the network topology. During this process, multiple models (sets of parameters) are received as inputs, but only a single aggregated model (set of parameters) is produced. Consequently, this reduces the amount of link resources required for transmission after aggregation.

The inter-arrival time between consecutive requests is modeled with an exponential distribution. The capacities requested at nodes and links during simulation events are generated from a uniform distribution [29] [30] as specified in Table IV. The clients participating in each training round represent a random subset of all clients. The number of clients associated with each edge node is denoted by the symbol ♯clients. To simulate the training at clients, the FL application randomly selects an image dataset size and a model architecture from a predefined set. The training times for a single image are detailed in Table IV. Additionally, we consider the number of weights for each model architecture in the computation of the aggregation latency.

B. Evaluating ILP, Heuristic and Baseline Algorithms on Medium-size Topology

In this section, we evaluate the performance of the ILP, HFEL-MESH and the baseline HFEL algorithm, proposed by Luo et al. [15], in terms of Mean Utilization Ratio (MUR) and objective function. We define the MUR for links and nodes respectively as follows:

TABLE IV: Overview of simulation parameters.

Network resources	
client	1 computing unit
edge node	200 computing units
cloud node	4000 computing units
end link	200 Mbps
edge link	2000 Mbps
cloud link	4000 Mbps
Training round request	
requested node capacity	$[1, 8]$ cores
requested link capacity	$[20, 40]$ Mbps
client number	$[\frac{\#\text{clients}}{4}, \frac{\#\text{clients}}{2}]$
dataset size	$[59, 118]$ images
Model architectures	single-image training time, #weights
Squeezenet	26.4 ms, 421 098 weights [31]
MobileNetV2	38.4 ms, 3 400 000 weights [32]
MNas	35.7 ms, 3 900 000 weights [33]
GoogleNet	35.9 ms, 6 797 700 weights [34]
Res18	21.8 ms, 11 689 512 weights [35]
Res50	77.8 ms, 25 557 032 weights [35]

$$MUR_{link} = \frac{\eta_e}{C_e} \quad \forall e \in E \tag{13}$$

$$MUR_{node} = \frac{\eta_n}{C_n} \quad \forall n \in N \tag{14}$$

where η_e and η_n are the used capacity on links and nodes, respectively, while C_e and C_n denote total available capacity on links and nodes.

The objective function aims at minimizing the cumulative weighted capacity, reflecting the effectiveness of capacity allocation. A lower objective function signifies a better distribution of capacity, indicating a balanced aggregation of FL data flow and a reduced cost on less critical nodes (e.g. congested nodes or links). In essence, achieving a lower objective function demonstrates optimal capacity utilization and efficient resource allocation across the network.

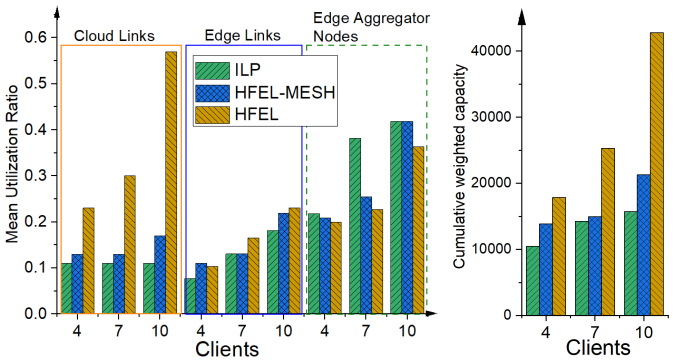
We conducted performance evaluations using the medium-size topology illustrated in Fig. 5a, accommodating a maximum of 10 requests due to scalability constraints with ILP. Each request involved allocating training rounds to different clients per edge node. In our experiments, we considered from 4 to 10 clients per request. Our assessment focused on the algorithms' effectiveness in minimizing the MUR of three key components: Cloud link, edge link, and edge aggregator nodes, as mentioned before. Fig. 6a shows the results of the MUR by varying the number of clients.

