Compressing and Fine-tuning DNNs for Efficient Inference in Mobile Device-Edge Continuum

Compressing and Fine-tuning DNNs for Efficient Inference in Mobile Device-Edge Continuum

Gurtaj Singh*†, Olga Chukhno*†, Claudia Campolo*†, Antonella Molinaro*†, Carla Fabiana Chiasserini§†
*University Mediterranea of Reggio Calabria, Italy. E-mail: name.surname@unirc.it
†Consorzio Nazionale Interuniversitario per le Telecomunicazioni (CNIT), Italy
§Politecnico di Torino, Italy. E-mail: carla.chiasserini@polito.it

***Abstract*—Pruning deep neural networks (DNN) is a well-known technique that allows for a sensible reduction in inference cost. However, this may severely degrade the accuracy achieved by the model unless the latter is properly fine-tuned, which may, in turn, result in increased computational cost and latency. Thus, upon deploying a DNN in resource-constrained edge environments, it is critical to find the best trade-off between accuracy (hence, model complexity) and latency and energy consumption. In this work, we explore the different options for the deployment of a machine learning pipeline, encompassing pruning, fine-tuning, and inference, across a mobile device requesting inference tasks and an edge server, and considering privacy constraints on the data to be used for fine-tuning. Our experimental analysis provides insights for an efficient allocation of the pipeline tasks across network edge and mobile device in terms of energy and network costs, as the target inference latency and accuracy vary. In particular, our results highlight that the higher the edge server load and the number of inference requests, the more convenient it becomes to deploy the entire pipeline at the mobile device using a pruned model, with a cost reduction of up to a factor two compared to deploying the whole pipeline at the edge.**

***Index Terms*—Edge computing, Edge-mobile device continuum, Machine learning pipeline, ML model compression.**

I. Introduction

Machine Learning (ML) is becoming pervasive and crucial to the provisioning of a plethora of intelligent services and applications in several domains, ranging from smart manufacturing to autonomous driving, healthcare and smart agriculture. Many of them rely on the execution of computation-intensive and resource-hungry Deep Neural Network (DNN) models, which can achieve high inference accuracy by relying on the training of a huge number of parameters.

In parallel, Sixth Generation (6G) networks envision a paradigm shift, moving away from reliance on traditional cloud-based, resource-rich services towards distributed intelligence in resource-constrained edge environments, dominated by the presence of mobile and embedded devices [1]. This transition helps address the privacy concerns raised by several smart applications with humans in the loop, as well as the growing demand for pervasive, low-latency DNN inference execution. However, implementing DNN models at the edge requires careful consideration to balance the model size, latency, energy consumption, and accuracy.

Model pruning is a widely used technique for compressing DNNs and facilitating inference in resource-limited environments [2]. By removing, e.g., the low-magnitude model parameters, hence, reducing the model size and complexity, pruning can help match the capabilities of mobile devices with the DNNs resource demand [3]. On the other hand, pruned DNNs may achieve lower accuracy. To mitigate this issue, the pruned DNN can be fine-tuned and customized using data locally collected by a mobile device to trade off model size with accuracy [4], [5].

Importantly, fine-tuning is also known for its inherent complexity, which may require a substantial amount of data and computational resources, and it may take quite a long time to be executed, namely, from minutes to hours [6], [7]). Additionally, although offloading model fine-tuning to the edge may be an option to save the limited resources of end-devices, transferring data to the edge may *(i)* lead to privacy leakage, thus necessitating encryption, *(ii)* require a large bandwidth, and *(iii)* increase inference latency whenever the edge computational load is high.

In this work, we tackle the above issues and aim to assess *(i)* whether the ML pipeline to be deployed for executing inference tasks required by a mobile device should include DNN pruning (and, if so, by which factor) and fine-tuning, and *(ii)* how the tasks of a DNN inference pipeline should be distributed between the resource-constrained end device and the more capable edge server, looking at crucial aspects such as latency, accuracy, energy consumption, network load, and privacy preservation. Specifically, the main contributions of our work are as follows:

- We characterize the DNN inference pipeline, including pruning, fine-tuning, and inference, executed either at the edge or at the mobile device, in terms of latency and accuracy, when considering different pruning factors;
- We compare different DNN inference pipelines and workflows across the edge and the mobile device in terms of overall and per-operation latency, amount of transferred data, and energy consumption. The analysis provides guidelines for selecting the most convenient option under different conditions, such as computational load at the edge server and number of requests to be executed using a given DNN.

