Machine Learning Performance at the Edge: When to Offload an Inference Task

Machine Learning Performance at the Edge: When to Offload an Inference Task

Olga Chukhno, Gurtaj Singh,
Claudia Campolo, Antonella Molinaro
University Mediterranea of Reggio Calabria and CNIT
Reggio Calabria, Italy
name.surname@unirc.it

Carla Fabiana Chiasserini
Politecnico di Torino and CNIT
Turin, Italy
name.surname@polito.it

ABSTRACT

Machine Learning (ML) techniques play a crucial role in extracting valuable insights from the large amounts of data massively collected through networked sensing systems. Given the increased capabilities of user devices and the growing demand for inference in mobile sensing applications, we are witnessing a paradigm shift where inference is executed at the end devices instead of burdening the network and cloud infrastructures. This paper investigates the performance of inference execution at the network edge and at end-devices, when using both a full and a pruned model. While pruning reduces model size, thus making the model amenable for execution at an end-device and decreasing communication footprint, trade-offs in time complexity, potential accuracy loss, and energy consumption must be accounted for. We tackle such trade-offs through extensive experiments under various ML models, edge load conditions, and pruning factors. Our results show that executing a pruned model provides time and energy (on the device side) savings up to 40% and 53%, respectively, w.r.t. the full model. Also, executing inference at the end-device may lead to 60% faster decision-making compared to inference execution at a highly loaded edge.

This work was supported by the SNS-JU-2022 project ADROIT6G under the European Union's Horizon Europe research and innovation programme under Grand Agreement No. 101095363 and by the European Union under the Italian National Recovery and Resilience Plan (NRRP) of NextGenerationEU (PE00000001 - program "RESTART").

Conference'28, October 2023, Madrid, Spain

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/XXXXXXX.XXXXXXX

KEYWORDS

Machine learning, inference, edge computing, pruning

ACM Reference Format:
Olga Chukhno, Gurtaj Singh,, Claudia Campolo, Antonella Molinaro and Carla Fabiana Chiasserini. 2023. Machine Learning Performance at the Edge: When to Offload an Inference Task. In *Proceedings of ACM Conference (Conference'28).* ACM, Madrid, Spain, 8 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

By enabling physical objects equipped with sensing and processing capabilities to be connected to the Internet, Internet of Things (IoT) paves the way to a plethora of innovative applications also contributing to the sustainability of our society and ranging from smart agriculture and smart cities to industry 4.0 and e-health [1]. Machine Learning (ML) allows gaining deep insights from the data collected by IoT devices. ML relies on two main phases: *training* and *inference*. The former involves training a model using an input dataset, while the latter uses the trained model to make decisions and predictions or provide knowledge.

Although inference tasks are instead less demanding than training procedures, in the case of a high number of inference requests, the total load over time on computational and networking resources can be substantial. Moreover, time-critical applications require fast and reliable decisions or predictions, which imply obtaining inference results in near real-time. Such considerations, along with the growing demand for swift inference tasks in pervasive environments and the privacy requirements of a large set of ML-based applications, call for moving inference execution to the network edge and far-edge (i.e., on the end-devices).

Another critical aspect of pervasive ML is the large number of model parameters, up to millions for Deep Neural Networks (DNNs), and operations, which require significant memory space and frequent memory access, resulting in high energy consumption. To face resource constraints of end-devices that may hamper inference in the far-edge, and to optimize energy consumption, ML models can be compressed. Techniques, such as model pruning, can be applied to reduce the size of an ML model for deployment at an

end-device [2]. Pruning aims to keep the most influential network's weights and remove the less influential ones (simply set to zero) while preserving the quality level of the model output. It is worth noting that, although pruning reduces the size of the full model with inherent benefits in terms of time execution and communication footprint, two major drawbacks may arise: *(i)* a degraded quality of the model output and *(ii)* the time complexity of the pruning operation itself, which may indeed be computationally intensive. Hence, it cannot be stated that pruning a model for a resource-limited end-device is always beneficial in terms of inference time and sustainability.

