Inference Load-Aware Orchestration for Hierarchical Federated Learning

Anna Lackinger*, Pantelis A. Frangoudis*, Ivan Čilić[†], Alireza Furutanpey*, Ilir Murturi*,
Ivana Podnar Žarko[†], Schahram Dustdar*

*Distributed Systems Group, TU Wien, Vienna, Austria
{a.lackinger, p.frangoudis, a.furutanpey, imurturi, dustdar}@dsg.tuwien.ac.at

[†]Faculty of Electrical Engineering and Computing, University of Zagreb, Zagreb, Croatia
{ivan.cilic, ivana.podnar}@fer.hr

2024 IEEE 49th Conference on Local Computer Networks (LCN) | 979-8-3503-8800-8/24/$31.00 ©2024 IEEE | DOI: 10.1109/LCN60385.2024.10639809

***Abstract*—Hierarchical federated learning (HFL) designs introduce intermediate aggregator nodes between clients and the global federated learning server in order to reduce communication costs and distribute server load. One side effect is that machine learning model replication at scale comes "for free" as part of the HFL process: model replicas are hosted at the client end, intermediate nodes, and the global server level and are readily available for serving inference requests. This creates opportunities for efficient model serving but simultaneously couples the training and serving processes and calls for their joint orchestration. This is particularly important for *continual learning*, where serving a model while (re)training it periodically, upon specific triggers, or continuously, takes place over shared infrastructure spanning the computing continuum. Consequently, training and inference workloads can interfere with detrimental effects on performance. To address this issue, we propose an inference load-aware HFL orchestration scheme, which makes informed decisions on HFL configuration, considering knowledge about inference workloads and the respective processing capacity. Applying our scheme to a continual learning use case in the transportation domain, we demonstrate that by optimizing aggregator node placement and device-aggregator association, significant inference latency savings can be achieved while communication costs are drastically reduced compared to flat centralized federated learning.**

***Index Terms*—Federated learning, service orchestration, continual learning, edge computing.**

I. Introduction

The traditional approach to machine/deep learning (ML/DL) is to train an ML model with data collected on a centralized cloud server and make it available for inference either by hosting it in the cloud or, more recently, serving it directly from end devices following the edge computing paradigm [1]. Both training and inference are, however, challenged in different ways. On the one hand, data often tend to be distributed across various devices, and collecting all data to a central server raises logistical and privacy concerns [2]. On the other hand, and despite the proliferation of AI accelerators of different scales, serving inference requests at the same time in a timely, accurate, and private manner is a hard, multi-faceted problem that calls for sophisticated edge-cloud synergy [3].

Federated learning (FL) has emerged as a response to some of these challenges and has gained significant popularity. In FL, clients train an ML model on local data and upload the result of their computation to a central server for aggregation. The server then produces and redistributes an updated global model, starting a new training round. This iterative process continues until some convergence criterion is met. FL operates under distinctly different assumptions than traditional deep learning: the number of participating nodes is often high, data are unbalanced and often not independent and identically distributed (non-IID), and performance depends on the communication capabilities of participating entities, which are often heterogeneous, resource-constrained and unreliable [4]. Moreover, even when the number of participating devices is high, each device typically only holds a relatively small data volume.

In its default centralized configuration, FL can suffer from increased network costs. Instead of uploading raw data, clients must exchange ML model copies with the FL server, which is particularly costly for long-lasting training processes and large DL models. At the same time, large-scale synchronous federated learning tasks require significant network capacity at the server end to deal with simultaneous model updates [5]. These limitations motivate *hierarchical* federated learning (HFL), where frequent aggregations can be enabled on the edge network, avoiding extensive cloud aggregations without sacrificing learning accuracy [6]–[15].

In a hierarchical FL process, the key decision is how to appropriately cluster FL clients in groups and designate local aggregator nodes per cluster to optimize for specific performance criteria, such as communication cost minimization or FL convergence time. As a side effect, ML model replication at hosts in the computing continuum comes "for free" as a natural part of the HFL process: besides client devices and the cloud, model replicas are now found at local aggregation points and are usable for serving inference requests, potentially with low latency due to proximity, and provided that privacy requirements allow it. At the same time, though, this creates a coupling between training and inference and calls for their *joint orchestration*. This coupling becomes more important in *continual learning (CL)* scenarios [16], [17], where training and inference processes overlap and compete for compute and network resources. The CL paradigm provides a realistic setting reflecting real-world learning processes, with new tasks and new data, potentially with different distributions, emerging over time. As such, CL may involve retraining an ML model while serving it, which is challenging to achieve with high performance when sharing the same resource-constrained infrastructure, as is often the case in FL. This challenge has started receiving attention only recently [18].

Our work focuses on continual hierarchical FL and contributes to answering the following: *How can a hierarchical FL process be orchestrated with inference workload awareness? Additionally, what are the performance benefits of this approach in terms of reducing inference latency and communication costs?*

In summary, our main contributions are:

- Introducing an architectural framework for HFL orchestration (§ III), in the context of which we formulate and solve the *inference-aware HFL Orchestration Problem (HFLOP)*. This leads to a cost-optimal assignment of FL devices to aggregator nodes, taking into account their inference workload processing capacities (§ IV). To the best of our knowledge, this is the first work that tackles joint training-inference orchestration in the *hierarchical* FL setting.
- Applying and evaluating our scheme to a continual federated learning use case in the transportation domain using a real-world dataset to handle dynamic and evolving traffic patterns (§ V). Our results show that our approach can feasibly reduce inference latency and communication costs.
- Providing our code as open source[1] for reproducibility and to allow other researchers to utilize it for their purposes.

