Native Support for Federated Learning Hyper-Parameter Optimization in 6G Networks

Mohammad Bariq Khan*†, Xueli An*, and Falko Dressler†
*AWTL, Munich Research Center, Huawei Technologies, Munich, Germany
†School for Electrical Engineering and Computer Science, Technische Universität Berlin, Berlin, Germany
mohammad.bariq.khan1@huawei.com, xueli.an@huawei.com, dressler@ccs-labs.org

***Abstract*—6G mobile communication networks are envisioned to be AI-native, that is, the provision of AI services to users as well as the network itself would be one of the most essential aspect for its system architecture and design. To provide these services and in order to make correct design decisions, deep understanding of the relationship between the wireless networks and the different AI paradigms along with their various functional aspects is needed. In this paper, we explore the relationship in one such AI paradigm, namely federated learning (FL). More specifically, we aim to look at the problem of hyper-parameter optimization (HPO) for hierarchical federated learning by exploring two design scenarios: (1) HPO is carried out at intermediary aggregators which are implemented at edge servers and (2) HPO is realized by a global server such as a cloud-based approach. Empirical results show that although offloading the HPO algorithm completely to the edge-servers does lower the costs, the hybridized approach whereby the global server realizes the HPO algorithm and the edge-server offers some assistance, can provide at least twice as much savings on training costs. We further discuss the two approaches for providing FL as a native service from the mobile communication network and the implications on the network architecture in light of the discussed design scenarios.**

***Index Terms*—AI-native Networks, Hierarchical Federated Learning, Hyper-Parameter Optimization**

I. Introduction

As the mobile communication networks continue to evolve toward 6G, they are expected to become more intelligent with the integration of new state-of-the-art artificial intelligence (AI) technologies [1]. The next generation of networks are envisioned to enable the transfer of intelligence from the central cloud to the mobile communication system [2], thereby leading to what is referred to as AI-native networks. The term AI-native network covers two broad areas: First referred to as AI for communication network (AI4Net), that is, the use of machine learning techniques to automate network operations and enhance performance. The other aspect, communication network for AI (Net4AI), focuses on enabling the network system with built-in AI capabilities to facilitate distributed learning and inference [3]. This would require integrating communication and computation capabilities to facilitate the shift from cloud-based centralized learning to distributed network infrastructure [4], which is in line with the edge support in 6G networks [5].

Federated learning (FL) is a prominent distributed learning paradigm that was introduced by Google researchers as a privacy-preserving distributed machine learning framework [6]. In FL, the data-owners or clients train local models on their private data and share only the model parameters with a central server, which in turn aggregates these parameters from all the clients and disseminates the aggregated model back to clients, thereby eliminating the need to share private data.

The implementation of federated learning over the mobile communications networks faces several challenges ranging from efficiency and effectiveness to security issues. These have been widely studied in the literature and various proposals have been put forward to address these issues [7]–[9]. The communication network plays a pivotal role with regards to the communication efficiency of FL. While most of the focus in the current literature is on how to improve the FL algorithm to suit the current networks, we aim to look at the requirements on the mobile communication networks to aid the implementation of FL as well as provide it as a service from the network itself. A rather obvious but not necessarily simple solution is to increase network capacity. While that remains one of the core motivations for every generation of mobile networks, we aim to look at other aspects of AI in general, and FL in particular, to make the networks AI-native.

In this paper we are concerned with the issue of hyper-parameter optimization (HPO) for hierarchical federated learning (HFL). Our contributions can be summarized as follows:

- We explore the problem of HPO in a hierarchical federated learning scenario for 6G edge-enabled networks;
- We integrate an HPO algorithm in the HFL framework and use it to optimize three hyper-parameters: number of participating clients, number of local epochs at the clients, and the number of epochs between the clients and the intermediary servers; and
- We empirically evaluate and compare local optimization of hyper-parameters at the edge servers and global optimization at the cloud server in terms of training costs.

II. Related Work

Although hyper-parameter optimization for centralized learning has been widely studied [10], there has been quite limited research on HPO for federated learning. Nevertheless, a few studies already have explored this problem. For example, Khodak, Tu, Li, Li, Balcan, Smith, and Talwalkar [11] proposed to use weight sharing to improve the round-to-accuracy performance of FL. Their method, called FedEx, manages to achieve higher accuracy than the baselines within a same budget.

