Energy-Efficient Service Placement for Latency-Sensitive Applications in Edge Computing

Gopika Premsankar and Bissan Ghaddar

***Abstract*—Edge computing is a promising solution to host artificial intelligence (AI) applications that enable real-time insights on user-generated and device-generated data. This requires edge computing resources (storage and compute) to be widely deployed close to end devices. Such edge deployments require a large amount of energy to run as edge resources are typically overprovisioned to flexibly meet the needs of time-varying user demand with a low latency. Moreover, AI applications rely on deep neural network (DNN) models that are increasingly larger in size to support high accuracy. These DNN models must be efficiently stored and transferred, so as to minimize their energy consumption. In this article, we model the problem of energy-efficient placement of services (namely, DNN models) for AI applications as a multiperiod optimization problem. The formulation jointly places services and schedules requests such that the overall energy consumption is minimized and latency is low. We propose a heuristic that efficiently solves the problem while taking into account the impact of placing services across time periods. We assess the quality of the proposed heuristic by comparing its solution to a lower bound of the problem, obtained by formulating and solving a Lagrangian relaxation of the original problem. Extensive simulations show that our proposed heuristic outperforms baseline approaches in achieving a low energy consumption by packing services on a minimal number of edge nodes, while at the same time keeping the average latency of served requests below a configured threshold in nearly all time periods.**

***Index Terms*—Deep neural network (DNN) model placement, edge computing, optimization, service placement.**

I. Introduction

EDGE computing allows applications to be hosted closer to end users by bringing computing, storage, and networking resources to the edge of the network [1]–[3]. Edge computing has the added benefit of reducing the amount of data that has to be sent to the cloud, which is beneficial for Internet of Things (IoT) applications that process high-bandwidth data in a privacy-preserving manner (e.g., live video analytics) [4], [5]. Recently, IoT applications that rely on artificial intelligence (AI) to draw insights from data generated by end-devices are increasingly being deployed on the edge [5], [6]. Such applications rely on inference from deep neural network (DNN) models on device-generated data. DNN models are hosted on the cloud today [7], [8], and are expected to run on the edge to support real-time, privacy-preserving and low-latency applications [2], [5]. This reduces the need to run inference on resource-constrained, battery-powered IoT devices while still meeting strict latency constraints [6].

Edge computing requires storage and compute resources to be widely distributed close to the end devices generating data. Edge resources must be always-on and overprovisioned to support on-demand, flexible scaling of applications to meet user demand [9], [10]. Such edge deployments require a large amount of energy to run, and contribute to the already-high energy consumption of the cloud today [11]. A recent study has shown that the overall utilization of edge computing infrastructure is lower than cloud data centers [12], implying that edge servers are even more overprovisioned than in the cloud. Moreover, the AI applications targeted to run on the edge increasingly rely on computationally expensive and large DNN models [8]. As mobile network operators deploy edge-based AI applications, the energy consumption of running DNN inference increases the operating expenditure of network operators [13]. Thus, there is an urgent need to design solutions that intelligently manage resources for DNN inference, i.e., how DNN models are stored, transferred, and run [8], to minimize energy consumption.

This article considers the optimal placement of services, i.e., DNN models, and scheduling of requests for AI applications on edge nodes (ENs) such that the energy consumed by ENs is minimized and the latency experienced by end devices is low. The problem is challenging as a solution that minimizes consumed energy must consolidate workloads by placing services on a few ENs. However, the latency of serving requests from end devices must remain low despite the lower number of available ENs. Previous research has either minimized the latency experienced by users [14], [15] or considered fixed energy budgets for ENs [16]. Next, edge workloads have predictable patterns [12], which may be used to decide a placement strategy for future time periods [17]. However, the optimal placement of services in ENs over multiple time periods is challenging to solve efficiently. This is because loading a service on an EN at the beginning of a time slot incurs in

Manuscript received 30 November 2021; revised 15 February 2022; accepted 17 March 2022. Date of publication 28 March 2022; date of current version 7 September 2022. The work of Gopika Premsankar was supported in part by the Academy of Finland under Grant 338854 and in part by the Postdoc Pool Grant from the Finnish Cultural Foundation. The work of Bissan Ghaddar was supported in part by the David G. Burgoyne Faculty Fellowship and in part by the NSERC Discovery under Grant 2017-04185. *(Corresponding author: Gopika Premsankar.)*

Gopika Premsankar is with the Department of Computer Science, University of Helsinki, 00560 Helsinki, Finland (e-mail: gopika.premsankar@helsinki.fi).

Bissan Ghaddar is with Ivey Business School, Western University, London, ON N6G 0N1, Canada (e-mail: bghaddar@ivey.ca).

Digital Object Identifier 10.1109/JIOT.2022.3162581

an overhead (e.g., due to the cost of downloading data over the network or loading data in memory); thus, energy-efficient placement decisions in a particular time period are dependent on whether the service is already present on a particular EN. Solving for the optimal placement of services in each time slot individually may result in a suboptimal solution over multiple time periods [17] unless the dependencies between time slots are captured in the formulation of the optimization problem.

The contributions of our work are threefold. First, we propose a novel multiperiod optimization problem that places services (DNN models) on ENs such that energy consumed is minimized while at the same time ensuring requests are served on average within a certain latency. The formulation decides both a placement plan for the DNN models and fraction of requests to be assigned to each EN for future time periods. It takes into account the effect of loading DNN models across multiple time periods, the limited capacity of ENs, and requirements to replicate services on multiple ENs. Second, we propose a heuristic that efficiently solves the problem in a step-by-step manner by taking into account the models placed in the previous time period. Third, we utilize a Lagrangian relaxation to exploit the structure of the multiperiod problem and decompose the problem into individual time periods. By solving the Lagrangian relaxation iteratively, we obtain a lower bound on the original multiperiod problem. This lower bound provides quality guarantees on our proposed heuristic. Finally, we evaluate our solution in several network instances as well as with real traces from a cloud provider. Our solution outperforms baseline approaches across different network instances in achieving a low energy consumption by packing services on minimal number of ENs, while at the same time keeping the average latency of served requests below the configured threshold in nearly all time periods. The solutions are on average within 1.9% of the lower bound obtained with the Lagrangian relaxation; this indicates a high-quality solution for the heuristic.

The remainder of this article is organized as follows. Section II reviews the state of the art. Section III presents the optimization problem, the Lagrangian relaxation of the multiperiod model, and the heuristic to solve the original problem. Section IV evaluates the proposed heuristic against baseline approaches. Finally, Section V provides concluding remarks.

II. Related Work

Placement of Services in Edge Computing: The placement of services in edge computing has been studied from various perspectives including increasing user throughput, reducing delay, or reducing energy consumed by end devices [10], [14]–[19]. We discuss the most relevant work next. Gao *et al.* [14] optimized the service placement in ENs such that the total delay experienced by users is minimized, given the mobility patterns of users. They propose an iterative framework that decomposes the problem into individual mixed-integer programming subproblems that are then solved using a commercial optimization solver. Sun *et al.* [15] optimized the placement of services considering the mobility of users, and aim to minimize user-experienced delay while constraining the energy consumed by each end-device. They propose a solution based on Lyapunov optimization and multiarmed bandit framework. However, they do not consider the energy consumption of ENs, and focus on energy-efficiency of end-devices. In contrast, our work aims to minimize the energy consumed by ENs, while constraining the average latency experienced by end users. Badri *et al.* [16] aimed to place applications on ENs based on the mobility patterns of end users. They identify the energy consumption of ENs as an important factor in placement decisions. Accordingly, they formulate an optimization problem that includes a fixed energy budget for ENs in the constraints. In contrast, we aim to minimize the total energy consumed by all ENs. The literature described above focuses on user-centric metrics, which, thus, require considering mobility patterns of users. In contrast, we focus on application-specific latency metrics as typically defined by cloud (or edge) provider service-level objectives [10], [20]. Farhadi *et al.* [17] considered the service placement and scheduling of requests to the ENs with the objective of maximizing the number of served requests in each time slot. The formulation does not consider the energy costs or latency experienced by users. The authors develop a two-stage approach to solve the placement and scheduling problem in each time slot, but leave the multiperiod formulation to future work. In contrast, our formulation considers the impact of placed services over multiple time periods, and aims to keep the average latency of requests below application-specific thresholds. Poularakis *et al.* [21] aimed to place services and route requests to maximize the number of requests served by ENs (i.e., equivalently minimize the number of requests sent to the cloud) while meeting resource utilization constraints at ENs. Their formulation does not consider the impact on latency. The problem of maximizing the number of requests is a variation of the knapsack problem, where items have to be placed into knapsacks (bins) of fixed capacity. In contrast, our formulation aims to minimize the number of running ENs, and is a variation of the bin packing problem [22]. Nguyen *et al.* [10] proposed an application provider-centric optimization problem for placing individual services on ENs. Their objective is to minimize the total cost of the application provider for deploying a single application. However, this does not include the energy costs of running applications on the EN. In contrast, we aim to consolidate load on the running ENs to minimize the number of ENs that run *multiple* applications. Recently, reinforcement learning (RL)-based approaches have been proposed to place services and to schedule requests. Sami *et al.* [23] proposed a deep RL approach to place services on ENs with the objective of minimizing the number of ENs, number of services that are not placed, number of requests that are not served, and the distances between ENs and users. However, their work does not consider the scheduling of requests to ENs once the services are placed, and the cost of the solution fluctuates from the optimal in a few cases. Hao *et al.* [24] proposed a deep RL-based approach to service placement and workload scheduling such that latency of service requests is minimized. However, they do not consider the energy consumed by ENs. The RL-based