II. Related Work

A. Continuous federated learning

Continuous learning, also known as lifelong learning, has been proposed by Thrun [19] and is a well-researched topic with extensive literature. It involves learning continuously from streaming data, developing a model based on previous learning, and then being able to reapply, adapt, and generalize it to new situations. Continuous learning and continual learning are often overlooked in federated learning, which is typically limited to one-time training. In practice, however, a model may need to be retrained over time to maintain its accuracy. Wang et al. [20] present a comprehensive survey on continual learning, mainly focusing on the theoretical foundations, representative methods, and practical applications to address catastrophic forgetting [21]–[23] and maintain model performance. Le et al. [24] present a federated continuous learning scheme (FCL) based on broad learning (FCL-BL), which proposes a weighted processing strategy to solve the catastrophic forgetting problem and develops a local-independent training solution for fast and accurate training. For an overview of existing FCL approaches, related challenges, and classification tasks, the reader is referred to the work of Yang et al. [25]. We should note that the aforementioned studies *do not focus on service orchestration aspects of FCL processes and do not address resource contention between (continuous) training and inference.*

B. Joint training-inference optimization in federated learning

Most research on federated learning focuses on model training (e.g., model pruning [26], gradient compression [27], etc.) rather than the inference stage (e.g., [28]). Especially in continuous learning, where models need to be constantly updated, it can happen that inference requests and training overlap. One of the first papers that emphasizes the problem of joint optimization of model training and inference is by Han et al. [18]. To characterize the clients' inference performance, the authors introduce the notion of the age of the model to represent how much the models on the client side differ from the global model. Another challenge mentioned in the paper is the close coupling between clients' decisions, including participation probability in FL, model download probability, and service rates. To address these challenges, the authors offer an online problem approximation that reduces complexity and optimizes resources for model training and inference. *Our work shares a similar motivation but focuses on the hierarchical setting*. Despite the extensive literature on HFL [6]–[15], we are not aware of any works that jointly consider training and inference serving in HFL.

C. Applications to traffic flow prediction

From a use-case perspective, one area where continuous learning and federated learning are often required is Traffic Flow Prediction (TFP), which is also the application scenario we experiment with in this work (§ V). Predicting traffic flow is crucial for reducing congestion and improving transportation efficiency in smart cities [29]. In TFP, distributed sensors measure traffic flow and train local models to predict how the traffic flow will evolve in the future. Due to the constantly changing traffic situation, the models have to be retrained continuously. Zhang et al. [30] present CTFed, a hierarchical FL approach to train Graph Neural Networks for TFP, clustering clients based on the similarity of their local models. Liu et al. [31] propose an FL-based gated recurrent unit neural network algorithm (FedGRU). Yuan et al. [29], on the other hand, propose an FL scheme based on the Spatial-Temporal Long and Short-Term Networks (FedSTN) algorithm for TFP. To integrate external factors into TFP, a Semantic Capture Network (SCN) is introduced to capture the regional flow relationship and the influence of Point of Interest (POI) on TFP. Hu et al. [32] propose an FL-based spatial-temporal traffic demand prediction model (FedSTTDP) while ensuring data security and privacy. In contrast to the mentioned works, *we also target low-latency TFP serving for the hierarchical FL setup in particular.*

III. System architecture

Hierarchical federated learning aims to reduce centralized FL's communication cost and alleviate the load of a single aggregation server. FL clients are organized in clusters based on criteria such as their network proximity and perform aggregations first on a local level with the mediation of a local aggregation server: client devices transmit their trained models to local servers after a set number of training epochs

[1] https://github.com/pfrag/hflop

and receive updated (aggregated) models by their local servers; this process constitutes a *local aggregation round*. After a configurable number of epochs and local aggregation rounds, the derived cluster-local model is sent to the global server for global aggregation and redistribution of a new version of the global model to clusters; this constitutes a *global aggregation round*.

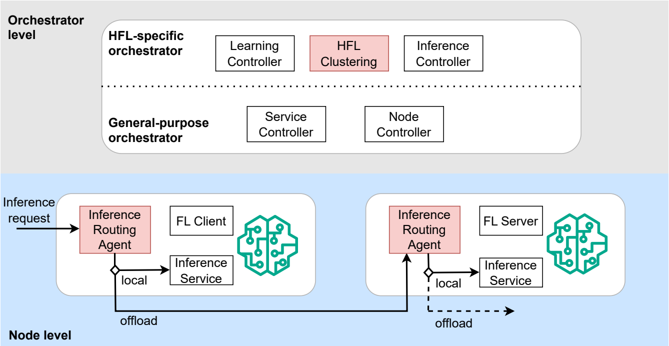


Fig. 1: Architectural view of our hierarchical federated learning orchestration framework. Our framework supports the joint orchestration of distributed training and inference serving.

We address the challenges associated with the dynamic orchestration of such processes and put our work in the context of an HFL orchestration system. While the technical details of such a system are outside the scope of this paper, in this section, we provide its high-level overview and demonstrate how the functionality we present fits into this framework. We apply a two-level orchestration approach, depicted in Fig. 1. Our design relies on two main components. The first is a *general purpose orchestrator* (GPO), such as Kubernetes, to enable the collection of information about the underlying compute infrastructure, e.g., nodes capable of hosting HFL service components, and manage the deployment of these components as containerized microservices distributed over the computing continuum. The second is a special-purpose, *HFL-specific service orchestrator*. This component makes decisions at the HFL level, and the mechanisms we present in this paper are within its scope.

One element of this service-specific orchestrator is the *learning controller*, which manages the HFL runtime. This controller retrieves information from the GPO on node resource states, underlying networks, and information on the clients requiring inference services. This information is provided as the input for the *HFL clustering* mechanism, whose role is to form a hierarchy of FL clients and local aggregation nodes, solving the optimization problems therein – a major focus of this paper is on this mechanism in particular (see Section IV). The output of the clustering mechanism prescribes a specific HFL configuration, which is eventually translated to (i) an actual containerized deployment implemented by the GPO at the instruction of the learning controller and, in turn, (ii) a concrete distributed HFL service instance, managed by the learning controller and implemented by microservices running on client devices, edge nodes, and the cloud. Besides assigning FL devices to aggregator nodes, and depending on the specific instantiation of our orchestration system, the clustering mechanism may decide on further FL service parameters, such as the number of local epochs and local/global aggregation frequencies.

