OffloaDNN: Shaping DNNs for Scalable Offloading of Computer Vision Tasks at the Edge

OffloaDNN: Shaping DNNs for Scalable Offloading of Computer Vision Tasks at the Edge

Corrado Puligheddu‡, Nancy Varshney‡, Tanzil Hassan†, Jonathan Ashdown*,
Francesco Restuccia† and Carla Fabiana Chiasserini‡

† Institute for the Wireless Internet of Things, Northeastern University, United States
* Air Force Research Laboratory, United States
‡ Politecnico di Torino, Italy

***Abstract*—Emerging mobile applications often require the execution of computer vision (CV) tasks based on compute- and memory-intensive deep neural networks (DNNs). Although offloading CV tasks to edge servers can decrease resource consumption at the mobile devices, it poses the challenge of handling multiple concurrent tasks with limited computing and memory capacity. In stark opposition with the existing state of the art, we tackle this challenge by jointly optimizing (i) the utilization of resources at the edge, among which memory – so-far widely overlooked – and the radio resources used for task offloading; (ii) which and how many offloaded tasks should be executed; and (iii) the structure of the DNNs. First, we formulate the DNN for scalable Offloading of Tasks (DOT) problem, prove that it is NP-hard, and envision a weighted-tree-based heuristic solution, named OffloaDNN, that efficiently solves the DOT problem. We evaluate OffloaDNN through extensive numerical analysis using state-of-the-art image classification ResNet-18, as well as real-world experiments on the Colosseum emulator. The numerical results show that, in small-scale scenarios, OffloaDNN matches the optimum very closely, and, in larger-scale scenarios, increases the number of admitted offloaded tasks by 26.9% with respect to the state of the art, while saving 82.5% memory and 77.4% per-inference computing time. The numerical results are confirmed by the real-world validation on Colosseum.**

I. Introduction

Computer vision (CV) has become essential for a wide range of applications, including autonomous driving, medical imaging, optical character recognition, and event detection. CV tasks are notoriously computationally demanding as they leverage complex deep neural networks (DNNs) to achieve high accuracy. For example, with respect to MobileNetv2 [1], a DNNs such as ResNet-152 [2] that is 8.7x larger in terms of number of parameters (60M vs 6.9M) can improve inference Top-1 accuracy in ImageNet by 5.2%. However, due to their excessively cheap, small, or light design, mobile devices may be unsuitable for hosting hardware and energy storage that meet the DNNs requirements for supporting CV tasks in a timely manner. To address this issue, prior art resorts to *task offloading*, where mobile devices, connected to an edge server through a radio link, delegate the processing of a CV task to the relatively more powerful edge computing platform. Then the task result is seamlessly returned to the mobile device. This is indeed often the only way to support CV tasks in a mobile scenario without depleting the resources of the devices collecting the data to be processed.

Existing issues. Resource availability at the edge, even if larger than at the mobile devices, is still limited. In particular, *memory* consumption of DNNs is rarely considered as a limiting factor to the execution of tasks at the edge. Further, the *structure* of the DNNs required for the execution of different CV inference tasks, and *the correlation among such structures*, has often been overlooked [3]–[7], thus failing to exploit the benefits that instead accounting for such factors can bring. In Sec. V, we show that carefully shaping and sharing the DNNs, i.e., acting on which blocks of DNNs layers are used for different CV tasks, increases the number of tasks admitted for execution at the edge by 26.9%, while reducing resource consumption by 54.7% compared to the state of the art, thus scaling very well with the diversity and number of tasks.

Scientific challenges. Joint task admission and layer configuration and sharing among different DNNs are highly challenging. This is because (i) the required DNNs depend on the offloaded tasks that can be admitted at the edge and, more specifically, on the required CV methods and object classes; (ii) the available configurations must consider the requirements of admitted tasks and the available resources; (iii) sharing is feasible only when two or more DNNs have a sequence of common layers, which requires freezing such layers during DNN training and fine-tuning the remaining task-specific layers. Additionally, the relationship between admitted task latency and accuracy and the DNN structures is highly nonlinear and cannot be expressed in closed form, which further complicates the issue of DNN shaping and sharing.

Our approach. To optimize the consumption of radio and compute resources, while respecting the requirements of the offloaded tasks, we propose a new framework, named OffloaDNN (scalable Offloading of DNN tasks), whose key innovations are detailed below.

• Hidden layers of a DNN specialize in different feature levels. The initial layers detect low-level features such as corners, edges, colors, etc., and, as we move towards the final layers, they become specialized in detecting high-level features like items, scenes, patterns, etc. [8]. While high-level features are task-specific and can be hardly used for tasks other than the one they have been originally trained on, lower-level features can instead be common among different tasks [9]. Our first intuition, depicted in Fig. 1(left), is to **leverage DNN layers that can be shared at the edge among DNNs serving**

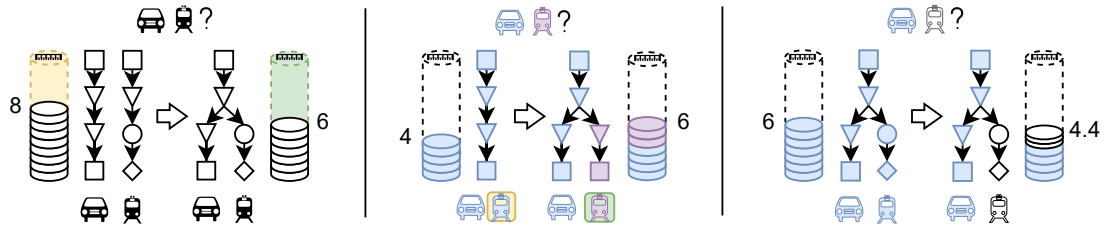

Fig. 1. Exemplified application of the proposed innovations to DNN structures for tasks aiming at classifying different objects (e.g., car and train): (left) DNN block sharing for memory savings; (center) fine-tuning of task-specific blocks to improve accuracy (e.g., for train classification); (right) DNN block pruning (pruned blocks are denoted in white and reduced by an 80% factor), to decrease resource footprint at the cost of possible reduced accuracy.

different offloaded tasks, to save memory while still offering acceptable DNN accuracy;

• While DNN layers trained for the most common CV tasks can be used for a variety of applications (i.e., they can be easily shared), they may be inadequate for other tasks for which they have not been specifically trained. In these cases, fine-tuning task-specific layers, although costly, can improve the level of accuracy attained for such tasks, and, at the same time, freezing the initial, general-purpose layers still allows them to be shared and help mitigate the well-known issue of catastrophic forgetting [10]. Our second intuition (Fig. 1(center)) is thus to **tailor which layers to share among different tasks and which layers to fine-tune, depending on the tasks requirements.** In so doing, we can effectively trade off training cost with tasks accuracy;

• Moreover, when an already-available DNN is accurate enough, we allow for structured pruning of (all or some of) the DNN layers to decrease the compute and memory footprint needed for inference execution while still offering acceptable accuracy. As depicted in Fig. 1(right), **blocks of DNNs layers can be pruned to decrease resource consumption and inference latency, while still meeting accuracy requirements**.

A. Summary of Novel Contributions

• We mathematically formulate the problem of *DNN for scalable Offloading of Tasks* (DOT) to find (i) the optimal composition of (possibly pruned and fine-tuned) blocks of DNNs layers, (ii) the set of offloaded tasks that can be admitted at the edge, and (iii) the allocation of (radio and computing) resources for tasks remote execution. DOT minimizes the task rejection rate and resource consumption, while meeting the accuracy and latency requirements of admitted tasks (Sec. III). The DOT problem is fundamentally different from the existing formulations as, for all admitted tasks, it considers different configurations of the DNNs that can serve for the execution of the inference tasks, with each configuration offering different opportunities for sharing layers with other DNNs and leading to different performance-resource consumption tradeoffs.

• We propose OffloaDNN, the first framework supporting scalable offloading of CV tasks to the edge, to allow for the solution of realistic instances of the DOT problem (which we prove to be NP-hard). OffloaDNN is an efficient heuristic based on weighted tree-based graph modeling of the feasible solutions, which accounts for DNN layers sharing, hence the possible correlation in memory utilization among the DNNs deployed to handle the admitted inference tasks (Sec. IV).