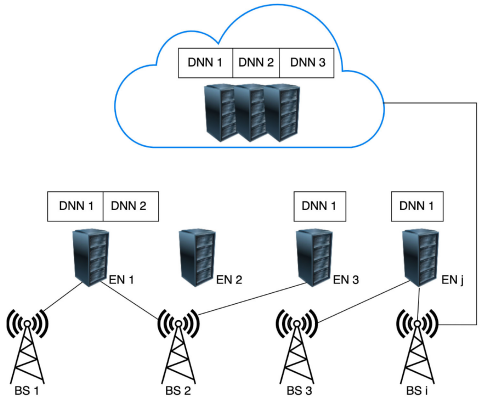


Fig. 1. System model.

approaches aim to place services and schedule requests in a single time period and in an online manner, whereas we focus on optimizing the placement across multiple time periods assuming that the demand can be predicted ahead of time.

Energy-Efficient Data Centers: The consolidation of workloads [e.g., running virtual machines (VMs)] to minimize the energy consumed by cloud data centers has been studied extensively [25], [26]. Dai *et al.* [27] proposed an optimization problem to minimize the total energy consumed by servers running VMs. Each VM must be assigned to only one server and VMs assigned to a server must not exceed the capacity constraints. The authors propose two greedy approximation algorithms to solve the problem. Zhou *et al.* [26] surveyed several state-of-the-art algorithms to consolidate and migrate VMs on cloud servers to minimize energy consumption. They propose a modified version of the best-fit-descending algorithm for bin packing. The VMs are long-running applications that span multiple time periods. The surveyed articles do not consider the scheduling of requests to shared VMs nor the latency of serving requests to users. Thus, the existing solutions cannot be directly applied to the energy-efficient placement of services on ENs.

III. Energy-Efficient Placement of Services in Edge Nodes

The target scenario (Fig. 1) comprises an edge (or cloud) service provider that has deployed a set of ENs to provide computational resources (e.g., GPUs) for DNN inference. The DNN models are pretrained and can be run either in the cloud or on an EN. End devices are connected to the nearest base station, and devices generate requests for DNN inference. The ENs are located close to end devices, and thus, are able to run inference and return the output with very low latency. The cloud provides computational resources for DNN inference, albeit, with a higher network latency. Edge service providers define a service-level objective that specifies the average latency for obtaining a response from a DNN model [8], [20]. To ensure that ENs can serve requests within this time limit, the appropriate DNN model(s) must be loaded in their memory and ready to serve any incoming requests. However, loading DNN models in memory increases the idle power of GPUs and can result in significant wasted energy when the number of incoming requests is low [28]. To this end, this article proposes an optimization problem that determines a *placement plan* that describes the DNN models that must be loaded at ENs during *future* time periods based on known demand patterns. Such a plan ensures that an edge service provider can meet the strict latency requirements of tasks while at the same time minimizing the energy consumed by the edge infrastructure.

A. System Model

Let $\mathcal{J}$ denote the set of ENs, and $\mathcal{S}$ denote the set of pretrained DNN models. Each EN is assumed to be equipped with a single GPU. Given the (predicted or average) demand in each future time slot t, the objective is to determine an optimal placement of DNN models in each EN such that energy is minimized and stringent latency constraints are met. The duration of each time slot t is equal to Δ seconds and there are T time slots in total. Several DNN models can be served on a single EN, and models can be cached in GPU memory to reduce expensive load operations and minimize latency [20]. Accordingly, decision variable $x_{s,j,t}$ indicates whether the DNN model s is loaded on EN j in time slot t. End devices submit their DNN inference requests through a set of base stations $\mathcal{I}$, and the demand ($d_{i,s,t}$) for inference exhibit variation over time [29]. For instance, vehicular applications may show different diurnal patterns than those for health-monitoring applications. Also, there may be regional variations due to the location of the actual base stations. The requests from each base station can be assigned to an EN j or to the cloud. Accordingly, $\mathcal{N}$ denotes the set of all compute nodes, including both the ENs in $\mathcal{J}$ ($j \in \{1, 2, 3, \ldots, |\mathcal{J}|\}$) and the cloud ($\{0\}$). The latency between the base stations and each compute node is represented by $t_{i,n}$, and is obtained as an average based on regular measurements. Variable $w_{i,s,n,t}$ indicates the fraction of total requests from base station i for model s that are assigned to compute node n in time slot t. This variable is known as a shadow scheduling variable; although this does not necessarily need to be used for actual scheduling, it is crucial to evaluate the placement of models in the compute nodes [17]. Finally, edge service providers define a target latency threshold (L_s) for each model s to indicate that the requests for this model must be served on average within L_s milliseconds.

B. Optimization Problem

We now present our service placement optimization problem called SPO. Table I summarizes the notation used in SPO. The objective is to minimize the energy consumed by the ENs while at the same time ensuring that the requests for each model s are served within the target latency thresholds (L_s). ENs are equipped with GPUs that are not energy-proportional, i.e., they consume a large portion of energy even if not serving any demand [28]. Thus, significant savings should be possible by switching off under-utilized GPUs and consolidating the inference tasks on a few ENs. Accordingly,

TABLE I
SUMMARY OF NOTATION IN THE OPTIMIZATION PROBLEM

Sym.	Description
$\mathcal{J}$	Set of edge nodes
$\mathcal{N}$	Set of all compute nodes, including edge nodes and cloud
$\mathcal{S}$	Set of DNN models
$\mathcal{I}$	Set of base stations
Δ	Duration of time slot in seconds
C_j	Capacity of edge node j
cl	Cost of loading a model in GPU memory
cd	Cost of downloading a model
$d_{i,s,t}$	Demand in terms of number of requests for DNN model s from BS i in time slot t
f_j	Energy cost to switch on an EN j
g_j	Operational cost (in terms of energy) to run inference tasks on an EN j
h_s	Cost of running inference with model s on the cloud
m_s	Memory (in MB) required to load model s
Mem_j	Memory capacity (in MB) of EN j
$t_{i,n}$	Network latency between BS i and compute node n
p_s	Cost for model s exceeding the average latency by 1 ms
L_s	Expected average latency (in ms) for inference with model s
l_s	Time (in ms) required to load model s weights in GPU memory
R_s	Number of ENs on which model s must be loaded
v_s	Giga operations required per inference request for DNN model s
k_s	Size of the input data in request for DNN model s
$\gamma_{s,t}$	Number of milliseconds by which average latency for DNN model s is exceeded in time slot t
$x_{s,j,t}$	Binary variable indicating whether DNN model s is loaded in memory of EN j in time slot t
$ON_{s,j,t}$	Binary variable indicating whether DNN model s is loaded for the first time in the memory of EN j in time slot t
$w_{i,s,j,t}$	Fraction of requests from BS i for model s scheduled to node j in time slot t
$z_{j,t}$	Binary variable indicating whether EN j is used to host any DNN model in time slot t
$\delta_{i,s,j,t}$	Continuous variable that represents product of $ON_{s,j,t}$ and $w_{i,s,j,t}$

a higher cost (f_j) is set to switch on an EN, and an operational cost (g_j) captures the increase in energy consumed as more demand is assigned to it. However, to avoid solutions where all requests are assigned to the cloud, it is also necessary to include a cost for scheduling requests to the cloud, denoted by h_s. This represents the additional overhead in sending requests to the cloud. Next, there is a cost p_s associated with exceeding the average expected latency for each model ($\gamma_{s,t}$). Finally, there are energy costs associated with loading the model in GPU memory, as well as downloading the model for the first time at EN j. These costs are denoted by *cl* and *cd*, respectively, where *cl* represents the cost for loading a model in an EN's memory [20], [28], whereas *cd* represents the cost associated with downloading the model at the beginning of a time slot, denoted by $\text{ON}_{s,j,t}$. Note that *cd* is typically higher than *cl* as it includes the overhead involved in transferring the model's weights from memory to GPU [20] and downloading the model weights from a different EN or the cloud. On the other hand, there is no extra cost with loading the model in the cloud as all DNN models are assumed to exist in the cloud. Finally, there is no cost associated with removing a model from EN j. Thus, the final objective function is as follows:

$$\min \sum_t \sum_j f_j z_{j,t} + \sum_t \sum_j \sum_i \sum_s g_j \cdot \frac{v_s d_{i,s,t} w_{i,s,j,t}}{C_j} + \sum_t \sum_i \sum_s h_s w_{i,s,0,t} + \sum_t \sum_s p_s \gamma_{s,t} + \sum_t \sum_s \sum_j cl x_{s,j,t} + \sum_t \sum_s \sum_j cd \text{ON}_{s,j,t}. \quad (1)$$

The constraints are as follows.