To enable model serving, the *inference controller* is triggered at the end of the deployment phase to deploy and configure an inference service and an inference request agent per node. In the context of hierarchical FL, an FL service can be a client that trains on local data or a server (aggregator) that provides model aggregation on a local or global level. The same model, updated during the learning process, can be used for inference. Therefore, on the same node, the orchestrator deploys an inference service, which is a server that provides an interface for interaction with the model. Also, each node hosts an inference routing agent, a proxy that decides where an inference request will be processed, implementing specific request routing logic. For example, if the node is under a heavy workload, such as when training a DL model, the routing agent forwards the requests to an inference service higher up in the hierarchy, i.e., the parent server (aggregator). Information on the load can be extracted from the GPO node controller.

We should finally note that orchestration tasks involve deploying and managing services at runtime. Therefore, orchestration must adapt to environmental changes, such as node failures and network changes. The learning controller monitors the learning pipeline and reacts to these changes by performing re-clustering on environmental events. On the other hand, orchestration also needs to adapt to service-specific events. For example, in the continual learning settings our architecture targets, a task of the inference controller is to monitor inference services and trigger a new HFL task if inference accuracy is below a specific threshold.

IV. The Inference-Aware HFL Orchestration Problem

We now focus on the internals of the HFL clustering component of our architecture. We introduce a model of our system that captures the co-existing and inter-dependent training and inference processes and we provide an integer linear programming formulation for the inference-aware HFL orchestration problem. A solution to this problem represents a cost-minimizing HFL cluster configuration.

A. System model

A set of n devices participate in an FL task, and m edge host locations exist that are eligible to place an aggregator. The communication cost for device-edge host pair (i, j) is given by $c_{ij}^{(d)}$. For example, $c_{ij}^{(d)} = 0$ implies that uploading data from device i to an aggregator placed at edge node j incurs no cost, as would be the case for an aggregator placed inside a device's local area network or, more generally, if the device is connected over an unmetered link. Similarly, the communication cost between the global server and edge node j is given by $c_j^{(e)}$.

After several *epochs*, clients upload the result of their computation to their associated aggregator (*local round*). After a specified number of *local* rounds, all aggregators submit their models to the global server, which computes a global model and broadcasts it to the aggregators, which in turn relay it to their associated client devices (*global round*).

At the same, device i generates inference requests at a rate of λ_i. Each inference request must be processed by a node hosting a version of the ML model under training, i.e., the device itself, its associated aggregator, or the global server. Edge node j has a specific inference request processing capacity r_j (in requests/s), while we assume for simplicity that the capacity of the cloud-hosted global server is infinite. Requests are routed using the following rules:

- **R1:** If the device generating an inference request is busy training, the request is always offloaded to the device's associated aggregator.
- **R2:** If the device is not participating in the current FL round, it independently decides to process the request locally or offload it to the closest aggregator.
- **R3:** The aggregator processes requests of its associated busy devices with priority and may serve requests by devices not active in the current FL round (or, more generally, requests by entities external to the FL process) only if the load generated by the former is sufficiently below its capacity. Otherwise, it will offload excess requests to the cloud; the aggregator thus operates as a device *proxy*.

B. Problem formulation

Given these inference workload processing rules, the HFL orchestrator needs to solve the following problem: *How to place aggregators at edge nodes and assign FL clients to them so that FL communication cost is minimized, subject to inference processing capacity constraints of edge hosts.*

A solution to this problem is represented by assigning values to binary variables x_{ij} for $i = 1, 2, \cdots, n$ and $j = 1, 2, \cdots, m$. These variables encode device-edge node associations, where $x_{ij} = 1$ if device i is associated with an aggregator placed at edge node j, and $x_{ij} = 0$ otherwise. We further introduce binary variables $y_j, j = 1, 2, \cdots, m$, where $y_j = 1$ if an aggregator is placed at edge node j and 0 otherwise.

We provide the following binary integer linear programming formulation for the *HFL Orchestration Problem (HFLOP)*, where the objective is to find a configuration that minimizes the communication costs of HFL:

$$\text{minimize} \quad \sum_{i=1}^{n}\sum_{j=1}^{m} x_{ij} c_{ij}^{(d)} l + \sum_{j=1}^{n} y_j c_j^{(e)} \tag{1}$$

$$\text{subject to} \quad x_{ij} \le y_j, \ \ \forall\, i \in [1, n]\,,\ \forall\, j \in [1, m] \tag{2}$$

$$y_j \le \sum_{i=1}^{n} x_{ij}, \ \ \forall\, j \in [1, m] \tag{3}$$

$$\sum_{i=1}^{n} x_{ij} \lambda_i \le r_j, \ \ \forall\, j \in [1, m] \tag{4}$$

$$\sum_{j=1}^{m} x_{ij} \le 1, \ \ \forall\, i \in [1, n] \tag{5}$$

$$\sum_{i=1}^{n}\sum_{j=1}^{m} x_{ij} \ge T \tag{6}$$

$$x_{ij}, y_j \in \{0, 1\}, \ \forall\, i \in [1, n], \ \forall\, j \in [1, m] \tag{7}$$

The objective function (1) consists of two terms. The LHS term expresses the cost of local aggregation rounds, where l is the number of local aggregations taking place before a global aggregation round. In practical terms, *assigning devices to local edge aggregators*, to which connectivity is cheap/free, tends to reduce this cost component. The RHS term is the cost of global aggregation, which involves each edge aggregator uploading a model update to the global server and receiving the global model. *Reducing the number of edge aggregators* tends to reduce this cost (limited by edge host capacity constraints and at the expense of potentially having clients associated with remote aggregators).

We have dropped the constants expressing the model size and that both local and global aggregation involve two communication directions; thus, in absolute terms, the cost doubles. Finally, we assume the uplink and downlink costs are the same, though our model is straightforward to extend in order to capture cost asymmetries. Constraint set (2) ensures that an aggregator is always placed at edge node j if at least one edge device is associated with it. Conversely, constraint set (3) does not allow an aggregator to be placed at an edge node without any associated devices. This set of constraints is often redundant, as operating an aggregator at node j typically comes at a cost; if no devices are associated with it, then $y_j = 1$ will not be part of the optimal solution. Edge node capacity constraints are given by (4). Constraint (5) allows each device to associate with at most one aggregator. Additionally, constraint (6) ensures that a minimum number of devices T participates in FL. Finally, (7) constrains the variables to binary values.