• We evaluate OffloaDNN (Sec. V) through a comprehensive numerical analysis and leveraging a proof-of-concept prototype implemented on the Colosseum network emulator [11], by considering real-world CV tasks and state-of-the-art DNNs. Our results show that, in small-scale scenarios, OffloaDNN performs very closely to the optimum. In large-scale scenarios, it allows for 26.9% more offloaded tasks and substantial resource savings (54.7%) compared to the state-of-the-art approach in [5], while always meeting task constraints. The real-world experiments on Colosseum confirm the validity of our approach. **To allow for the reproducibility of our results and foster further development, we pledge to release our solution as open-source upon paper acceptance**.

II. Motivations

Task offloading requires to optimize both radio resources as well as edge computational resources. Indeed, task input data acquisition for inference execution (and transmitting back the results to mobile devices) requires optimal usage of radio resources. Also, besides the memory taken by the deployed DNNs, fine-tuning the DNN architectures for specific CV tasks and compressing them before deployment and performing inference on the offloaded data require significant computational effort from CPUs and GPUs of edge servers. In this section, we present two experiment-driven motivating factors to propose OffloaDNN. The first one is related to optimizing the *training expense* of DNN models for specific tasks while converging to target performance. The second is related to optimizing *inference compute time* while deploying a DNN in a resource-constrained edge server.

First experiment. Before deploying a DNN for a specific task, the first step is to train the model parameters efficiently. Training a DNN from random initialization of the parameters results in slow convergence up to a target accuracy [12]. Thus, training a new DNN from scratch for each task is not always the best solution. Alternatively, fine-tuning a pre-trained DNN with a new task-related dataset may be a better approach [13]. If the new dataset is similar to the pre-training dataset (i.e., for tasks related to natural images, ImageNet [14] is widely used as a pre-training dataset), we can freeze the early layers of the DNN while preserving the learned features from the larger dataset.

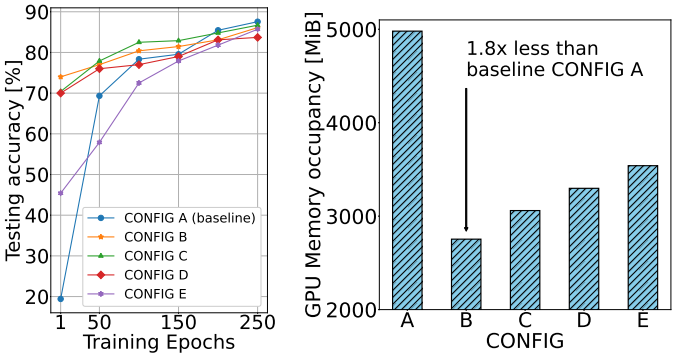

Fig. 2. Comparison between different DNN block training configurations (listed in Table I) applied on ResNet-18 as a feature extractor: (left) Progression of average testing accuracy of the DNN model after each epoch training; (right) Peak GPU Memory Occupancy in mebibyte (MiB) during training.

Fig. 2(left) compares the training costs of various DNN configurations as listed in Table I, and refers to the training of the ResNet-18 [2] feature extractor. The DNN was initially trained on a subset of the ImageNet dataset (Table II), featuring 60 object categories. These are the pre-trained parameters that form the backbone for the configurations in Table I. Subsequently, we train these DNN configurations using a new dataset that contains an additional object class. For fair comparison between all the configuration training, we use batch size of 256, 'Adam' optimizer, 'CosineAnnealing' learning rate scheduler, starting learning rate of 0.2, decay rate of 0.001, Cross-Entropy as loss function. In this experiment, we emulate the fact that a new task is to detect grocery items (e.g., mushrooms). To reach near 80% testing accuracy, CONFIG A takes more than 200 training epochs, while CONFIG B and CONFIG C converge to 80% accuracy faster. Further, CONFIG C outperforms CONFIG D and CONFIG E, which share and freeze fewer layers than CONFIG C. In Fig. 2(right), it is important to notice the reduced GPU memory occupancy by CONFIG B and CONFIG C compared to the other configurations, which indicates that the shared layer-blocks are not using processing resources to train the model parameters. Eventually, after more than 250 epochs, the fully fine-tuned CONFIG A configuration achieves better accuracy than the shared configurations. On the contrary, CONFIG B and CONFIG C models get overfit to the training data after long training epochs and achieve lower accuracy than CONFIG A baseline. Similarly, CONFIG D and CONFIG E converge more slowly to 80% accuracy than CONFIG C. The reason is that both of the DNN structures have more parameters to train during the training process compared to CONFIG C. **The key takeaway is thus that we can opt for respectable accuracy according to task demand with shared configurations with less training cost, or we can fully fine-tune DNN parameters for better performance.**

Second experiment. Using DNNs containing millions of parameters tends to be costly in terms of computational resources. This motivates reducing the number of parameters to make the DNN more compact to deploy in a resource-constrained edge computing scenario. To this end, DNN pruning is a very popular method to alleviate the over-parameterization while maintaining the accuracy of the original DNN [15]. General pruning steps start with training the larger network, then applying the pruning criterion on the model, and then fine-tuning the smaller network again. This iterative process goes on until a satisfactory sparsity level and model performance is reached. In the recent literature [16] [17], authors argue against iterative pruning considering fine-tuning budget limitations in a resource-constrained scenario. To save time and effort for this iterative pruning, single-shot pruning methods [18]–[20] are getting popular.

In this experiment, we validate the intuition that pruning DNN blocks after fine-tuning for the target task will greatly reduce the inference compute time on the edge server, with a potential risk of degrading performance. We use the same configurations as listed in Table I with ResNet-18 feature extractor and apply pruning after 100 epochs of fine-tuning for a new task to detect 'Musical Instruments'. To make a fair comparison between different configurations, we choose 100 epochs of fine-tuning and a constant pruning ratio of 80% for the training phase. After fine-tuning, we apply magnitude pruning from DepGraph [21] to the fine-tuned layer-blocks only, as shared layer-blocks are to be used for other tasks in hand. In Fig. 3(left), we report the experimental results of inference compute time for a dummy input tensor of each configuration compared to the non-pruned version of it. It is evident that CONFIG B-pruned has a smaller inference compute time difference compared to other configurations. Due to 4 shared layers-blocks from the base model, CONFIG B-pruned DNN has the least number of pruned blocks, hence, larger amount of parameters, and it takes longer to infer an input. For the same reason, CONFIG C-pruned, CONFIG D-pruned and CONFIG E-pruned take less and less time than CONFIG B-pruned. Smallest inference compute time is found with CONFIG A-pruned, compared to its baseline CONFIG A, mostly because the entire DNN was fine-tuned and pruned for task-specific purposes.

Fig. 3(right) depicts the Average Class Accuracy of a specific object we want to detect (in this case it is "Electric guitar") for all the configurations and their pruned versions. After pruning, each of the configurations performs a bit worse compared to its original version. In this case, CONFIG B, shows better performance after pruning because most of its layer-blocks are inherited from the base DNN model. Further fine-tuning from this stage of the pruned model may increase the accuracy of all the configurations, but it will also increase the training expense. **The key takeaway from this second experiment is thus that to minimize the inference compute time of CV tasks at the edge, we can choose among different pruned configurations depending on the trade-off they offer between accuracy and inference compute time. Or, we can choose a DNN configuration without pruning if task requirements are very accuracy-intensive.**

Overall, from these two experiments, it is evident that **selecting the best DNN model for task requirements is not**

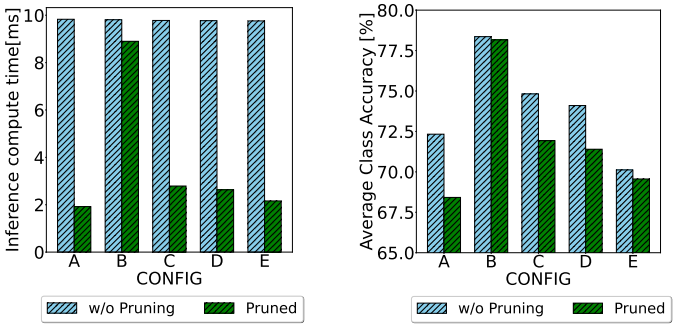

Fig. 3. Effects of applying pruning on different DNN layer-blocks with ResNet-18 feature extractor architecture and configurations listed in Table I: (left) Comparison of inference compute time in milliseconds when input is a dummy tensor (standard procedure to estimate DNN model inference compute time in a system). (right) Average Class Accuracy measured for class "Electric guitar" images. DNN Models were fine-tuned for 100 epochs before applying pruning with ratio of 80%.

a straightforward problem that can be solved with simple heuristics. Rather, it is a complex problem that involves multiple intertwined factors.