Another approach called FLORA [12] has been proposed in which the authors create a set of global hyper-parameters from the ones that work the best for the clients individually and then select the best of these using four proposed techniques. The authors in [13] propose a genetic algorithm-based tuning framework for clustered federated setup, where they use a combined density and hyper-parameter based clustering and optimize the hyper-parameters using the evolutionary genetic algorithm. In [14], the authors proposed a particle swarm optimization approach for tuning the FL parameters which accelerates the training process as compared to the genetic algorithm based tuning. Zhang, Zhang, Liu, Mohapatra, and DeLucia [15] proposed a light-weight automatic HPO tuning framework for FL, named FedTune, which takes into consideration the diverse training preferences of applications in terms of computational and transmission times and loads achieving an improvement of about 8-26% over considered baselines. However, all these works consider the classical two-layer FL scenario with a number of clients and a global server. To extend the work to a more generic framework, we assume a three layer FL scenario whereby a set of clients is associated with an edge-server and a number of such edge servers are connected to a global server.

III. Problem Statement and System Model

To provide native support for federated learning within the mobile communication network, we need to understand the dependency of various aspects of FL on the communication networks. While throughput capacity remains a core communication factor that determines FL convergence, the selection of optimized hyper-parameters for FL plays a critical role. As such, it is interesting to investigate how to design a HPO framework for FL and its implications on the mobile network architecture. The result of these investigations can then be used to make design decisions and add new services/interfaces in the mobile network architecture. In an HFL scenario, the choice of whether the optimization occurs at the global or intermediate edge-servers would thereby determine the interactions between the FL and network entities. As such, we first need to determine which of the two cases mentioned above reap the maximum performance benefits. In this work we try to answer the following question: How does the hyper-parameter optimization at the edge-servers impact the training costs for federated learning in comparison to global optimization?

A. System Model

We first define the analytical models used for evaluating the system costs. Table I summarizes the different symbols used in the system model.

1) Hierarchical Federated Learning Model: We consider the hierarchical model for federated learning whereby **N** clients are split among s intermediary edge-servers. Indexed by l, each edge-server has a disjoint subset with a total of n^l clients associated with it. The edge-servers aggregate the parameters from these clients and transmit them to a central server (e.g. cloud-based) for further aggregation and are then disseminated back to the clients through the edge servers. Due to space limitation we skip the analytical description here and refer the reader to our previous work with a similar model [16].

TABLE I
Symbols used in the system model

Symbol	Description
N	The total number of clients available for participation
P_t	The client set selected for a particular epoch
K	Number of clients in P_t
D_j	Data-set at client j
E_c	Number of local epochs or SGD iterations at each client
E_b	Number of edge epochs or averaging iterations at edge server
r_j	Transmission rate at client j
B	Total bandwidth at each base station
z	Size of the learned model at the client or edge server
p_j	Uploading power at client j
a_j	Proportion of bandwidth allocated to client j
h_j	Wireless channel gain of client j
f_j	Processor frequency at client j
α, β	Computation and transmission time co-efficients
γ, δ	Computation and transmission load co-efficients

2) Communication Model: The transmission rate of communication of j^{th} client with its associated edge-server is given by

$$r_j = a_j B log_2(1 + \frac{p_j h_j}{a_j B N_0}), \tag{1}$$

where B is the total bandwidth, h_j is the channel gain, N_0 is the noise power density and a_j is the proportion of the bandwidth allocated to client j. With z as the size of the transmitted model, the time cost of uploading, t_u, for the j^{th} client of the l^{th} server would be

$$t^l_{u,j} = \frac{z}{r_j} = \frac{z}{a_j B log_2(1 + \frac{p_j h_j}{a_j B N_0})}. \tag{2}$$

Assuming that there is a fixed rate connection between each base station and the cloud with a rate r_l, the time cost for uploading is given by

$$t_{u,l} = \frac{z}{r_l}. \tag{3}$$

For one global epoch the transmission time can then be defined as

$$TransmissionTime = \max_{l=1}^{s}[E^l_b \cdot \max_{j=1}^{n^l}(t^l_{u,j})] + \sum_{l=1}^{s} t_{u,l} + t_d, \tag{4}$$

where E^l_b is the number of rounds between the l^{th} edge server and it's associated clients. For simplicity, we assume a constant download time, t_d for the whole epoch.

The transmission load for each global epoch depends on the number of client-edge rounds, E_b and the total number of participating clients K as

$$TranmissionLoad = C \sum_{l=1}^{s} E^l_b \cdot K^l, \tag{5}$$

where C is a constant corresponding to the model size or the number of model parameters.