Our analysis reveals that the mean utilization ratio of cloud links for HFEL-MESH is between 15% - 23.5% higher than that of ILP, while being between 23.5% - 43% lower than that of the baseline HFEL algorithm. These findings underscore the effectiveness of our HFEL-MESH algorithm in substantially reducing utilization and congestion on the cloud links compared to the baseline HFEL approach. Furthermore, HFEL-MESH achieves a performance level closely aligned with ILP, as expected.

However, HFEL-MESH exhibits a maximum 30% optimality gap in edge aggregator node utilization compared to ILP, and 5% lower utilization compared to the HFEL baseline algorithm. This discrepancy can be attributed to the distinct objectives pursued by these algorithms. With HFEL-MESH, multiple aggregations occur on the edge nodes, resulting in higher edge node utilization compared to HFEL. In particular,

for scenarios involving requests with 7 clients, HFEL-MESH achieves a 15% higher utilization of cloud links compared to ILP, while ILP incurs a 52% higher utilization of edge aggregator nodes compared to HFEL-MESH. This disparity arises from the higher weight assigned to cloud links in the ILP's objective function, which drives it to prioritize reducing cloud link utilization, even at the cost of increased edge aggregator node utilization. In contrast, HFEL-MESH achieves a more balanced utilization across these components. The ILP is designed to minimize cloud link usage to prevent cloud congestion, while the heuristic approach focuses more on reducing the TRFR and makes the model more balanced. HFEL aims for a more balanced outcome overall. Although ILP is expected to deliver better performance, HFEL-MESH, while not optimal, offers a practical balance between resource utilization and TRFR.

Furthermore, as depicted in Fig. 6b, we observe that the objective function of HFEL-MESH is at most 32% higher than ILP, yet significantly lower than the baseline algorithm HFEL. This reaffirms the superior performance of HFEL-MESH over HFEL, with the performance gap widening as the complexity of requests increases.



(a) Utilization ratio (b) Objective function

Fig. 6: Comparative analysis between ILP, HFEL, and HFEL-MESH.

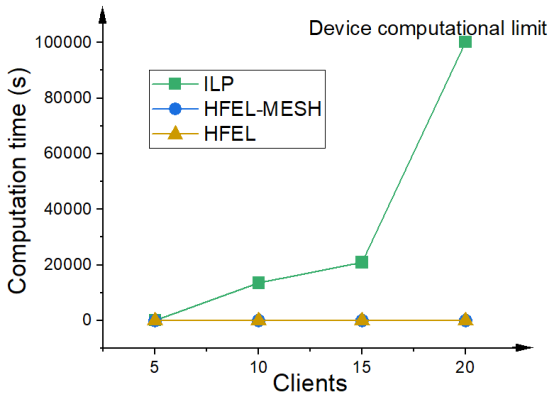


Fig. 7: Computational time.

Then, we examine the complexity of three different methods. Given the scalability challenges associated with the ILP approach, we tested the computational time of the system using a single training request with varying numbers of clients. As illustrated in Fig. 7, the computational time for ILP increases nearly exponentially as the number of clients grows. When the training request involves 20 clients, the ILP method reaches the computational limit of our device. In contrast, HFEL and HFEL-MESH exhibit nearly constant computational times across all scenarios, as shown in Fig. 7: even as the number of clients increases, the computational time for both HFEL and HFEL-MESH remains relatively stable.

C. Evaluating Heuristic and Baseline Algorithms on Large-size Topology

In this section, we evaluate the performance of the HFEL-MESH heuristic and the baseline HFEL algorithms in terms of MUR, as defined by Eq. 13 and Eq. 14, and Training Rounds Failure Rate (TRFR) defined by Eq. 11.

We conducted performance evaluations using the medium-size and large-size topology shown in Fig. 5a and Fig. 5b, respectively.