1) *Loading Constraint:* Binary variable ON_{sj} represents whether model s is loaded for the first time on EN j in time slot t, i.e., when the model was not already loaded in the previous time slot ($x_{s,j,t-1}$)

$$-x_{s,j,t-1} + x_{s,j,t} - \text{ON}_{s,j,t} \leq 0 \quad \forall j \in \mathcal{J}, s, t. \quad (2)$$

2) *Scheduling Constraints:* The demand for model s can be assigned to EN j only if the model is loaded, where $M_1 = |\mathcal{I}|$

$$\sum_i w_{i,s,j,t} \leq M_1 x_{s,j,t} \quad \forall j \in \mathcal{J}, s, t. \quad (3)$$

Next, all incoming requests must be scheduled to an EN or to the cloud

$$\sum_j w_{i,s,j,t} d_{i,s,t} + w_{i,s,0,t} d_{i,s,t} = d_{i,s,t} \quad \forall i, s, t. \quad (4)$$

3) An EN j is required to be on in time slot t if any model s is loaded on it, where $M_2 = |\mathcal{S}|$

$$\sum_s x_{s,j,t} \leq M_2 z_{j,t} \quad \forall j \in \mathcal{J}, t. \quad (5)$$

4) *Capacity Constraints:* The requests being served by EN j cannot exceed its compute or memory capacity. The computational requirements of DNN inference are characterized by giga operations per input byte [30], [31]; accordingly, v_s indicates the number of giga operations required per incoming request. Parameter C_j denotes the maximum number of giga operations per second that a GPU can support. The capacity is restricted to 70% of the total available compute, as is common in data center servers, where the utilization should never exceed 70% [32]

$$\sum_s v_s \sum_i d_{i,s,t} w_{i,s,j,t} \leq 0.7 C_j \quad \forall j \in \mathcal{J}, t. \quad (6)$$

Next, the memory required to load a DNN model and serve the scheduled requests cannot exceed the memory capacity of EN j (denoted by Mem_j). Here, we consider the memory required for loading the model [20] and limit the memory allocated to loaded models to 70% (7). The input data may be potentially large for certain image or video-based applications that require inference to be run on high-resolution data. Thus, the total memory, including that required to load the input data in user requests, is restricted to 95% of the total memory capacity (8)

$$\sum_s m_s x_{s,j,t} \leq 0.7 \ \text{Mem}_j \quad \forall j \in \mathcal{J}, t \quad (7)$$

$$\sum_s m_s x_{s,j,t} + \frac{\sum_i \sum_s w_{i,s,j,t} d_{i,s,t} k_s}{\Delta} \leq 0.95\ \text{Mem}_j \quad \forall j \in \mathcal{J}, t. \tag{8}$$

5) *Latency Constraint:* The requests for each model s must be served on average within a certain threshold, L_s, in each time slot t. First, to quantify the impact of loading the model on average latency [20], variable $\delta_{i,s,j,t}$ is introduced to represent the fraction of requests that are served by model s on EN j in time slot t if the model is loaded in that particular time slot. In particular, $\delta_{i,s,j,t}$ represents the product of $\text{ON}_{s,j,t}$ and $w_{i,s,j,t}$, i.e., the product of a binary and continuous variable. Accordingly, $\delta_{i,s,j,t}$ is linearized through

$$\delta_{i,s,j,t} \leq w_{i,s,j,t} \quad \forall j \in \mathcal{J}, i, s, t \tag{9}$$
$$\delta_{i,s,j,t} \leq \text{ON}_{s,j,t} \quad \forall j \in \mathcal{J}, i, s, t \tag{10}$$
$$\delta_{i,s,j,t} \geq w_{i,s,j,t} - \left(1 - \text{ON}_{s,j,t}\right) \quad \forall j \in \mathcal{J}, i, s, t \tag{11}$$
$$\delta_{i,s,j,t} \geq 0 \quad \forall j \in \mathcal{J}, i, s, t. \tag{12}$$

The average latency for model s includes the time required to load the memory weights on EN j and the time required to send the requests from BS i to EN j or compute node 0 (representing the cloud), where $t_{i,n}$ represents the latency between the BS i and the compute node n. The parameter $\gamma_{s,t}$ represents the slack variable, that is, penalized in the objective function (i.e., if the latency threshold L_s is exceeded)

$$\frac{\sum_i \sum_j \delta_{i,s,j,t} d_{i,s,t} l_s}{\Delta \sum_i d_{i,s,t}} + \frac{\sum_i t_{i,0} w_{i,s,0,t} d_{i,s,t}}{\sum_i d_{i,s,t}} + \frac{\sum_i \sum_j t_{i,j} w_{i,s,j,t} d_{i,s,t}}{\sum_i d_{i,s,t}} \leq L_s + \gamma_{s,t} \quad \forall s, t. \tag{13}$$

6) *Resilience Constraint:* Each application must be hosted on at least R_s ENs for resilience. This is required so that edge service providers can provide reliable services even if an EN fails [10].

$$\sum_j x_{s,j,t} \geq R_s \quad \forall s, t. \tag{14}$$

7) *Constraints on Variables:* Finally, the following constraints specify valid ranges for the variables:

$$\gamma_{s,t} \geq 0 \quad \forall s, t \tag{15}$$
$$0 \leq w_{i,s,n,t} \leq 1 \quad \forall i, s, n, t \tag{16}$$
$$x_{s,j,t} \in \{0, 1\} \quad \forall j \in \mathcal{J}, s, t \tag{17}$$
$$\text{ON}_{s,j,t} \in \{0, 1\} \quad \forall j \in \mathcal{J}, s, t \tag{18}$$
$$z_{j,t} \in \{0, 1\} \quad \forall j \in \mathcal{J}, t. \tag{19}$$

The single-period version of SPO is related to the well-known *bin packing problem* [22] and *distributed caching problem* [33], [34] with additional constraints. The objective of a general bin packing problem is to pack items of different volumes into a minimum number of capacitated bins. In addition, our formulation aims to meet the service latency thresholds when assigning items to bins. Moreover, certain items must be packed on R_s or more bins. On the other hand, the objective of the distributed caching problem is to minimize storage costs and access latencies. Both sets of problems are known to be NP-complete [22], [33]. Our formulation also incorporates the effect of placing DNN models across multiple time periods; specifically, the items placed in each time period also depend on the items placed in the previous time period (2). However, linking the time periods increases the difficulty of the problem. In the next section, we exploit the structure of the problem by utilizing a Lagrangian relaxation to decompose the problem and obtain a lower bound.

C. Lagrangian Relaxation

Lagrangian relaxation is a well-known relaxation method to obtain a bound on difficult optimization problems [35]. It has been widely used in vehicle routing [36], water pump scheduling [37], and more recently in orchestration of services in edge networks [38]. In our formulation, constraint (2) links the time periods and complicates the problem. Relaxing this constraint decomposes the original problem into smaller problems that can be solved individually. Accordingly, we apply a Lagrangian relaxation to constraint (2) with multiplier μ, to obtain the following the subproblem $Z_{LR}(\mu)$:

$$\begin{aligned} \min\ & \sum_t \sum_j f_j z_{j,t} + \sum_t \sum_j \sum_i \sum_s g_j \cdot \frac{v_s d_{i,s,t} w_{i,s,j,t}}{C_j} \\ & + \sum_t \sum_i \sum_s h_s w_{i,s,0,t} + \sum_t \sum_s p_s \gamma_{s,t} \\ & + \sum_s \sum_j \sum_t c l x_{s,j,t} + \sum_s \sum_j \sum_t c d \text{ON}_{s,j,t} \\ & + \sum_s \sum_j \sum_t \mu_{s,j,t}\left(-x_{s,j,t-1} + x_{s,j,t} - \text{ON}_{s,j,t}\right) \quad (20) \\ \text{s.t.}\ & (3)-(19) \\ & \mu_{s,j,t} \geq 0. \quad (21) \end{aligned}$$

The terms in the objective function can be rearranged to group the variables based on the time period t. For example, if there are three time periods, the terms related to the Lagrangian multiplier μ are as follows:

$$\begin{aligned} \sum_{s,j} & \mu_{s,j,1}\left(-x_{s,j,0} + x_{s,j,1} - \text{ON}_{s,j,1}\right) \\ & + \mu_{s,j,2}\left(-x_{s,j,1} + x_{s,j,2} - \text{ON}_{s,j,2}\right) \\ & + \mu_{s,j,3}\left(-x_{s,j,2} + x_{s,j,3} - \text{ON}_{s,j,3}\right). \end{aligned}$$