The rest of the paper is organized as follows. After discussing some relevant literature and the work motivation in Sec. II, we introduce the reference scenario and the considered DNN inference workflows in Sec. III. We then present our

experimental analysis and results in Sec. IV. Finally, we draw our conclusions and highlight the major lessons learned in Sec. V.

II. Related Work and Motivation

Pruning of Neural Networks (NNs) aims to reduce model complexity while maintaining the model predictive capabilities. It consists in removing non-essential neural connections and/or neurons, thereby reducing the network size and optimizing computational resources [8]. One prominent technique is magnitude-based pruning [9], which removes the neural connections associated with the least relevant weights [10]. The underlying assumption is that such NN weights have a negligible impact on the overall model performance, and eliminating them does not significantly lower the accuracy level. Magnitude-based pruning methodologies range from the simple removal of weights below a predefined threshold to complex iterative strategies that consider sensitivity variations towards specific weights to preserve critical information [11].

In our previous study [12], we characterized the cost and performance of edge inference by considering pruned versions of different ML models with varying pruning factor values. Our results show that executing a pruned model at a device can yield time and energy savings of up to 40% and 53%, respectively, w.r.t. a full-size model. Also, executing inference at the end device may speed up decision-making by 60% compared to the execution at a highly loaded edge.

On the negative side, pruning may not preserve model accuracy for high values of pruning factor. Hence, whenever lightweight model variants are needed to match the limited resource availability of the network edge and mobile devices, fine-tuning is needed [13]. This involves additional training of the pruned model, allowing the remaining weights to adapt to information previously overlooked during pruning. The critical issue in fine-tuning is to ensure that a pruned NN maintains its predictive performance, by ensuring effective specialization without overfitting. Importantly, fine-tuning may require the use of private data [4], [5], which may pose significant challenges in terms of privacy preservation. It follows that fine-tuning may imply either using the resource-limited hardware and energy reserve of mobile devices or encrypting data before transferring it to the edge server. It is worth noting, however, that dataset encryption cannot entirely prevent malicious attacks, and additional techniques for privacy preservation may be needed [14]. It follows that, despite the inherent benefits of retaining a competitive accuracy, the costs of fine-tuning have to be assessed carefully upon deploying a DNN pipeline, wisely distributing fine-tuning and inference operations between edge and end-devices, which motivates our work.

III. DNN Pipeline Options

A. Reference scenario

We consider a mobile device within the coverage area of a base station (BS) that needs to perform inference tasks over its own input data. The BS is connected to an edge server, denoted with e. The device can either request the edge server for a DNN model for local inference execution or offload a task to the edge server. As data distribution and requirements may evolve over time, ML models may yield reduced accuracy, thus necessitating periodic retraining or replacement. We denote with Q the number of inference requests generated by the mobile device before a DNN update is required [15].

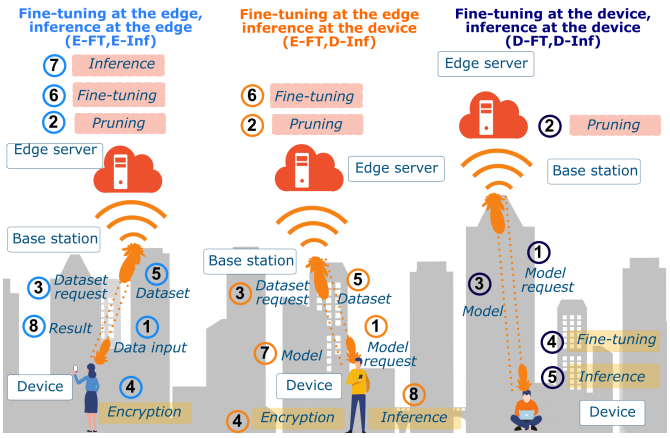


Fig. 1: DNN inference pipeline options and associated workflows.

