Dynamic Offloading for Compute Adaptive Jobs

Agrim Bari*, Gustavo de Veciana*, Kerstin Johnsson †, Alexander Pyattaev ‡
*Department of Electrical and Computer Engineering, The University of Texas at Austin
†Intel Corporation, Santa-Clara, California
‡YL-Verkot OY, Tampere, Finland

***Abstract*—The increasing demands of computationally intensive device applications are driving advancements in edge technologies and the need for improved computation offloading policies. This paper focuses on "adaptive" offloading and computation, i.e., adapting the amount of offload data and, consequently, the associated compute adaptive job's quality to the wireless channel quality, network congestion, and the compute adaptive job's computational options. For example, when the channel quality is poor and a job has a tight deadline, the amount of offloaded data can be reduced in exchange for a loss in the associated compute adaptive job's quality. In this paper, we show the substantial advantages of adapting the amount of offloaded data to channel quality and network congestion. We begin by defining a reward model for compute adaptive jobs based on the amount of offloaded data and resulting computation quality. We then develop an upper bound on the achievable revenue rate and propose/compare various offloading policies: Greedy, Predictive Abandonment (PA), Probabilistic Admission Control and Layer Assignment (PACLA), and combinations thereof. We evaluate our policies via simulation and observe that the combination of PACLA + PA provides the best offloading performance for homogeneous or heterogeneous compute adaptive jobs for devices with varied channel qualities.**

***Index Terms*—Computation offloading, Upper bound.**

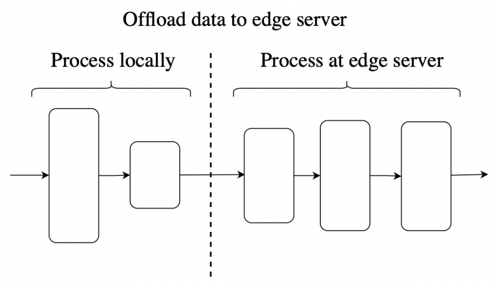
I. Introduction

Mobile Edge Computing (MEC): MEC holds the promise of augmenting the capabilities of devices, resulting in improved Quality of Service (QoS) and reduced cost/energy burdens for computationally-intensive applications [1], [2]. By leveraging processing on devices, edge and cloud, and wireless communication to transfer compute adaptive job-related input/output data among nodes, MEC can support compute adaptive jobs associated with AR/VR, computational perception, and new forms of personalization. To achieve this promise, one must address the central problem of how to jointly optimize applications and system design to make the most of available resources and adapt to system load, heterogeneity, and variability in wireless channel qualities.

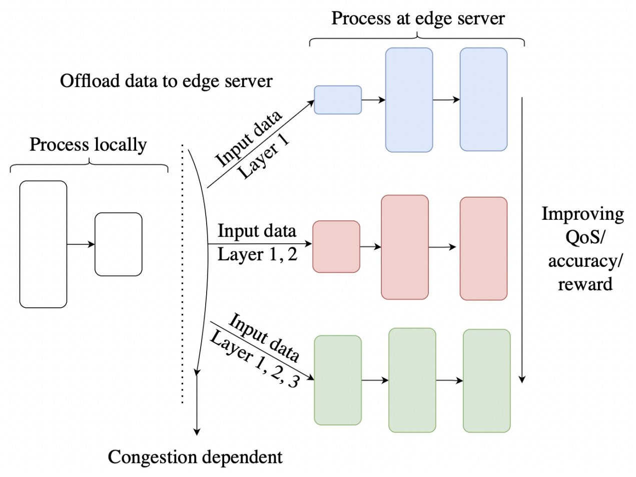
Adaptive offloading and computation: A central problem in optimizing the execution of applications consisting of several processing steps across MEC lies in judiciously deciding when/where to partition and offload them. Offloading requires sharing data over typically limited wireless resources interconnecting devices with edge/cloud compute servers (e.g., [1], [2], [3]). In this paper, we consider scenarios where application partitioning decisions have already been made, yet one can adapt the amount of data at the pre-selected partition point that gets offloaded to the user's current channel quality and/or network's overall congestion. Like the video-rate/quality adaptation found in scalable video coding, layered compression standards, etc., offloading policies that adapt the amount of offloaded data to changing network conditions in real-time are better equipped to deliver robust performance. To that end, we consider applications designed to support varying compute adaptive job models (and quality) based on the amount of input data that can be offloaded to the edge within a specified time budget. Input data is structured as a sequence of layers, $1, \ldots, L$, which an offloading policy will seek to progressively deliver to the edge, see e.g., [4]. Each additional layer improves the compute adaptive job's quality, which translates to increased user satisfaction and additional reward for the service provider.

Definition: A **layered family of compute adaptive models for a job** include: a) a shared computation model which results in output data $\mathcal{D}$, and b) a family of compatible computation models $1, 2 \ldots L$ depending on a sequence of nested data subsets, $\mathcal{D}_1 \subset \mathcal{D}_2 \subset \ldots \mathcal{D}_L = \mathcal{D}$ outputted by shared model – input for family of compatible models. We refer to $\mathcal{D}_1$ as the data required to execute the 1st model and $\mathcal{D}_l \setminus \mathcal{D}_{l-1}$ as the additional data corresponding to layer l and $d_l = |\mathcal{D}_l|$ the cumulative data required to execute the lth model. Successful execution of the shared model, combined with data offloading of $\mathcal{D}_l$ and execution of the compatible lth model, results in a compute adaptive job quality/reward α_l, which is assumed to be increasing in l.

Applicability of compute adaptive job model: There are several classes of applications that can be structured to allow for variable input data size and adaptive computing. Most notably, applications such as speech recognition, image classification, computational perception, etc., based on Deep Neural Networks (DNNs) can be designed in this manner. As shown in Fig. 1a, one can partition a DNN at various points, and depending on where this occurs, the amount of data that must be transmitted to the edge may vary. Determining the best partition point depends on the DNN requirements (deadline, processing size per node, etc), processing capabilities of the device vs. edge/cloud, DNN's input/output data sizes, wireless channel conditions, energy consumption for computation and communication, and the energy restrictions of device [5]. We assume that the optimal partition point has been pre-selected. However, the data offloading process (when multiple users are simultaneously offloading) may not be robust to the heterogeneity and variability in users' wireless channel capacity and network congestion. Thus, we explore a layered



(a) Partitioning of a DNN.



(b) Partitioning and Layered offloading of a DNN.