The objective of our study is thus multifold. We investigate the performance of inference execution at an end-device in the far-edge versus the case where such a task is offloaded to an edge node by exploring the option of deploying a pruned model, instead of its full version, on the resource-limited end-device. Moreover, we consider that the edge server may adapt a pruned model *on demand* upon a request from an end-device, e.g., because of the evolving dynamics of a specific domain [3]. In such a situation, an edge server will have either to update or further prune a model, which is then sent over the wireless link to the end-device. The latter will use the new version of the model for inference execution until the model gets outdated. Our main contributions can be summarized as follows:

- We characterize the performance of pruning and inference executed at edge servers and end-devices through extensive experiments in terms of execution time and accuracy for different ML models and pruning factors;
- We determine under which conditions, such as load at the edge and different ML models and pruning factors, executing inference on an end-device through a pruned ML model is more convenient than a conventional inference offloading approach. While doing so, we assess the inference delay, the quantity of exchanged data, and the energy consumption of the overall process.

The remainder of this work is organized as follows. Section 2 discusses related work and highlights the novelty of our study. Section 3 describes the reference scenario and our system model. Section 4 presents our experimental analysis, and Section 5 draws some conclusions and sketches directions for future work.

2 BACKGROUND AND MOTIVATION

Different techniques can be applied to adapt an ML model to the constrained resource capabilities of IoT devices [2]. One such technique is model pruning, which involves removing a percentage of edges between neurons, or the neurons themselves, that are associated with small magnitude weights, thus leading to a version of the model of reduced size and complexity [4]. Pruning offers advantages such as reducing model memory footprint and speeding up inference time. However, a higher pruning factor may cause a decreased level of output quality and reduced ability to generalize to slightly different input data distributions. Therefore, it is crucial to determine the pruning factor that optimally trades off model size with accuracy [5].

Magnitude pruning involves sorting weights by their absolute value and eliminating those with the lowest value, assuming that less important weights are closer to zero [6]. Sensitivity-based pruning calculates the sensitivity of each weight to the Neural Network (NN) output instead and removes the weights with the lowest sensitivity value [7]. As both techniques reduce the number of model parameters to be transmitted when the model is transferred across the network, they also have a lower communication footprint compared to the full model.

In [8], the effectiveness of a pruning algorithm for NN has been evaluated through a series of experiments with the aim of identifying the appropriate pruning factor, considering computational complexity, accuracy, and size requirements. The experiments cover various NN, including ResNet, and prove that the time required for pruning operations may range from seconds to hours. This is because model fine-tuning after pruning, essential for maintaining model generalization, may take hours [9]. The study in [10] addresses ML model compression for various devices and optimal training location decisions based on resource availability. The proposal considers model compression, node and data selection, and training duration and quality. The work also discusses the challenges of formulating and optimizing a cooperative training process that integrates model and node switching.

Novelty: Unlike previous studies, we investigate both the benefits and the cost in terms of energy, time, and accuracy of executing ML inference tasks at the end-devices, when either a full or a pruned model is used. To do so, we envision different ways the network edge and the end-devices can interact and create synergies, and we evaluate the efficiency and the effectiveness of such interaction by accounting for *(i)* the impact of model pruning on inference accuracy, latency, and computational cost, *(ii)* the computational footprint of inference, and *(iii)* the energy and latency cost due to data transfer between the edge and an end-device. Through such comprehensive evaluation, we offer novel insights that can facilitate the optimization of ML model deployment in resource-constrained IoT environments.

3 SYSTEM MODEL

This section first introduces the network and application scenario we consider in our study (Sec. 3.1); then, it details the main elements of our system and how they interact for

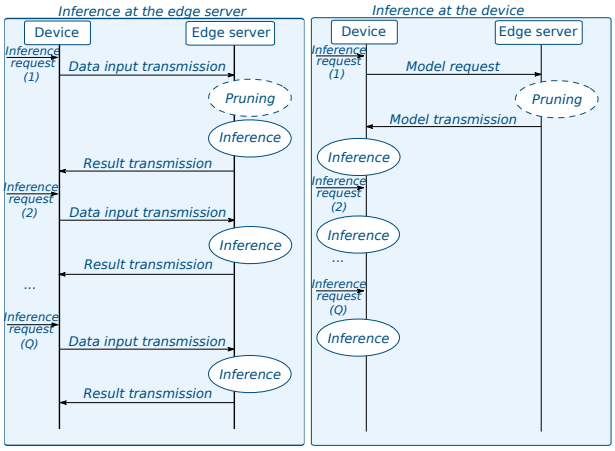


Figure 1: Workflow required for inference execution at an edge server (left) and at a mobile device (right).

inference execution at the network edge (Sec. 3.2) and at an end device (Sec. 3.3).