3) Computation Model: For the computation model, we first note that the computation latency is directly proportional to the number of local epochs that are executed, E_c and the size of the data that is available at a particular client. Therefore,

we define the computation latency, t_n, for client j of the l^{th} server as

$$t^l_{n,j} = \frac{\alpha_0}{f_j} E^l_c |D_j|, \tag{6}$$

where f_j is the processor frequency, $|D_j|$ is the number of data points at the given client and α_0 is an empirical parameter that depends on the model being trained and the software running on the client.

Thus, for one global epoch, the total computation time is

$$ComputationTime = \max_{l=1}^{s}[E^l_b \cdot \max_{j=1}^{n^l}(t^l_{n,j})]. \tag{7}$$

Note that in federated learning most of the compute intensive work is carried out at the local clients and the edge-servers execute only the averaging computation which is relatively lightweight. Hence for simplicity, we ignore the computation at the edge servers in the computation model.

The computation load in each global epoch is the sum of all the computation load at the clients and depends on the number of data points at each client and the number of edge-client rounds. It can thus be formulated as

$$ComputationLoad = \alpha_0 \cdot \sum_{l=1}^{s} E^l_b \cdot E^l_c \sum_{j=1}^{n^l} |D_j|. \tag{8}$$

IV. Hierarchical Federated Learning with Hyper-Parameter Optimization

Federated learning can be expensive both in terms of computation as well as communication. Besides the commonly known hyper-parameters in deep learning training, like number of training epochs and learning rate, hierarchical federated learning introduces a few more hyper-parameters like the number of participating clients, number of local epochs at the clients, and the number of iterations between the clients and edge servers. The choice of these parameters significantly affects the training costs. As such, we integrate a HPO algorithm with HFL to design and explore the three hyper-parameter optimization strategies that we describe in the following sections.

Algorithm 1 HFL with Local HPO

```
Initialize the model on the cloud server
while Global model accuracy lower than target accuracy do
    At each edge server l = 1, 2 ..., s in parallel
    Set the edge model same as server
    Initialize K & E_c
    while Edge model accuracy lower than target accuracy do
        P_t ← ClientSelection(K)
        for i ϵ P_t clients in parallel do
            Set the client model same as associated edge server
            for t_1 = 1, 2 ..., E_l do
                local training at client
            Upload model to associated edge server
        Aggregation at Edge servers
        K, E_c ← HPO()
    Upload model to cloud server
    Aggregation at cloud server
```

A. Optimization at the Edge Server

The first strategy is to implement the HPO at the edge servers, where each server optimizes the hyper-parameters for its associated set of clients independently (local optimization). Here, we consider the optimization of two FL hyper-parameters: total number of clients selected $\mathbf{K}$ and number of local epochs $\mathbf{E}_c$. The pseudo-code for the algorithm is described in Algorithm 1. Both HPO as well as client selection is implemented at the edge servers.

In this scenario, we first initialize the two hyper-parameters with some baseline values at each edge server in parallel. The servers then select a random set of clients among their associated clients. Next, the training rounds between the edge servers and clients are executed and the hyper-parameters are updated based on the HPO decision after each edge-client round. This continues until the target accuracy is reached at all the edge-server models, following which the models are aggregated at the global server. The global epochs are executed until the target accuracy is reached for the global model.

B. Optimization at the Global Server

The second approach is to implement both client selection and HPO at the global server. The pseudocode for this scenario is described in Algorithm 2. Here, in addition to total number of clients selected $\mathbf{K}$ and number of local epochs $\mathbf{E}_c$, we also optimize for the number of communication rounds between the edge-servers and the clients, $\mathbf{E}_b$.

The HPO decision is carried out after each global epoch following which the three hyper-parameters are updated. As such, the entire client set is selected at the global server which remains constant for the whole global epoch.

C. Hybrid Optimization

In addition to the two cases above, we can adopt another approach which is essentially a hybridized variant of the above two cases. Here, both the global as well as the edge server take part in the optimization. The number of clients selected $\mathbf{K}$ and number of local epochs $\mathbf{E}_c$ are determined by the global server at the beginning of the local epoch, while the number of edge-client iterations $\mathbf{E}_b$ are determined by the edge-servers

Algorithm 2 HFL with Global HPO

```
Initialize the model on the cloud server
Initialize K, E_c & E_b
P_t ← ClientSelection(K)
while accuracy lower than target accuracy do
    for l = 1, 2 ..., s edge servers in parallel do
        Set the edge model same as server
        for t_2 = 1, 2 ..., E_b do
            Set the client model same as associated edge server
            for t_1 = 1, 2 ..., E_c do
                local training at client
            Upload model to associated edge server
        Aggregation at Edge servers
        Upload model to cloud server
    Aggregation at cloud server
    K, E_c, E_b ← HPO()
```

independently. As in the edge-optimization scenario, the servers stop the edge-client training iterations as soon as the target accuracy is achieved for the edge-model. This decoupling could potentially allow to avail the benefits of heterogeneity in the training resources at different locations while reducing the complexity of the HPO algorithm at the global server.