TABLE I
DNN BLOCK CONFIGURATIONS (RESNET)

Name	Description
CONFIG A	Entire DNN structure trained from scratch
CONFIG B	First 4 layer-blocks shared from the base DNN
CONFIG C	First 3 layer-blocks shared. Last layer-block + classifier layers fine-tuned
CONFIG D	First 2 layer-blocks shared. Last 2 layer-blocks + classifier layers fine-tuned
CONFIG E	First 1 layer-blocks shared. Last 3 layer-blocks + classifier layers fine-tuned
CONFIG A-pruned	CONFIG A DNN architecture with pruning ratio 80%
CONFIG B-pruned	CONFIG B + Fine-tuned layer-blocks are pruned with ratio of 80%
CONFIG C-pruned	CONFIG C + Fine-tuned layer-blocks are pruned with ratio of 80%
CONFIG D-pruned	CONFIG D + Fine-tuned layer-blocks are pruned with ratio of 80%
CONFIG E-pruned	CONFIG E + Fine-tuned layer-blocks are pruned with ratio of 80%

III. SYSTEM MODEL AND PROBLEM FORMULATION

This section first introduces the model we developed to represent the edge computing system that handles CV tasks offloaded by mobile devices to an edge server (Sec. III-A). Then we present the DOT (DNNs for scalable Offloading of Tasks) problem, which identifies (i) the optimal ratio of tasks to offload to the edge and (ii) the best DNNs configurations

TABLE II
BASE DATASET DESCRIPTION

Objects	Description
Vehicle	12 vehicle categories(e.g., Bus)
Wild animals	18 wild animal categories(e.g., koala)
Snakes	10 snake categories(e.g., green snake)
Cats	6 cat categories (e.g., Persian cat)
Household Objects	14 household objects (e.g., toaster)
Total	60 categories of objects

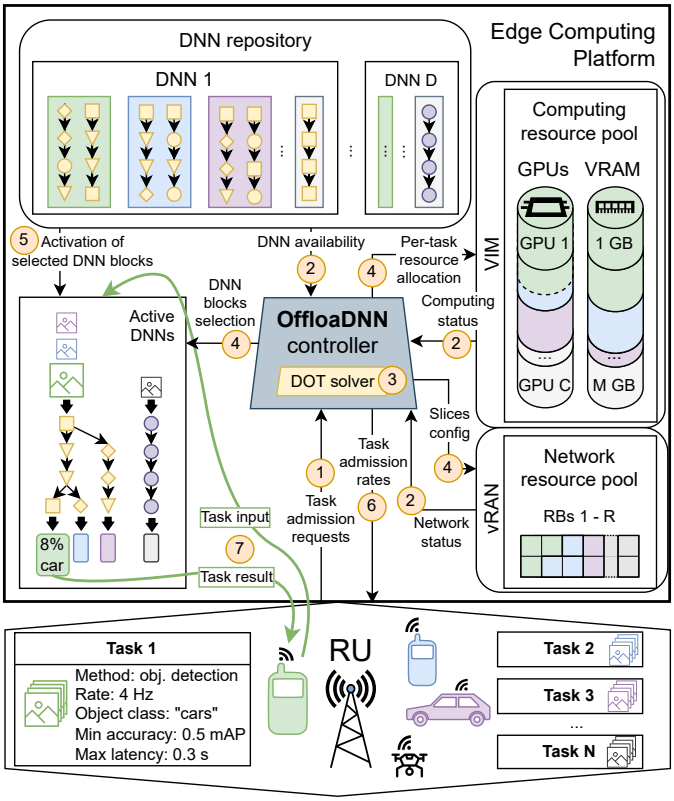

Fig. 4. Architecture and workflow of OffloaDNN. To offload tasks, mobile devices submit task admission requests to the OffloaDNN controller (step 1), which runs the DOT problem solver. Besides the tasks definition, the DOT solver requires as input the available DNNs blocks and their resource cost, and the available computing and radio resources capacity, which are pulled from the Virtual Infrastructure Manager (VIM) and the vRAN (step 2). After obtaining the DOT solution (step 3), the controller allocates the radio slice and the computing resources (step 4), and deploys the selected DNN blocks for the soon-to-be-admitted tasks (step 5). Then the controller notifies the mobile devices about the admitted tasks rates (step 6); after that, the mobile devices can transmit task input data and receive task results (step 7).

to handle such tasks, accounting for both their training and inference costs (Sec. III-B).

A. Reference Model

We consider a scenario, exemplified in Fig. 4, where mobile devices can benefit from offloading CV tasks to an edge computing platform connected with the base station covering the devices. As the edge platform has limited resource capacity, it has to determine which tasks to admit and how to serve them.

Let $\mathbb{T}$ be a set of T inference tasks, with each $\tau \in \mathbb{T}$ representing a CV method that can be implemented through a DNN out of a set, $\mathbb{D}$, of available models, to be applied to the images generated by the mobile devices. Examples include image classification performed through ResNet-18, or MobileNetv2. Each task τ is associated with a minimum required accuracy, A_τ (e.g., mean Average Precision for an object detection task), and a maximum end-to-end latency L_τ, accounting for both networking and processing latency. A task is also associated with priority p_τ, which is a real value between 0 (lowest priority) and 1 (highest priority) indicating the importance of a task, and a request rate λ_τ,

TABLE III
NOTATION

Symbol	Description
$\tau \in \mathbb{T}$	Requested tasks, with $\lvert\mathbb{T}\rvert = T$
$d \in \mathbb{D}$	Set of possible dynamic DNN structures, each able to serve multiple tasks
$s^d \in \mathbb{S}^d$	Block belonging to the dynamic DNN structure d
p_τ	Priority of task τ
$\pi_\tau^d \in \Pi_\tau^d$	Sequence of blocks $[s^d]_d$ belonging to the dynamic DNN structure d, suitable for executing task τ
λ_τ	Request rate of task τ
A_τ	Minimum accuracy tolerable for task τ
L_τ	Maximum latency tolerable for task τ
$\mathbb{Q}_\tau$	Set of auxiliary variables denoting the possible quality levels for the data that are input to task τ
R	Number of available RBs
C	Available compute time (CPU/GPU)
M	Available memory (RAM/VRAM)
σ_τ	SINR of mobile devices requesting task τ
$B(\sigma_\tau)$	No. of bits carried by an RB assigned to a mobile device generating data for τ
$\beta(q_\tau)$	No. of bits associated with transferring data with quality level q_τ as input to task τ
$c(s^d)$	Compute time required by block s^d
$\mu(s^d)$	Memory required by block s^d
$c_t(s^d, \cdot)$	Cost of training s^d
x_τ^d	Binary decision variable taking on 1 when DNN type d is used for task τ
$y_{\pi_\tau^d}$	Binary decision variable, taking on 1 when π_τ^d is selected for task τ
z_τ	Real-valued decision variable representing the task requests admission ratio
r_τ	Integer decision variable indicating the no. of RBs assigned to mobile devices offloading task τ
$m(s^d)$	Binary auxiliary variable, indicating whether s^d is used by at least one task

which specifies the number of images over which the inference task is requested per second.

A task τ can be offloaded by mobile devices to an edge server using a radio network slice specifically allocated for the task. The number of resource blocks (RBs) allocated to a slice, r_τ, for task τ may vary over time but the sum for all admitted tasks requests cannot exceed the available capacity R (expressed in RBs). Also, denoting with σ_τ the average signal-to-noise ratio (SNR) experienced by mobile devices offloading task τ over the allocated radio network, we define $B(\sigma_\tau)$ as the number of bits that a single RB can carry.

An offloaded task is executed by a dynamic DNN structure, d, which is built using blocks $s^d \in \mathbb{S}^d$. Notice that such blocks can represent one or multiple layers of a DNN, or versions thereof pruned by an arbitrary factor. The sequence of blocks in $\mathbb{S}^d$ selected to serve task τ is identified by the *path* on the DNN structure, $\pi_\tau^d = [s^d]_d \in \Pi_\tau^d$. As detailed later, training or fine-tuning the DNNs blocks has a computational cost (CPU/GPU time in seconds); also, let $c(s^d)$ and $\mu(s^d)$ be, respectively, the inference computing time and the utilized memory, associated with block s^d, which can be derived experimentally. The fundamental difference between how computing time and memory utilization associated with active DNN blocks consume the available edge resources is that, for every offloaded task, computing time increases proportionally with the task rate, while memory utilization remains constant.