Figure 1: Example of adaptive computation for DNN.

approach to offloading the input data, adjusting the number of transmitted data layers to changing channel conditions and/or wireless network congestion. Fig. 1b, depicts how a DNN-based compute adaptive job's quality improves as more input data layers are offloaded within the time budget. A DNN-based family of compute adaptive models for a job can be trained/designed to support computation based on varying input data sizes. Thus adapting the number of input data layers offloaded selects different computation models and thus different quality of the result.

Remark 1:. Consider a DNN based family of compute adaptive models for a job trained to detect items in a picture. The family of models share the same initial architecture, which is executed locally. Thus local computation burden is fixed. However, we can adapt the amount of input data (output of shared model) that gets offloaded based on the transmission rate and network congestion. The detection accuracy depends on how many nested data layers we manage to offload in time to allow the edge to complete execution of the compatible model and return the result before the deadline.

Design objectives for offloading policies: The following are some of the key objectives/characteristics that we posit are most important in the design of layered offloading policies:

- **Edge-based:** able to make good offloading decisions with minimal state information from, coordination with, and changes to Base Station (BS) (or Access Point) schedulers.
- **Adapt to uncertainty:** able to adapt to heterogeneity and temporal variability (load, interference, contention) in wireless systems.
- **Delay constrained:** able to complete compute adaptive jobs within the heterogeneous time budgets associated with diverse applications.
- **Performance management:** able to achieve high system revenue rate/user satisfaction in systems shared by heterogeneous compute adaptive job types.

A. Related Work

Offloading: Computation offloading strategies have been extensively studied in the context of applications, such as image recognition (DNN based), augmented reality, and autonomous driving. In early studies related to offloading policies (e.g., [6], [7]), the focus is on deciding whether to execute a job entirely locally or on the edge server. Decisions were usually made based on server speed, server load, required input data, and channel quality. In particular, in [6], the authors attempt to develop a delay-optimal computation offloading policy using a Markov Decision Process (MDP) framework, but the proposed approach is limited since it does not consider time-varying channel. The authors of [7] also formulate the problem as an MDP and consider time-varying communication resources and offloading time budgets. However, their analysis focuses on a representative user with a time-varying wireless channel, but there is no consideration of possible network congestion.

DNN offloading: Recent studies (e.g., [8], [5], [9], [10], [11]) on DNN-based applications have explored the advantages of partial offloading, i.e., process a part of the job locally and the rest on edge server. However, they do not consider heterogeneity in channel quality, network congestion, and do not impose delay constraints. The authors of [4] consider varying channel capacity and delay constraint, but still, only consider settings where a single device is offloading at a given time instance.

Offload friendly DNNs: Another related research area and perspective lies in devising offload friendly DNNs (e.g., [12], [13], [14], [15], [16]), where the DNN structure is adjusted per the device's processing capabilities, time budget, and channel capacity (e.g., introduce a partition point which is composed of few nodes, in the early stages of the DNN structure, enabling in-network compression of input data and reducing the load on a device). However, for the most part, these studies ignore key aspects of a wireless network. For example, in [15], a modified DNN model is developed along with real-time offloading decisions but again there is no delay constraint on data transmission, and the focus is on a single user.

Video layered coding - rate adaptation: The idea of structuring jobs' input data into layers and progressively transmitting them as channel capacity allows, parallels layered coding for adaptive bit rate video streaming [17], [18]. Layered coding generates a base layer and additional enhancement layers. The base layer is necessary for decoding the media stream, whereas enhancement layers improve video quality. This framework allows one to prioritize uninterrupted video streaming at the cost of (a temporary) quality reduction in scenarios where network congestion or packet losses occur. The central idea underlying this paper is the design of adaptive applications that can operate with varying amounts of input data and permit one to perform dynamic offloading decisions of compute adaptive jobs in response to heterogeneous and time-varying channel capacity/congestion.

B. Contributions and Organization

In this paper, we study offloading policies for compute adaptive jobs that support adaptive computation based on the amount of offloaded data. Our focus is on stochastic settings with homogeneous, then heterogeneous compute adaptive jobs sharing wireless network resources, and our contributions are as follows.

We begin by deriving a simple universal upper bound on the revenue rate achievable by any offloading policy. This is motivated by [19]. The revenue rate captures the benefit derived from offloading compute adaptive jobs with possibly different amounts of input data, i.e., number of nested layers. We propose and compare several opportunistic data offloading policies for homogeneous compute adaptive job types and show the benefits of adaptive computation in terms of revenue rate and the fraction of completed offloads.

Our results show that data offloading policies that transmit input data layers until the compute adaptive job times out, e.g., Greedy, lead to throughput collapse as the rate of offered load grows, resulting in zero completion rate for offloads under heavy loads. By contrast, Predictive Abandonment (PA) based policies, that only transmit the next input data layer if it is estimated to arrive on time and otherwise quit, avoid throughput collapse and sustain a reasonable revenue rate even under high loads. We also investigate a Probabilistic Admission Control and Layer Assignment (PACLA) policy that predetermines the number of data layers to offload based on user channel quality and the overall load, channel qualities, etc. in the network. The combination of PACLA + Greedy significantly outperforms Greedy policy, and out/under-performs PA depending on scenario. Combining PACLA with PA always outperforms PACLA + Greedy and also outperforms PA.

In settings with heterogeneous compute adaptive jobs that have different rewards and different demands on the channel, PACLA is critical to maximizing total network revenue. We observe that PACLA + PA results in a high revenue rate and is robust to heavy loads, due to PACLA's ability to maximize revenue based on knowledge of the offered loads and channel/revenue models of different types of compute adaptive jobs in the system, while PA adjusts to congestion on the wireless network.

The paper is organized as follows: In Section II we introduce our basic system model. In Section III we explore different input data offloading policies and develop an upper bound on the overall system performance. We end the section with some simulation results. Finally Section V concludes the paper.

II. System Model

We begin by proposing a model for the study of offloading policies for homogeneous compute adaptive jobs with layered input data in systems with stochastic and heterogeneous wireless channels.

A. Compute adaptive job Model

A compute adaptive job may involve processing on the device, transmission of input data to the edge, processing on the edge, and then transmission of results back to the device. We assume a fixed time budget, τ, for the transmission of a compute adaptive job's input data to the edge, as this is typically the bottleneck [1]. It is derived from the job's overall delay budget once the time required to compute the job on the user + edge and return of results is accounted for. As introduced in Section I, our compute adaptive job can be executed on a range of nested data subsets. If we let $\mathcal{L} = \{1, 2, ..., L\}$ denote the discrete set of input layers – capturing data subsets, then d_l for $l \in \mathcal{L}$ denotes the *cumulative* input data associated with delivering Layers 1 through l. The reward α_l represents the quality of output when the compute adaptive job is computed using Layers 1 through l, where $\alpha_l = \alpha(d_l)$. We consider $\alpha(\cdot)$ concave, representing applications with diminishing incremental reward per input layer, see Fig. 2.