3.1 Reference scenario

Our study aims to investigate a scenario where mobile devices within the coverage area of a Base Station (BS) of a cellular network request inference services (e.g., objection detection). For the sake of simplicity and without loss of generality, an edge server is co-located with the BS and stores in a local repository a set of pre-trained ML models, $\mathcal{M} = \{m_1, ..., m_M\}$, either full or pruned models.

Inference tasks can be executed either at the edge server or the end-device, provided that the pre-trained ML model is downloaded from the edge server. In the former case, the mobile device sends the input data (e.g., an image) to the edge server, which performs the inference, through the full ML model or through a pruned version thereof, and sends back the result to device d (e.g., the object detected in the image). In the latter case, the mobile device sends a model request to the edge server, also specifying its capabilities, and waits for the (full or pruned) model to perform inference on board. It follows that each model has a set of possible configuration options, $\mathcal{K} = \{k_1, ..., k_K\}$, that depend on the model compression parameters (i.e., pruning factors).

The pruned version of the model may already be available in the repository, e.g., because it is a popular requested model, or the edge server can prune the model on demand. After a certain time lapse, the model may become outdated due to advancements in knowledge or drift of the input data distribution, thus requiring retraining the model with up-to-date data to maintain the desired level of output quality. Therefore, the same model can serve multiple inference requests, say Q requests, from the same mobile device before the model needs to be retrained.

In the case of inference at the edge, the entire workflow in Fig. 1(left) (i.e., data input transmission, inference execution, and result transmission) needs to be repeated Q times. On the contrary, if the inference is performed on the mobile device, once the model is retrieved and locally stored, there is no need for the mobile device to undergo the entire workflow again and again: as shown in Fig. 1(right), only the last step, i.e., the inference execution, must be repeated.

3.2 Inference at the edge

Data input/result transmission. Let D_{ul} and D_{dl} be the uplink and downlink data rates (in bps), estimated using the Shannon's theorem and the path loss model in [11]. Then the latency experienced by device d, respectively, to transmit the input data of l^{input} bits to the edge server, and to receive the inference result of model m with configuration k from the BS, can be written as:

$$T^{input} = \frac{l^{input}}{D_{ul}} \text{ and } T^{res}_{m,k} = \frac{l^{res}_{m,k}}{D_{dl}}, \forall m \in \mathcal{M}, k \in \mathcal{K}. \quad (1)$$

Pruning execution. Upon receiving the input data, the edge server may need to prune the model to be used for inference. The pruning time of model m with configuration option k, denoted by $T^{pr}_{m,k}$, is experimentally measured (Section 4).

Inference execution. The edge server executes the inference using model m with configuration k in a time that is denoted by $T^{inf}_{m,k,e}$ and is experimentally measured (see Section 4). The total cycle delay to undergo for an inference task in the case of inference execution at the edge is given by:

$$T(edge) = T^{input} + T^{pr}_{m,k} + T^{inf}_{m,k,e} + T^{res}_{m,k}. \quad (2)$$

Correspondingly, the energy spent at the device can be computed as:

$$E^d(edge) = P^{tx}_d T^{input} + (P^{rx}_d + P^E_d) T^{res}_{m,k}, \quad (3)$$

where P^{rx}_d / P^{tx}_d is the received/transmitted power (in Watts) spent at the device d, P^E is the baseband electric circuit power unit consumed by the BS or the device (in Watts).

3.3 Inference at an end device

Model request/model transmission. Indicating with l^{req} the inference request size and with $l^{size}_{m,k}$ the size of model m with configuration k, the respective transmission delays on uplink and downlink are given by:

$$T^{req} = \frac{l^{req}}{D_{ul}} \text{ and } T^{model}_{m,k} = \frac{l^{size}_{m,k}}{D_{dl}}, \forall m \in \mathcal{M}, k \in \mathcal{K}. \quad (4)$$

Pruning execution. Experimental measurements for the pruning time of model m with configuration option k, $T^{pr}_{m,k}$, are the same as in Section 3.2 and are reported in Section 4.