V. Performance Evaluation

We evaluate the performance of the three algorithms on two standard image classification data-sets: MNIST and CIFAR-10. For MNIST, we use a CNN with two convolution layers and two fully connected layers and for CIFAR-10 data set, we train a standard ResNet-18 model. The data is split among $N = 100$ clients and these clients are then randomly split into 4 groups, each associated with one edge server. For these experiments, the clients are assumed to have the same computation capacity, and the clients are selected randomly. The bandwidth at each base station is assumed to be $B = 20MHz$ and $N_0 = 5 \times 10^{-20}$. The channel gains are generated from an exponential distribution, $h_j = g_0(d_0/d)^{\theta} exp(1)$ with $g_0 = 10^{-4}$, $d_0 = 1$, $\theta = 4$ and $d = 200$. The model is implemented using Python 3.8 with simulations running on an Intel i7-8650U CPU. For the HPO, we integrate FedTune [15] in Steps 14 and 15 in Algorithm 1 and Algorithm 2 respectively.

A. FedTune Preliminaries

FedTune is a hyper-parameter tuning algorithm that automatically adjusts the hyper-parameters for federated learning to cater to the specific requirements of different FL applications. It aims to optimize the system overhead for FL by reducing computation and/or communication loads and times. For a specific application, FedTune takes into consideration the training preferences in terms of the four metrics: Computation Time (α), Transmission Time (β), Computation Load (γ), and Transmission Load (δ), as described above, where $\alpha + \beta + \gamma + \delta = 1$. The higher the co-efficient for a metric means the application is more concerned about that particular metric.

Given two sets of parameters, S_{cur} & S_{nxt} and the corresponding training metrics of computation time(t_{cur}, t_{nxt}), transmission time (q_{cur}, q_{nxt}), computation load(z_{cur}, z_{nxt}), and transmission load (v_{cur}, v_{nxt}), FedTune minimizes the following objective function:

$$G(S_{nxt}) = \alpha \times \frac{t_{nxt} - t_{cur}}{t_{cur}} + \beta \times \frac{q_{nxt} - q_{cur}}{q_{cur}} + \gamma \times \frac{z_{nxt} - z_{cur}}{z_{cur}} + \delta \times \frac{v_{nxt} - v_{cur}}{v_{cur}} \quad (9)$$

FedTune finds the next set of optimal parameters by calculating the derivative of Equation (9) over the candidate hyper-parameters. Based on the signs of these derivatives, it determines whether to increase or decrease the candidate hyper-parameters. Given its light-weight nature and the feature of tuning the parameters during training while considering different training preferences for different applications, we choose FedTune as the optimizer in our algorithms.

B. Relative Performance Improvement

In the first set of experiments, we evaluate the relative performance gain in the three optimization scenarios over the baseline for a set of training preferences. For the baseline measurements, we set arbitrary values for the candidate hyper-parameters and measure the four metrics required to achieve a set accuracy target. Then, for each training preference, we enable FedTune with local, hybrid and global optimization and measure the relative performance improvement over the baseline using Equation (9).

For MNIST, the baseline measurements were taken with $K = 60$, $E_c = 20$, $E_b = 8$, and the target model accuracy was set to 0.8. For CIFAR-10, the baseline was set at $K = 60$, $E_c = 50$, $E_b = 8$ with a target accuracy of 0.4. The performance of the three scenarios on CIFAR-10 and MNIST is shown in Tables II and III, respectively. The first row shows the baseline measurement and the remaining ten rows depict the performance for the different training preferences. The rounded average values of the candidate hyper-parameters are also shown. Note that, in the local scenario, K' is the round up average value at each edge server (so total K = 4 $\times$ K').

From the two tables we clearly observe that local and hybrid optimization performs significantly better than the global optimization achieving an improvement between 11-50% for both the data sets. On the contrary, in the case of global optimization the improvement is of much lower order, between 0.5-16%. The hybrid optimization case, although of the same order as the local one, generally outperforms the other two.