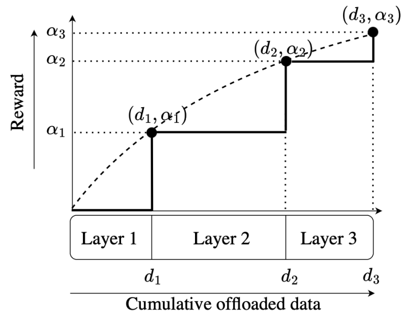


Figure 2: Reward versus cumulative offloaded data when $\alpha(.)$ is concave.

B. Model for load

Compute adaptive job offloads are initiated stochastically by a population of users. When a user initiates an offload, it is assumed to have one of a discrete set $\mathcal{C}$ of channel qualities/classes with a baseline uplink transmission rate r_c, where $c \in \mathcal{C}$, to its associated BS. The channel quality is assumed constant throughout the offload process. We assume each compute adaptive job is generated by a distinct user. We model the "arrivals" (i.e., initiations) of compute adaptive job offloads on type c channel as a stationary stochastic process with an average arrival rate of λ_c, and let $\boldsymbol{\lambda} = (\lambda_c, c \in \mathcal{C})$ denote a vector capturing the arrival rate of compute adaptive job offloads on each type of channel. The total arrival rate of compute adaptive job offloads is $\lambda = \sum_{c \in \mathcal{C}} \lambda_c$.

C. Model for shared base station resources

In our model, there is a set of users $\mathcal{U}(t)$ concurrently offloading compute adaptive jobs at time t, where $N(t) = |\mathcal{U}(t)|$ denotes the number of active users. Users are associated with the same BS, and share its uplink resources via a policy that mimics proportional fair scheduling, where a user with channel type c is served at a rate $r_c \frac{1}{N(t)}$. A user can abandon the offloading of a compute adaptive job at any time. The state of the system at time t includes the set of active offloads, $\mathcal{U}(t)$, and for each offload $u \in \mathcal{U}(t)$, its channel class, c_u, the time elapsed since initiating offloading, $\mathrm{e}_u(t)$, the layer it is currently offloading, $\mathrm{y}_u(t)$, and the residual data, $\mathrm{s}_u(t)$, that needs to be offloaded to complete the delivery of the current layer, $\mathrm{y}_u(t)$.

D. Stationary ergodic offloading policies

Consider a set of stationary ergodic offloading policies, Π, including any combination of admission control and scheduling/resource allocation – where the latter may go beyond the proportionally fair policy introduced in our system model. For an offered load, $\boldsymbol{\lambda}$, under policy $\pi \in \Pi$, we define $\mathbf{q}(\boldsymbol{\lambda}, \pi) = (q_{c,l}(\boldsymbol{\lambda}, \pi) : c \in \mathcal{C}, l \in \mathcal{L})$, where $q_{c,l}(\boldsymbol{\lambda}, \pi)$ denotes the long term fraction of compute adaptive jobs with channel quality c that *successfully* offloaded up through input layer l within time budget τ. Since we assumed ergodicity and stationarity, long term fraction is well defined. We must have $\sum_l q_{c,l}(\boldsymbol{\lambda}, \pi) \le 1$ for all $c \in \mathcal{C}$, and note that this may be a strict inequality if a fraction of the offloads do not complete. We define $\mathcal{F}(\boldsymbol{\lambda}) = \{\mathbf{q}(\boldsymbol{\lambda}, \pi) \mid \pi \in \Pi\}$, as the set of *achievable* long term fractions. By time sharing, i.e., alternating over long periods, between two policies, say $\pi_1, \pi_2 \in \Pi$, it is easy to see that $\mathcal{F}(\boldsymbol{\lambda})$ is a convex set.

E. Revenue rate

We define the revenue rate for offered load $\boldsymbol{\lambda}$ under policy $\pi \in \Pi$ as:

$$\beta(\mathbf{q}(\boldsymbol{\lambda}, \pi), \boldsymbol{\lambda}) = \sum_{c \in \mathcal{C}} \lambda_c \sum_{l \in \mathcal{L}} \alpha_l q_{c,l}(\boldsymbol{\lambda}, \pi).$$

III. Homogeneous Compute Adaptive Jobs and Offloading policies

In this section, we study different offloading policies with the design objectives discussed in Section I in mind. We focus on networks with homogeneous compute adaptive jobs (i.e., compute adaptive jobs that have the same number of input layers and the same reward per number of layers delivered). Offloads may, however, have different channel qualities and get different shares of the channel depending on how many other offloads they have to share it with. We begin by developing an upper bound for the revenue rate of any stationary ergodic offloading policy.

Table I: Summary of Notation introduced

Notation	Description
$\mathcal{L}$	discrete set of input layers for a compute adaptive job
d_l	cumulative input data up through layer $l \in \mathcal{L}$
$\alpha(.)$	reward function based on cumulative offloaded data
τ	time budget
Π	set of stationary ergodic offloading policies
$\mathcal{C}$	discrete set of channel qualities/classes
r_c	baseline capacity (bits/sec) for channel of quality/class $c \in \mathcal{C}$
λ_c	arrival rate of compute offloads with channel c
λ	total arrival rate of compute offloads
$\boldsymbol{\lambda}$	vector capturing the arrival rate of compute offloads per channel type
$\mathcal{U}(t)$	set of users performing offloading at time t
$N(t)$	number of users performing offloading at time t
c_u	channel quality/class of user $u \in \mathcal{U}(t)$
$\mathrm{e}_u(t)$	time since user $u \in \mathcal{U}(t)$ initiated offloading
$\mathrm{y}_u(t)$	layer that user $u \in \mathcal{U}(t)$ is currently offloading
$s_u(t)$	residual data that user $u \in \mathcal{U}(t)$ needs to offload to deliver current layer $\mathrm{y}_u(t)$

A. Upper bound

We let $\mathbf{q} = (q_{c,l} : c \in \mathcal{C}, l \in \mathcal{L})$, where $q_{c,l}$ denotes the fraction of compute adaptive jobs with channel quality c that successfully offload up through layer l within the time budget. We capture the set of such possible vectors by

$$\mathcal{Q} = \{\mathbf{q} \mid \mathbf{q} \ge 0, \; \sum_{l \in \mathcal{L}} q_{c,l} \le 1, \; \forall c \in \mathcal{C}\}$$

where we note that in some settings, a fraction of compute adaptive jobs may not succeed, whence they need not sum to 1. For a given $\mathbf{q}$, we define the channel utilization by

$$\rho(\mathbf{q}) = \sum_{c \in \mathcal{C}} \lambda_c \sum_{l \in \mathcal{L}} q_{c,l} \frac{d_l}{r_c}.$$

Recall that we defined $\mathcal{F}(\boldsymbol{\lambda})$ to be the set of *achievable* long term fractions of offloads per channel type and number of input layers that complete on time when the load is $\boldsymbol{\lambda}$. Here we define the set of all *possible* successful long term fractions

$$\overline{\mathcal{F}}(\boldsymbol{\lambda}) = \{\mathbf{q} \mid \mathbf{q} \in \mathcal{Q} \text{ and } \rho(\mathbf{q}) \le 1\}$$

as a natural outer bound.