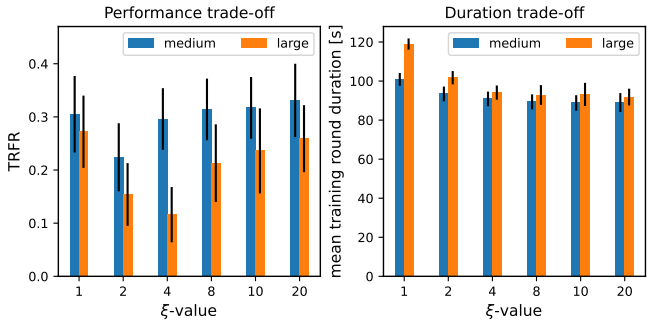
Fig. 8a illustrates the outcomes of the TRFR and the mean training round duration of the HFEL-MESH algorithm in response to variations of the hyper-parameter ξ. This parameter, ξ, serves as a tunable factor enabling adjustments to the algorithm's inclination towards reducing the number of edge aggregators or vice versa. A lower value of ξ favors aggregation on edge nodes (towards the edge nodes directly connected to the clients), whereas a higher value prioritizes aggregation on the cloud node (towards a total aggregation in the cloud). As depicted in Fig. 8a, we can notice that leveraging edge nodes for aggregations provides a distinct advantage, resulting in approximately an 11% of TRFR in the large topology.

Interestingly, when ξ=1, there is an observed increase in the TRFR for both network topologies. This suggests that an excessive emphasis on (far) edge nodes directly connected to the clients becomes counterproductive, and leads to longer training round durations as shown in Fig. 8a (right). In other words, the algorithm prioritizes aggregating every local model update on the nearest edge node, which may result in routing detours and unnecessary capacity waste. This wastage contributes to an increase in TRFR and in a longer training round durations.

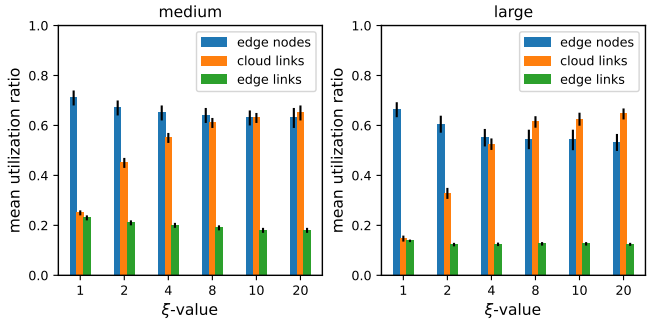
The higher TRFR observed for the medium topology can be attributed to its increased utilization ratio, as depicted in Fig. 8b (left). Conversely, the large topology, characterized by a lower TRFR, exhibits a correspondingly lower utilization ratio compared to the medium topology, as shown in Fig. 8b (right). It is interesting to note the increase in the cloud links usage with growing ξ for both topologies, while edge nodes utilization gradually decreases, albeit at a slower rate. This highlights the advantage of an aggregator overlay topology: the additional occupation of edge nodes, associated with the presence of edge aggregators, results in a smaller increase in utilization compared to the reduction in cloud links utilization.

Considering the duration of a training round, it is noteworthy from Fig. 8a and Fig. 8b that increasing edge nodes utilization results in longer training round durations. This observation suggests that although higher load distribution and lower TRFR are achieved, they are counteracted by increased

request durations. The rationale behind these findings lies in the prioritization of load distribution through edge aggregators during the creation of the aggregator overlay network, without considering request duration as a constraint. This leads to longer training round durations due to the additional aggregation steps introduced.



(a) TRFR of HFEL-MESH (left) and training round duration (right) for different ξ-values [30].



(b) Mean utilization ratio of HFEL-MESH for different ξ-values.

Fig. 8: Demonstration of edge-cloud trade-off by tuning ξ-parameter.

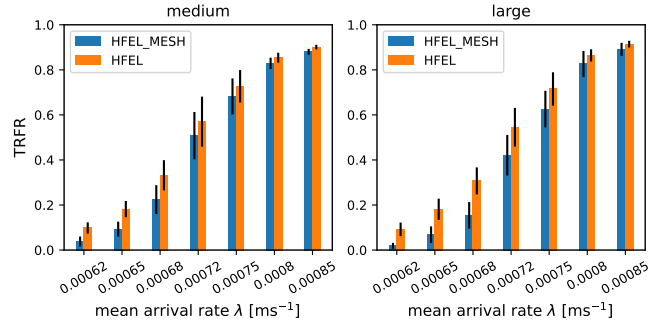


Fig. 9: Training round failure rate over arrival rate λ. The system reacts sensibly to a change in the arrival rate.