C. Cost Evaluation

In the following sections, we study the variation of the training costs across a range of different accuracy targets. First, we set the training preferences so as to favor each of the four cost metrics individually by setting the corresponding co-efficient to one, and then measure the said cost for different accuracy targets in the three optimization scenarios. Furthermore, we compare the three with the baseline cost measured with fixed hyper-parameters as before.

The results are depicted in Figure 1. Clearly, the general observation from all the four cases is that the hybrid and local optimization outperforms the global one across the entire range of accuracy targets for both the data-sets. The performance between the local and hybrid optimization is comparable particularly in the higher accuracy regions.

The two most commonly referred to performance indicators for the evaluation of federated training are (i) service response latency, which is determined by computation and communication time, and (ii) service energy efficiency, which is determined by the computation and communication energy expenditure. In terms of the four metrics we have defined, it is easy to see that service response latency is proportional to computation and transmission time while the service energy efficiency is proportional to computation and communication load. Therefore, to evaluate the overall performance, we use the combined values of these defined metrics. However, as the order of the magnitude varies significantly among these

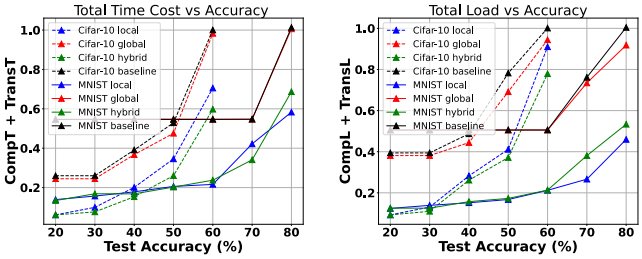
TABLE II
TRACE FOR CIFAR10 DATASET

α	β	γ	δ	Final $\mathbf{K}', \mathbf{E}_c'$	Local Optimization				Gain(%)	Final $\mathbf{K}, \mathbf{E}_c'$	Hybrid Optimization				Gain(%)	Final $\mathbf{K}, \mathbf{E}_c', \mathbf{E}_b'$	Global Optimization				Gain(%)
					CompT	TransT	CompL	TransL			CompT	TransT	CompL	TransL			CompT	TransT	CompL	TransL	
0	0	0	0													60, 50, 8	12000	9.75	720000	1440	
0.25	0.25	0.25	0.25	16, 50	9797	7.25	542930	1075	+23.5	60, 50	6020	5.33	362220	722	**+48.7**	60, 50, 6	11067	8.84	667400	1326	+8.1
1	0	0	0	19, 41	8363	8.24	572580	1269	+30.3	62, 48	5980	6.57	351780	706	**+50.2**	62, 48, 6	10933	8.64	659267	1326	+8.9
0	1	0	0	19, 59	10267	8.63	691323	1252	+11.5	62, 52	7060	5.3	403200	799	**+45.6**	62, 52, 6	11067	8.97	667400	1327	+8.0
0	0	1	0	5, 40	11913	5.37	355146	777	+50.7	59, 49	5980	5.3	351320	705	**+51.2**	59, 49, 6	10685	8.06	637915	1283	+11.4
0	0	0	1	4, 60	19513	4.86	406340	745	+48.3	58, 52	7020	4.5	357680	713	**+50.5**	58, 52, 7	10815	8.41	645585	1283	+10.9
0.5	0.5	0	0	19, 51	9840	8.64	624606	1237	+14.7	63, 51	6020	4.9	362220	722	**+49.8**	63, 51, 5	11815	9.12	745215	1482	+4.0
0	0	0.5	0.5	4, 49	13265	5.68	385290	790	+45.8	58, 50	7020	5.1	373180	744	**+48.2**	58, 50, 6	12065	9.27	718085	1428	+0.5
0.5	0	0	0.5	17, 51	10197	7.24	523903	1033	+21.6	60, 50	8020	6.1	430720	812	**+38.4**	60, 50, 6	10815	9.47	652215	1295	+9.9
0	0.5	0.5	0	16, 51	9197	6.58	534713	1083	+29.1	60, 50	7020	5.3	407220	859	**+44.5**	60, 50, 7	10060	8.13	606660	1206	+16.2
0.5	0	0.5	0	15, 41	9373	7.84	500626	115	+26.2	61, 49	5520	5.5	374780	752	**+50.9**	61, 49, 8	12125	9.78	730686	1476	-1.2
0	0.5	0	0.5	13, 59	10430	6.29	509240	937	+35.2	60, 52	6480	4.9	332220	662	**+51.9**	60, 52, 6	10815	8.62	652215	1296	+10.8