Inference execution. We experimentally characterize the inference time of model m with configuration k at device d, denoted with $T^{inf}_{m,k,d}$, as detailed in Section 4. Thus, the total cycle delay in the case of a single inference execution at an end device can be written as:

$$T(device) = T^{req} + T^{pr}_{m,k} + T^{model}_{m,k} + T^{inf}_{m,k,d}. \quad (5)$$

Being the device typically battery-powered and more concerned with the energy consumption, we focus only on the corresponding energy spent at its side, which is computed as follows:

$$E^d(device) = P^{tx}_d T^{req} + (P^{rx}_d + P^E_d) T^{model}_{m,k} + P_d T^{inf}_{m,k,d}, \quad (6)$$

where P_d is the power consumption of the device (in Watts).

4 PERFORMANCE EVALUATION

This section presents the system performance when inference is executed at the edge and at the device. After detailing the system settings under which we derived our results (Sec. 4.1), we show the impact of the pruning factor on the ML model and system performance as well as the latency and energy consumption for the two considered inference execution options (Sec. 4.2).

4.1 Evaluation settings

For the estimation of the communication latency experienced in transferring the input data, pruned model, and inference results, we consider a 5G New Radio (NR) BS operating at a frequency of 3.5 GHz. The transmit power is set to 23 dBm at the BS and 10 dBm at the device. We assume an available bandwidth of 100 MHz and that transmitter and receiver use an antenna array consisting of 4×4 antenna elements. We model the sub-6 GHz channel using the Third Generation Partnership Project (3GPP) Urban Micro (UMi) street canyon [11].

As for inference and pruning times, we measure them through experiments. To mimic an edge server, we set up a virtual machine (VM), Azure Standard D4ds v5, equipped with 4 virtual CPUs. The underlying physical CPU is an Intel(R) Xeon(R) Platinum 8370 C processor with 16 GB of RAM. The VM works at full CPU power. We utilize up to 100 Docker containers active in the VM for the execution of inference tasks to resemble different load conditions on the edge server. The mobile device has 1 physical core (Intel Core i7-9750H), 4 GB of RAM, and 30 GB of disk space. We focus on image recognition inference tasks, employing a combination of deep learning models, datasets, and pruning techniques. To assess the impact of model size on inference performance, we consider highly heterogeneous models: ResNet-152 [8], DenseNet201 [12], and MobileNet [13], with ResNet-152 being the most complex model and MobileNet the simplest one. We then consider the ImageNet-1000 dataset[1] – a widely used dataset for image recognition tasks – as it provides a robust evaluation environment with a diverse set of categorized images. For the inference task, a 300 kB-large image is given as input data, while the inference output size is equal to 2 kB. With regard to model compression, we considered magnitude weight pruning and applied different tools to perform it, namely, Tensorflow Model Garden, Tensorflow Model Optimization, and Keras Surgeon.

We assess energy consumption at the device, by referring to the following settings: P^E_d=5.34 W, P^{tx}_d=0.01 W, P^{rx}_d=0.1 W, P_d=2.8 W [14].

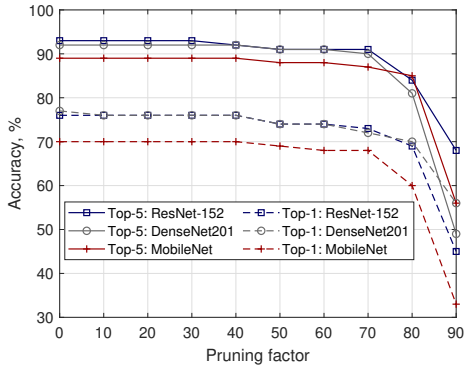
4.2 Numerical Results

Impact of pruning factor. The first set of results focuses on evaluating the impact of the pruning factor on the system metrics. Fig. 2(a) shows the performance of the considered ML models in terms of two popular accuracy metrics, Top-1 and Top-5, for different pruning factors (pf). The former (dashed line curves) measures the percentage of instances in which the model predicts the correct class as its highest-rank prediction. The latter (solid line curves), instead, measures the percentage of instances where the correct class is included within the top five predictions of the model. Both values of accuracy remain stable for a pruning factor less than or equal to 70%; beyond this value, instead, the accuracy degrades visibly, indicating the need for fine-tuning.