By gathering terms according to the time period t, we obtain

$$\begin{aligned} \sum_{s,j} & -\mu_{s,j,1} x_{s,j,0} + x_{s,j,1}\left(\mu_{s,j,1} - \mu_{s,j,2}\right) - \mu_{s,j,1}\text{ON}_{s,j,1} \\ & + x_{s,j,2}\left(\mu_{s,j,2} - \mu_{s,j,3}\right) - \mu_{s,j,2}\text{ON}_{s,j,2} + x_{s,j,3}\mu_{s,j,3} \\ & - \mu_{s,j,3}\text{ON}_{s,j,3}. \end{aligned}$$

Since no DNN models are placed initially, the first term containing $x_{s,j,0}$ can be removed, and the remaining terms are grouped by time slot t. Accordingly, in the general case, the problem $Z_{LR}(\mu)$ can be decomposed into T subproblems for each time period t, represented as $Z_t(\mu)$ where $Z_t(\mu)$ is defined

as follows:

$$\begin{aligned}\min\ & \sum_j f_j z_{j,t} + \sum_j \sum_i \sum_s g_j \cdot \frac{v_s d_{i,s,t} w_{i,s,j,t}}{C_j} \\ & + \sum_i \sum_s h_s w_{i,s,0,t} + \sum_s p_s \gamma_{s,t} \\ & + \sum_s \sum_j \big(cl - \mu_{s,j,t+1} + \mu_{s,j,t}\big) x_{s,j,t} \\ & + \sum_s \sum_j \big(cd - \mu_{s,j,t}\big) \mathrm{ON}_{s,j,t} \\ \text{s.t.}\ & (3)-(19).\end{aligned}$$

Each subproblem $Z_t(\mu)$ can be solved independently as the time periods are no longer linked. The solution of each subproblem is a vector $[z^h_{j,t}, x^h_{s,j,t}, w^h_{i,s,j,t}, \mathrm{ON}^h_{s,j,t}, w^h_{i,s,0,t}, \gamma^h_{s,t}]_{h\in\mathcal{H}_t}$, where $\mathcal{H}_t$ is the set of indices of the feasible solutions. The objective function of $Z_t(\mu)$ can be rewritten as

$$\begin{aligned}\min_{h\in\mathcal{H}_t}\ & \sum_j f_j z^h_{j,t} + \sum_j \sum_i \sum_s g_j \cdot \frac{v_s d_{i,s,t} w^h_{i,s,j,t}}{C_j} \\ & + \sum_i \sum_s h_s w^h_{i,s,0,t} + \sum_s p_s \gamma^h_{s,t} \\ & + \sum_s \sum_j \big(cl - \mu_{s,j,t+1} + \mu_{s,j,t}\big) x_{s,j,t} \\ & + \sum_s \sum_j \big(cd - \mu_{s,j,t}\big) \mathrm{ON}_{s,j,t}. \end{aligned} \tag{22}$$

$\sum_t Z_t(\mu)$ gives a lower bound on the optimal objective function of the original problem, for a given set of values for μ. The Lagrangian bound is the best lower bound, that is, given by the following problem:

$$\begin{aligned}&\max_\mu \sum_t Z_t(\mu) \\ &= \max_\mu \sum_t \min_{h\in\mathcal{H}_t} \Bigg(\sum_j f_j z^h_{j,t} + \sum_j \sum_i \sum_s g_j \cdot \frac{v_s d_{i,s,t} w^h_{i,s,j,t}}{C_j} \\ &\qquad + \sum_i \sum_s h_s w^h_{i,s,0,t} + \sum_s p_s \gamma^h_{s,t} \\ &\qquad + \sum_s \sum_j \big(cl - \mu_{s,j,t+1} + \mu_{s,j,t}\big) x^h_{s,j,t} \\ &\qquad + \sum_s \sum_j \big(cd + \mu_{s,j,t}\big) \mathrm{ON}^h_{s,j,t} \Bigg).\end{aligned}$$

Before defining the Lagrangian master problem Z_{MP}, we set θ_t as follows:

$$\begin{aligned}\theta_t \coloneqq \min_{h\in\mathcal{H}_t} \Bigg(& \sum_j f_j z^h_{j,t} + \sum_j \sum_i \sum_s g_j \cdot \frac{v_s d_{i,s,t} w^h_{i,s,j,t}}{C_j} \\ & + \sum_i \sum_s h_s w^h_{i,s,0,t} + \sum_s p_s \gamma^h_{s,t} \\ & + \sum_s \sum_j \big(cl - \mu_{s,j,t+1} + \mu_{s,j,t}\big) x^h_{s,j,t} \\ & + \sum_s \sum_j \big(cd + \mu_{s,j,t}\big) \mathrm{ON}^h_{s,j,t} \Bigg).\end{aligned}$$

Then, the master problem, Z_{MP}, is defined as follows:

$$\begin{aligned}\max\ & \sum_t \theta_t \\ \text{s.t.}\ & \theta_t \le \Bigg(\sum_j f_j z^h_{j,t} + \sum_j \sum_i \sum_s g_j \cdot \frac{v_s d_{i,s,t} w^h_{i,s,j,t}}{C_j} \\ & \qquad + \sum_i \sum_s h_s w^h_{i,s,0,t} + \sum_s p_s \gamma^h_{s,t} + \sum_s \sum_j cl x^h_{s,j,t} \\ & \qquad + \big(\mu_{s,j,t} - \mu_{s,j,t+1}\big) x^h_{s,j,t} + \sum_s \sum_j cd \mathrm{ON}^h_{s,j,t} \\ & \qquad - \mu_{s,j,t} \mathrm{ON}^h_{s,j,t} \Bigg) \quad \forall h \in \mathcal{H}_t \quad \forall t \end{aligned} \tag{23}$$

$$\mu_{s,j,t} \ge 0 \quad \forall j \in \mathcal{J}, s, t. \tag{24}$$

Since the feasible solutions are not known in advance, similar to [37] and [36], we use an iterative algorithm to solve the master problem, starting with an empty set $\mathcal{H}_t$. In each iteration, the subproblems $Z_t(\mu)$ are solved given fixed values of the Lagrangian multipliers $\mu_{s,j,t}$ to obtain a new feasible solution $h \in \mathcal{H}_t$. This solution is used to generate a cut of the form (23) that is added to the master problem Z_{MP}. The master problem is then solved to obtain optimal values of the Lagrangian multipliers $\mu_{s,j,t}$ that are used to solve the subproblems in the next iteration. In each iteration, $\sum_t Z_t(\mu)$ gives a lower bound on the best Lagrangian bound, whereas the solution of Z_{MP} provides an upper bound. The algorithm proceeds iteratively until the gap between the lower and upper bound is within 0.1%. The resulting Lagrangian bound is a lower bound on the original optimization problem SPO.

In addition to exploiting the Lagrangian relaxation to decompose the problem, one can take advantage of the structure of the subproblem to solve it more efficiently. The subproblem described above $Z_t(\mu)$ exhibits a structure that can be exploited using a Benders decomposition approach. We apply a standard Benders decomposition approach to the subproblem where the subproblem is now composed of a master and a cut generation linear problem. All continuous variables are projected out and we keep only the binary variables in the master problem. At each step of the iterative approach, the master is first solved to optimality. A cut generation problem is then constructed using the solution of master and from its solution, cuts for the master are derived (Benders cuts herein). This process is repeated until no cuts are found by the cut generation problem. The theorem of Benders guarantees that at this point the original problem is solved to optimality. In this work, we use the automatic Benders decomposition implemented in the commercial solver IBM CPLEX [39] to solve each $Z_t(\mu)$ to optimality. The automatic Benders strategy of CPLEX will have the master problem decide on the allocation of DNN models to ENs for different time slots, i.e., $x_{s,j,t}$ and $z_{j,t}$, while the subproblem will decide on the fraction and the value of latency being exceeded, i.e., $w_{i,s,j,t}$ and $\gamma_{s,t}$.

TABLE II
APPLICATION AND DNN MODEL CHARACTERISTICS

Application	v_s	l_s	m_s	k_s	L_s
Video	3	15	1192	0.4	50
Compute	433	10	1100	1.0	20
AR	8	15	1320	0.6	20
Vehicular	8	2	1240	0.3	40

D. Heuristic: One-Step Ahead

We present a heuristic called `One-step ahead` to efficiently solve the multiple-period model `SPO`. The heuristic solves the complete problem by obtaining solutions for each time period sequentially with a commercial solver such as CPLEX. The key intuition behind the heuristic is that once an optimal placement has been determined for time period $t-1$, the solution for time period t can be obtained by modifying constraint (2) based on the known values of $x_{s,j,t-1}$. Specifically, in a given time period t, if model s is placed on EN j in the previous time period (i.e., $x_{s,j,t-1}$ is 1), $\text{ON}_{s,j,t}$ is set to 0, implying there is no download cost or impact on latency if the DNN model is kept or removed in time period t. Alternatively, if model s is not placed on EN j in the previous time period $t-1$, $\text{ON}_{s,j,t}$ is set to the value of variable $x_{s,j,t}$. This implies that if the model s is loaded in time period t ($x_{s,j,t}=1$), it would be the first time the model is loaded and consequently $\text{ON}_{s,j,t}$ is set to 1. In the first time period ($t=0$), we assume no models were placed previously and consequently $x_{s,j,t-1}=0\ \forall s,j$. By doing so, we obtain solutions for each time period quickly, while still efficiently solving for the multiple-period model `SPO`. Implementation details are provided in the next section.