Evaluating different training round arrival rates λ provides insight into the system's capacity to handle requests. Fig. 9 reports a TRFR below $10\,\%$ for $\lambda = 0.00062\,\mathrm{ms}^{-1}$ and nearly $90\,\%$ for $\lambda = 0.00085\,\mathrm{ms}^{-1}$ across different topologies. These λ values result in average inter-arrival times between $1.2\,\mathrm{s}$ and $1.6\,\mathrm{s}$ on average, emphasizing the delicate balance that needs to be maintained by the algorithms. This fragility stems from the fact that a training round has a significantly longer lifespan compared to the arrival rate, making the system susceptible to saturation with just a few additional arriving requests.

HFEL-MESH consistently achieves a lower TRFR over topologies and arrival rates on average. For instance, the difference reaches up to 15% for the large topology, underscoring the value added by routing and aggregation between aggregators. Combining this with the reported resource utilization in Fig. 8b, we can state that HFEL-MESH optimizes computing resource utilization by increasing the usage of edge nodes while achieving a lower TRFR. Consequently, HFEL-MESH operates more reliably and efficiently. However, the difference in TRFR reported is smaller for the medium topology and diminishes further with an increased arrival rate. This is attributed to the lower availability of edge computing nodes, which restricts the aggregator routing capabilities of HFEL-MESH. Regarding the mean request placement time, we observe that even in the large topology the overhead of routing between aggregators is acceptable with $0.55\,\mathrm{s}$ for HFEL-MESH compared to $0.5\,\mathrm{s}$ for HFEL ($\lambda = 0.00068\,\mathrm{ms}^{-1}$).

This research underpins the WatchEDGE project, which aims to investigate an advanced edge computing architecture distributed across multiple geographically distant sites. Our primary focus is on orchestrating limited network resources while reducing cloud link utilization. The results show that HFEL-MESH achieves balanced resource utilization and significantly reduces cloud link usage compared to the HFEL baseline. While HFEL-MESH exhibits a slight performance gap when compared to the optimal ILP solution, it still proves to be a more efficient alternative, particularly in lowering the Training Round Failure Rate (TRFR) compared to the HEFL algorithm. This suggests that HFEL-MESH enhances FL performance relative to existing methods. The topology employed in this research is based on a real-world scenario, with plans to develop a custom topology in future work. Our approach to cloud utilization has demonstrated its effectiveness, further validating the potential of HFEL-MESH in practical applications. In conclusion, online resource-aware aggregation proves to be both advantageous and feasible for optimizing network resource utilization and reducing TRFR in networks with sub-optimal cloud connectivity and limited edge node resources. However, the presented analysis disregards other relevant aspects of edge computing, i.e. edge node reliability and trust [36], which remain beyond the scope of this paper.

VI. Conclusions

In this paper, we investigate the integration of FL into the WatchEDGE network infrastructure, a specialized MEC architecture powered by SD-WAN and AI-driven end devices. Through the utilization of ILP-based and HFEL-MESH algorithms, we assess the impact of FL training on network resource usage. Our findings reveal a discernible trade-off between computation at the edge and communication to the cloud, favoring edge computing for reduced TRFR and enhanced network capacity utilization. Furthermore, we introduce a novel online aggregation technique that introduces routing capabilities between edge aggregators. Our performance analysis demonstrates the superiority of this technique over state-of-the-art aggregation methods in adapting to the

network's resource characteristics. This novel approach yields a significant reduction in TRFR by up to 15%, concurrently mitigating the cloud communication overhead induced by FL. Looking ahead, our future research endeavors will delve into the incorporation of additional FL-specific metrics, such as energy cost and node reliability, into our aggregator routing strategy. By exploring further implications of FL in MEC networks, we aim to provide comprehensive insights into the optimization of FL-enabled network infrastructures.

ACKNOWLEDGEMENT

This work was partially supported by the European Union under the Italian National Recovery and Resilience Plan (NRRP) of NextGenerationEU, partnership on "Telecommunications of the Future" (PE00000001 - program "RESTART")

REFERENCES

[1] A. Filali, A. Abouaomar, S. Cherkaoui, A. Kobbane, and M. Guizani, "Multi-access edge computing: A survey," *IEEE Access*, vol. 8, pp. 197017–197046, 2020.