Notably, HFLOP is a generalization of the *Capacitated Facility Location Problem (CFLP) with unsplittable flows*, i.e., the CFLP version where a single facility must serve the whole demand of an entity. This problem is known to be NP-hard [33]. Any instance of this CFLP version can be transformed to an instance of HFLOP by (i) creating one edge node for each potential facility location and one device for each location to be served, (ii) setting $c_j^{(e)}$ to the setup cost of each corresponding location, and device communication costs $c_{ij}^{(d)}$ to the transportation costs between locations i and j, and (iii) setting $T = n$, i.e., all devices (cf. locations) need to be covered.

C. Performance considerations

Given the hardness of the problem, dealing with large instances can be challenging. Fig. 2 presents the time to derive the optimal solution to HFLOP for varying problem instance sizes using the branch and cut algorithm by CPLEX on a host

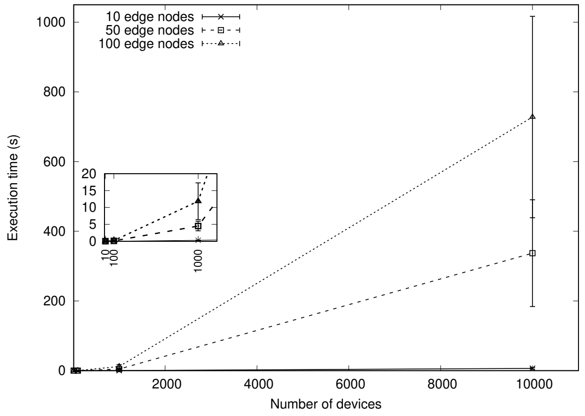


Fig. 2: Execution times of deriving the optimal solution to HFLOP using a commercial solver (CPLEX). Mean execution times are reported with 95% confidence intervals.

with an 8-core AMD Ryzen CPU and 32 GB of RAM. For scenarios with 10.000 devices and 100 edge host locations, it takes hundreds of seconds to find an optimal assignment, while for larger instances, it can become prohibitively expensive computationally to solve the problem optimally due to severely increasing CPU and memory requirements. However, this approach is feasible for many practical cases, given that HFLOP may not need to be solved frequently. For larger-scale scenarios, adaptations of heuristics and approximation algorithms for versions of the facility location problem [33], [34] can be considered.

V. Evaluation

The following experiments aim to evaluate the performance of our scheme (HFLOP) on a real-world dataset. We wish to quantify the effects of using HFLOP on the accuracy of federated learning clients, the communication cost of aggregation rounds, and the inference serving time. By examining these factors, we can determine whether HFLOP improves communication costs and inference processing time without negatively impacting training.

A. Methodology and use case

Fig. 3 illustrates a hierarchical federated learning setup with inference requests and an overview of the use case that shapes our experiment setup. Federated learning clients continuously train models on their local data. After some time, they send their data to local servers that either aggregate the models and send them back to the clients or forward the aggregated models to the global server. While some federated learning clients are training, some entities may send inference requests to the closest server, which is a federated learning client in most cases. The inference request will be forwarded to the closest local server if the FL client is busy training. If this server is busy, the request will be forwarded to the global server.

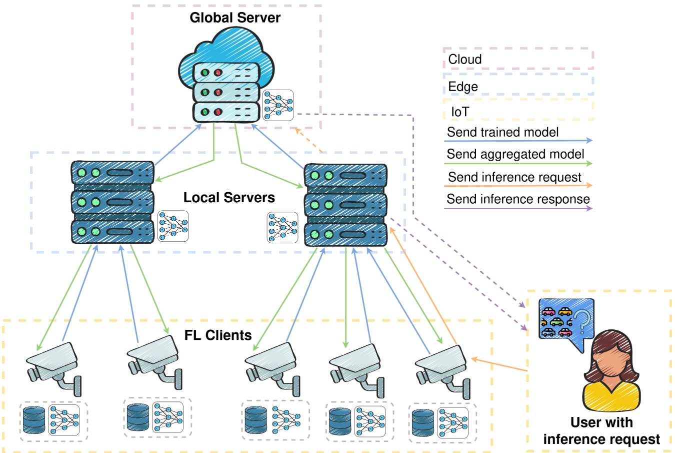


Fig. 3: Different FL clients are clustered and connected to their closest local server. An inference request is sent to the closest server, which processes the request on its model or sends it to a local server if it is busy training. The local server processes the request or, if it reaches its processing capacity, forwards it to a cloud server, which then answers the request.

We use the traffic forecasting dataset based on Los Angeles Metropolitan traffic (METR-LA) for the experiments. Its sensors are depicted in Fig. 4. This dataset contains traffic information collected from loop detectors in the highway of Los Angeles County [35]. In the dataset, there are 207 sensors and 4 months of data ranging from March 1st, 2012, to June 30th, 2012. Traffic information is recorded every 5 minutes for 34.272 time stamps.

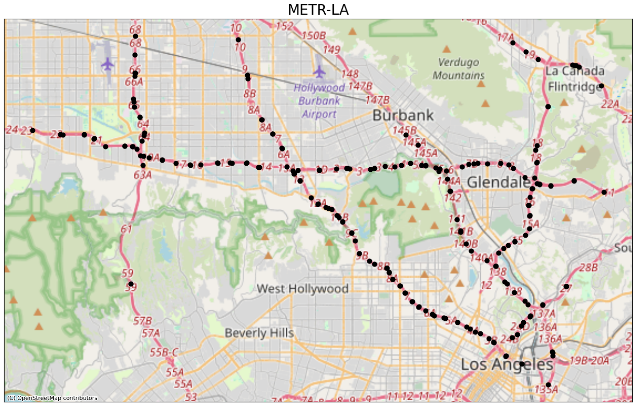


Fig. 4: Sensor distribution of the METR-LA dataset.