IV. EVALUATION

This section presents the results from evaluating our proposed solution `One-step ahead` in different types of edge networks. Our solution provides a placement plan that defines the DNN models to be loaded in the ENs over the next 24-h time period; assuming that the average demand is known in 15-min intervals for the next day. We evaluate the total cost of implementing the obtained placement plans and compare it to different baseline approaches. We first experiment with networks having random topologies and with varying levels of randomly generated demand (Section IV-A). As a second step, we also present results (Section IV-B) for a realistic network using traces from a cloud service provider [40]. In both cases, we consider a combination of four different categories of IoT applications that require inference to be carried out on device-generated data within different latency thresholds. Video, AR, and vehicular applications rely on DNN inference on video streams from cameras, AR headsets, and cars, respectively. The compute-intensive applications represent object detection on video streams using a more computationally heavyweight and accurate DNN model [30]. The DNN models fall into one of the four application categories, and have different compute requirements, memory requirements, input size, and loading costs (see Table II). The computation and memory requirements for different DNN models are obtained from [20] and [30] rounded up from decimal values to the nearest integer values.

In all experiments, we evaluate the performance of the solution obtained from our heuristic, `One-step ahead`. The solutions from `Lagrangian without Benders` and `Lagrangian with Benders`, the Lagrangian relaxation and the Lagrangian relaxation using Benders decomposition for the subproblem, respectively, provide a lower bound on the original multiperiod optimization problem. This allows us to assess the quality of the solution obtained using `One-step ahead`. We solve the individual optimization problems with CPLEX (version 20.1) through its Python API on a server with an Intel Xeon Silver 4114 CPU. The source code for the optimization and evaluation is available at https://github.com/gpremsan/energy-efficient-opt-placement.

Baseline Methods: We compare the performance of the solutions obtained from `One-step ahead` with the following two approaches.

`Greedy-by-Capacity` aims to load DNN models and assign demand to the ENs such that the number of ENs is minimized. It is based on a popular offline approximation algorithm for bin packing [22]. In this algorithm, the models are first sorted in order of increasing latency thresholds, so as to fit demand for latency-critical DNN models first. The demand for each model is assigned to the EN with smallest residual capacity and having sufficient capacity to load the model in memory. Similar to `SPO`, we limit the compute utilization of all ENs to 70% and the memory utilization to 95%. `Greedy-by-capacity` is a heuristic solution and provides an upper bound on the optimal solution.

`Greedy-by-Latency` aims to reduce the latency experienced by user requests for latency-critical applications. The algorithm sequentially considers DNN models sorted by increasing latency thresholds, and assigns the demand for the model from each base station to the nearest EN. The demand is assigned as long as neither the compute nor memory capacity of the EN is exceeded (limited to 70% and 95%, respectively). `Greedy-by-latency` is a heuristic and provides an upper bound on the optimal solution.

Metrics: We evaluate the performance of the different approaches in terms of the cost of implementing the placement plans (1). The *energy costs* of running an EN include a large cost (f_j) if the EN is running without serving any demand [28], followed by a cost (g_j) that increases proportionally to the demand served. Additionally, there are energy costs associated with loading and downloading DNN models. The *total cost* of the placement plan includes the energy costs, as well as costs for assigning demand to the cloud, and for the average latency exceeding the latency threshold. The values of the cost parameters f_j, g_j, h_s, p_s, cl, and cd are set to 1000, 350, 200, 200, 1, and 1, respectively. We also evaluate how many ENs are open in each time period, how the utilization of the ENs vary, and how the DNN models are loaded and unloaded from the ENs. We report the time required to obtain a solution with `One-step ahead`, `Lagrangian with Benders` and `Lagrangian without Benders` for all networks. Note that the

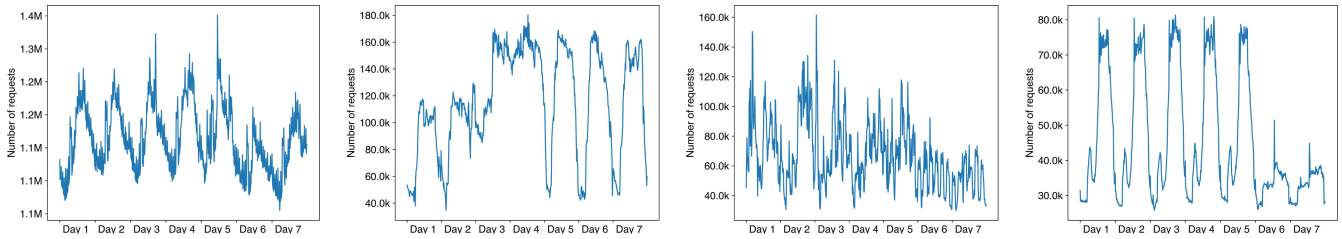


Fig. 2. Number of requests per 15-min interval for (a) video, (b) compute, (c) AR, and (d) vehicular applications.

TABLE III
NETWORK CHARACTERISTICS

Setting	Demand	C_j
1	U(0,1000)	U(10000,15000)
2	U(1000,2000)	U(10000,15000)
3	U(0,1000)	U(5000,10000)
4	U(1000,2000)	U(5000,10000)
5	U(0,1000)	U(20000,25000)
6	U(1000,2000)	U(20000,25000)

baseline approaches, namely, `Greedy-by-capacity` and `Greedy-by-latency`, have very short running times that are not reported.

A. Computational Results

We first experiment with networks having random topologies and demand drawn from a uniform random distribution. There are three categories of networks, namely, small, medium and large, depending on the number of BSs, ENs and DNN models. Small networks comprise 10 BSs, 5 ENs, and 5 DNN models, medium networks comprise 18 BSs, 10 ENs, and 13 DNN models, and finally, large networks comprise 25 BSs, 15 ENs, and 20 DNN models. Under each category, we experiment with six different settings of demand and compute capacity (Table III). Accordingly, networks 1–6 correspond to small networks, networks 7–12 to medium networks, and networks 13–18 to large networks. In each setting, U(min,max) indicates that the average demand in each 15-min interval for each DNN model from each base station is drawn from a uniform distribution within the range (min,max). All ENs have a fixed memory capacity of 32 GB. Each DNN model is randomly assigned to one of the four application categories (Table II). Finally, each BS is assumed to be close to at least two ENs, i.e., the latency is drawn from U(10,15) ms, whereas the remaining ENs can be accessed with an average latency of U(25,50) ms. The average latency between a BS and the cloud is drawn from U(100,120) ms.

The goal of this section is to evaluate the performance of `One-step ahead` as compared to other baseline approaches, and whether its objective value is close to the lower bound provided by the Lagrangian relaxation, which allows us to assess the quality of the solution provided by `One-step ahead`. Table IV presents the results for the 18 considered networks. We observe that `One-step ahead` achieves the lowest cost in all cases, with a value, that is, between 53% and 89% lower than the next best approach `Greedy-by-capacity`. Note that the solution for the complete multiperiod problem is not included, as the relative gap in CPLEX even for the smallest network was still 7% after 5 h of running time. Nevertheless, the Lagrangian relaxation provides a lower bound on the complete optimization problem. We observe that `One-step ahead` achieves a total cost, that is, close to the Lagrangian relaxation in all networks (with an average gap of 1.9%). Given the promising performance of the `One-step ahead`, we further examine the results of this approach in a realistic network with real demand patterns in the next section.