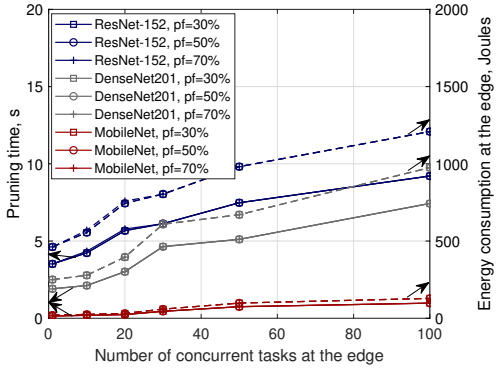
Fig. 2(b) depicts the time, $T^{pr}_{m,k}$, required for the edge server to perform pruning (solid line curves), as the edge load varies. Interestingly, such time remains almost constant for varying values of the pruning factor, since the operation only implies the removal of a portion of the weight vector. Conversely, the experienced delay varies depending on the considered ML model, due to the different number of model parameters, which affects the ordering times of the weight vector before the actual pruning is performed. Furthermore, as expected, latency increases as the load of the edge server grows. Consistently with the behavior noted for the delay, the energy consumed by the edge server for pruning a full model (as the product of pruning time and power, with the latter one set to 131.26 W [14]), increases with the edge load (dashed line curves in Fig. 2(b)).

Based on the above observations, in the following we fix the pruning factor to 70% whenever the pruned version of a model is considered. Indeed, such a value provides an accuracy level almost as good as that of the full model while

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



(a) Top-1 and Top-5 accuracy metrics vs. pruning factor.



(b) Latency and energy consumption at the edge due to ML model pruning vs. number of concurrent tasks at the edge.

Figure 2: Impact of pruning for different ML models.

exhibiting lower complexity w.r.t. it and no additional latency at the edge compared to other pruned models.

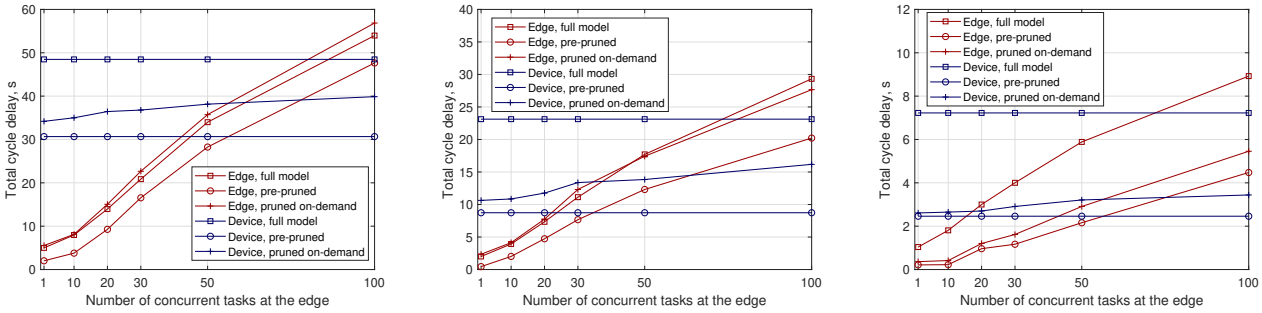
Inference at the edge vs. inference at the device. Fig. 3 presents the average time for obtaining an inference result at the end device once the request has been issued, for inference executed at the edge (as per (2), curves labeled as *Edge*), and at the device (as per (5), curves labeled as *Device*). In the plot, the performance is shown when inference is executed through *(i)* the full model, *(ii)* a pre-pruned model stored at the edge server, or *(iii)* a model pruned at edge on-demand, i.e., upon the edge receives a request from a device.

As expected, for all the considered models, in the case of inference at the edge, the greater the model complexity and the edge load, the larger the inference time. The difference in performance between using the pruned model and the full model increases as the model complexity decreases. As expected, running the full model requires the longest time, with the exception of the case when ResNet-152 is executed at the edge. Indeed, the pruning operation for ResNet-152 is so time consuming that using the full model for inference may be more convenient than pruning the model and using this one for inference. Moreover, it can be observed that executing the pre-pruned model, either at the device or at the edge, leads to the lowest latency. Interestingly, notice that, for all the three models, beyond a given threshold on the level of load at the edge server, it is faster to perform inference at the device (blue curves) than at the edge (red curves), with the threshold value getting smaller as a lighter model is considered.