TABLE III
TRACE FOR MNIST DATASET

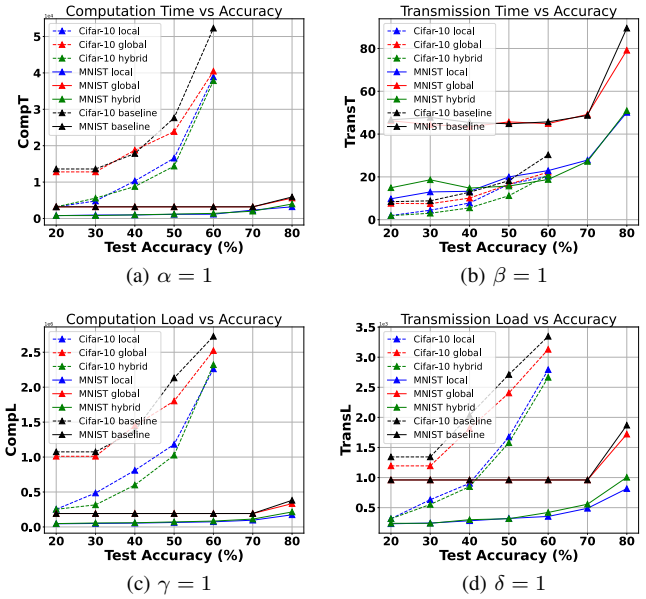
α	β	γ	δ	Final $\mathbf{K}', \mathbf{E}_c'$	Local Optimization				Gain(%)	Final $\mathbf{K}, \mathbf{E}_c'$	Hybrid Optimization				Gain(%)	Final $\mathbf{K}, \mathbf{E}_c', \mathbf{E}_b'$	Global Optimization				Gain(%)
					CompT	TransT	CompL	TransL			CompT	TransT	CompL	TransL			CompT	TransT	CompL	TransL	
0	0	0	0													60, 20, 8	4800	65	288000	1440	
0.25	0.25	0.25	0.25	17, 22	3820	47	222033	1067	+24.2	61, 21	2800	36	155000	775	**+44.7**	61, 21, 7	4670	61	281670	1387	+3.6
1	0	0	0	25, 7	3035	93	234810	2013	+36.7	62, 18	2980	41	161580	814	**+38.0**	62, 18, 6	4530	71	273130	1511	+5.6
0	1	0	0	25, 30	4273	58	345300	1302	+10.7	62, 22	3020	38	166020	824	**+41.5**	62, 22, 6	5110	61	308950	1151	+6.1
0	0	1	0	2, 6	7583	40	123203	991	**+57.2**	58, 18	2780	38	163020	821	+43.4	58, 18, 6	4890	71	270470	1489	+6.0
0	0	0	1	5, 30	6886	27	156126	634	**+55.9**	59, 21	3220	40	170380	846	+41.2	58, 22, 6	5110	70	304250	1373	+4.6
0.5	0.5	0	0	25, 20	3480	61	269510	1337	+16.8	62, 20	3020	44	172420	856	**+34.7**	61, 21, 7	4670	62	281670	1387	+3.6
0	0	0.5	0.5	5, 21	4187	27	131920	652	**+54.4**	58, 20	3020	45	171380	851	+40.7	58, 20, 6	4670	60	278730	1373	+3.9
0.5	0	0	0.5	17, 22	3790	47	211323	1016	+25.6	60, 20	2820	42	163020	809	**+42.5**	60, 20, 6	4670	64	281670	1387	+3.2
0	0.5	0.5	0	16, 23	3680	48	215360	1036	+25.6	60, 20	2820	42	166620	827	**+38.8**	60, 20, 6	4670	66	281670	1387	+0.3
0.5	0	0.5	0	16, 7	3083	65	178786	1455	+36.8	61, 19	2980	40	167180	842	**+39.9**	60, 18, 6	4530	70	273130	1507	+5.4
0	0.5	0	0.5	16, 30	3850	46	230946	913	+32.9	60, 22	2820	37	160220	795	**+43.9**	60, 22, 6	5110	65	281670	1387	+1.8

metrics, we first use min-max normalization for each of them to scale the values to $[0, 1]$, and then add computation time to communication time and computation load to communication load, respectively. Thus, we have an indirect measure of total training time and energy expenditure. Then we set the training preference to optimize the two added metrics together for a range of accuracy targets. The results are depicted in Figure 2. As before, we can observe that the overall performance for both time and load metrics is the better in the case of hybrid and local optimization cases in comparison the global case for both the data-sets.