[2] P. Mach and Z. Becvar, "Mobile edge computing: A survey on architecture and computation offloading," *IEEE communications surveys & tutorials*, vol. 19, no. 3, pp. 1628–1656, 2017.

[3] Z. Yang, Y. Cui, B. Li, Y. Liu, and Y. Xu, "Software-defined wide area network (sd-wan): Architecture, advances and opportunities," in *2019 28th International Conference on Computer Communication and Networks (ICCCN)*, pp. 1–9, IEEE, 2019.

[4] K. G. Yalda, D. J. Hamad, and N. Ţăpuş, "A survey on software-defined wide area network (sd-wan) architectures," in *2022 International Congress on Human-Computer Interaction, Optimization and Robotic Applications (HORA)*, pp. 1–5, IEEE, 2022.

[5] W. Y. B. Lim, N. C. Luong, D. T. Hoang, Y. Jiao, Y.-C. Liang, Q. Yang, D. Niyato, and C. Miao, "Federated learning in mobile edge networks: A comprehensive survey," *IEEE Communications Surveys & Tutorials*, vol. 22, no. 3, pp. 2031–2063, 2020.

[6] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Artificial intelligence and statistics*, pp. 1273–1282, PMLR, 2017.

[7] S. Niknam, H. S. Dhillon, and J. H. Reed, "Federated learning for wireless communications: Motivation, opportunities, and challenges," *IEEE Communications Magazine*, vol. 58, no. 6, pp. 46–51, 2020.

[8] C. Zhang, Y. Xie, H. Bai, B. Yu, W. Li, and Y. Gao, "A survey on federated learning," *Knowledge-Based Systems*, vol. 216, p. 106775, 2021.

[9] S. Trindade, L. F. Bittencourt, and N. L. da Fonseca, "Resource management at the network edge for federated learning," *Digital Communications and Networks*, 2022.

[10] G. Maier, A. Albanese, M. Ciavotta, N. Ciulli, S. Giordano, E. Giusti, A. Salvatore, and G. Schembra, "Watchedge: Smart networking for distributed ai-based environmental control," *Computer Networks*, vol. 243, p. 110248, 2024.

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

[12] P. Qi, D. Chiaro, A. Guzzo, M. Ianni, G. Fortino, and F. Piccialli, "Model aggregation techniques in federated learning: A comprehensive survey," *Future Generation Computer Systems*, 2023.

[13] J. Wu, F. Dong, H. Leung, Z. Zhu, J. Zhou, and S. Drew, "Topology-aware federated learning in edge computing: A comprehensive survey," *ACM Computing Surveys*, vol. 56, no. 10, pp. 1–41, 2024.

[14] A. Brecko, E. Kajati, J. Koziorek, and I. Zolotova, "Federated learning for edge computing: A survey," *Applied Sciences*, vol. 12, no. 18, p. 9124, 2022.

[15] S. Luo, X. Chen, Q. Wu, Z. Zhou, and S. Yu, "Hfel: Joint edge association and resource allocation for cost-efficient hierarchical federated edge learning," *IEEE Transactions on Wireless Communications*, vol. 19, no. 10, pp. 6535–6548, 2020.

[16] Z. Xu, D. Zhao, W. Liang, O. F. Rana, P. Zhou, M. Li, W. Xu, H. Li, and Q. Xia, "Hierfedml: Aggregator placement and ue assignment for hierarchical federated learning in mobile edge computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 34, no. 1, pp. 328–345, 2023.

[17] X. Chen, G. Zhu, Y. Deng, and Y. Fang, "Federated learning over multi-hop wireless networks with in-network aggregation," *IEEE Transactions on Wireless Communications*, vol. 21, no. 6, pp. 4622–4634, 2022.

[18] Z. Wang, H. Xu, J. Liu, H. Huang, C. Qiao, and Y. Zhao, "Resource-efficient federated learning with hierarchical aggregation in edge computing," in *IEEE INFOCOM 2021-IEEE Conference on Computer Communications*, pp. 1–10, IEEE, 2021.