Fig. 4 depicts the total amount of data exchanged over the radio interface, in both uplink and downlink directions, between the device requesting the inference and the edge server, for several inference requests issued within the model lifetime, Q. In the case of inference at the edge, the communication footprint is independent of the model, because it accounts for the input data (l^{input}) and the inference result ($l^{res}_{m,k}$). Furthermore, the metric increases with the number of requests, as the input data for which inference should be performed must always be sent to the edge server. On the contrary, in the case of inference executed at the device (curves with circle markers), the full/pruned model is transferred only once to the device and re-used for several inference tasks of the same type. It is clear that the more complex the model, the larger the amount of transferred data, which decreases when the pruned version of a model is used.

For a single inference request, it is more convenient executing inference at the edge, because the burden on the network for transferring the model from the edge server to the device is higher than that for delivering the input data from the device to the edge server, e.g., nearly x1000 for ResNet-152. Instead, if the model can be reused several times ($Q > 1$), it is more beneficial to execute inference at the device. This holds for all models, with the exception of ResNet-152, for which performing inference at the edge almost always implies the lowest communication footprint (at least up to Q=300). For the other models, the Q value above which inference at the device leads to a lower amount of transferred data compared to inference at the edge varies, being lower for the lighter models (i.e., MobileNet and pruned versions).

Energy consumption at the device. Fig. 5 presents the device energy consumption as Q varies, when inference is executed at the edge (as per (3)) and at the device (as per (6)). In the former case, the energy consumption does not change regardless of the model size (full or pruned model) due to the negligible impact of input and output data transfer over the wireless channel. Not surprisingly, the use of a full model for inference execution at the device imposes a higher energy toll than a pruned model: this is because the device has *(i)* to receive a larger model from the edge and *(ii)* execute a computation-heavier model.



(a) ResNet-152 (b) DenseNet201 (c) MobileNet

Figure 3: Total cycle delay for the two different workflows, full and pruned models (with pf=70%), and edge server load, when varying the ML model: (a) ResNet-152, (b) DenseNet201, (c) MobileNet.

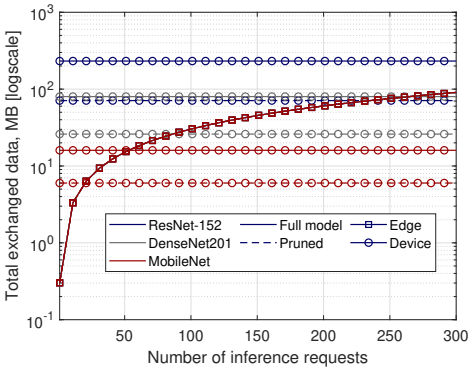


Figure 4: Total exchanged data vs. number of inference requests from the same device within the model lifetime (Q), for different ML models (with pf=70% for pruned models). Curves identified with a square marker are overlapping.

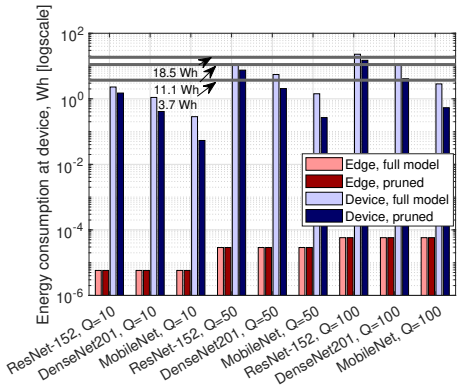


Figure 5: Energy consumption at the device for different ML models (pf=70% for pruned models), for a varying number of inference requests from the same device within the model lifetime (Q).