The edge server stores multiple DNN models, denoted by $\mathcal{M}=\{1,\ldots,M\}$. Due to limited storage capabilities, we assume that only the most popular models are stored at server e. Furthermore, DNNs can be pruned on demand at the edge server to better match the resource-constrained capability of requesting mobile devices, or to ensure fast inference at the edge while requiring fewer computational resources – a crucial issue under loaded edge conditions. We refer to a DNN model pruned with a given pruning factor (pf) as a configuration option and denote the set of such options by $\mathcal{K}=\{1,\ldots,K\}$. Pruned DNN models can be fine-tuned to improve their accuracy by using a dedicated dataset already labeled to improve the performance of the DNN on more specific inputs [5], [16].

While we consider that model pruning is executed at the edge server, fine-tuning and inference can be performed either at the mobile device or at the edge server. Thus, the following DNN inference pipelines are possible: *(i)* fine-tuning and inference execution at the edge server (E-FT,E-Inf for short); *(ii)* fine-tuning execution at the edge server, inference at the device (E-FT,D-Inf for short); and *(iii)* fine-tuning and inference execution at the device (D-FT,D-Inf for short).

B. Workflows

The workflows corresponding to the considered inference pipelines are detailed below and depicted in Fig. 1.

E-FT, E-Inf. *If both fine-tuning and ML inference are performed at the edge*, the device has just to send its sample data to the edge and wait for the inference result. Thus, the steps to be performed are: (1) a sample is transmitted from the device to the BS and, consequently, to the edge server

where (2) pruning is executed; (3) the edge server requests the dataset to the device, which (4) encrypts and then (5) transmits the dataset to the edge server; the edge server executes (6) fine-tuning and (7) inference, and (8) returns the inference result to the device. If the device requests subsequent inference tasks using the same model, after the first inference request is fulfilled, the workflow includes only sample data transmission, inference execution, and inference result delivery.

E-FT, D-Inf. *Performing fine-tuning at the edge and inference at the device* needs encrypted customized[1] dataset transfer for model fine-tuning at the edge. Thus, it consists of the following steps: (1) the device requests the DNN model from the edge server; (2) the edge server prunes the DNN and (3) requests the dataset from the device; (4) the device encrypts and (5) transfers the dataset to the edge. Then, (6) the edge server fine-tunes and (7) sends the model to the device. Finally, (8) the device executes the inference task. Once the device obtains the model, it can use it locally for subsequent inference tasks until the model needs to be updated.

D-FT, D-Inf. *When fine-tuning and inference are both executed at the device*, the workflow entails: (1) the device requests the pruned model to the edge server; (2) the edge prunes and (3) transmits the model; (4) the device executes fine-tuning and (5) inference. As in the E-FT, D-Inf workflow, when the device has to perform inference for multiple subsequent sample data and the same model can be reused, only local inference (step (5)) needs to be performed.

C. Performance metrics

We now characterize the three DNN inference pipelines in terms of the following metrics:

- *ML pipeline latency*, defined as the time from when the device generates the first inference request until it receives the result of the last inference request, given the total number of inference requests Q;
- *Energy consumption* incurred by the device from the first to the last inference request.

E-FT, E-Inf. The ML pipeline latency is given by:

$$T(\text{E-FT,E-Inf}) = QT^{req,res} + T^{pr}_{m,k} + T^{req,dataset} + T^{enc} + T^{dataset} + T^{ft,e}_{m,k} + QT^{inf,e}_{m,k} + QT^{res}_{m,k}, \quad (1)$$

where the latency components related to communication include the latency contributions due to the transmission of the data sample, $l^{req,res}$, and the inference result, $l^{res}_{m,k}$, at a data rate D, i.e.,

$$T^{req,res} = \frac{l^{req,res}}{D} \text{ and } T^{res}_{m,k} = \frac{l^{res}_{m,k}}{D}. \quad (2)$$

Similarly, the contributions from the dataset request, $l^{req,dataset}$, and the dataset transfer, $l^{dataset}$, are given by:

$$T^{req,dataset} = \frac{l^{req,dataset}}{D} \text{ and } T^{dataset} = \frac{l^{dataset}}{D}. \quad (3)$$

[1]More sophisticated methods can be applied to better preserve privacy [14], which is beyond the scope of this study. Moreover, they are expected to add negligible overhead compared to the encryption operation we consider.

$T^{pr}_{m,k}$ accounts for the time needed to perform pruning of the ML model m with configuration option k at the edge, whereas T^{enc} is the time required to encrypt the dataset at the device. $T^{ft,e}_{m,k}$ and $T^{inf,e}_{m,k}$ represent the time required to perform, respectively, fine-tuning and inference at the edge. In our computation, the values for executing pruning, encryption, fine-tuning, and inference are obtained through experimental measurements under the settings presented in Sec. IV.