[19] J. Feng, L. Liu, Q. Pei, and K. Li, "Min-max cost optimization for efficient hierarchical federated learning in wireless edge networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 11, pp. 2687–2700, 2022.

[20] C. T. Dinh, N. H. Tran, M. N. H. Nguyen, C. S. Hong, W. Bao, A. Y. Zomaya, and V. Gramoli, "Federated learning over wireless networks: Convergence analysis and resource allocation," *IEEE/ACM Transactions on Networking*, vol. 29, no. 1, pp. 398–409, 2021.

[21] Q. Wu, X. Chen, T. Ouyang, Z. Zhou, X. Zhang, S. Yang, and J. Zhang, "Hiflash: Communication-efficient hierarchical federated learning with adaptive staleness control and heterogeneity-aware client-edge association," *IEEE Transactions on Parallel and Distributed Systems*, vol. 34, no. 5, pp. 1560–1579, 2023.

[22] C. Du, J. Xiao, and W. Guo, "Bandwidth constrained client selection and scheduling for federated learning over sd-wan," *IET Communications*, vol. 16, no. 2, pp. 187–194, 2022.

[23] R. Wang, J. Zhang, M. Ibrahimi, Z. Gu, Y. Xiao, F. Musumeci, M. Tornatore, and Y. Ji, "Network for ai: Communication-efficient federated learning with mst-based scheduling and multi-aggregation over optical networks," in *2024 Optical Fiber Communications Conference and Exhibition (OFC)*, pp. 1–3, 2024.

[24] R. Wang, J. Zhang, Q. Zhang, B. Zhang, Z. Gu, A. Attarpour, Y. Ji, and M. Tornatore, "Poster: Flexible scheduling of network and computing resources for distributed ai tasks," in *Proceedings of the ACM SIGCOMM 2024 Conference: Posters and Demos*, pp. 60–62, 2024.

[25] S. AbdulRahman, H. Tout, A. Mourad, and C. Talhi, "Fedmccs: Multicriteria client selection model for optimal iot federated learning," *IEEE Internet of Things Journal*, vol. 8, no. 6, pp. 4723–4735, 2020.

[26] C. Li, H. Wang, and R. Song, "Mobility-aware offloading and resource allocation in noma-mec systems via dc," *IEEE Communications Letters*, vol. 26, no. 5, pp. 1091–1095, 2022.

[27] B. Xiang, J. Elias, F. Martignon, and E. Di Nitto, "A dataset for mobile edge computing network topologies," *Data in Brief*, vol. 39, p. 107557, 2021.

[28] A. Rajagopal and C.-S. Bouganis, "perf4sight: A toolflow to model CNN training performance on edge GPUs," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV) Workshops*, pp. 963–971, 2021.

[29] Y. Liu, H. Lu, X. Li, Y. Zhang, L. Xi, and D. Zhao, "Dynamic service function chain orchestration for nfv/mec-enabled iot networks: A deep reinforcement learning approach," *IEEE Internet of Things Journal*, vol. 8, no. 9, pp. 7450–7465, 2020.

[30] T. D. Burd and R. W. Brodersen, "Processor design for portable systems," *Journal of VLSI signal processing systems for signal, image and video technology*, vol. 13, no. 2-3, pp. 203–221, 1996.

[31] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size," *arXiv preprint arXiv:1602.07360*, 2016.

[32] M. Zhu and A. Z. L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 4510–4520, 2018.

[33] M. Tan, B. Chen, R. Pang, V. Vasudevan, M. Sandler, A. Howard, and Q. V. Le, "Mnasnet: Platform-aware neural architecture search for mobile," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 2820–2828, 2019.

[34] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1–9, 2015.

[35] T. Contributors, "resnet18, resnet50 - Torchvision main documentation." https://pytorch.org/vision/master/models/generated/torchvision.models.resnet18.html
https://pytorch.org/vision/master/models/generated/torchvision.models.resnet50.html, 2023.

[36] G. Carvalho, B. Cabral, V. Pereira, and J. Bernardino, "Edge computing: current trends, research challenges and future directions," *Computing*, vol. 103, pp. 993–1023, 2021.