Theorem 1: *Given an offered load $\boldsymbol{\lambda}$ we have that $\mathcal{F}(\boldsymbol{\lambda}) \subseteq \overline{\mathcal{F}}(\boldsymbol{\lambda})$ and*

$$\beta^*(\boldsymbol{\lambda}) := \max_{\mathbf{q} \in \mathcal{F}(\boldsymbol{\lambda})} \beta(\mathbf{q}, \boldsymbol{\lambda}) \le \max_{\mathbf{q} \in \overline{\mathcal{F}}(\boldsymbol{\lambda})} \beta(\mathbf{q}, \boldsymbol{\lambda})$$

where $\beta^*(\boldsymbol{\lambda})$ denotes the maximum achievable revenue rate given an offered load $\boldsymbol{\lambda}$ by any stationary offloading policy.

Proof. We first argue that $\mathcal{F}(\boldsymbol{\lambda}) \subseteq \overline{\mathcal{F}}(\boldsymbol{\lambda})$. Indeed suppose $\mathbf{q} \in \mathcal{F}(\boldsymbol{\lambda})$. Recall that $q_{c,l}$ represents the fraction of offloaded compute adaptive jobs with channel type c that can be completed on time when offloading Layer 1 through l. Since each compute adaptive job is offloading till a particular layer l these fractions always sum to less than or equal to 1. Thus $\mathbf{q}$ is clearly in $\mathcal{Q}$. But suppose the load on the channel is greater than 1 given these fractions, i.e., $\rho(\mathbf{q}) > 1$. If this is true, then not all compute adaptive jobs will complete on time, contradicting our earlier statement. Thus, the channel load must be less than or equal to 1, i.e., $\rho(\mathbf{q}) \leq 1$ if $\mathbf{q} \in \mathcal{F}(\boldsymbol{\lambda})$, which implies $\mathbf{q} \in \overline{\mathcal{F}}(\boldsymbol{\lambda})$. Thus $\mathcal{F}(\boldsymbol{\lambda}) \subseteq \overline{\mathcal{F}}(\boldsymbol{\lambda})$. This then implies the following bound

$$\max_{\mathbf{q}\in\mathcal{F}(\boldsymbol{\lambda})} \beta(\mathbf{q}, \boldsymbol{\lambda}) \leq \max_{\mathbf{q}\in\overline{\mathcal{F}}(\boldsymbol{\lambda})} \beta(\mathbf{q}, \boldsymbol{\lambda}).$$

$\square$

Remark 2: We let $\mathbf{q}^* = \operatorname{argmax}_{\mathbf{q}\in\overline{\mathcal{F}}(\boldsymbol{\lambda})} \beta(\mathbf{q}, \boldsymbol{\lambda})$. Note that $\mathbf{q}^*$ may not actually be achievable yet it provides some guidance on the fraction of compute adapitve jobs to admit across wireless channels with different capacity and up to which layers offloading should take place.

B. Offloading policies

Greedy: Under the Greedy offloading policy, users progressively transmit input layers until the compute adaptive job's offloading time budget, τ, expires. The reward for a user is determined by the cumulative number of layers that were successfully delivered.

Predictive Abandonment (PA): The Predictive Abandonment (PA) policy adapts to changes in channel capacity and uplink congestion as the number of users trying to offload simultaneously varies. Under PA, a user, $u \in \mathcal{U}(t)$, continues to transmit input data only as long as the residual data associated with the current input layer, $\mathrm{y}_u(t)$, is estimated to be successfully delivered within the remaining time budget, $(\tau - \mathrm{e}_u(t))$. We estimate the time required to finish offloading of the current layer by dividing the bits remaining to be transmitted, $\mathrm{s}_u(t)$, by the throughput, $\mathrm{h}_u(t)$, seen by the user since it began offloading, t_u, where

$$\mathrm{h}_u(t) = \frac{1}{\mathrm{e}_u(t)} \sum_{t'=t_u}^{\mathrm{e}_u(t)+t_u} \frac{r_{c_u}}{N(t')}.$$

The user stops offloading input data if

$$\frac{\mathrm{s}_u(t)}{\mathrm{h}_u(t)} > (\tau - \mathrm{e}_u(t)).$$

This prediction method is crude, but roughly captures the average throughput seen by the user. It is user-oriented and does not require any coordination with BS. More accurate estimates could be made by considering the total number of users in the system during each user's offload, remaining service requirements of currently offloading compute adaptive jobs, and/or completed service requirements of offloaded compute adaptive jobs. See, [20] for a discussion of such considerations in a processor sharing scenario without layered input data and abandonment.

The PA policy effectively performs a sort of dynamic admission control, since it blocks a user from initiating offload it it estimates that the user would not be able to offload the first input layer within the time budget given the user's channel quality and the current number of co-channel users.

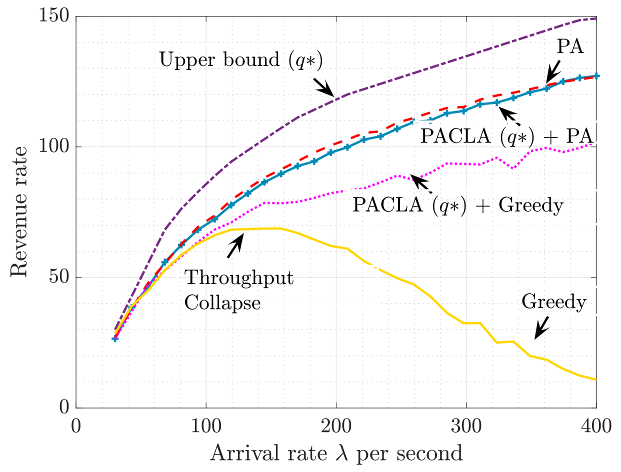
In the homogeneous scenario, where users' compute adaptive jobs have identical offloading time budgets and input layers, those with better channel qualities will not only transfer more data per second, they will also stay on the system longer, since they have a higher probability of estimating that their "next layer" will transfer within the remaining time budget. Thus, the PA policy is opportunistic in terms of channel quality, promoting more aggressive layer offloading when a user has good throughput. Both the Greedy and PA policies are decentralized and require no information about the overall system load $\boldsymbol{\lambda}$. Next, we consider policies that take the overall system load into account.