According to the task context (e.g., image lighting conditions or camera sensor resolution), a quality level $q_\tau \in \mathbb{Q}_\tau$ is associated with the task and determines the achievable accuracy level a_τ, derived again experimentally due to its high non-linearity with respect to the image quality and the processing path pi_τ^d on the DNN. The quality level also determines the number of bits per image, $\beta(q_\tau)$, to be transmitted over the radio link from the device offloading task τ to the edge.

A task experiences end-to-end latency, including both networking and processing components, defined as $l_\tau(q_\tau, r_\tau, \sigma_\tau, y_{\pi_\tau^d}, c(s^d)) = \frac{b(q_\tau)}{B(\sigma_\tau)\cdot r_\tau} + \sum_{s^d \in \pi_\tau^d} c(s^d)$, where the networking component is the transmission time of $\beta(q_\tau)$ bits over a link of capacity $B(\sigma_\tau)\cdot r_\tau$, and the processing one is given by the sum of processing times of the DNN blocks that belong to the selected path π_τ^d. Finally, the computational resource capacity that the edge server can devote to offload tasks is limited to C and M, indicating, respectively, the available compute time (CPU/GPU time in seconds) and memory (RAM/VRAM in GB).

The notations used here and in the following are summarized in Table III.

B. DOT Formulation

We consider the following decision variables:

- the task admission vector $\mathbf{z} = [z_1, \ldots, z_T]$ where $z_\tau \in [0, 1]$ indicates the fraction of task τ request rate that is admitted for offloading;
- the task-DNN mapping vector $\mathbf{x}^d = [x_1^d, \ldots, x_T^d]$, $\forall d \in \mathbb{D}$, where x_τ^d is a binary variable indicating whether task τ is served by the DNN d or not;
- the DNN path selection vector $\mathbf{y}_{\pi^d} = [y_{\pi_1^d}, \ldots, y_{\pi_T^d}]$, $\forall d \in \mathbb{D}$ and path on d, where $y_{\pi_\tau^d}$ is a binary variable that takes on 1 if π_τ^d is used to execute task τ;
- the radio resource allocation vector $\mathbf{r} = [r_1, \ldots, r_T]$ where r_τ is the number of RBs allocated for offloading task τ.

The goal of the DOT problem is to minimize the rejection rate of the tasks (also accounting for their priority), the cost of the radio resources for offloading the admitted tasks, and the cost of the edge resources for both training and inference of the DNNs deployed to handle such tasks. Thus, the DOT problem can be formulated as in the colored box below, where the objective function (1a) weights by parameter α the task admission term and the resource allocation term. More specifically, the latter accounts for: (i) the cost of training each block $s^d \in \mathbb{S}$ that is selected for serving one or multiple admitted tasks normalized to the full DNN training cost (for brevity, we denote with $\mathbb{S}$ the set of all possible blocks of all available DNNs); (ii) the fraction of total radio resources allocated for offloading the admitted tasks, and (iii) the cost of CPU/GPU time consumed by each DNN block to execute inference for the admitted tasks, normalized to the cost yielded by the full DNN. We remark that the training cost (c_t) of a block s^d also depends on the subset of tasks that will use that block (represented through the decision variables $y_{\pi_\tau^d}$), thus accounting for the possible savings that sharing a block among different tasks can bring [22]. Also, such a cost is zero when no

task makes use of s^d. Further, it is worth noting that both the training (c_t) and inference (c) costs can be computed off-line; hence, given the set of tasks $\mathbb{T}$ and possible subsets thereof, the values of such costs are inputs to the DOT problem.

Computing resource budget requirements are enforced by (1b) for the memory and (1c) for the CPU/GPU time, which ensure that they do not exceed the available capacity. We remark that, whenever multiple tasks use the same DNN block s^d, the memory utilization due to s^d is counted only once; this is done by introducing the binary auxiliary variable $m(s^d)$, which takes 1 when s^d is used by at least one task. Conversely, the consumed compute time is summed over all tasks using DNN block s^d, scaled according to the admission task rate $z_\tau \cdot \lambda_\tau$. Similarly, the requirements related to the radio resources are expressed by (1d) and (1e). The former ensures that the number of RBs assigned to the radio network slices serving a task does not exceed the available capacity. The latter imposes that each task is allocated a radio slice that has sufficient bandwidth to transmit task input data of quality q_τ generated by the mobile device experiencing channel quality σ_τ, given the selected admission task rate $z_\tau \cdot \lambda_\tau$.

DNNs for scalable Offloading of Tasks (DOT)

$$\min_{\substack{\mathbf{z},\mathbf{x}^d,\\ \mathbf{y}_{\pi^d},\mathbf{r}}} \sum_{\tau\in\mathbb{T}} \alpha(1-z_\tau)p_\tau + (1-\alpha)\left[\sum_{s^d\in\mathbb{S}} \frac{c_t\left(s^d, \{y_{\pi^d_\tau}\}_{\pi^d_\tau : s^d\in\pi^d_\tau}\right)}{C_t} + \sum_{\tau\in\mathbb{T}} z_\tau\lambda_\tau\left(\frac{r_\tau}{R} + \sum_{d\in\mathbb{D}}\sum_{\pi^d_\tau\in\Pi^d_\tau}\sum_{s^d\in\pi^d_\tau} x^d_\tau y_{\pi^d_\tau}\frac{c(s^d)}{C}\right)\right] \quad \text{(1a)}$$

s.t.

$$\sum_{d\in\mathbb{D}}\sum_{s^d\in\mathbb{S}^d} m(s^d)\cdot\mu(s^d) \le M, \quad \text{(1b)}$$

$$\sum_{\tau\in\mathbb{T}} z_\tau\lambda_\tau \sum_{d\in\mathbb{D}}\sum_{\pi^d_\tau\in\Pi^d_\tau} x^d_\tau y_{\pi^d_\tau}\sum_{s^d\in\pi^d_\tau} c(s^d) \le C, \quad \text{(1c)}$$

$$\sum_{\tau\in\mathbb{T}} z_\tau r_\tau \le R, \quad \text{(1d)}$$

$$z_\tau\lambda_\tau\cdot\beta(q_\tau) \le B(\sigma_\tau)\cdot r_\tau,\ \forall\tau\in\mathbb{T}, \quad \text{(1e)}$$

$$a_\tau(q_\tau, y_{\pi^d_\tau}) \ge A_\tau \mathbb{1}_{z_\tau>0},\ \forall\tau\in\mathbb{T}, \quad \text{(1f)}$$

$$l_\tau(q_\tau, c(s^d,\tau), r_\tau, y_{\pi^d_\tau}, \sigma_\tau)\mathbb{1}_{z_\tau>0} \le L_\tau,\ \forall\tau\in\mathbb{T}, \quad \text{(1g)}$$

$$\sum_{\tau\in\mathbb{T}}\mathbb{1}_{z_\tau>0}\sum_{\pi^d_\tau\in\Pi^d_\tau} y_{\pi^d_\tau}\mathbb{1}_{s^d\in\pi^d_\tau} \le K\cdot m(s^d),\ \forall s^d\in\mathbb{S}_d, \quad \text{(1h)}$$

$$\sum_{\tau\in\mathbb{T}}\mathbb{1}_{z_\tau>0}\sum_{\pi^d_\tau\in\Pi^d_\tau} y_{\pi^d_\tau}\mathbb{1}_{s^d\in\pi^d_\tau} \ge m(s^d),\ \forall s^d\in\mathbb{S}_d \quad \text{(1i)}$$

Task requirements compliance is enforced by constraints (1f) and (1g). Notice that $\mathbb{1}_{z_\tau>0}$ is an indicator function that takes on 1 if $z_\tau>0$. In (1f), the accuracy function associated with task τ must comply with the minimum accuracy requirement A_τ, as long as tasks τ are admitted with a non-zero ratio z_τ. Similarly, in (1g), for every task with a non-zero admission ratio, the maximum tolerable end-to-end latency must be satisfied by the task latency function. Finally, (1h) and (1i) are used to force the values of the auxiliary binary variable $m(s^d)$. Here, $\mathbb{1}_{s^d\in\pi^d_\tau}$ is an indicator function taking 1 when DNN block s^d is part of path π^d_τ. More in detail, (1h) uses the Big M method notation (here, K takes a large value) to force $m(s^d)$ to take 1 when s^d is used by at least one task. Conversely, according to (1i), $m(s^d)$ must take 0 when s^d is not used by any task.

The following proposition holds.

Proposition 1: The DOT problem is NP-hard.