B. Continuous learning performance

1) Continuous learning: The tests conducted in this work for continuous learning, continuous federated learning, and continuous federated learning with inference requests ran on a server that is equipped with 32 CPUs, specifically the AMD Ryzen 9 5950X 16-Core Processor, each with a maximum clock speed of 5083.3979 MHz. Additionally, the server incorporates 2 NVIDIA GeForce RTX 3090 GPUs, each featuring 24576 MB of memory.

To test the benefits of continuous learning and whether it improves the accuracy of the model, we conducted some experiments using a Gated Recurrent Units (GRU) architecture. Recurrent Neural Networks (RNNs) are often used for time-series predictions, and GRUs have gating mechanisms to improve the information flow. Therefore, they are often used in federated time-series prediction [31], [35]. The GRU was trained for 20 epochs on 4 weeks of the data, and the remaining data was used for testing. The best results were achieved using a batch size of 16, a hidden size of 128, 2 layers, and a learning rate of 0.0001. After training for 100 epochs, the Mean Squared Error (MSE) predicting the test data was 0.04470, and the MSE of the retrained model that continuously retrained on new data was 0.04284. These results demonstrate that continuously training the model decreased the MSE.

2) Continuous federated learning: After the first tests were conducted to find the best hyperparameters for the model and to show the benefits of continuous training, the next step was to evaluate continuous federated learning. We first clustered the clients for the hierarchical experiments based on their location. This ensures that when an edge aggregator is added, its network cost and latency are reduced. In Fig. 5, one can see the street map after the sensors have been clustered. Each color represents a different cluster. The minimal number of participating devices given in Constraint (6) is for these experiments set to 20. Since there are 4 clusters, 5 random sensors were chosen from each cluster for training an ML model. These sensors that were chosen are marked in red and are also shown in Fig. 5.

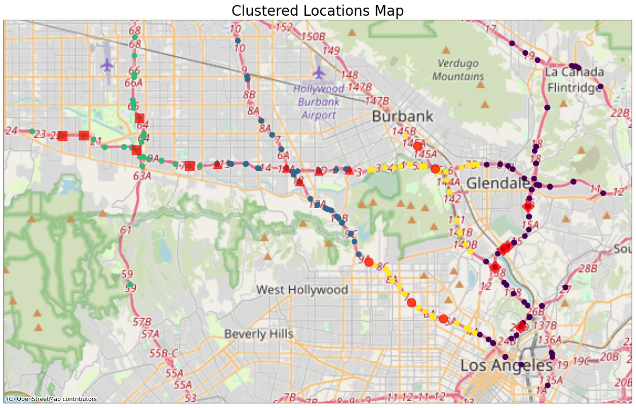
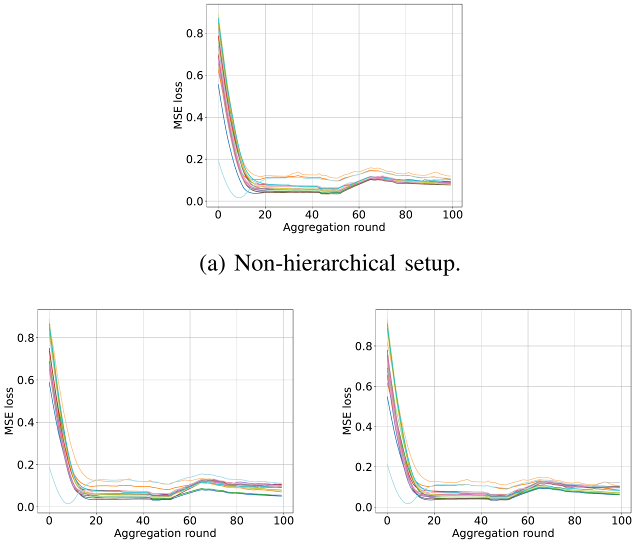


Fig. 5: Clustered sensors using the METR-LA dataset.

To simulate continuous learning, we use 3 weeks of training and 1 week of validation. After each aggregation round, the global time shifts for some timestamps so that the number of training and test samples stays the same, but it is shifted to simulate time passing. The MSE results for 100 aggregation rounds, using 20 clients trained for 5 epochs and 3 different setups, are shown in Fig. 6.

In this figure, one can see the MSE that has been calculated for each client directly after receiving the global model for up to 100 aggregation rounds when having a non-hierarchical setup (Fig. 6a), a hierarchical setup based only on the locations (Figure 6b) and a hierarchical setup using HFLOP (Fig. 6c).



(a) Non-hierarchical setup.

(b) Hierarchical setup with 4 edge servers and 2 local aggregations before a global aggregation.

(c) HFLOP setup with 4 edge servers and 2 local aggregations before a global aggregation.

Fig. 6: MSE of 20 clients, each of them is represented in a different color. Each client tests on its local data after receiving the global model.

Fig. 6 demonstrates that the global MSE improves after the first few aggregation rounds as the MSE significantly decreases for most clients. For all settings, the model converges after about 20 aggregation rounds; for the non-hierarchical setup, this means that about 20 aggregation rounds with the cloud server are needed, whereas in the hierarchical setups, 20 aggregation rounds with the local servers were required and only 10 cloud aggregations. The MSE slightly increases after around 50 aggregation rounds. One reason for this increase may be the changing data. However, after some time, the MSE decreases again, showing that there are some oscillations, but overall, the model indeed converges.

C. Inference serving performance

1) Benchmarks: The subsequent experiments demonstrate the performance when inference requests are served while the clients continuously train. The clustering strategies for the hierarchical experiments remain the same. In addition, each of the federated learning devices is assigned a rate λ_i at which it produces inference requests. For the hierarchical federated learning experiments, edge servers are added, each having a certain capacity r_j to handle inference requests. If an edge server receives more inference requests than it can handle, it forwards the additional requests to the cloud server.

a) Non-hierarchical FL benchmark: In the non-hierarchical experiment, the federated learning clients busy with training forward their inference requests to a cloud server.

b) Hierarchical FL benchmark: For the hierarchical FL benchmark experiments, the clients are clustered based on their location and send their inference requests to their local aggregator. If the number of inference requests exceeds capacity, the local aggregator forwards the request to the global aggregator.

c) Hierarchical FL with HFLOP: For these experiments, the clients are clustered using HFLOP. Therefore, not only their location but also their inference request rate is considered when clustering. If the number of inference requests exceeds available capacity, the local aggregator forwards the request to the global aggregator.