The energy gain when moving from a full model to a pruned version is in the order of 1.5, 2.7, and 5.4 for ResNet-152, DenseNet201, and MobileNet, respectively. However, it is worth remarking that such a gain is achieved at the expense of additional energy consumption at the edge server, unless the model is not pre-pruned, as highlighted in Fig. 2(b). The same figure also shows three common battery capacity values: *(i)* 3.7 Wh for embedded devices [15], *(ii)* 11.1 Wh for smartphones [16], and *(iii)* 18.5 Wh for PCs [17] (grey lines). While the energy consumption spent by a single on-device inference task (Q=1) is well below all the above thresholds, running inference at the device multiple times may drain its battery. This holds for instance for the most complex model, ResNet-152, for Q=100, no matter whether it is pruned or not. MobileNet, instead, can be executed for values of Q higher than 100, before the device runs out of battery, confirming the suitability of this model for embedded devices.

5 CONCLUSIONS AND FUTURE WORK

We addressed the execution of ML inference tasks in mobile networks when different ways of interaction and cooperation between edge servers and end devices are considered. Furthermore, through measurements and extensive experiments, we assessed the system performance when either a full or a pruned version of an ML model is used at the network edge or at the end device. Notwithstanding the inherent energy saving at the device when inference is executed at the edge, experimental results show that inference offloading may not always be beneficial, e.g., for heavy computational load at the edge, inference at the device turns out to be significantly faster. This is especially true if a pruned version of

a model is used, which, however, may be obtained at the cost of a non-negligible energy footprint at the edge server if the model has to be pruned on demand. In a nutshell, whether inference should be performed at the edge or at the device depends on how complex the ML model is, for how many instances of an inference task the model can be reused before its validity expires, on the overall computational load on the edge server, on the accuracy and time constraints imposed by the target application, and on the battery availability at the device.

Our findings pave the way to several future research directions, including the design of judicious mechanisms for the support of ML tasks in mobile networks that account for the aforementioned aspects and jointly orchestrate inference placement and caching of (pruned) models at the edge.

REFERENCES

[1] Mohammad Saeid Mahdavinejad et al. 2018. Machine learning for Internet of Things data analysis: A survey. *Digital Communications and Networks*, 4, 3, 161–175.

[2] Lachit Dutta et al. 2021. TinyML Meets IoT: A Comprehensive Survey. *Internet of Things*, 16, 100461.

[3] Firas Bayram et al. 2022. From Concept Drift to Model Degradation: An Overview on Performance-Aware Drift Detectors. *Knowledge-Based Systems*, 108632.

[4] Tailin Liang et al. 2021. Pruning and Quantization for Deep Neural Network Acceleration: A Survey. *Neurocomputing*, 461, 370–403.

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

[6] Trevor Gale et al. 2019. The State of Sparsity in Deep Neural Networks. *arXiv preprint arXiv:1902.09574*.

[7] Cankun Zhong et al. 2022. A Sensitivity-based Pruning Method for Convolutional Neural Networks. In *2022 IEEE International Conference on SMC*. IEEE, 1032–1038.

[8] Hyeji Kim et al. 2019. Efficient Neural Network Compression. In *Proceedings of the IEEE/CVF conference on CVPR*, 12569–12577.

[9] Pavlo Molchanov et al. 2016. Pruning Convolutional Neural Networks for Resource Efficient Inference. *arXiv preprint arXiv:1611.06440*.

[10] Francesco Malandrino et al. 2022. Matching DNN Compression and Cooperative Training with Resources and Data Availability. *arXiv preprint arXiv:2212.02304*.

[11] 3GPP. 2022. 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, (Mar. 2022).

[12] Gao Huang et al. 2017. Densely Connected Convolutional Networks. In *Proceedings of the IEEE Conference on CVPR*, 4700–4708.

[13] Andrew G Howard et al. 2017. Mobilenets: Efficient Convolutional Neural Networks for Mobile Vision Applications. *arXiv preprint arXiv:1704.04861*.

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

[15] EEMB Battery. 2023. Datasheet: LP603449-Standard Type Lithium Polymer Battery. [Online] (accessed August 4, 2023) https://www.eemb.com/product-147. (2023).

[16] Microsoft. 2023. Datasheet: Lumia 950. [Online] (accessed August 4, 2023) https://news.microsoft.com/wp-content/uploads/2016/04/Lumia-950-Datasheet.pdf. (2023).

[17] BatterySpace.com. 2023. Datasheet: LiFePO4 battery. [Online] (accessed August 4, 2023) https://www.batteryspace.com/prod-specs/lfp-32650%20Spec.pdf. (2023).