Proof: The proof is based on a reduction in polynomial time from the binary multi-dimensional knapsack problem, which is known to be NP-hard (details omitted for brevity). ■

We remark that, although the DOT formulation above has been given considering no preexisting DNNs already deployed at the edge for previously admitted tasks, it can be trivially extended to deal with a *dynamic scenario* where new tasks offloaded by mobile devices may need to be incrementally accommodated at the edge server. In this case, it is indeed enough to consider the training cost and memory occupancy of already-deployed DNN blocks equal to zero, discount the radio, compute, and memory capacity, and only account for the additional blocks and RBs that may be needed by the set of newly requested tasks.

IV. The OffloaDNN Solution Strategy

In light of the NP-hardness of the DOT problem, we present a heuristic algorithm, named scalable Offloading of DNN tasks (OffloaDNN), for serving inference tasks through efficient sharing and configuration of dynamic DNN structures. Before detailing our approach, we highlight the characteristics of the DOT problem and the main challenges it poses, and how OffloaDNN can effectively address them. The fundamental aspects and the challenges of the DOT problem include:

(i) Size of the solution space: The solution space, especially concerning integer optimizing variables, includes the various permutations of DNN paths for different tasks, which is very large. Solving the DOT problem thus demands a method to explore and evaluate potential solutions efficiently while still being able to make high-quality decisions.

(ii) Accounting for heterogeneous resources: The DOT problem deals with cost and allocation of resources that are diverse, in terms of both type and dimensionality, which need to be considered and effectively utilized in the solution.

(iii) DNN block sharing among different inference tasks: The DOT problem involves choosing the best DNN blocks sequence (i.e., paths) for the offloaded tasks, where these paths share memory resources and training costs. Achieving efficiency in terms of total memory utilization and training cost requires considering the fact that DNN blocks can be shared by different tasks and accounting for such possible correlations while selecting the optimal paths for the various tasks.

In our OffloaDNN approach, we address the characteristics and challenges of DOT as follows:

(i) Graph-based model: We tackle the problem complexity by modeling the solution space through a graph that is *built*

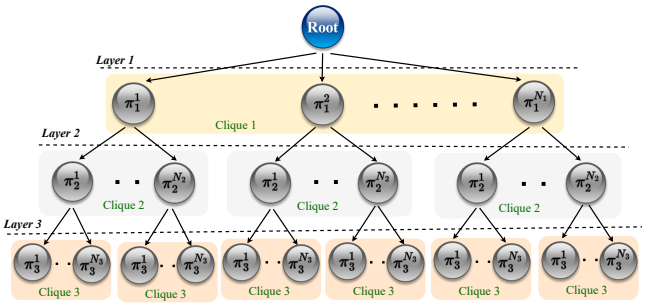

Fig. 5. Hierarchical tree representation for T=3 tasks. Here, task τ=1 has highest priority with N_1 siblings in clique 1, τ=2 has second highest priority with N_2 siblings in clique 2, and τ=3 has third highest priority with N_3 siblings in clique 3. Each vertex v_j=π_τ^j in the clique at t-th layer corresponds to a possible path on a DNN that can serve the task of priority t.

by processing the tasks sequentially (instead of in parallel) according to their priority level, from highest to lowest. Each vertex then represents a possible decision for a task, i.e., a path on a dynamic DNN structure that can be used for that task execution, and edges connect different deployment options when transitioning between tasks. To reduce the solution space and explore it in an efficient manner, upon processing a new task, only the vertices corresponding to feasible solutions, i.e., honoring accuracy and inference latency constraints, are included.

(ii) Assigning attributes to vertices: Vertices carry multi-dimensional and heterogeneous attributes capturing the diverse nature of the resources to be allocated: total memory consumption, training and inference compute time, and radio resource allocation for data transfer. Additionally, attributes represent the task admission ratio, and the accuracy and latency associated with a given path. Attributes also provide flexibility in modeling the system, in that their values can vary in dynamism: they can be fixed, dynamic, or subject to optimization upon graph traversal.

(iii) Tree structure: The graph we build has a tree structure, with each layer corresponding to the decisions that are possible for a given task, from the highest-priority task to the lowest-priority one (the tree root represents the process starting point). Every layer accommodates "sibling" groups of vertices, i.e., groups of vertices that are replicated almost the same but for the value taken by the total memory utilization and the training cost attributes. The group of siblings at layer t is referred to as clique t (Fig. 5.) Each repetition of a clique connects to a single parent vertex, enabling us to track the dependencies of memory and training cost from previously selected paths during graph traversal (i.e., the choices concerning the higher-priority tasks). Indeed, the vertex total memory consumption and training cost attributes update dynamically during traversal. Further, by properly ordering the vertices within each clique according to their inference compute time, we dramatically reduce the complexity of tree exploration by selecting the tree branch that minimizes this metric.

The tree construction and branch selection are detailed next.

A. Tree Construction

We begin by defining a directional tree $\mathcal{T}$=$(\mathcal{V},\mathcal{E})$ where each vertex, v_i∈$\mathcal{V}$, models a possible DNN configuration (path) for the execution of a task. The set of edges $\mathcal{E}$ represents directional connections, from parent to child vertices. The tree has T=$|\mathbb{T}|$ layers: each layer in $\mathcal{T}$ corresponds to a specific task τ, with the sequence of layers from root to leaves matching the descendent order of the tasks priority.

Each layer is constructed by replicating a group of vertices referred to as a clique (Fig. 5): all vertices within a clique share the same parent and represent suitable DNN paths for that specific task. Within each clique, we arrange vertices $\{v_i\}$ from left to right based on the increasing inference compute time of the DNN paths represented by the vertices.

At a given layer, different cliques differ by the value taken by the total memory utilization attribute and the associated training cost. Given $|\mathbb{D}|$ DNNs suitable for task τ of priortiy t, with each DNN d offering N_d=$|\Pi_\tau^d|$ potential paths, a clique for task τ involves aggregating N_τ=$\sum_{d\in\mathbb{D}} N_d$ vertices, with the j-th vertex representing a DNN path, i.e., v_j=π_τ^j. These vertices encapsulate information about the utilized DNN blocks, their characteristics, and their cost. Specifically, each vertex embodies static, dynamic, and to-be-optimized attributes. The static attributes of a vertex include the attained accuracy a_τ, inference latency l'_τ, required computational time by the inference task $\sum_{s^d\in\pi_\tau^j} c(s^d)$, and number of bits $\beta(q_\tau)$ to be transmitted over the radio interface from the mobile device providing the edge with the input data for the task with quality q_τ. The variable attributes are the total memory consumption and associated training cost attributed to the corresponding blocks for the tasks processed till this tree layer. The attributes to optimize include task admission ratio z_τ and resource block allocation r_τ, which are initialized to 1 during the tree construction phase.

Edges e∈$\mathcal{E}$ link each vertex within a layer to all vertices in the subsequent task's clique, maintaining task priority order. This creates a complete tree structure comprising $\mathbf{\Pi}_{\tau\in\mathbb{T}} N_\tau$ vertices; thus, traversing all vertices proves computationally challenging. To manage complexity, at every layer, vertices violating the accuracy constraint or associated with an inference compute time greater than L_τ are removed.

Each branch within the tree, denoted by b_k, comprises vertices corresponding to distinct layers, ensuring that each branch contains a single vertex representing a specific DNN path (i.e., configuration option) for a task τ from layer t. Each branch holds records of attributes associated with the vertices traversed in that branch, as detailed next.

B. Tree Traversal

We define the cost function representing the total cost associated with a tree branch as the DOT objective function in (1a). In so doing, the optimal solution can be found by selecting the least-cost branch. Traversing the tree to compute the branches cost involves employing a Depth-First Search (DFS) approach to track branch costs and update the dynamic attributes of the vertices within each branch.

More specifically, while traversing the vertices of a branch b_k, the memory consumption and training cost at each subsequent vertex $v_j{=}\pi_\tau^j$ at layer t and task τ are updated, considering the additional costs incurred by employing new blocks $\{s^d\}$ compared to those used by the preceding vertices within the same branch up to layer $t{-}1$. If the memory consumption exceeds the threshold M at any $v_j{=}\pi_\tau^j$ vertex on a branch b_k, exploration of that branch halts. Thus, after exploring all branches, the cost for each branch can be computed by solving an optimization problem with (i) the objective function as in (1a) but with $\mathbf{x}_\tau^d$, $\mathbf{y}_\tau^d$, π_τ^d given (note that the latter are the vertices on the considered branch); (ii) the constraints are as in (1c)–(1e) and (1g), since while building the tree we already made sure that those related to memory, accuracy, and latency were honored. Finally, the branch with the least cost, denoted by b^*, provides the optimal task admission ratio $\mathbf{z}_\tau^*$ and resource allocation $\mathbf{r}_\tau^*$. Within this branch, the vertex information π_τ^j at layer t identifies the optimal DNN d and the corresponding DNN path for task τ, implying that $x_\tau^d{=}1$ and $y_{\pi_\tau^d}{=}1$.