The energy consumption at the mobile device is given by:

$$E^d(\text{E-FT,E-Inf}) = QP^{tx}T^{req,res} + (P^{rx}+P^E)T^{req,dataset} + P^{enc}T^{enc} + P^{tx}T^{dataset} + Q(P^{rx}+P^E)T^{res}_{m,k}, \quad (4)$$

where P^{rx} (P^{tx}) represents the power spent by the device in receiving (transmitting), P^E denotes the power consumed by the baseband electric circuit of the device, and P^{enc} is the power consumed for encrypting the dataset (all these values are expressed in Watts).

E-FT, D-Inf. The ML pipeline latency is as follows:

$$T(\text{E-FT,D-Inf}) = T^{req,model} + T^{pr}_{m,k} + T^{req,dataset} + T^{enc} + T^{dataset} + T^{ft,e}_{m,k} + T^{model}_{m,k} + QT^{inf,d}_{m,k}, \quad (5)$$

which accounts for the communication latency for requesting and receiving the ML model of size $l^{model}_{m,k}$ given, respectively, by:

$$T^{req,model} = \frac{l^{req,model}}{D} \text{ and } T^{model}_{m,k} = \frac{l^{model}_{m,k}}{D}. \quad (6)$$

$T^{ft,e}_{m,k}$ and $T^{inf,d}_{m,k}$ refer to the time required to perform fine-tuning at the edge and inference at the device, respectively.

The energy consumption at the device can be written as:

$$E^d(\text{E-FT,D-Inf}) = T^{req,model}P^{tx} + (P^{rx}+P^E)T^{req,dataset} + P^{enc} \cdot T^{enc} + P^{tx}T^{dataset} + (P^{rx}+P^E)T^{model}_{m,k} + QP^{inf}T^{inf,d}_{m,k}, \quad (7)$$

where P^{inf} is the power consumption to perform the inference at the device (in Watts).

D-FT, D-Inf. In this case, we have:

$$T(\text{D-FT,D-Inf}) = T^{req,model} + T^{pr}_{m,k} + T^{model}_{m,k} + T^{ft,d}_{m,k} + QT^{inf,d}_{m,k}, \quad (8)$$

where $T^{ft,d}_{m,k}$ denotes the time to perform fine-tuning at the device. The energy consumption at the device is then given by:

$$E^d(\text{D-FT,D-Inf}) = T^{req,model}P^{tx} + (P^{rx}+P^E)T^{model}_{m,k} + P^{inf}T^{ft,d}_{m,k} + QP^{ft}T^{inf,d}_{m,k}, \quad (9)$$

where P^{ft} is the device power consumption (in Watts) for fine-tuning the pruned model locally.

IV. Performance Evaluation

A. Experimental settings

We consider a 5G New Radio BS operating at a frequency of 3.5 GHz and evaluate the communication latency contributions as per (2), (3), and (6). The transmission power of the BS

and the device are, respectively, 23 dBm and 10 dBm. The available bandwidth is 100 MHz, and both the transmitter and receiver use an antenna array with 4×4 antenna elements. The sub-6 GHz channel is modeled using the Urban Micro (UMi) street canyon model [17], while the data rate is estimated using Shannon's theorem.

An Oracle VirtualBox virtual machine (VM) with 4 virtual CPUs acts as edge server, using an Intel(R) Xeon(R) Platinum 8370 C physical processor with 16 GB of RAM. The performance metrics are reported for the tasks running as Docker containers, with the hosting VM operating at maximum CPU power, either without additional load (case denoted with "edge, no load" in the following) or with 50 active Docker containers for inference tasks (case denoted with "edge, loaded"); all containers in the background also run inference tasks. The mobile device has 1 physical core (Intel Core i7-9750H), 4 GB of RAM, and 30 GB of disk space.

We use MobileNet [18] as a DNN model and a 412-MB subset of ImageNet-1000[2] dataset, including samples for 100 classes, as private dataset for fine-tuning. This mimics the case in which the device operates in a specific environment, and the number of classes it can detect is limited [5]. The size of an image sample is 300 kB, and the DNN output size is 2 kB. We use Tensorflow Model Garden, Tensorflow Model Optimization, and Keras Surgeon for pruning. Other settings are as follows: P^{E}=5.34 W, P^{tx}=0.01 W, P^{rx}=0.1 W, and $P^{inf/ft}$=2.8 W [19].