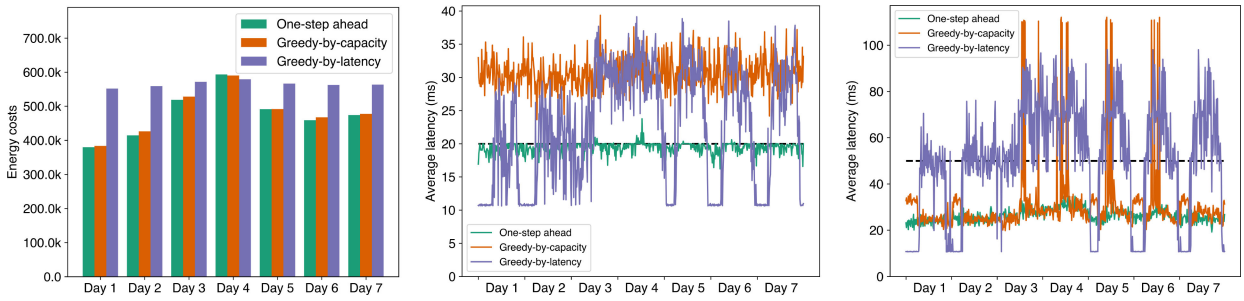
B. Trace-Driven Results

This section describes the performance of our proposed `One-step ahead` approach with real workload traces from Microsoft Azure's Function as a Service platform [40]. The dataset comprises the number of invocations of serverless functions in 1-min intervals. We use this dataset to represent the invocations of DNN models due to the lack of public datasets for DNN inference workloads. Nevertheless, the dataset is representative of DNN workloads as serving DNN models at the edge using such a Function as a Service paradigm has been explored extensively [41]–[43]. Accordingly, we use the data from the first seven days and consider the top 20 functions by the number of median invocations. Next, the demand is split randomly between each base station, with the assumption that there are groups of busy and moderately busy base stations. This demand ($d_{i,s,t}$) is used as input to our solution. The functions are mapped to one of the four application categories presented in Table II, and Fig. 2 shows the average number of requests in 15-min intervals per application. The resilience (R_s) is set to 2 for the latency-critical AR and compute applications, i.e., these applications must be loaded on at least two ENs. The network comprises of 20 DNN models, 5 ENs, and 20 base stations. The latency distribution is similar to that in the previous section. All ENs have a computation capacity of 22 000 GOPS and memory capacity of 32 GB, which are in line with the hardware used for DNN inference [31].

Overall Results: Table V shows the total cost of the placement plans obtained by `One-step ahead`, `Greedy-by-capacity`, and `Greedy-by-latency` and the time taken to obtain the plan for each day. First, `One-step ahead` is able to find solutions within a short time and under seven minutes in the worst case. Next, we observe that `One-step ahead` determines placement plans

TABLE IV
PERFORMANCE EVALUATION FOR RANDOMLY GENERATED NETWORKS

	Greedy-by-latency	Greedy-by-capacity	One-step ahead		Lagrangian (with Benders)		Lagrangian (without Benders)	
Network	UB	UB	UB	Time	LB	Time	LB	Time
1	490 009.72	417 304.95	195 467.35	18	187 035.56	87	187 061.27	2911
2	504 872.43	505 293.16	209 622.54	21	209 352.69	83	209 395.06	1167
3	496 288.70	425 232.43	203 904.40	21	191 221.23	142	191 240.06	947
4	523 494.96	596 748.47	222 251.80	20	222 107.92	198	222 225.77	1154
5	486 431.48	413 190.35	192 346.26	21	184 069.17	86	184 114.01	769
6	494 276.17	425 518.94	205 639.28	21	201 312.34	66	201 340.35	697
7	911 578.96	1 597 967.93	325 257.65	171	321 209.26	9067	319 710.71	18 270
8	986 359.38	2 337 598.18	496 334.51	357	485 004.36	18 397	493 116.18	18 112
9	937 913.17	2 091 487.44	345 888.47	147	342 519.16	5732	341 445.26	18 236
10	8 376 453.68	2 699 927.68	854 537.51	599	852 078.06	19 251	846 816.51	18 414
11	895 220.10	1 451 470.84	309 883.29	151	306 324.94	18 099	305 531.69	18 140
12	935 412.36	1 696 320.50	347 070.09	223	346 271.57	3549	345 308.40	18 187
13	1 263 538.43	3 148 716.40	468 060.72	1763	—	—	—	—
14	13 235 923.09	4 670 235.91	1 278 416.50	3208	—	—	—	—
15	4 065 322.50	3 840 354.41	663 916.63	6577	—	—	—	—
16	26 615 518.51	31 314 474.32	3 335 555.72	1283	—	—	—	—
17	1 222 723.01	2 863 749.73	360 693.79	698	—	—	—	—
18	3 780 326.29	3 520 103.67	718 782.67	4085	—	—	—	—



(a) (b) (c)

Fig. 3. (a) Energy costs. Average latency for DNN models used in (b) compute and (c) video applications.

TABLE V
PERFORMANCE EVALUATION FOR TRACE-DRIVEN RESULTS

	Greedy-latency	Greedy-capacity	One-step ahead	
Day	Total cost	Total cost	Total cost	Time
1	4 513 719.40	2 073 381.60	382 915.29	203
2	6 383 555.38	2 021 602.11	422 064.67	255
3	11 193 442.50	3 619 752.44	538 187.16	305
4	14 171 177.51	3 919 459.62	626 144.76	411
5	9 661 269.99	3 653 876.90	506 801.72	295
6	8 758 828.77	3 167 598.94	476 186.34	306
7	8 750 251.26	2 322 052.85	481 940.36	299

with a lower cost than other approaches. This trend is similar to that observed in the previous section. However, if we take a look at the energy costs [Fig. 3(a)], we observe that `One-step ahead` is able to achieve costs that are similar to `Greedy-by-capacity` that packs services (DNN models) on the fewest number of ENs. Thus, both approaches are able to achieve a high utilization of ENs while keeping energy costs low. On the other hand, `Greedy-by-latency` results in running more ENs and thus, the energy cost is higher than other approaches. On day 4, we observe that the energy costs are similar for all methods as they load DNN models on all five ENs. The energy costs for `One-step ahead` are slightly higher than `Greedy-by-capacity` as `One-step ahead` loads DNN models for latency-critical applications on multiple ENs so as to lower the average latency. Fig. 3(b) and (c) shows the average latency achieved by the three approaches over all seven days for compute and video applications. Overall, `One-step ahead` is able to maintain the average latency within the configured threshold (denoted by the dashed horizontal line) on most days. In particular, for compute-intensive applications, `One-step ahead` exceeds the configured threshold only in four time periods, whereas `Greedy-by-capacity` exceeds the latency at all times. For video applications, `One-step ahead` maintains the average latency below the threshold at all times, whereas `Greedy-by-capacity` exceeds the threshold 23 times. Even on days with high demand, `One-step ahead` is able to keep the average latency within 5 ms of the configured threshold, whereas the latency experienced by other solutions are much higher (even up to 20 ms above the threshold). We do not include the plots for AR and vehicular applications as the trend is similar to that presented for compute and video applications. Thus, the total cost is higher for `Greedy-by-capacity` and `Greedy-by-latency` as they are not able to maintain the average latency below application-specific thresholds. Finally, we examine the utilization of the ENs during the duration of the simulation run. Fig. 5 shows the cumulative distribution function (CDF) of the utilization of the ENs achieved in each 15-min interval during the simulation run. As expected, `Greedy-by-latency`

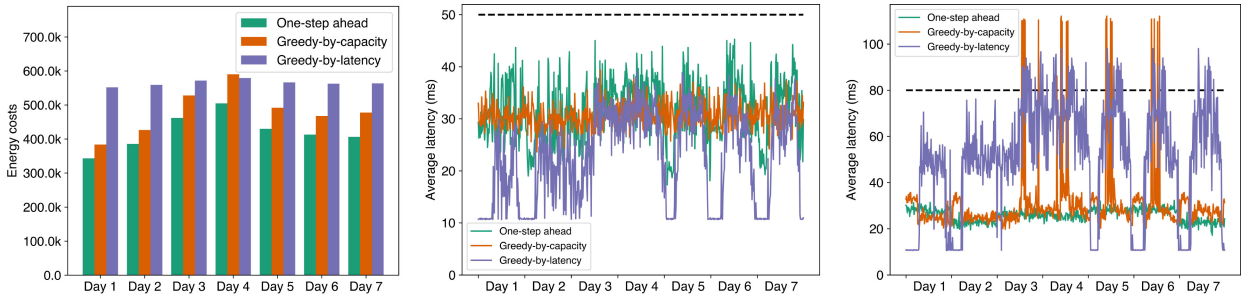


Fig. 4. Results from increasing latency thresholds. (a) Energy costs. Average latency for DNN models used in (b) compute and (c) video applications.

TABLE VI
PERFORMANCE EVALUATION FOR TRACE-DRIVEN RESULTS WITH INCREASED LATENCY THRESHOLDS

	Greedy-latency	Greedy-capacity	One-step ahead	
Day	Total cost	Total cost	Total cost	Time
1	2 200 262.70	406 132.58	369 279.57	117
2	2 992 108.58	451 731.10	410 862.82	123
3	5 462 382.58	1 554 947.33	512 597.53	150
4	7 010 342.54	1 640 843.83	580 108.88	167
5	4 837 887.85	1 435 522.95	482 331.42	146
6	4 351 836.75	1 101 098.88	454 491.14	163
7	4 347 175.75	519 091.72	458 372.45	173

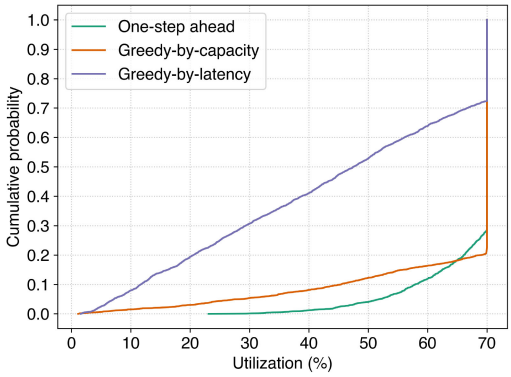


Fig. 5. Empirical CDF of ENs' utilization in 15-min intervals across the simulated seven days.

keeps several ENs running with a low utilization (median value of 47.5%). `One-step ahead` is able to keep the utilization above 60% in the vast majority of the time periods, i.e., in over 87% of the time periods. `Greedy-by-capacity` achieves a high utilization of ENs, however, there are a few instances (3%) where the utilization of ENs is below 20%. `One-step ahead` avoids such cases, and the lowest observed utilization is 23%. This indicates that `One-step ahead` is able to keep the ENs running at high utilization, thereby serving a large number of user requests when running.