Probabilistic Admission Control and Layer Assignment (PACLA): PACLA pre-determines the probability, $p_{c,l}$, with which a user with channel quality c should attempt to offload its compute adaptive job up through layer l. We let $\mathbf{p} = (p_{c,l} : c \in \mathcal{C}, l \in \mathcal{L})$, where we require $\sum_l p_{c,l} \leq 1$ for all $c \in \mathcal{C}$ and denote the associated policy as PACLA $(\mathbf{p})$. Determining a good choice for $\mathbf{p}$ is challenging given the complex interplay among contending loads from sets of channel types, possible congestion on the network, delay constraint, and the revenue rate the offloading policy seeks to optimize. We propose to choose $\mathbf{p}$ based on the upper bounds in Theorem 1, i.e., we let $\mathbf{p} = \mathbf{q}^*$, the solution that would in principle optimize the revenue rate. These probabilities are determined based on knowledge of the overall system load, $\boldsymbol{\lambda}$, thus the policy should be viewed as relying on a form of centralized coordination. Since PACLA $(\mathbf{p})$ can vary its admission probabilities across users based on channel quality, it can be made opportunistic across users with different channel qualities.

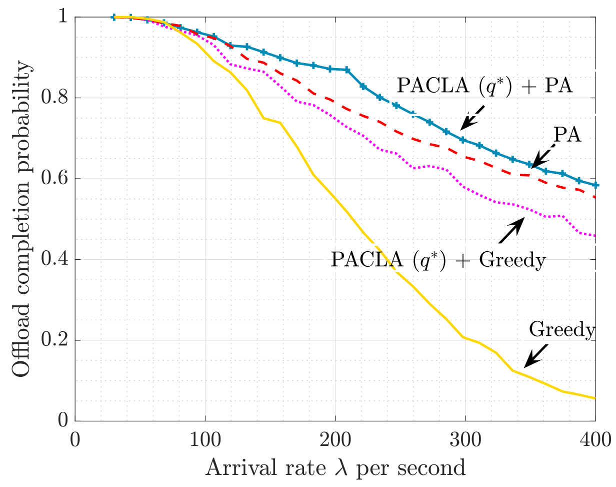
Users can then either greedily transmit their data up through their assigned layer l or expiration of their time budget, whichever comes first. Alternatively, users can use PA to avoid wasting channel resources when their current input layer is not estimated to arrive on time. We refer to the former policy as PACLA $(\mathbf{q}^*)$ + Greedy and the latter as PACLA $(\mathbf{q}^*)$ + PA.

C. Simulation results

In this subsection, we compare the different offloading policies discussed in the previous section. The overall compute adaptive job arrival process is modeled as a Poisson arrival process with intensity λ. We assume each compute adaptive job is initiated by a distinct user, and users have 1 of 3 equally likely channel qualities, i.e., $\lambda_c = \lambda/3$ for all c in $\mathcal{C}$. In our simulation, we perform a discrete-time simulation of each policy. The slot size is 100 µs. We simulate for 5e4 time slots and averaged over 100 Monte Carlo simulations. The simulation parameters are given in Table II, where b is a scale factor. We compare policies based on their achieved revenue rates and offload completion probability, which is defined as the fraction of offloads that transmit at least up through Layer 1 within time budget τ. Fig. 3 exhibits the revenue rates and offload completion probabilities generated by our proposed offloading policies as a function of offload rate λ, when the relationship between reward and number of successfully offloaded input layers is concave. In this case, $\alpha(d) = \sqrt{d/b}$, where b is a constant.



(a) Revenue rate from successful offloading.



(b) Offload completion probability.

Figure 3: Comparison of different offloading policies.

Table II: Simulation parameters

Parameter	Value	Units/Remarks
(d_1, d_2, d_3)	$b \cdot (1/3, 2/3, 1)$	bits
τ	1/30	sec
r_c	$b \cdot (50, 75, 100)$	bits/sec
$(\alpha_1, \alpha_2, \alpha_3)$	$(0.58, 0.82, 1)$	Concave

From Fig. 3a, we see that the Greedy policy suffers from throughput collapse as λ increases. By contrast, policies like PACLA that limit the load on the channel via admission control and PA that stop the transmission of input data when the current layer's offload is estimated to exceed the time budget, perform well even under high loads.

With a concave $\alpha(.)$, most of a compute adaptive job's reward is gained with the 1st input layer. As a result, a policy like PA, which reduces the number of input layers a user transmits as channel congestion increases, will perform quite well as the load grows. As Fig. 3 shows, PACLA + PA outperforms all the other policies in terms of revenue rate and the fraction of successful offloads. When $\lambda = 100$, its performance is within 82% of the (unachievable) upper bound developed in Theorem 1, suggesting that any additional revenue from some smarter scheduling and/or admission control would be limited.

In Fig. 4, we show the revenue rate and offload completion probability of the highest revenue rate-achieving policy (PACLA + PA) with and without the benefit of layered input data. We do this to emphasize the benefits of adaptive computation (i.e., the ability to adapt computation based on amount of offloaded data). In the figures, we refer to the case where input data is delivered in layers and compute adaptive jobs can be executed on anything from 1-3 layers as "Adaptive computation" while "No adaptive computation" refers to the case when there are no layers and all input data must be delivered to do the computation. As Fig. 4 shows, PACLA + PA improves significantly both with respect to revenue rate and offload completion probability when adaptive computation is supported.

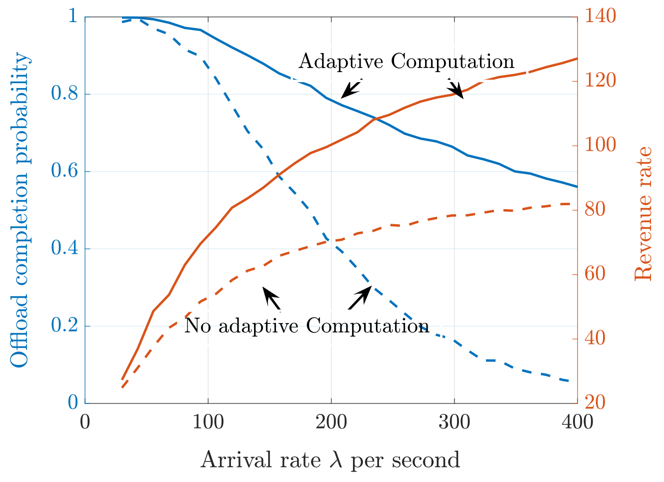


Figure 4: Benefits of adaptive computation - For PACLA + PA.