We use magnitude-weight pruning, which removes the least relevant neural connections [9], [10]. For fine-tuning, the loss function is extended to include validation data. The process uses incremental backpropagation, updating weights incrementally and applying regularization techniques such as *dropout* to prevent overfitting. The learning rate is set to 0.001, and detailed evaluation of the performance on a test dataset followed. For best fine-tuning, the process is iterated with parameter tuning, balancing computational efficiency, and prediction accuracy. Regarding dataset encryption, we use the public and private key Rivest–Shamir–Adleman (RSA) scheme.

B. Results

The first set of results evaluates the impact of the pruning factor (pf) on the target performance metrics. Fig. 2 highlights the trade-off between model accuracy (Top-1) and model size as the pruning factor varies. Notice that, according to the Top-1 metric, a prediction is considered as correct only if the top prediction corresponds to the correct class. Beyond a pruning factor of 40%, the accuracy level degrades significantly, indicating the need for fine-tuning.

When fine-tuning and inference get faster at the mobile device. Table I shows that fine-tuning and inference times generally decrease with higher pruning factors (i.e., smaller models). Also, while they get larger at the edge with an increased traffic load, their increase with the pruning factor becomes more evident at the edge in the case of high load. As expected, the execution of fine-tuning and inference gets faster when moving from a loaded edge to the device.

Achieving high accuracy by fine-tuning a pruned model. The model can be significantly pruned while still achieving high accuracy if properly fine-tuned. Specifically, at a pruning factor of 70%, fine-tuning enhances the accuracy, increasing it from 70% to 85% (with the latter value corresponding to the accuracy of the full model). Henceforth, we will focus on this pruning factor and consider fine-tuning after pruning.

Impact of the number of inference instances (Q) on ML pipeline efficiency. 1) Latency: Figures 3(a)–(b) illustrate the ML pipeline latency when the number of inference requests for the three DNN pipeline options varies. In addition, the plots show the curves denoted with "E-Inf, full model" and "D-Inf, full model", which represent the baseline approaches for the execution of the full model at the edge and device, respectively, with no pruning and fine-tuning procedures.

In the case of no loaded edge (see Fig. 3(a)), the faster option is to execute the full DNN model at the edge. This is