Varying Latency Threshold: Next, we experiment with increasing the latency threshold by 30 ms for all application groups (Table VI) while keeping the network and demand the same as before. Here, on days 1 and 2, both `Greedy-by-capacity` and `One-step ahead` are able to achieve a placement plan with low overall costs. However, on other days (days 3–6) when the overall demand is higher, `One-step ahead` outperforms `Greedy-by-capacity` by keeping the average latency low. Fig. 4(b) and (c) shows that `One-step ahead` keeps the average latency below the threshold on all days, including the fourth day when the demand is high. On the other hand, `Greedy-by-capacity` is able to keep the average latency low for the compute applications, whereas the average latency for video applications exceeds the threshold by at least 10 ms when the total demand for all DNN models is high (days 5–7). We also observe that `One-step ahead` is able to lower the energy costs [Fig. 4(a)] as compared to `Greedy-by-capacity`, although both open similar number of ENs and keep the average utilization of ENs close to 70%. This is because `One-step ahead` reduces the number of times the DNN models are loaded and unloaded throughout the day. For example, for a single day in the evaluation, `One-step ahead` loads DNN models at the beginning of a 15-min period 80 times during the day whereas `Greedy-by-capacity` loads the DNN models 252 times during the day. Thus, `One-step ahead` keeps the model loaded in memory as long as the placement is able to meet the latency thresholds. This saves energy by reducing the need to frequently download and load DNN models in memory. Finally, the utilization of ENs is similar to that observed in Fig. 5. `One-step ahead` keeps the ENs running with a median utilization of 70% with a lowest observed utilization of 32%. Although the median utilization is 70% with `Greedy-by-capacity`, the utilization is below 30% in 5% of the cases. `Greedy-by-latency` achieves a median utilization of only 47% as it aims to minimize latency of served requests. Again, `One-step ahead` ensures that the running ENs serve requests with a high utilization, while balancing the requirements of both latency and energy costs.

Varying Resilience Requirement: Next, we experiment with setting different values for R_s in `One-step ahead`. This value affects the number of ENs each DNN model must be placed in, and thus, the number of running ENs. As expected, increasing the R_s value results in increasing the overall cost. However, even with R_s set to 4 for each DNN model, the overall cost is still lower than other approaches with a maximum total cost of 633 476.85 on any single day (day 4). This is 83.8% lower than the total cost of

`Greedy-by-capacity` and 95.5% lower than the total cost obtained with `Greedy-by-latency` on the same day. Thus, both the placement of DNN models and fraction of requests assigned to each model are crucial in ensuring that the placement plan is able to both save energy and meet a target average latency.

Summary and Discussion: Overall, the solutions obtained by `One-step ahead` are able to minimize the total cost of placing DNN models in ENs by balancing the requirements of both low energy consumption as well as an average response latency below a target threshold. Following the daily patterns of the workload trace [40], the number of running ENs varies between 2 and 4 on days 1 and 2 and between 2 and 5 on days 3 and days 5–7. On day 4, all five ENs run throughout the day. This demonstrates that having a placement plan can enable network operators to save energy based on the predicted traffic patterns by proactively placing DNN models and minimizing the number of running ENs. Moreover, by considering the models loaded in the previous time period, `One-step ahead` decides when to keep the same model loaded in memory instead of loading it on a new EN. Such a placement plan is able to account for geographic variations in traffic patterns. However, when the latency requirements are very strict, the energy consumption may slightly increase as `One-step ahead` loads the models on more ENs. Apart from the actual placement of DNN models, the fraction of requests assigned to each EN also play a crucial role in lowering average latency. For instance, `Greedy-by-capacity` aims to minimize the number of ENs that are open, but this results in an increased average latency. `Greedy-by-latency` results in high energy consumption as more ENs are open than the other two approaches. Despite this, the average latency exceeds the threshold in many cases, indicating that the placement of the models themselves and the fraction of requests that are assigned to the cloud or EN are important factors. Finally, this article focused on the energy-efficient placement of DNN models in edge computing, using parameters (Table II) that apply to DNN inference. Nevertheless, our formulation is general enough to support the placement of services that follow a request–response pattern, simply by changing the values of the parameters.

V. Conclusion

This article considered the optimal placement of DNN models for AI applications on ENs such that the energy consumption of the ENs is minimized and the latency of serving requests is low. We proposed a novel multiperiod optimization problem that takes into account the impact of placing services across time periods, the limited capacity of ENs, and a target average latency for serving requests. Additionally, we utilized a Lagrangian relaxation to decompose the multiperiod problem and thereby, obtained a lower bound on the optimal solution. We proposed a heuristic that efficiently obtains solutions that are able to simultaneously save energy while meeting strict latency constraints. Extensive evaluations showed that our heuristic achieved placement plans for DNN models at a lower cost than other baseline approaches across different network instances. As part of future work, we plan to include the prediction of demand into our formulation and consider the impact of uncertainty in the demand patterns on service placement and scheduling. Comparing such an approach with recent deep RL-based approaches that operate in an online manner is also an interesting direction for future work. Finally, it will be interesting to consider dependencies of DNN inference on additional CPU-based computation and storage services that also contribute to the energy consumed by AI applications at the edge.

Acknowledgment

The authors are extremely grateful to the reviewers and the Editor for their constructive comments which have contributed to significant improvements to the manuscript.

References

[1] G. Premsankar, M. Di Francesco, and T. Taleb, "Edge computing for the Internet of Things: A case study," *IEEE Internet Things J.*, vol. 5, no. 2, pp. 1275–1284, Apr. 2018.

[2] E. Li, L. Zeng, Z. Zhou, and X. Chen, "Edge AI: On-demand accelerating deep neural network inference via edge computing," *IEEE Trans. Wireless Commun.*, vol. 19, no. 1, pp. 447–457, Jan. 2020.

[3] J. Chen and X. Ran, "Deep learning with edge computing: A review," *Proc. IEEE*, vol. 107, no. 8, pp. 1655–1674, Aug. 2019.

[4] B. Jedari, G. Premsankar, G. Illahi, M. Di Francesco, A. Mehrabi, and A. Ylä-Jääski, "Video caching, analytics, and delivery at the wireless edge: A survey and future directions," *IEEE Commun. Surveys Tuts.*, vol. 23, no. 1, pp. 431–471, 1st Quart., 2020.

[5] Y. Tian, L. Njilla, J. Yuan, and S. Yu, "Low-latency privacy-preserving outsourcing of deep neural network inference," *IEEE Internet Things J.*, vol. 8, no. 5, pp. 3300–3309, Mar. 2021.

[6] Z. Chang, S. Liu, X. Xiong, Z. Cai, and G. Tu, "A survey of recent advances in edge-computing-powered artificial intelligence of things," *IEEE Internet Things J.*, vol. 8, no. 18, pp. 13849–13875, Sep. 2021.

[7] E. Chung *et al.*, "Serving DNNs in real time at datacenter scale with project brainwave," *IEEE Micro*, vol. 38, no. 2, pp. 8–20, Mar./Apr. 2018.

[8] H. Liu *et al.*, "JIZHI: A fast and cost-effective model-as-a-service system for Web-scale online inference at Baidu," 2021, *arXiv:2106.01674*.

[9] S. Kekki *et al.*, "MEC in 5G networks," vol. 28, White Papar, ETSI, Sophia Antipolis, France,, 2018.

[10] D. T. Nguyen, H. T. Nguyen, N. Trieu, and V. K. Bhargava, "Two-stage robust edge service placement and sizing under demand uncertainty," *IEEE Internet Things J.*, vol. 9, no. 2, pp. 1560–1574, Jan. 2022.

[11] B. Knowles, *ACM TechBrief: Computing and Climate Change*. New York, NY, USA: Assoc. Comput. Mach., 2021. [Online]. Available: https://dl.acm.org/doi/book/10.1145/3483410. doi: 10.1145/3483410.

[12] M. Xu *et al.*, "From cloud to edge: A first look at public edge platforms," 2021, *arXiv:2109.03395*.

[13] J. A. Ayala-Romero, A. Garcia-Saavedra, X. Costa-Perez, and G. Iosifidis, "EdgeBOL: Automating energy-savings for mobile edge AI," in *Proc. 17th Int. Conf. Emerg. Netw. Exp. Technol.*, 2021, pp. 397–410.