IV. Heterogeneous Compute Adaptive Jobs

In this section, we explore how different offloading policies perform when compute adaptive jobs are heterogeneous, i.e., when they no longer have the same deadline, number of layers, data size per layer, and reward per layer.

We let $\mathcal{J} = \{1, 2, ..., J\}$ denote the set of different compute adaptive job types and $\boldsymbol{\Lambda} = (\boldsymbol{\lambda}^1, ..., \boldsymbol{\lambda}^J)$ their arrival rates, where $\boldsymbol{\lambda}^j = (\lambda_c^j, c \in \mathcal{C})$ denotes a vector capturing the arrival rate of type j compute adaptive job per channel class. The total arrival rate of type j compute adaptive jobs is thus denoted by $\lambda^j = \sum_{c \in \mathcal{C}} \lambda_c^j$. We let τ^j denote the offloading time budget and $\mathcal{L}^j = \{1, ..., L^j\}$ denote the set of input layers for type j compute adaptive jobs.

For an offered load, $\mathbf{\Lambda}$, under policy $\pi \in \Pi$, we let $\mathbf{Q}(\mathbf{\Lambda}, \pi) = (\mathbf{q^1}(\mathbf{\Lambda}, \pi), ..., \mathbf{q^J}(\mathbf{\Lambda}, \pi))$, where $\mathbf{q^j}(\mathbf{\Lambda}, \pi) = (q^j_{c,l}(\mathbf{\Lambda}, \pi) : j \in \mathcal{J}, c \in \mathcal{C}, l \in \mathcal{L}^j)$, and $q^j_{c,l}(\mathbf{\Lambda}, \pi)$ denotes the long term fraction of type j compute adaptive jobs with channel quality c that successfully offload up through layer l within time budget τ^j. We must have $\sum_{l=1}^{L^j} q^j_{c,l}(\mathbf{\Lambda}, \pi) \leq 1 \ \forall j \in \mathcal{J}, \ \forall c \in \mathcal{C}$, where again this may be a strict inequality if a fraction of offloads do not complete. We define $\mathcal{S}(\mathbf{\Lambda}) = \{\mathbf{Q}(\mathbf{\Lambda}, \pi) \mid \pi \in \Pi\}$, as the set of feasible long term fractions. As in the homogeneous case this set can be seen to be convex.

We define a performance metric for systems with heterogeneous compute adaptive jobs similar to what we did for the homogeneous case. For an offered load $\mathbf{\Lambda}$ under policy π, weighted revenue rate is given by:

$$\beta(\mathbf{Q}(\mathbf{\Lambda}, \pi), \mathbf{\Lambda}) = \sum_{j \in \mathcal{J}} w^j \left(\sum_{c \in \mathcal{C}} \lambda^j_c \sum_{l=1}^{L^j} \alpha^j_l q^j_{c,l}(\mathbf{\Lambda}, \pi) \right)$$

where α^j_l is the reward for type j compute adaptive jobs that successfully offload up through layer l and w^j is relative importance of different compute adaptive job types.

Weights w^j allow us to prioritize the revenue rates of one compute adaptive job over another. However, the underlying channel scheduler need not favor offloads of one type over another. In this section we shall continue to assume the proportionally fair scheduler used in the case when compute adaptive jobs were homogeneous.

Table III: Summary of Notation introduced

Notation	Description
$\mathcal{J}$	discrete set of compute adaptive job types
$\mathbf{\Lambda} = (\boldsymbol{\lambda}^1, ..., \boldsymbol{\lambda}^J)$	vector of vectors capturing offered load of different types of compute adaptive jobs
$\boldsymbol{\lambda}^j$	vector capturing type j compute adaptive job's arrival rate per channel class
λ^j	total arrival rate of type j compute adaptive job
τ^j	time budget for type j compute adaptive job
$\mathcal{L}^j$	discrete set of layers for type j compute adaptive job

A. Upper bound

Let $\mathbf{Q} = (\mathbf{q^1}, ..., \mathbf{q^J})$ be a vector of vectors, where $\mathbf{q^j} = (q^j_{c,l} : j \in \mathcal{J}, c \in \mathcal{C}, l \in \mathcal{L}^j)$, and $q^j_{c,l}$ is the fraction of type j compute adaptive jobs with channel quality c that successfully offload up through layer l within time budget τ^j. We define

$$\Sigma = \{\mathbf{Q} \mid \mathbf{q^j} \geq \mathbf{0} \text{ and } \sum_{l=1}^{L^j} q^j_{c,l} \leq 1, \ \forall j \in \mathcal{J}, \ \forall c \in \mathcal{C}\}$$

as the set of such *possible* vector of vectors. We then define the channel utilization per type j compute adaptive job based on $\mathbf{Q}$, which is the long term fraction of completed compute adaptive jobs, as

$$\rho^j(\mathbf{Q}) = \sum_{c \in \mathcal{C}} \lambda^j_c \sum_{l=1}^{L^j} q^j_{c,l} \frac{d^j_l}{r_c}$$

where d^j_l is the amount of data associated with offloading type j compute adaptive job up through layer l, and r_c is the transmission rate for channel type c. Let $\rho(\mathbf{Q}) = \sum_{j \in \mathcal{J}} \rho^j(\mathbf{Q})$ denote the total network utilization.

Recall that we defined $\mathcal{S}(\mathbf{\Lambda})$ to be the set of *achievable* long term fractions of successful compute adaptive job completions by stationary offloading policies when the system load is $\mathbf{\Lambda}$. Here we define

$$\overline{\mathcal{S}}(\mathbf{\Lambda}) = \{\mathbf{Q} \mid \mathbf{Q} \in \Sigma \text{ and } \rho(\mathbf{Q}) \leq 1\}$$

as a natural outer bound.