(a) $\alpha = 0.5, \beta = 0.5$ (b) $\gamma = 0.5, \delta = 0.5$

Fig. 2. Total time and total load joint optimization in both scenarios.



(a) $\alpha = 1$ (b) $\beta = 1$

(c) $\gamma = 1$ (d) $\delta = 1$

Fig. 1. Cost metrics across a range of accuracy targets for both scenarios. For each metric, the corresponding training preference co-efficient was set to 1.

The explanation for the above results lies in the general trend that can be observed from the plots in Figures 1 and 2, that is, a large performance gap between the local and global optimization which tends to decrease as the required number of global epochs increases for higher accuracy targets. In Algorithm 1, Step 6 inherently optimizes the number of edge-client iterations by stopping as soon as the target accuracy is reached. This means, if the local algorithm is run even with fixed hyper-parameters, it would still get better performance than the other one. Because in the global scenario, even for lower accuracy targets, it adds unnecessary iterations. Even though this parameter has been added as an hyper-parameter for FedTune, the global target is already reached before FedTune could converge to the optimum number. With higher accuracy targets, the required number of (edge-client and global) epochs are relatively higher and as such the performance gap narrows as the effect of this inherent optimization in the local algorithm vanishes. When this feature of local optimization is combined with the the HPO running at the global server, the best results are achieved in the hybrid optimization case. This leads to the conclusion that running the HPO for the all the hyper-parameters independently at the edge-servers is not of much benefit in itself. However, optimization for certain hyper-parameters, like in this case edge-client training iterations, $\mathbf{E}_b$ at the edge-servers can help to maximize the benefits of the Global HPO.

VI. Implications and Vision for the FL-Native Architecture

With the above HFL-HPO framework in consideration, the mobile communication network can play two roles: (i) assistance to third party FL applications in terms of exposing networking related analytics for the hyper-parameter optimization or (ii) provisioning FL training as native service.

The first approach could be thought of as a first step towards AI-Native networks. For example, Network Data Analytics Function (NWDAF) has recently been introduced in 5G to provide network performance-related analytics [17]. The service of this network function can be extended to assist third party distributed learning applications that reside on the data-networks. In the case of FL, the application can request the network and UE-related metrics collected by NWDAF to optimize its training hyper-parameters. One such metric could be average packet loss rate, which could be used to determine the number of clients that are selected in each round of the FL algorithm as shown in [16]. Additionally, as suggested by the experiments above, the HPO algorithm can benefit from the UE computation related metrics like computation load and time. In the case where the global server is running in the data network, the UE can expose these metrics directly to the server via the application layer. In such a case, no new data collection capability needs to be added to the NWDAF and the only requirement is to define the standardized interface to the application server to enable the exposure of NWDAF services.

However, for a true native service, we envision a new core network function, Federated Learning Service Function (FLSF), that is capable of provisioning the federated learning services to third party clients. In such a case, the global FL server would be realized by this network function while the edge-servers would be realized at the base-stations. As we have observed in this work, for the hyper-parameter optimization, this FL service provisioning network function would also need to realize the HPO algorithm, while at the base-stations, a lightweight optimizer function could bring additional cost savings. In addition, the Data Collection and Coordination Function (DCCF) that manages the collection of data within the network for NWDAF, could be extended to have distributed instances that are deployed in the access network. This would enable the network to collect and expose non-network related metrics (e.g., computation-related metrics) from the UE to the network.

VII. Conclusions

In this paper, we integrated a hyper-parameter optimization algorithm, FedTune, with hierarchical federated learning in the context of providing native FL support in the next generation of mobile networks. We devised three strategies such that the optimization can be carried out either at the base stations, the global server or both. From the experiments on standard classification data-sets, it was observed that, implementing the HPO at the global server with some aid from the optimization at the edge-servers, in general, produces the best results in terms of saving the overhead training costs, with an average improvement between 40-48% as compared to 30-34% and 4-8% in the local and global scenarios respectively. We also discussed the two approaches for providing native support for FL in the mobile communication network and the broad requirements for new features in the network architecture.

References

[1] C.-X. Wang, X. You, X. Gao, X. Zhu, Z. Li, C. Zhang, H. Wang, Y. Huang, Y. Chen, H. Haas, J. S. Thompson, E. G. Larsson, M. D. Renzo, W. Tong, P. Zhu, X. Shen, H. V. Poor, and L. Hanzo, "On the Road to 6G: Visions, Requirements, Key Technologies and Testbeds," *IEEE Communications Surveys & Tutorials*, vol. 25, no. 2, pp. 905–974, Feb. 2023.