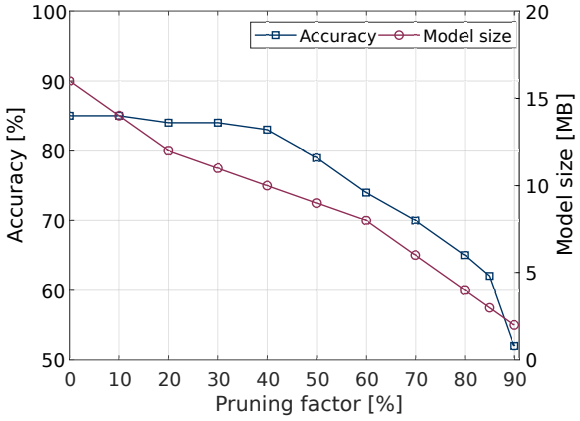
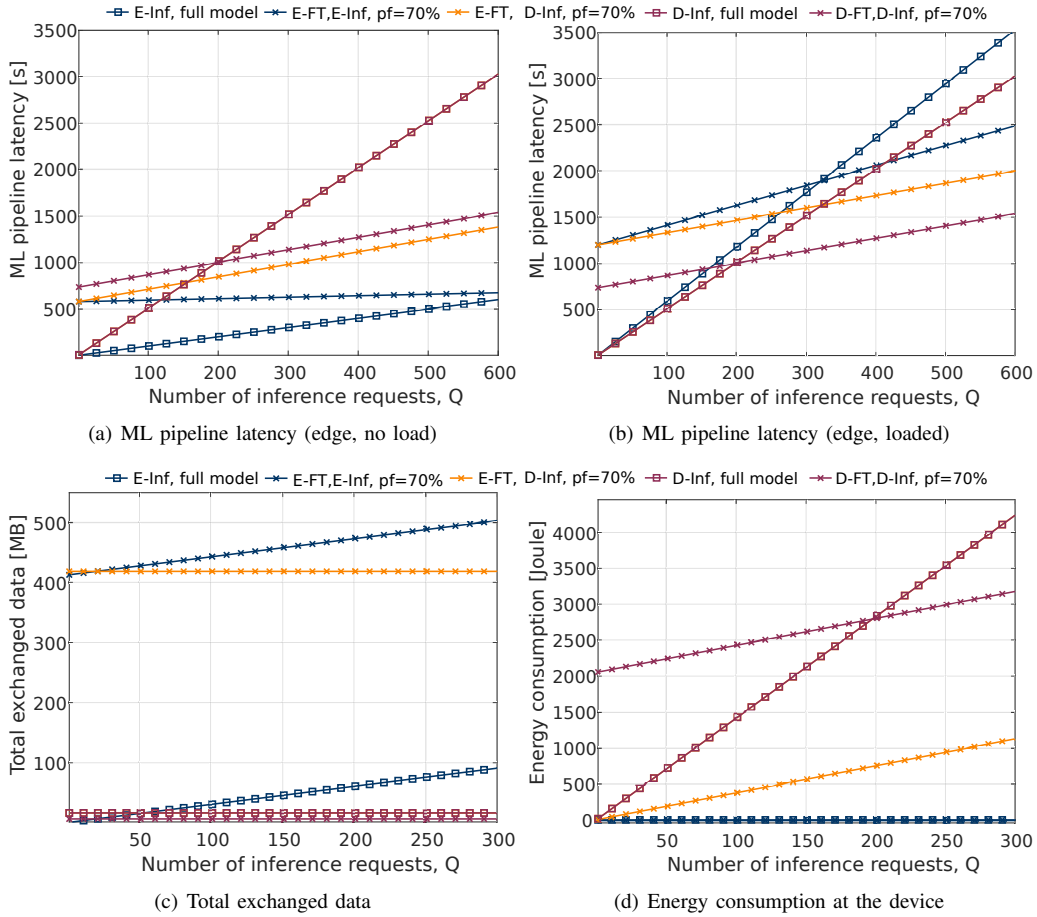


Fig. 2: Accuracy and model size vs. pruning factor.

[2]https://www.image-net.org/

TABLE I: Performance metrics vs. pruning factor

Pruning factor [%]	Accuracy [%]		Fine-tuning time [min]			Inference time [s]		
	w/o fine-tuning	w fine-tuning	edge, no load	edge, loaded	device	edge, no load	edge, loaded	device
50	79	87	11.09	21.53	15.35	0.3	3.156	2.121
60	74	86	10.10	21.13	14.52	0.187	2.932	1.954
70	70	**85**	09.22	19.54	12.22	0.143	2.135	1.34
80	65	82	08.54	16.14	10.15	0.11	1.982	0.5
85	62	79	08.32	15.01	10.09	0.091	1.75	0.32



(a) ML pipeline latency (edge, no load)

(b) ML pipeline latency (edge, loaded)

(c) Total exchanged data

(d) Energy consumption at the device

Fig. 3: Metrics vs. number of inferences executed through the same ML model (Q).