Theorem 2: *Given an offered load $\mathbf{\Lambda}$ we have that $\mathcal{S}(\mathbf{\Lambda}) \subseteq \overline{\mathcal{S}}(\mathbf{\Lambda})$ and*

$$\beta^*(\mathbf{Q}, \mathbf{\Lambda}) := \max_{\mathbf{Q} \in \mathcal{S}(\mathbf{\Lambda})} \beta(\mathbf{Q}, \mathbf{\Lambda}) \leq \max_{\mathbf{Q} \in \overline{\mathcal{S}}(\mathbf{\Lambda})} \beta(\mathbf{Q}, \mathbf{\Lambda}),$$

where $\beta^*(\mathbf{Q}, \mathbf{\Lambda})$ denotes the maximum achievable revenue rate given an offered load $\mathbf{\Lambda}$ by any stationary offloading policy.

Proof. Similar to Proof of Theorem 1 □

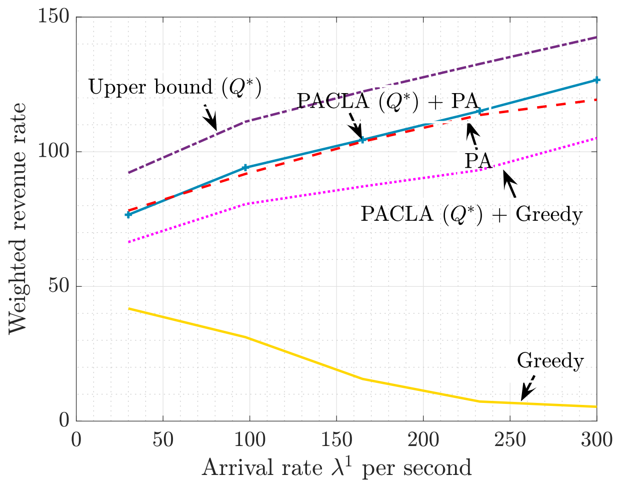
Remark: We let $\mathbf{Q}^* = \text{argmax}_{\mathbf{Q} \in \overline{\mathcal{S}}(\mathbf{\Lambda})} \beta(q, \mathbf{\Lambda})$ denote the maximizers associated with the bound. As before $\mathbf{Q}^*$ may not actually be achievable.

B. Simulation results

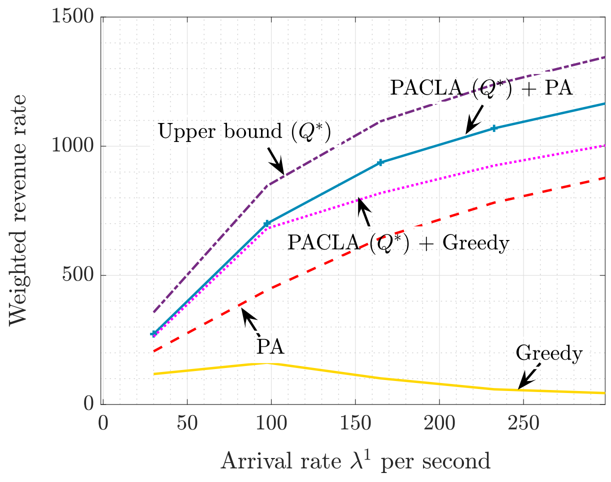
In this subsection, we compare the performance of our proposed offloading policies when compute adaptive jobs are heterogeneous. We set PACLA's admission probabilities to $\mathbf{Q}^*$, which we denote as PACLA ($\mathbf{Q}^*$) or simply PACLA for brevity. We then schedule admitted input layers using the Greedy or PA policy, i.e., PACLA + Greedy or PACLA + PA. We consider two types of compute adaptive jobs in our simulations. Each type of compute adaptive job is equally likely to experience 1 of 3 channel qualities, i.e., $\lambda^j_c = \lambda^j/3$ for all $c \in \mathcal{C}$. The reward based on amount of successfully offloaded data is a concave function for both types of compute adaptive job, i.e., $\alpha(d) = \sqrt{d/b}$. The complete set of simulation parameters are given in Table IV where b is a scale factor.

Table IV: Simulation parameters

Parameter	Value	Units/Remarks
(d^1_1, d^1_2, d^1_3)	$b \cdot (1/3, 2/3, 1)$	bits
(d^2_1, d^2_2, d^2_3)	$b \cdot (2/3, 4/3, 2)$	bits
r_c	$b \cdot (50, 75, 100)$	bits/sec
(τ^1, τ^2)	$(10/225, 20/225)$	sec
$(\alpha^1_1, \alpha^1_2, \alpha^1_3)$	$(0.58, 0.82, 1)$	Concave
$(\alpha^2_1, \alpha^2_2, \alpha^2_3)$	$(0.82, 1.16, 1.41)$	Concave

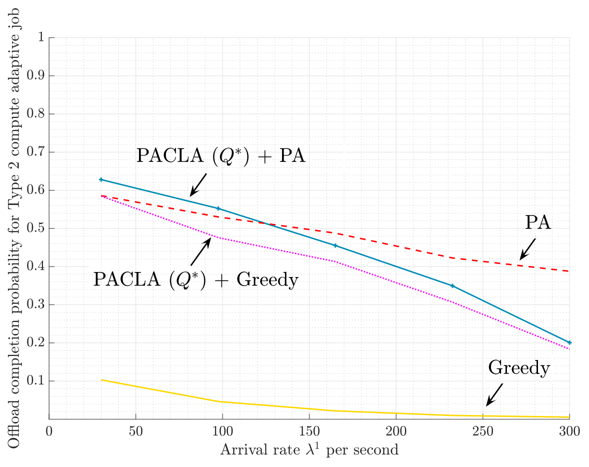


(a) For $(w^1, w^2) = (1, 1)$.

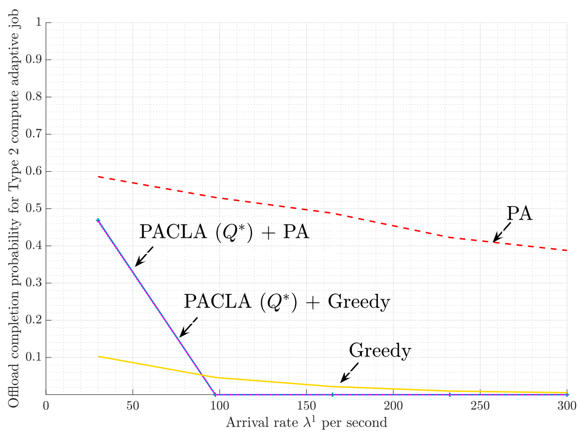


(b) For $(w^1, w^2) = (10, 1)$.

Figure 5: Weighted revenue rate for fixed load of Type 2 ($\lambda^2 = 98$) as load of Type 1 increases.