For the experiments, we assume that the latency for sending requests to the global server/cloud is between 50 and 100 ms. To make reasonable assumptions on latency, we measured HTTP response times (round-trip) to different cloud servers from other devices and networks. The latency cost to the local/edge servers is much lower and estimated between 8 and 10 ms.

2) Inference serving latency: Fig. 7 depicts the time in seconds it took until a client received a response to an inference request. This figure shows that inference serving latency using the vanilla/non-hierarchical architecture is much higher than the other methods. The higher response times are because it takes much more time to send the requests to the cloud server than to edge servers. In these tests, we assumed that the edge servers had comparable processing power for the inference requests.

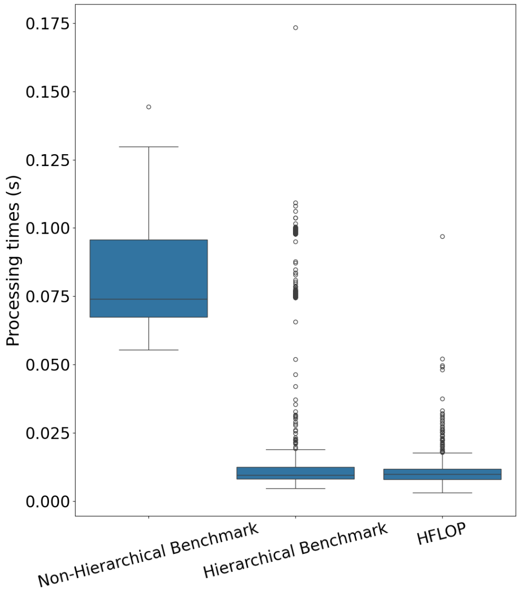
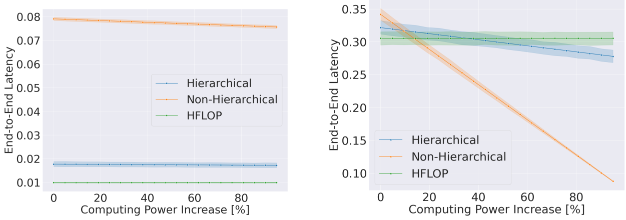


Fig. 7: Response times of inference requests.

The average response times were for the non-hierarchical benchmark 79.07 ms ± 15.94, for the hierarchical benchmark 17.72 ms ± 24.26, and for HFLOP 9.89 ms ± 4.63.

3) End-to-end latency across different computing capacities: This section compares the end-to-end latency when assuming different computing capacities on edge and cloud servers. In the previous section, the assumption was made that the servers placed on edge nodes are as powerful as cloud servers. Since this is often not the case, we compare the methods, considering a theoretical speedup of up to 95%.



(a) End-to-end latency with speedup and inference request rates $\lambda_i,\ \forall\, i \in [1, n]$.

(b) End-to-end latency with speedup and inference request rates $\lambda_i \times 10,\ \forall\, i \in [1, n]$.

Fig. 8: End-to-end latency with different speedups and inference request rates.

In Fig. 8, one can see the end-to-end latency of the three methods. The non-hierarchical benchmark is depicted in orange, the hierarchical benchmark is visualized in blue, and HFLOP is visualized in green. Since the inference time is much lower than network-induced latency, there is almost no difference when increasing the speedup in Fig. 8a. When the inference request rate is increased 10 times (Fig. 8b), the non-hierarchical method has a lower average latency than the other methods when the speedup is above 14.25.

D. Cost savings

We then turn our attention to communication costs. We simulate a scenario with $n = 500$ devices and increasing edge node densities with the following setup: For each device, there is a single edge node accessible at zero cost (e.g., a host in the same local area network), while the rest are accessible at unit cost. All edge hosts communicate with the global server at unit cost, and we force all clients to participate in training. Inference workloads and capacities are drawn uniformly at random. We compare the following three mechanisms: (i) vanilla FL, (ii) HFLOP, and (iii) the uncapacitated variant of HFLOP, where each edge host is assumed to have infinite request processing capacity. The ML model under training is the one applied in our use case, whose size in serialized format is 594 KB. This is the payload uploaded/downloaded by an entity that submits a model update and receives a new model version. We configure HFL so that one global aggregation round takes place every two local ones; this is a rather conservative approach from a cost perspective, as it leads to a larger number of costly global aggregations. We compare the three approaches in terms of their communication cost *until convergence*, expressed as *the volume of traffic exchanged over metered links*, i.e., excluding the traffic exchanged over zero-cost connections. In the experiments presented in Fig. 6, convergence is ensured after approximately 100 aggregation rounds in all scenarios, which is what we also assume for this experiment. (If we assume convergence after 20 rounds instead, as discussed in Section V-B2, the results in absolute traffic volume terms we report below are scaled down accordingly.) Given our configuration, this corresponds to 50 global aggregation rounds for the hierarchical FL experiments. The

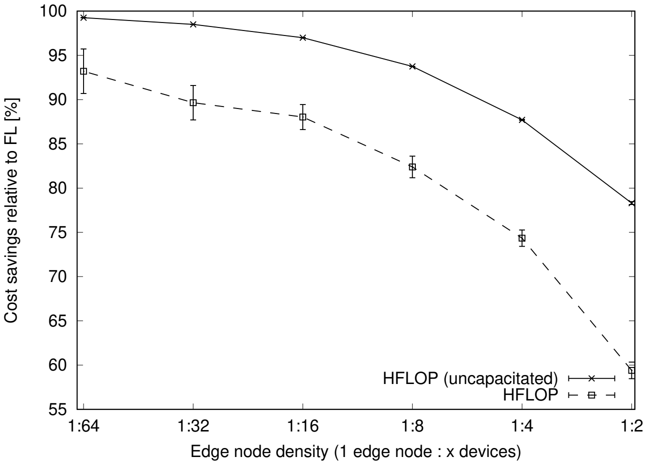


Fig. 9: Cost savings relative to standard FL for increasing edge node densities and a fixed number of $n = 200$ devices. Mean values are reported with 95% confidence intervals.

uncapacitated HFLOP variant serves as the lower bound in terms of communication cost: since inference workloads of devices and the inference processing capacity of individual edge nodes are ignored, the derived solution is the best one could expect from a cost perspective.