In this new optimization problem, the objective function is a linear combination of known constants and decision variables $\mathbf{z}_\tau$ and $\mathbf{r}_\tau$. The first two and the fourth constraints are linear and, hence, convex. The third constraint involves linear combinations of z_τ and r_τ. Also, $B(\sigma_\tau)$ is positive and $b(q_\tau)$ is a convex function. Hence, the problem is convex in $\mathbf{z}_\tau$ and $\mathbf{r}_\tau$ and can be solved to the optimum by using any convex optimizer. Nevertheless, the above solution strategy has exponential complexity, namely, $\mathcal{O}(N_{\max}^T T^2)$. Indeed, the possible number of vertices in a clique is $\sum_{d\in\mathbb{D}]} |\Pi_\tau^d|$ (see Sec. IV-A); hence, the total number of vertices in $\mathcal{T}$ is at most $N_{\max}{=}\mathbf{\Pi}_{\tau\in\mathbb{T}} \sum_{d\in\mathbb{D}]} |\Pi_\tau^d|$. Also, the worst case complexity for solving the linear optimization problem for $\mathbf{z}_\tau$ and $\mathbf{r}_\tau$ with $2(T{+}1)$ constraints and maximum $2T$ variables for a branch is $\mathcal{O}(4T{\times}(T{+}1))$.

We therefore leverage the method followed while constructing the tree (i.e., the fact that at each layer vertices appear in increasing order w.r.t. their inference compute time), and, while traversing the tree from the root to the leaves, we select the first branch. The rationale is that, in (1a), the total inference cost is minimized when the corresponding compute time of all tasks is minimal. Notably, though the inference compute cost of the branches increases from left to right, the value of the DOT objective function may not follow the same trend, as it includes further terms. However, OffloaDNN exhibits now a *polynomial, and indeed dramatically reduced complexity*, namely, $\mathcal{O}(T^2)$, at the expense of achieving a sub-optimal solution. We will show in Sec. V that, in spite of this, OffloaDNN matches the optimum very closely.

V. Experimental Evaluation

In this section, we first introduce the setup configuration used to derive our numerical results and the performance of OffloaDNN against the optimum in a small-scale scenario, and against the SEM-O-RAN state-of-the-art competitor [5] in a larger-scale scenario (Sec. V-A). Then we describe the settings adopted for the validation of OffloaDNN on the Colosseum emulator and the end-to-end latency that we obtained for the considered inference tasks (Sec. V-B).

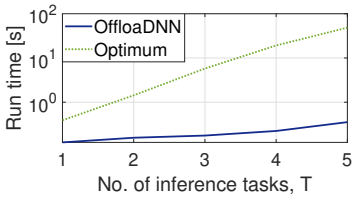

Fig. 6. Comparison of the average runtime taken by the optimum and the OffloaDNN solution strategies, in the small-scale scenario, as the number of inference tasks T varies.

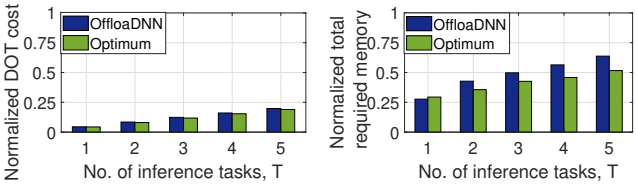

Fig. 7. Small-scale scenario: Comparison of the total DOT cost and memory utilization due to active DNN blocks, under the optimal and the OffloaDNN strategies, as the number of inference tasks T varies.

TABLE IV
Scenarios Parameters

Parameter	Small scenario	Large scenario
T	$\{1,\dots,5\}$	20
$\mathbb{T}$	$\{1\},\dots\{1,\dots,5\}$	$\{1,\dots,20\}$
λ_τ [req./s]	$5\,\forall\tau$	$\forall\tau$ low: 2.5; medium: 5; high: 7.5
A_τ [top-1]	$[0.9, 0.8, 0.7, 0.6, 0.5]$	$0.8 - 0.015\cdot\tau$, $\tau{\in}\mathbb{T}$
L_τ [ms]	$[200, 300, 400, 500, 600]$	$200 + 20\cdot\tau$, $\tau{\in}\mathbb{T}$
$\lvert\mathbb{D}\rvert$	3	125
$\lvert\Pi_\tau^d\rvert$	5	10
C [s]	2.5	10
C_t [s]	1000	1000
M [GB]	8	16
$\beta(\sigma_\tau)$ [Kb]	350	350
$B(\sigma_\tau)$ [Mbps]	0.35	0.35
α	0.5	0.5
p_τ	$[0.8, 0.7, 0.6, 0.5, 0.4]$	$[1, 0.95, \dots, 0.1, 0.05]$
R [RBs]	50	100

A. Numerical Results

Small-scale scenario. We crafted a small-scale scenario to compare OffloaDNN to the optimum, which can be practically derived only for problem instances with few tasks, from 1 to 5. Tasks are ordered according to decreasing priority, while their request rate is fixed at 5 req/s for every task. Each task is also assigned a distinct latency and accuracy requirement as specified in Table IV. For the DNNs and paths reported in the table, we remark that each DNN path is composed of four blocks; the costs implied by the different blocks were experimentally characterized under settings similar to those used in Sec. II.

As outlined in Sec. IV-B, the optimal solution of the DOT problem can be obtained by traversing all branches of the tree $\mathcal{T}$. For each branch, the optimization involves adjusting $\mathbf{z}_\tau$ and $\mathbf{r}_\tau$, followed by the computation of the total DOT cost for that

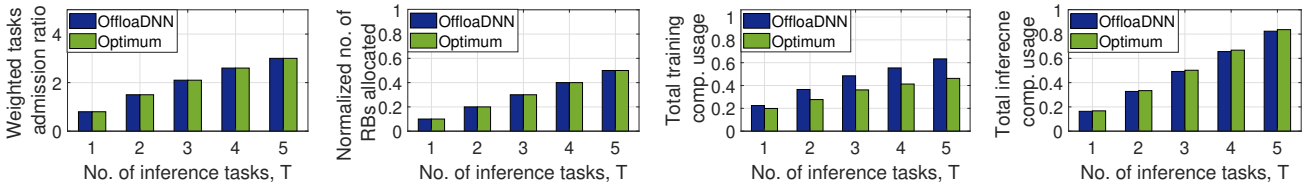

Fig. 8. Small-scale scenario: comparison between OffloaDNN and the optimum as a function of the number of inference tasks T: (left) average task admission ratio weighted by the tasks' priority; (center-left) total number of RBs allocated to the tasks' slices, normalized to the maximum available; (center-right) total compute usage for the training of the active DNNs; (right) total compute usage for the admitted inference tasks, normalized to the maximum available.

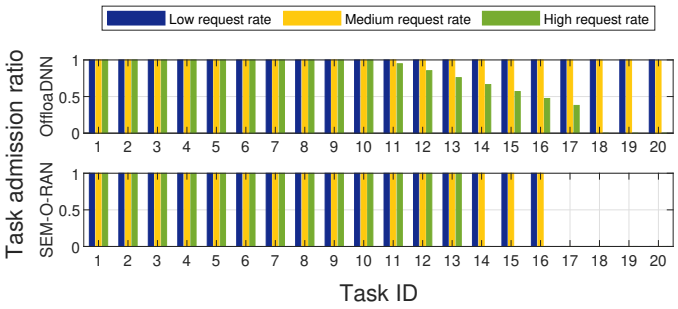

Fig. 9. Large-scale scenario: admission rate of each task for OffloaDNN (top) and SEM-O-RAN (bottom).

specific branch. The branch with the lowest cost provides the optimal solution. This benchmark serves here as the basis for evaluating our proposed OffloaDNN.

The reduction in run-time complexity exhibited by OffloaDNN compared to solving the DOT problem to the optimum is highlighted in Fig. 6. Already for a number of tasks greater than 1, the runtime of OffloaDNN is over one order of magnitude less than that of the optimum. Remarkably, this comes at the expense of a negligible increase in the cost, as shown in Fig. 7(left) reporting the normalized value of the DOT objective function obtained with OffloaDNN and the optimum. Fig. 7(right) shows that also OffloaDNN memory consumption is just slightly higher than that of the optimum, suggesting that OffloaDNN lets tasks share fewer DNN blocks than they could. It is worth noticing, however, that the objective of DOT is not to reduce the total memory consumption at the edge, rather to maintain it below the available quota M – a goal that is amply achieved as memory usage is at most 64% of the available budget.