(a) For $(w^1, w^2) = (1, 1)$.



(b) For $(w^1, w^2) = (10, 1)$.

Figure 6: Offload completion probability of Type 2 adaptive job for fixed load of Type 2 ($\lambda^2 = 98$) as load of Type 1 increases.

Fig. 5 shows the weighted revenue rate generated from different offloading policies when the arrival rate of Type 2 compute adaptive job is fixed while that of Type 1 varies. As in the case of homogeneous compute adaptive jobs, PACLA + PA outperforms all other offloading policies. PA alone performs nearly as well as PACLA + PA when compute adaptive jobs are weighted equally, but its performance relative to PACLA + PA drops significantly when compute adaptive job weights are heavily skewed. This is because PA admits input layers solely based on how much data is left in a compute adaptive job's current input layer relative to its average throughput and remaining time budget. It pays no attention to compute adaptive job priorities/weights, unlike PACLA which statistically admits a compute adaptive job's input layers based on how much relative reward they will generate. Revenue rate maximization does, however, come at the cost of lower offload completion probability for the lesser weighted compute adaptive job type when the weights are heavily skewed, as seen in Fig. 6.

V. Conclusion

As the performance results demonstrate, our probabilistic admission control and layer assignment policy, PACLA, maximizes total revenue by adapting the admission probabilities for different numbers of input layers per compute adaptive job type to the revenue per compute adaptive job type and input layer, the average arrival rate per compute adaptive job type and channel class, and potentially compute adaptive job's weight. Our predictive abandonment policy, PA, then adapts PACLA's admission control decisions to current channel conditions by ending offloads when the input layers they are currently offloading are no longer estimated to complete within time budget. The combination of the two provides a robust approach to managing computation offloads of both homogeneous and heterogeneous compute adaptive jobs.

Acknowledgements

This work was supported by Intel, an affiliate of the 6G@UT center within the Wireless Networking and Communications Group at The University of Texas at Austin.

References

[1] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "A survey on mobile edge computing: The communication perspective," *IEEE Communications Surveys Tutorials*, vol. 19, no. 4, pp. 2322–2358, 2017.

[2] P. Mach and Z. Becvar, "Mobile edge computing: A survey on architecture and computation offloading," *IEEE Communications Surveys Tutorials*, vol. 19, no. 3, pp. 1628–1656, 2017.

[3] H. Sun, Y. Fan, S. Yuan, and Y. Cai, "Survey on computation offloading schemes in resource-constrained mobile edge computing," in *ICBDS*, Singapore, 2020, pp. 433–444.

[4] J. Huang, C. Samplawski, D. Ganesan, B. Marlin, and H. Kwon, "CLIO: Enabling automatic compilation of deep learning pipelines across IoT and Cloud," in *MobiCom*, 2020, p. 1–12.

[5] Y. Kang, J. Hauswald, C. Gao, A. Rovinski, T. Mudge, J. Mars, and L. Tang, "Neurosurgeon: Collaborative intelligence between the cloud and mobile edge," *ASPLOS*, vol. 52, no. 4, p. 615–629, 2017.

[6] J. Liu, Y. Mao, J. Zhang, and K. B. Letaief, "Delay-optimal computation task scheduling for mobile-edge computing systems," in *2016 IEEE International Symposium on Information Theory (ISIT)*, 2016, pp. 1451–1455.

[7] X. Chen, H. Zhang, C. Wu, S. Mao, Y. Ji, and M. Bennis, "Optimized computation offloading performance in virtual edge computing systems via deep reinforcement learning," *IEEE Internet of Things Journal*, vol. 6, no. 3, pp. 4005–4018, 2019.

[8] M. Xu, F. Qian, M. Zhu, F. Huang, S. Pushp, and X. Liu, "Deepwear: Adaptive local offloading for on-wearable deep learning," *IEEE Transactions on Mobile Computing*, vol. 19, no. 2, pp. 314–330, 2020.

[9] J. H. Ko, T. Na, M. F. Amir, and S. Mukhopadhyay, "Edge-host partitioning of deep neural networks with feature space encoding for resource-constrained internet-of-things platforms," in *IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS)*, 2018, pp. 1–6.

[10] H.-J. Jeong, I. Jeong, H.-J. Lee, and S.-M. Moon, "Computation offloading for machine learning web apps in the edge server environment," in *IEEE International Conference on Distributed Computing Systems (ICDCS)*, 2018, pp. 1492–1499.

[11] W. Shi, Y. Hou, S. Zhou, Z. Niu, Y. Zhang, and L. Geng, "Improving device-edge cooperative inference of deep learning via 2-step pruning," in *IEEE Conference on Computer Communications Workshops*, 2019, pp. 1–6.

[12] A. E. Eshratifar, A. Esmaili, and M. Pedram, "Bottlenet: A deep learning architecture for intelligent mobile cloud computing services," in *IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, 2019, pp. 1–6.

[13] Y. Matsubara and M. Levorato, "Neural compression and filtering for edge-assisted real-time object detection in challenged networks," in *International Conference on Pattern Recognition (ICPR)*, 2021, pp. 2272–2279.

[14] Y. Matsubara, S. Baidya, D. Callegaro, M. Levorato, and S. Singh, "Distilled split deep neural networks for edge-assisted real-time systems," in *Hot Topics in Video Analytics and Intelligent Edges*, 2019, p. 21–26. [Online]. Available: https://doi.org/10.1145/3349614.3356022

[15] D. Callegaro, Y. Matsubara, and M. Levorato, "Optimal task allocation for time-varying edge computing systems with split DNNs," in *IEEE Global Communications Conference*, 2020, pp. 1–6.

[16] Y. Matsubara and M. Levorato, "Neural compression and filtering for edge-assisted real-time object detection in challenged networks," in *International Conference on Pattern Recognition (ICPR)*, 2021, pp. 2272–2279.

[17] H. Schwarz, D. Marpe, and T. Wiegand, "Overview of the scalable video coding extension of the H.264/AVC standard," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 17, no. 9, pp. 1103–1120, 2007.

[18] W. Hu, J. Mao, Z. Huang, Y. Xue, J. She, K. Bian, and G. Shen, "Strata: Layered coding for scalable visual communication," 2014, p. 79–90.

[19] F. P. Kelly, "Bounds on the performance of dynamic routing schemes for highly connected networks," *Mathematics of Operations Research*, vol. 19, no. 1, pp. 1–20, 1994.

[20] A. R. Ward and W. Whitt, "Predicting response times in processor-sharing queues," in *In Proc. of the Fields Institute Conf. on Comm. Networks*, 2000, pp. 1–29.