Fig. 9 reports the communication cost reduction (%) the two HFL variants achieve against standard FL. In our setup, these savings are more drastic when edge host density is low: devices always have an edge host reachable at zero cost, which may be used if capacity constraints (if any) allow it and costly aggregator-cloud links are only a few. In practice, though, this would necessitate a few very high-capacity *and* cheaply reachable edge nodes. In more general settings (i.e., where devices and edge nodes have diverse communication costs and not unit ones), the cost gap between the two HFL variants is much more narrow. Also, this cost gap gets narrower the more the overall available capacity; the capacity configurations we selected for this experiment favor the uncapacitated version of HFLOP. Finally, in absolute traffic volume terms and for the clustered topology we experimented with in our use case (4 edge nodes, 20 devices), approximately 2.37 GB, 0.53 GB, and 0.24 GB was transmitted over metered links until GRU model training convergence using FL, HFLOP, and its uncapacitated version, respectively.

VI. Discussion

Variants of the orchestration problem. HFLOP can be extended in various ways to capture different operational requirements. Straightforward extensions include enforcing privacy-related constraints, where a device is allowed to associate only with edge nodes that it trusts, and considering device reliability in assignment decisions as long as such per-device information exists. These can be implemented with modified or additional HFLOP constraints. More importantly, current hierarchical federated learning designs often consider the effects of differences in client data distributions on learning performance, adding this dimension in their configuration decisions [15]. Solving HFLOP while ensuring balanced data distributions among device clusters is an interesting and challenging problem.

Alternatives for inference serving. In our problem settings, we assume that a device that is busy training can offload all its inference workload to its associated edge node. Other inference-serving schemes, however, can be considered. For instance, trading inference quality for fast local execution, a device whose GPU is busy with an FL task can use a lower-complexity ML model for the same inference task suitable for CPU execution, such as a quantized version of the global model. Furthermore, more intelligent request routing strategies could be implemented at the device and the edge host (aggregator) end.

Dealing with environment dynamics. In practical FL settings, it is reasonable to expect that the device population fluctuates and that the conditions at edge nodes may change, e.g., in the presence of other application workloads that may affect the request processing capacity of edge nodes. Such changes may imply that a solution to HFLOP is not optimal anymore, and the question of what conditions should trigger its re-execution remains open. Adapting an HFL configuration in response to such changes requires considering reconfiguration costs and potentially applying heuristic schemes to deal with the complexity of HFLOP. This is the subject of our ongoing work on *adaptive FL orchestration*.

VII. Conclusion

This work addressed the challenge of inference serving during training in a federated learning environment. This challenge is particularly evident in continuous learning settings, where models must be retrained while serving inference requests. To this end, we introduced the inference-aware Hierarchical Federated Learning Orchestration Problem (HFLOP), which clusters federated learning clients considering their individual inference request workload and network costs, ensuring that inference requests are served as close to the clients as possible, lowering communication costs and response times. Our systematic experiments show that our orchestration scheme decreases the communication cost without affecting the continuous training of the federated learning clients. Our experiments also demonstrated significantly reduced end-to-end inference latency, achieving better results by processing inference requests on nearby edge servers. These performance benefits were sustained even for inference processing workloads that exceeded edge infrastructure capacity and in the presence of highly asymmetric compute resource capabilities between edge and cloud hosts.

Acknowledgment

This work has been supported by the European Union's Horizon Europe research and innovation program under grant agreements No. 101079214 (AIoTwin) and No. 101135576 (INTEND).

References

[1] S. Deng, H. Zhao, W. Fang, J. Yin, S. Dustdar, and A. Y. Zomaya, "Edge intelligence: The confluence of edge computing and artificial intelligence," *IEEE Internet Things J.*, vol. 7, no. 8, pp. 7457–7469, 2020.

[2] W. Y. B. Lim, J. S. Ng, Z. Xiong, J. Jin, Y. Zhang, D. Niyato, C. Leung, and C. Miao, "Decentralized edge intelligence: A dynamic resource allocation framework for hierarchical federated learning," *IEEE Trans. Parallel Distrib. Syst.*, vol. 33, no. 3, pp. 536–550, 2022.

[3] A. Furtuanpey, P. Raith, and S. Dustdar, "Frankensplit: Efficient neural feature compression with shallow variational bottleneck injection for mobile edge computing," *IEEE Trans. Mobile Comput.*, pp. 1–17, 2024.

[4] J. Wang, Y. Hong, J. Wang, J. Xu, Y. Tang, Q.-L. Han, and J. Kurths, "Cooperative and competitive multi-agent systems: From optimization to games," *IEEE/CAA J. Autom. Sinica*, vol. 9, no. 5, pp. 763–783, 2022.

[5] K. Wong, M. Nguyen-Duc, K. Le-Huy, L. Ho-Tuan, C. Do-Danh, and D. L. Phuoc, "An empirical study of federated learning on iot-edge devices: Resource allocation and heterogeneity," *CoRR*, vol. abs/2305.19831, 2023. [Online]. Available: https://doi.org/10.48550/arXiv.2305.19831

[6] L. Liu, J. Zhang, S. Song, and K. B. Letaief, "Client-edge-cloud hierarchical federated learning," in *Proc. IEEE ICC*, 2020.

[7] M. S. H. Abad, E. Ozfatura, D. Gündüz, and Ö. Erçetin, "Hierarchical federated learning ACROSS heterogeneous cellular networks," in *Proc. IEEE ICASSP*, 2020.

[8] Z. Wang, H. Xu, J. Liu, H. Huang, C. Qiao, and Y. Zhao, "Resource-efficient federated learning with hierarchical aggregation in edge computing," in *Proc. IEEE INFOCOM*, 2021.

[9] 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 Trans. Parallel Distributed Syst.*, vol. 34, no. 1, pp. 328–345, 2023.