The cost breakdown in its components is reported in Fig. 8. The weighted tasks admission ratio (Fig. 8(left)) is computed as the summation over all tasks of the product of the task admission ratio and the task corresponding priority. We observe that OffloaDNN allows for the same weighted task admission ratio as the optimum. Similarly, looking at the total number of RBs allocated to the tasks offloading (normalized to the available bandwidth R) in Fig. 8(center-left), one can observe that OffloaDNN performs as good as the optimum. Fig. 8(center-right) reveals that the slightly higher DOT cost under OffloaDNN relatively to the optimum is due to an increased cost of training the selected DNNs structures. However, remarkably, Fig. 8(right) underlines that OffloaDNN requires a lower inference compute usage than the optimum, with such usage accounting for the aggregate compute resource fraction necessary for executing all inference tasks admitted at the edge. The reduction of this relevant metric is attained thanks to the way OffloaDNN has been designed. Specifically, it is due to the combination of sorting the vertices within each clique based on the required compute time while building the weighted-tree graph, *and* the selection of the first branch of the tree performed by OffloaDNN.

Large-scale scenario. We now consider 20 inference tasks, which, again, have the same requesting rate but distinct values of priority as well as accuracy and latency requirements. In this case, we also investigate the system performance for different values of tasks requests rate, from low to medium and high (see Table IV), thus varying the load of task requests at the edge. Other relevant parameters are defined in Table IV. As for the DNN blocks, we consider the individual layers of the state-of-the-art ResNet-18 for image classification, in their full version as well as fine-tuned and then pruned by 80%. Again, each DNN path is composed of four blocks, and the blocks costs were experimentally characterized.

As obtaining the optimum in the large-scale scenario is impractical, we compare OffloaDNN against the SEM-O-RAN state-of-the-art solution [5]. We recall that SEM-O-RAN aims to maximize the total number of admitted offloaded tasks, multiplied by their value (i.e., the priority level in our scenario), till there are enough resources available. Furthermore, SEM-O-RAN does not admit/reject individual inference task requests, rather either it admits all requests for a given inference task, or it rejects them all.

The experiments reveal that, at a low request rate, all tasks achieve admission ratio equal to 1 in OffloaDNN, while only 16 get admitted in SEM-O-RAN (Fig 9). With a medium request rate, OffloaDNN admits 19 out of 20 tasks with ratio 1 and the lowest priority task with 0.99 admission ratio. In contrast, SEM-O-RAN still admits only 16 out of 20 tasks; this occurs because the task requests deplete all RBs (Fig. 10(left-center)). A similar trend is observed at high task requests rate, where OffloaDNN admits the top 10 priority tasks with admission ratio 1, while the next 7 had diminishing admission ratio, and the last three are all rejected due to RB saturation. SEM-O-RAN, instead, admits only 13 tasks and rejects all the others. Consequently, for both OffloaDNN and SEM-O-RAN, the weighted tasks admission ratio (Fig. 10(left)) decreases as the task request rate increases, but our solution always outperforms its counterpart.

In terms of allocated RBs, Fig. 10(left-center) underlines

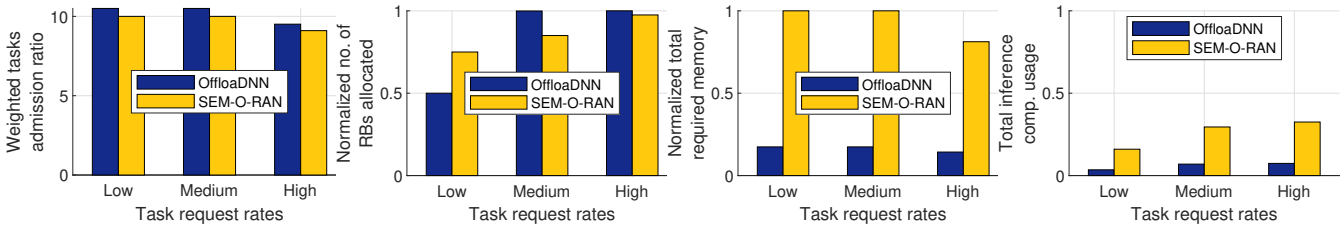

Fig. 10. Large-scale scenario: comparison between OffloaDNN and SEM-O-RAN as the tasks requests rate varies: (left) average task admission ratio weighted by the tasks priority; (center-left) total number of RBs allocated to the tasks slices, normalized to the maximum available; (center-right) total memory utilization for the active DNNs; (right) total compute usage for the admitted inference tasks, normalized to the maximum available.

that OffloaDNN saves nearly 33% of the RBs when the task request rate is low. Moreover, both OffloaDNN and SEM-O-RAN tend to saturate the available RBs as the task request rate grows from low to medium and high. As for the total required memory, Fig. 10(right-center) shows that OffloaDNN achieves significant savings compared to SEM-O-RAN due to block sharing among 20 tasks. Notice also that, under OffloaDNN, memory usage remains the same for low and medium task request rates because our solution selects the same tree branch and uses the same amount of memory. In the case of high task request rate, OffloaDNN still chooses the same branch, but the admission ratio of the last three tasks being zero results in fewer active blocks, thus reducing memory utilization.

Additionally, although the total inference compute usage increases with the task request rate for both schemes (Fig. 10(right)), OffloaDNN substantially outperforms SEM-O-RAN, thanks to the way it sorts and selects the DNN paths. Instead, SEM-O-RAN, consistently with its objective, tends to serve the tasks with higher priority (i.e., higher value) and discards entirely the low-priority ones whenever there are not enough resources. It follows that, as the tasks requests rates grow, SEM-O-RAN can accommodate fewer and fewer "difficult" tasks that, due to their stringent accuracy and latency requirements, would require large resource usage.

For completeness, we also report the total DOT cost: [0.35,0.44,0.74], and training cost: [0.81,0.81,0.67], for low, medium, and high traffic load (resp.), obtained under OffloaDNN. With a penalty for reduced admission rates, the DOT cost rises with increasing task request rates. Meanwhile, the total training compute usage remains constant for low and medium task request rates but decreases for high task request rates, aligning with the explanation provided for the total required memory.

As a result, when compared to SEM-O-RAN, on average OffloaDNN exhibits a **sensible gain in the number of offloaded tasks that can be served at the edge (26.9% increase), while substantially reducing memory and compute usage (by 82.5% and 77.3% resp.), as well as radio resources consumption (by 4.4%)**.

B. Experimental Results on Colosseum

We conduct real-world experiments of a small-scale scenario on the Colosseum wireless network emulator [11], to recreate a realistic LTE network scenario, with a Standard Radio Node (SRN) acting as an Edge Platform, hosting the

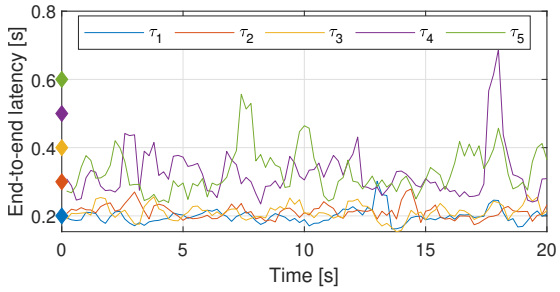

Fig. 11. Experiments in Colosseum: time evolution of task end-to-end latency and maximum latency targets (diamond markers). For clarity, a moving average with a window size of 3 samples is used.

vRAN Base Station, the computing platform for the execution of offloaded tasks, and the OffloaDNN controller, while 5 SRNs generate task requests and act as UEs. For the massive channel emulator (MCHEM) configuration, we set a 20 MHz FDD bandwidth (100 RBs), entirely dedicated to the LTE cell, and a static 0 dB path loss. We set tasks with λ_τ, A_τ, and L_τ as specified in Table IV for the small-scale scenario. We also consider 10 different DNN paths, each exhibiting distinct values of accuracy, training cost, memory usage, and inference compute time. The details of these DNNs are input into the OffloaDNN controller, which outputs, for each task, the optimal DNN path, task admission ratio, and no. of allocated RBs. Such values are used to configure Colosseum's computing and networking environment. Specifically, the RB allocation is set through SCOPE [23], while the admission ratio is sent to the UEs to set the task inference rate. Fig. 11, which reports the evolution over time of the end-to-end latency experienced by the different tasks, validates our framework from the operational viewpoint and demonstrates that the solution generated by OffloaDNN provides latency values within the specified task constraints.