[14] B. Gao, Z. Zhou, F. Liu, and F. Xu, "Winning at the starting line: Joint network selection and service placement for mobile edge computing," in *Proc. IEEE INFOCOM*, 2019, pp. 1459–1467.

[15] Y. Sun, S. Zhou, and J. Xu, "EMM: Energy-aware mobility management for mobile edge computing in ultra dense networks," *IEEE J. Sel. Areas Commun.*, vol. 35, no. 11, pp. 2637–2646, Nov. 2017.

[16] H. Badri, T. Bahreini, D. Grosu, and K. Yang, "Energy-aware application placement in mobile edge computing: A stochastic optimization approach," *IEEE Trans. Parallel Distrib. Syst.*, vol. 31, no. 4, pp. 909–922, Apr. 2020.

[17] V. Farhadi *et al.*, "Service placement and request scheduling for data-intensive applications in edge clouds," *IEEE/ACM Trans. Netw.*, vol. 29, no. 2, pp. 779–792, Apr. 2021.

[18] V. Farhadi *et al.*, "Service placement and request scheduling for data-intensive applications in edge clouds," in *Proc. IEEE INFOCOM*, 2019, pp. 1279–1287.

[19] T. He, H. Khamfroush, S. Wang, T. L. Porta, and S. Stein, "It's hard to share: Joint service placement and request scheduling in edge clouds with sharable and non-sharable resources," in *Proc. IEEE ICDCS*, 2018, pp. 365–375.
[20] A. Gujarati *et al.*, "Serving DNNs like clockwork: Performance predictability from the bottom up," in *Proc. 14th USENIX Symp. Oper. Syst. Design Implement. (OSDI)*, Nov. 2020, pp. 443–462.
[21] K. Poularakis, J. Llorca, A. M. Tulino, I. Taylor, and L. Tassiulas, "Joint service placement and request routing in multi-cell mobile edge computing networks," in *Proc. IEEE INFOCOM*, 2019, pp. 10–18.
[22] S. Martello and P. Toth, "Knapsack problems: Algorithms and computer implementations," in *Interscience Series in Discrete Mathematics and Optimization*. New York, NY, USA: Wiley, 1990.
[23] H. Sami, A. Mourad, H. Otrok, and J. Bentahar, "Demand-driven deep reinforcement learning for scalable fog and service placement," *IEEE Trans. Services Comput.*, early access, Apr. 27, 2021, doi: 10.1109/TSC.2021.3075988.
[24] Y. Hao, M. Chen, H. Gharavi, Y. Zhang, and K. Hwang, "Deep reinforcement learning for edge service placement in softwarized industrial cyber-physical system," *IEEE Trans. Ind. Informat.*, vol. 17, no. 8, pp. 5552–5561, Aug. 2021.
[25] A. Verma, G. Dasgupta, T. K. Nayak, P. De, and R. Kothari, "Server workload analysis for power minimization using consolidation," in *Proc. Conf. USENIX Annu. Tech. Conf.*, 2009, p. 28.
[26] Q. Zhou *et al.*, "Energy efficient algorithms based on VM consolidation for cloud computing: Comparisons and evaluations," in *Proc. 20th IEEE/ACM Int. Symp. Clust. Cloud Internet Comput. (CCGRID)*, 2020, pp. 489–498.
[27] X. Dai, J. M. Wang, and B. Bensaou, "Energy-efficient virtual machines scheduling in multi-tenant data centers," *IEEE Trans. Cloud Comput.*, vol. 4, no. 2, pp. 210–221, Apr.–Jun. 2015.
[28] A. Jahanshahi, H. Z. Sabzi, C. Lau, and D. Wong, "GPU-NEST: Characterizing energy efficiency of multi-GPU inference servers," *IEEE Comput. Archit. Lett.*, vol. 19, no. 2, pp. 139–142, Jul.–Dec. 2020.
[29] Q. Weng *et al.*, "MLaaS in the wild: Workload analysis and scheduling in large-scale heterogeneous GPU clusters," in *Proc. 19th USENIX Symp. Netw. Syst. Design Implement. (NSDI)*, Apr. 2022, pp. 1–16.
[30] V. J. Reddi *et al.*, "MLPerf inference benchmark," in *Proc. ACM/IEEE 47th Annu. Int. Symp. Comput. Archit. (ISCA)*, 2020, pp. 446–459.
[31] M. Blott, L. Halder, M. Leeser, and L. Doyle, "QuTiBench: Benchmarking neural networks on heterogeneous hardware," *ACM J. Emerg. Technol. Comput. Syst.*, vol. 15, no. 4, pp. 1–38, 2019.
[32] D. Chou *et al.*, "Taiji: Managing global user traffic for large-scale Internet services at the edge," in *Proc. 27th ACM Symp. Oper. Syst. Principles*, 2019, pp. 430–446.
[33] I. Baev, R. Rajaraman, and C. Swamy, "Approximation algorithms for data placement problems," *SIAM J. Comput.*, vol. 38, no. 4, pp. 1411–1429, 2008.
[34] N. Golrezaei, K. Shanmugam, A. G. Dimakis, A. F. Molisch, and G. Caire, "Wireless video content delivery through coded distributed caching," in *Proc. IEEE Int. Conf. Commun. (ICC)*, 2012, pp. 2467–2472.
[35] M. L. Fisher, "The Lagrangian relaxation method for solving integer programming problems," *Manag. Sci.*, vol. 50, pp. 1861–1871, Dec. 2004.
[36] B. Lin, B. Ghaddar, and J. Nathwani, "Electric vehicle routing with charging/discharging under time-variant electricity prices," *Transp. Res. C Emerg. Technol.*, vol. 130, Sep. 2021, Art. no. 103285.
[37] B. Ghaddar, J. Naoum-Sawaya, A. Kishimoto, N. Taheri, and B. Eck, "A Lagrangian decomposition approach for the pump scheduling problem in water networks," *Eur. J. Oper. Res.*, vol. 241, no. 2, pp. 490–501, 2015.
[38] Y. Liu *et al.*, "A Lagrangian relaxation based approach for service function chain dynamic orchestration for the Internet of Things," *IEEE Internet Things J.*, vol. 8, no. 23, pp. 17071–17089, Dec. 2021.
[39] IBM. *ILOG CPLEX Optimization Studio: Bender's Algorithm*. [Online]. Available: https://www.ibm.com/docs/en/icos/20.1.0?topic=optimization-benders-algorithm (Accessed: Apr. 3, 2022).
[40] M. Shahrad *et al.*, "Serverless in the wild: Characterizing and optimizing the serverless workload at a large cloud provider," in *Proc. USENIX Annu. Tech. Conf. (USENIX ATC)*, 2020, pp. 205–218.
[41] T. Rausch, W. Hummer, V. Muthusamy, A. Rashed, and S. Dustdar, "Towards a serverless platform for edge AI," in *Proc. 2nd USENIX Workshop Hot Topics Edge Comput. (HotEdge)*, Renton, WA, USA, Jul. 2019. [Online]. Available: https://www.usenix.org/conference/hotedge19/presentation/rausch
[42] A. Bhattacharjee, A. D. Chhokra, Z. Kang, H. Sun, A. Gokhale, and G. Karsai, "Barista: Efficient and scalable serverless serving system for deep learning prediction services," in *Proc. IEEE Int. Conf. Cloud Eng. (IC2E)*, 2019, pp. 23–33.
[43] A. Ali, R. Pinciroli, F. Yan, and E. Smirni, "BATCH: Machine learning inference serving on serverless platforms with adaptive batching," in *Proc. IEEE Int. Conf. High Perform. Comput. Netw. Storage Anal. (SC)*, 2020, pp. 1–15.

Gopika Premsankar received the Ph.D. degree in computer science from Aalto University, Espoo, Finland, in 2020.

She was a Postdoctoral Researcher with Ivey Business School, London, ON, Canada. She is a Postdoctoral Researcher with the University of Helsinki, Helsinki, Finland, working on energy-efficient networked systems. Her research interests include edge computing, wireless communications, augmented reality, and energy-efficient networked systems. Her research has been supported by grants from the Academy of Finland and the Finnish Cultural Foundation.

Bissan Ghaddar received the Ph.D. degree in operations research from the University of Waterloo, Waterloo, ON, Canada, in 2011. She is a David G. Burgoyne Faculty Fellow, a Dean's Faculty Fellow, and an Associate Professor of Management Science and Sustainability with Ivey Business School, London, ON, Canada, working on problems at the intersection of smart cities, mathematical optimization, and machine learning. Prior to joining Ivey Business School, she was an Assistant Professor with the Department of Management Sciences, University of Waterloo, Waterloo, ON, Canada. She has also worked with IBM Research, Dublin, Ireland, and the Centre for Operational Research and Analysis, Department of National Defence Canada, Ottawa, ON, Canada. Her research has been supported by national and international grants, including NSERC, OCE, Cisco, H2020, and FP7 IIF European Union Grant.

Dr. Ghaddar currently serves as an Associate Editor for the *EURO Journal on Computational Optimization*.