[10] 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 Trans. Parallel Distributed Syst.*, vol. 34, no. 5, pp. 1560–1579, 2023.

[11] Y. Guo, F. Liu, T. Zhou, Z. Cai, and N. Xiao, "Privacy vs. efficiency: Achieving both through adaptive hierarchical federated learning," *IEEE Trans. Parallel Distributed Syst.*, vol. 34, no. 4, pp. 1331–1342, 2023.

[12] Y. Sai, X. Wu, J. Jiang, Y. Huang, Q. Yan, Z. Li, and H. Huang, "Optimizing hierarchical federated learning: A reinforcement learning approach," *IEEE Trans. Consum. Electron.*, 2024, early access.

[13] M. F. Pervej, R. Jin, and H. Dai, "Hierarchical federated learning in wireless networks: Pruning tackles bandwidth scarcity and system heterogeneity," *IEEE Trans. Wireless Commun.*, 2024, early access.

[14] S. Trindade and N. L. S. da Fonseca, "Client selection in hierarchical federated learning," *IEEE Internet Things J.*, 2024, early access.

[15] Y. Deng, F. Lyu, T. Xia, Y. Zhou, Y. Zhang, J. Ren, and Y. Yang, "A communication-efficient hierarchical federated learning framework via shaping data distribution at edge," *IEEE/ACM Trans. Netw.*, vol. 32, no. 3, pp. 2600–2615, 2024.

[16] D. Shenaj, M. Toldo, A. Rigon, and P. Zanuttigh, "Asynchronous federated continual learning," in *Proc IEEE/CVF CVPR Workshops*, 2023.

[17] M. F. Criado, F. E. Casado, R. Iglesias, C. V. Regueiro, and S. Barro, "Non-iid data and continual learning processes in federated learning: A long road ahead," *Information Fusion*, vol. 88, pp. 263–280, 2022.

[18] P. Han, S. Wang, Y. Jiao, and J. Huang, "Federated learning while providing model as a service: Joint training and inference optimization," in *Proc. IEEE INFOCOM*, 2024.

[19] S. Thrun, "A lifelong learning perspective for mobile robot control," in *Intelligent robots and systems*. Elsevier, 1995, pp. 201–214.

[20] L. Wang, X. Zhang, H. Su, and J. Zhu, "A comprehensive survey of continual learning: Theory, method and application," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 46, no. 8, pp. 5362–5383, 2024.

[21] M. McCloskey and N. J. Cohen, "Catastrophic interference in connectionist networks: The sequential learning problem," in *Psychology of learning and motivation*. Elsevier, 1989, vol. 24, pp. 109–165.

[22] Z. Li and D. Hoiem, "Learning without forgetting," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 40, no. 12, pp. 2935–2947, 2017.

[23] J. Kirkpatrick, R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska *et al.*, "Overcoming catastrophic forgetting in neural networks," *Proceedings of the National Academy of Sciences*, vol. 114, no. 13, pp. 3521–3526, 2017.

[24] J. Le, X. Lei, N. Mu, H. Zhang, K. Zeng, and X. Liao, "Federated continuous learning with broad network architecture," *IEEE Trans. Cybern.*, vol. 51, no. 8, pp. 3874–3888, 2021.

[25] X. Yang, H. Yu, X. Gao, H. Wang, J. Zhang, and T. Li, "Federated continual learning via knowledge fusion: A survey," *IEEE Trans. Knowl. Data Eng.*, vol. 36, no. 8, pp. 3832–3850, 2024.

[26] D. Wen, K.-J. Jeon, and K. Huang, "Federated dropout—a simple approach for enabling federated learning on resource constrained devices," *IEEE Wirel. Commun. Lett.*, vol. 11, no. 5, pp. 923–927, 2022.

[27] Z. Jiang, Y. Xu, H. Xu, Z. Wang, and C. Qian, "Heterogeneity-aware federated learning with adaptive client selection and gradient compression," in *Proc. IEEE INFOCOM*, 2023.

[28] Q. Zhang, Y. Li, D. Zhang, I. Murturi, V. C. Pujol, S. Dustdar, and S. Wang, "Intelligence inference on IoT devices," in *Learning Techniques for the Internet of Things*, P. K. Donta, A. Hazra, and L. Lovén, Eds. Cham: Springer Nature Switzerland, 2024, pp. 171–195.

[29] X. Yuan, J. Chen, J. Yang, N. Zhang, T. Yang, T. Han, and A. Taherkordi, "FedSTN: graph representation driven federated learning for edge computing enabled urban traffic flow prediction," *IEEE Trans. Intell. Transp. Syst.*, vol. 24, no. 8, pp. 8738–8748, 2023.

[30] C. Zhang, L. Cui, S. Yu, and J. James, "A communication-efficient federated learning scheme for iot-based traffic forecasting," *IEEE Internet Things J.*, vol. 9, no. 14, pp. 11 918–11 931, 2021.

[31] Y. Liu, J. James, J. Kang, D. Niyato, and S. Zhang, "Privacy-preserving traffic flow prediction: A federated learning approach," *IEEE Internet Things J.*, vol. 7, no. 8, pp. 7751–7763, 2020.

[32] S. Hu, Y. Ye, Q. Hu, X. Liu, S. Cao, H. H. Yang, Y. Shen, P. Angeloudis, L. Parada, and C. Wu, "A federated learning-based framework for ride-sourcing traffic demand prediction," *IEEE Trans. Veh. Technol.*, vol. 72, no. 11, pp. 14 002–14 015, 2023.

[33] D. B. Shmoys, É. Tardos, and K. I. Aardal, "Approximation algorithms for facility location problems," in *Proc. ACM STOC*, 1997.

[34] V. Arya, N. Garg, R. Khandekar, A. Meyerson, K. Munagala, and V. Pandit, "Local search heuristic for k-median and facility location problems," in *Proc. ACM STOC*, 2001.

[35] Y. Li, R. Yu, C. Shahabi, and Y. Liu, "Diffusion convolutional recurrent neural network: Data-driven traffic forecasting," *Proc. International Conference on Learning Representations (ICLR)*, 2018.