[2] J. Wu, R. Li, X. An, C. Peng, Z. Liu, J. Crowcroft, and H. Zhang, "Toward Native Artificial Intelligence in 6G Networks: System Design, Architectures, and Paradigms," arXiv, cs.NI 2103.02823, Mar. 2021.

[3] W. Tong and G. Y. Li, "Nine Challenges in Artificial Intelligence and Wireless Communications for 6G," *IEEE Wireless Communications*, vol. 29, no. 4, pp. 140–145, Aug. 2022.

[4] X. Li, H. Zhang, C. Peng, Z. Liu, and F. Wang, "NET4AI: Supporting AI as a Service in 6G," Huawei Research, Communications of HUAWEI RESEARCH, Sep. 2022, pp. 26–39.

[5] F. Dressler, C. F. Chiasserini, F. H. P. Fitzek, H. Karl, R. Lo Cigno, A. Capone, C. E. Casetti, F. Malandrino, V. Mancuso, F. Klingler, and G. A. Rizzo, "V-Edge: Virtual Edge Computing as an Enabler for Novel Microservices and Cooperative Computing," *IEEE Network*, vol. 36, no. 3, pp. 24–31, May 2022.

[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*, PMLR, 2017, pp. 1273–1282.

[7] T. Nishio and R. Yonetani, "Client Selection for Federated Learning with Heterogeneous Resources in Mobile Edge," in *IEEE International Conference on Communications (ICC 2019)*, Shanghai, China: IEEE, May 2019.

[8] C. Xie, S. Koyejo, and I. Gupta, "Asynchronous Federated Optimization," arXiv, cs.DC 1903.03934, Mar. 2019.

[9] C. Ma, J. Li, M. Ding, H. H. Yang, F. Shu, T. Q. S. Quek, and H. V. Poor, "On Safeguarding Privacy and Security in the Framework of Federated Learning," *IEEE Network*, vol. 34, no. 4, pp. 242–248, Jul. 2020.

[10] L. Yang and A. Shami, "On hyperparameter optimization of machine learning algorithms: Theory and practice," *Neurocomputing*, vol. 415, pp. 295–316, Nov. 2020.

[11] M. Khodak, R. Tu, T. Li, L. Li, M.-F. F. Balcan, V. Smith, and A. Talwalkar, "Federated Hyperparameter Tuning: Challenges, Baselines, and Connections to Weight-Sharing," in *Advances in Neural Information Processing Systems (NeurIPS 2021)*, M. Ranzato, A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, Eds., vol. 34, Virtual Conference: Curran Associates Inc., Dec. 2021, pp. 19 184–19 197.

[12] Y. Zhou, P. Ram, T. Salonidis, N. Baracaldo, H. Samulowitz, and H. Ludwig, "FLoRA: Single-shot Hyper-parameter Optimization for Federated Learning," arXiv, cs.LG 2112.08524, Dec. 2021.

[13] S. Agrawal, S. Sarkar, M. Alazab, P. K. R. Maddikunta, T. R. Gadekallu, and Q.-V. Pham, "Genetic CFL: Hyperparameter Optimization in Clustered Federated Learning," *Computational Intelligence and Neuroscience*, vol. 2021, pp. 1–10, Nov. 2021.

[14] Z. Li, H. Li, and M. Zhang, "Hyper-parameter Tuning of Federated Learning Based on Particle Swarm Optimization," in *7th IEEE International Conference on Cloud Computing and Intelligent Systems (CCIS 2021)*, Xi'an, China: IEEE, Nov. 2021.

[15] H. Zhang, M. Zhang, X. Liu, P. Mohapatra, and M. DeLucia, "FedTune: Automatic Tuning of Federated Learning Hyper-Parameters from System Perspective," in *IEEE Military Communications Conference (MILCOM 2022)*, Rockville, MD: IEEE, Nov. 2022.

[16] M. B. Khan, X. An, and C. Peng, "Towards Native Support for Federated Learning in 6G," in *2023 IEEE Wireless Communications and Networking Conference (WCNC)*, 2023, pp. 1–6.

[17] 3GPP, "Architecture enhancements for 5G System (5GS) to support network data analytics services," 3rd Generation Partnership Project (3GPP), Technical Specification (TS) 23.288, version 18.2.0, Jun. 2023.