TABLE II: Contributions to the ML pipeline latency

	Result exchange		**Model exchange**		**Pruning**		**Encryption**		**Dataset exchange**		**Inference**		**Fine-tuning**		**ML pipeline latency [s]**	
	[s]	**[%]**	**[s]**	**[%]**	**[s]**	**[%]**	**[s]**	**[%]**	**[s]**	**[%]**	**[s]**	**[%]**	**[s]**	**[%]**	**Q=1**	**Q>1**
Full model, no load at the edge																
E-Inf	**0.017**	1.7	-	-	-	-	-	-	-	-	0.98	98.3	-	-	**1**	1
D-Inf	-	-	0.5203	9.3	-	-	-	-	-	-	5.04	90.7	-	-	5.56	5.04
Full model, edge loaded																
E-Inf	**0.017**	0.3	-	-	-	-	-	-	-	-	5.86	99.7	-	-	5.88	5.88
D-Inf	-	-	0.5203	9.3	-	-	-	-	-	-	5.04	90.7	-	-	**5.56**	5.04
Pruned model, pf=70%, no load the the edge																
E-FT,E-Inf	**0.017**	0.003	-	-	**0.15**	0.026	**0.12**	0.02	**23.75**	4.113	**0.14**	0.025	**553**	95.81	577	**0.16**
E-FT,D-Inf	-	-	**0.195**	0.034	**0.15**	0.026	**0.12**	0.02	**23.75**	4.103	1.34	0.232	**553**	95.58	579	1.34
D-FT,D-Inf	-	-	**0.195**	0.027	**0.15**	0.020	-	-	-	-	1.34	0.182	733	99.77	735	1.34
Pruned model, pf=70%, edge loaded																
E-FT,E-Inf	**0.017**	0.001	-	-	0.75	0.063	**0.12**	0.01	**23.75**	1.981	2.14	0.178	1172	97.77	1201	2.16
E-FT,D-Inf	-	-	**0.195**	0.016	0.75	0.063	**0.12**	0.01	**23.75**	1.981	1.34	0.112	1172	87.82	1198	**1.34**
D-FT,D-Inf	-	-	**0.195**	0.027	0.75	0.102	-	-	-	-	1.34	0.182	733	99.69	735	**1.34**

no longer valid for inference requests higher than 600. After that, a lighter version of the DNN model with fine-tuning and inference at the edge becomes a more convenient option. Executing the full model at the device is also more convenient than executing the pruned version either at the edge or device, but just until Q=120 and Q=200, respectively. This is because fine-tuning takes a considerable time, namely, in the order of 10 minutes. As shown in Fig. 3(b), the load at the edge server also affects the decision on where to perform fine-tuning and inference, making it preferable to execute both at the device in the case of a loaded edge. As expected, the latency for the case of fine-tuning at the edge and inference at the device lies between the extreme E-FT,E-Inf and D-FT,D-Inf cases.

The individual latency contributions and their corresponding percentages are detailed in Table II, which also reports in the last column: *(i)* the ML pipeline latency for the first

inference request (Q=1), which accounts for pruning and fine-tuning whenever pruning is part of the ML pipeline, and *(ii)* the latency for subsequent inference requests (Q>1), which consists of an inference task execution using the already pruned and fine-tuned model.

2) Bandwidth consumption: Fig. 3(c) depicts the amount of data exchanged in all considered DNN inference pipeline options. We observe that for Q values lower than 50, performing inference at the edge using the full model is always advantageous, as in this case, the device only needs to transmit the data samples, and the edge returns the inference result (less than 1 MB exchanged in total). This no longer holds for higher values of Q. In this case, the network burden is reduced if either the full model or a pruned version thereof is retrieved (only once) and executed at the device. The execution of the pruned model at the edge is the most expensive option in terms of exchanged data since it implies transferring the encrypted dataset (encryption accounts for only an additional 0.193 MB). The same holds for fine-tuning at the edge.

3) Energy consumption: Fig. 3(d) presents the energy consumption of the device as the number of inference requests varies. As expected, performing inference at the edge is consistently preferable because all computational load is offloaded, and the device incurs the lowest energy cost of transmitting and receiving data. However, this option may not be convenient in terms of latency when the edge is loaded, and offloading the execution of model fine-tuning proves to be the most energy-efficient choice. Furthermore, it is interesting to note that when more than Q=200 inference tasks can be executed with the same model, the energy spent in fine-tuning at the device becomes lower than that experienced if the inference tasks are executed locally using the full DNN. The energy consumption due to the execution of the ML pipeline at the device remains within the device's energy budget (i.e., 3.7 Wh [12]) for up to $3,000$ inference tasks executed with the pruned model.

V. Conclusions and Major Lessons Learned

We addressed the efficient deployment of DNN inference models across the mobile device-edge continuum. First, we explored the extent to which fine-tuning can help increase the accuracy of pruned DNN models when inference tasks are requested by applications running at a mobile device. We then experimentally investigated the performance of different DNN inference pipeline configurations, distributing the required operations across the edge and mobile device, in terms of latency, amount of transferred data, and energy consumption. Our results show that using a full-size DNN is the best option in terms of latency and energy consumption in the case of a limited number of requested inference tasks. Conversely, on-device execution is to be preferred in the case of a loaded edge server.