VI. Related Work

Several recent works have addressed the task offloading optimization problem, which is generally formulated as a mixed integer non-linear problem (MINLP) and, due to its complexity, in most cases, cannot be solved to optimality. A formulation (again as a MINLP) of the joint optimization of the offloading strategy and the allocation of edge computing resources can be found in [24], where the authors solve it by quasi-convex/convex optimization methods and an efficient heuristic algorithm. The study in [25] formulates the MINLP

problem of joint offloading, content caching, and resource allocation and solves it by tree-search and branch and bound.

Another body of work aims to solve the above problems, and variants thereof, through machine-learning techniques. For instance, [26] develops a deep learning method for multi-label classification, while minimizing the system overhead in a single-user, single-cell scenario. In [24], the authors propose a feedforward neural network to jointly optimize the offloading decision and the allocation of computing resources at the edge, considering latency constraints. Latency constraints are also the focus of [27], which deals with statistical rather than deterministic latency guarantees. In particular, [27] formulates a model to correlate QoS guarantees with task offloading strategies, and proposes an algorithm to offload tasks with statistical QoS using convex optimization.

In the context of CV tasks execution, [28] proposes a solution for real-time CV applications that tries to achieve the best DNN accuracy and delay according to the hardware type for which the DNN workload is intended. The authors of [29] use a feature specific for CV, i.e., adaptive quality optimization, to offload tasks by selecting a suitable execution version according to task latency constraints. A similar concept of adaptive task quality is employed by SEM-O-RAN [5], a task offloading framework that maximizes the number of admitted tasks by (i) applying semantic compression to task input images, which decreases the consumption of edge resources, and (ii) allocating edge resources of different types in a balanced manner, as to avoid resource starvation. Contrary to ours, the above approaches do not leverage DNN blocks sharing, optimizations of the DNNs structure, or fine-tuning and pruning, and only consider binary task admission decisions.

In summary, to the best of our knowledge, no existing work specifically addresses the scalability of CV tasks offloading, or looks at the CV DNN structures as a way to optimize the execution of such tasks at the edge.

VII. Conclusions

We tackled the problem of executing multiple computer vision tasks offloaded by mobile devices to the edge. In so doing, we leveraged three main innovations: (1) a proper set of DNN layers can be shared among diverse offloaded tasks to save memory resources at the edge; (2) a tailored number of common layers can be "frozen" while task-specific layers can be fine-tuned, to limit training costs, preserve previously acquired knowledge, and, at the same time, fulfill tasks accuracy requirements; (3) task-specific layers can be pruned in a customized manner to further save memory and computing time at the edge while meeting accuracy requirements. To best apply such innovations, we formulated an optimization problem that determines the most efficient DNNs configurations to be deployed, the tasks offloading rate that the edge can support, and the radio and computing resources to be allocated. We then envisioned a low-complexity solution strategy that efficiently and effectively solves the above (NP-hard) problem. Using ResNet-18, extensive numerical results, and real-world experiments on the Colosseum emulator, we validated our framework and demonstrated that our solution matches the optimum very closely in small-scale scenarios, and substantially outperforms its state-of-the-art alternative (with 82.5% reduction in memory usage, 77.3% decrease in inference compute time, and 26.9% more admitted offloaded tasks) in larger-scale scenarios.

References

[1] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "MobileNetV2: Inverted Residuals and Linear Bottlenecks," in *IEEE CVPR*, pp. 4510–4520, 2018.

[2] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *IEEE CVPR*, 2016.

[3] S. Jošilo and G. Dán, "Computation offloading scheduling for periodic tasks in mobile edge computing," *IEEE/ACM Trans. on Netw.*, vol. 28, no. 2, 2020.

[4] X. Wang, J. Ye, and J. C. Lui, "Decentralized task offloading in edge computing: A multi-user multi-armed bandit approach," in *IEEE INFOCOM 2022*, 2022.

[5] C. Puligheddu, J. Ashdown, C. F. Chiasserini, and F. Restuccia, "SEM-O-RAN: Semantic O-RAN slicing for mobile edge offloading of computer vision tasks," *IEEE Trans. on Mob. Comp.*, 2023.

[6] Z. Dong and et al., "EdgeMove: Pipelining device-edge model training for mobile intelligence," in *ACM WWW*, 2023.

[7] H. Sun and et al., "BIRP: Batch-aware inference workload redistribution and parallel scheme for edge collaboration," in *ACM ICCP*, 2023.

[8] B. Neyshabur, H. Sedghi, and C. Zhang, "What is being transferred in transfer learning?," *Adv. in Neur. Inf. Proc. Syst.*, vol. 33, 2020.

[9] K. Ahmed, M. H. Baig, and L. Torresani, "Network of experts for large-scale image categorization," in *Computer Vision – ECCV 2016* (B. Leibe, J. Matas, N. Sebe, and M. Welling, eds.), (Cham), Springer International Publishing, 2016.

[10] G. I. Parisi, R. Kemker, J. L. Part, C. Kanan, and S. Wermter, "Continual lifelong learning with neural networks: A review," *Neur. Netw.*, 2019.

[11] L. Bonati and et al., "Colosseum: Large-scale wireless experimentation through hardware-in-the-loop network emulation," in *IEEE DySPAN*, 2021.

[12] K. He, R. Girshick, and P. Dollár, "Rethinking ImageNet pre-training," in *IEEE/CVF ICCV*, 2019.

[13] A. Sharif Razavian, H. Azizpour, J. Sullivan, and S. Carlsson, "CNN features off-the-shelf: An astounding baseline for recognition," in *IEEE CVPR Workshops*, 2014.

[14] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A large-scale hierarchical image database," in *IEEE CVPR*, 2009.

[15] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *arXiv:1510.00149*, 2015.

[16] X. Ma, , *et al.*, "Sanity checks for lottery tickets: Does your winning ticket really win the jackpot?," *Advances in Neural Information Processing Systems*, vol. 34, 2021.

[17] Z. Liu, M. Sun, T. Zhou, G. Huang, and T. Darrell, "Rethinking the value of network pruning," *arXiv:1810.05270*, 2018.

[18] N. Lee, T. Ajanthan, and P. H. S. Torr, "SNIP: Single-shot network pruning based on connection sensitivity," 2018.

[19] J. van Amersfoort, M. Alizadeh, S. Farquhar, N. Lane, and Y. Gal, "Single shot structured pruning before training," *arXiv:2007.00389*, 2020.

[20] H. Kohama, H. Minoura, T. Hirakawa, T. Yamashita, and H. Fujiyoshi, "Single-shot pruning for pre-trained models: Rethinking the importance of magnitude pruning," in *IEEE/CVF ICCV*, 2023.

[21] G. Fang, X. Ma, M. Song, M. B. Mi, and X. Wang, "Depgraph: Towards any structural pruning," in *IEEE/CVF CVPR*, 2023.

[22] R. Aljundi, P. Chakravarty, and T. Tuytelaars, "Expert gate: Lifelong learning with a network of experts," in *IEEE CVPR*, 2017.

[23] L. Bonati, S. D'Oro, S. Basagni, and T. Melodia, "Scope: An open and softwarized prototyping platform for nextg systems," in *ACM MobiSys*, 2021.

[24] B. Yang, X. Cao, J. Bassey, X. Li, and L. Qian, "Computation offloading in multi-access edge computing: A multi-task learning approach," *IEEE Trans. on Mob. Comp.*, vol. 20, no. 9, 2021.

[25] J. Zhang *et al.*, "Joint resource allocation for latency-sensitive services over mobile edge computing networks with caching," *IEEE Internet of Things J.*, 2019.
[26] S. Yu, X. Wang, and R. Langar, "Computation offloading for mobile edge computing: A deep learning approach," in *IEEE PIMRC*, 2017.
[27] Q. Li, S. Wang, A. Zhou, X. Ma, F. Yang, and A. X. Liu, "Qos driven task offloading with statistical guarantee in mobile edge computing," *IEEE Trans. on Mobile Computing*, 2022.
[28] Z. Fang, J.-H. Lin, M. B. Srivastava, and R. K. Gupta, "Multi-tenant mobile offloading systems for real-time computer vision applications," in *ACM ICDCN*, 2019.
[29] A. Toma, J. Wenner, J. E. Lenssen, and J.-J. Chen, "Adaptive quality optimization of computer vision tasks in resource-constrained devices using edge computing," in *IEEE/ACM CCGRID*, 2019.