As for the cost of fine-tuning a pruned model, this is offset when the DNN model can be reused for multiple inference tasks, and swift inference results at the device are required. In this case, the decision about where to perform fine-tuning and inference should also account for the level of computational load at the edge and bandwidth availability. The following findings hold: *(i)* If the edge is loaded, inference at the device reduces latency by nearly a factor 4 w.r.t. the local execution of the full-size model; *(ii)* Fine-tuning at the edge saves precious energy resources at the end device at the cost of transferring a very large amount of data over the radio link; *(iii)* Although being the fastest option with a latency reduction of up to a factor 6 compared to the full-size model execution on the device, inference at the edge is convenient only when the edge is lighted loaded.

Future work will focus on the optimization of the DNN pipeline and of its deployment across edge servers and end-devices, as well as on extending the experimental analysis to a larger set of use cases.

References

[1] C. Campolo *et al.*, "Network for Distributed Intelligence: a Survey and Future Perspectives," *IEEE Access*, vol. 11, pp. 52840 – 52861, 2023.

[2] Y. Jiang *et al.*, "Model Pruning Enables Efficient Federated Learning on Edge Devices," *IEEE Trans. on Neural Networks and Learning Systems*, vol. 94, no. 12, pp. 10374 – 10386, 2022.

[3] M. Zhu *et al.*, "To Prune, or Not to Prune: Exploring the Efficacy of Pruning for Model Compression," *arXiv preprint arXiv:1710.01878*, 2017.

[4] P. Guo *et al.*, "Mistify: Automating DNN Model Porting for On-Device Inference at the Edge," in *18th USENIX Symposium on Networked Systems Design and Implementation*, pp. 705–719, 2021.

[5] Y.-D. Ma *et al.*, "OCAP: On-device class-aware pruning for personalized edge DNN models," *Journal of Systems Architecture*, vol. 142, p. 102956, 2023.

[6] H. Kim *et al.*, "A Framework for Fast and Efficient Neural Network Compression," *CoRR*, vol. abs/1811.12781, 2018.

[7] B. J. Eccles *et al.*, "DNNShifter: An efficient DNN pruning system for edge computing," *Future Generation Computer Systems*, vol. 152, pp. 43–54, 2024.

[8] T. Liang *et al.*, "Pruning and Quantization for Deep Neural Network Acceleration: A Survey," *Neurocomputing*, vol. 461, pp. 370–403, 2021.

[9] T. Gale *et al.*, "The State of Sparsity in Deep Neural Networks," *arXiv preprint arXiv:1902.09574*, 2019.

[10] J. Frankle *et al.*, "Pruning Neural Networks at Initialization: Why Are We Missing the Mark?," *arXiv preprint arXiv:2009.08576*, 2020.

[11] M. Zullich *et al.*, "Speeding-up Pruning for Artificial Neural Networks: Introducing Accelerated Iterative Magnitude Pruning," in *2020 ICPR*.

[12] O. Chukhno *et al.*, "Machine Learning Performance at the Edge: When to Offload an Inference Task," in *Workshop on Networked Sensing Systems for a Sustainable Society*, pp. 180–186, 2023.

[13] P. Molchanov *et al.*, "Importance Estimation for Neural Network Pruning," in *Proc. of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 11264–11272, 2019.

[14] B. Liu *et al.*, "Pmc: A privacy-preserving deep learning model customization framework for edge computing," *ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, vol. 4, no. 4, pp. 1–25, 2020.

[15] K. Rahmani *et al.*, "Assessing the Effects of Data Drift on the Performance of Machine Learning Models Used in Clinical Sepsis Prediction," *International Journal of Medical Informatics*, vol. 173, p. 104930, 2023.

[16] Y. Mao *et al.*, "A privacy-preserving deep learning approach for face recognition with edge computing," in *HotEdge Workshop*, pp. 1–6, 2018.

[17] 3GPP, "Technical Specification Group Radio Access Network; Study on channel model for frequencies from 0.5 to 100 GHz (Release 17)," tech. rep., 3GPP TR 38.901 V17.0.0, March 2022.

[18] A. Howard *et al.*, "Mobilenets: Efficient Convolutional Neural Networks for Mobile Vision Applications," *arXiv preprint arXiv:1704.04861*, 2017.

[19] Q. Liang *et al.*, "AI on the Edge: Rethinking AI-based IoT Applications Using Specialized Edge Architectures," *arXiv preprint arXiv:2003.12488*, 2020.