Energy-efficient cooperative inference via adaptive deep neural network splitting at the edge

Ibtissam Labriji, Mattia Merluzzi†, Fatima Ezzahra Airod†, Emilio Calvanese Strinati†
† CEA-Leti, Université Grenoble Alpes, F-38000 Grenoble, France
email: ibtissam.labriji@gmail.com,{mattia.merluzzi, fatima-ezzahra.airod, emilio.calvanese-strinati}@cea.fr.

***Abstract*—Learning and inference at the edge is all about distilling, exchanging, and processing data in a cooperative and distributed way, to achieve challenging trade-offs involving energy, delay, and accuracy. This calls for a joint orchestration of radio and computing resources. We propose an online adaptive resource allocation algorithm to choose *where to compute*, and *how to offload computations*, exploiting the concept of Deep Neural Network (DNN) splitting. The latter allows a device to locally execute part of an inference related processing, and delegate the other portion to a nearby Mobile Edge Host (MEH), which receives intermediate results from the device via a time varying wireless communication channel. Our method deals with dynamic parameters involving wireless channels, data arrivals, and MEH's CPU availability, by taking online control actions including the best splitting point, and the uplink data rate to transfer raw data or intermediate results (e.g., extracted features). The decision is taken only based on instantaneous observations of context parameters, to minimize the long-term device energy consumption, while guaranteeing the end-to-end delay not to exceed a predefined threshold, on average and probabilistic sense. Besides a theoretical analysis, numerical simulations show the effectiveness of our adaptive method in selecting the best partial offloading decision (DNN splitting) under different network conditions. Differently from previous works on edge inference, we exploit recently developed empirical models for the energy consumption of NVIDIA® edge boards, to evaluate the performance of DNN splitting at the edge, when exploring the typical offloading trade-off between energy and delay, both entailing communication and computing.**

***Keywords*—Multi-access Edge Computing (MEC), Edge AI, DNN splitting, energy efficiency, edge inference**

I. Introduction

Today, as part of the race to 6G, wireless networks are requested to evolve from pure communication infrastructures connecting people and things, to efficient platforms connecting heterogeneous intelligent agents, to enable complex cooperative tasks. However, analyzing the myriads of data continuously generated by sensors, machines, robots, etc., through complex Machine Learning (ML) models, needs a tremendous amount of computation resources, which is not feasible for end devices with limited hardware resources, memory size, and battery lifetime. In this regard, Multi-access Edge Computing (MEC) [1] is a promising framework to remedy this issue, as it provides access to shared pools of computing and storage resources, in close proximity to the end service consumers, e.g., within the radio access network. Such a computing paradigm, coupled with ML, can more intelligently assist end devices and reduce end-to-end (E2E) latency and energy consumption. Computation offloading [2] can be affected by user preferences, privacy, radio and backhaul connection quality, on-board processing capabilities, and MEC availability. Indeed, computing resources can be limited and volatile in MEC scenarios, compared to central cloud processing settings. Computation offloading can be exploited to transfer (part of) the execution of an ML model (e.g., a Deep neural Network - DNN), to perform real-time inference at the edge of wireless networks [3]. There are three options for DNN inference at the edge [4]: *i)* perform all computations at the device side, which can be referred to as *full local inference*, *ii)* offload the entire DNN, referred to as *full offloading*, and *iii)* partition the DNN and offload only a part of it to the MEC network, referred to as *partial offloading*. Full local inference can be computationally demanding for the end device, while full offloading requires a significant amount of raw data to be transferred to a Mobile Edge Host (MEH), through the wireless connection with an Access Point (AP). This is not ideal from privacy and latency perspectives. Additionally, full offloading shifts the whole model inference, which is highly dependent on (unpredictable) server availability, typical of edge resources. Partial offloading for edge inference involves a cooperative inference between the end device and the MEH, by splitting the DNN into two parts, with some layers executed locally and the others offloaded. Of course, this assumes both sides to be equipped with the DNN model, but it can reduce the latency and energy consumption, if the splitting is optimized. This work proposes an adaptive algorithm to select the best splitting point (SP) and wireless resources, to minimize the device energy consumption under E2E delay constraints, entailing (local and remote) computation and communication.

Related Works. Previous works in the literature address DNN splitting problems for edge inference [5]–[11]. To overcome the excessive E2E latency experienced with full offloading, [6] proposes a compression technique to reduce the size of the transmit data. Partial offloading requires a co-inference between the end-user and the MEH. In [7], the authors propose a strategy to partition the DNN under different network conditions to minimize the overall processing delay. Also, [8] minimizes the energy consumption on the client-side by partitioning Convolutional Neural Network (CNN) computations between the client and the cloud. Mao *et al.* partition the DNN always after the first convolutional layer to minimize the cost of mobile devices and use the differentially private mechanism to preserve the privacy [9]. None of these works jointly optimizes energy consumption and E2E delay, in a dynamic and adaptive fashion. Indeed, the splitting point is chosen in a static way, ignoring time-varying unreliable channel conditions and MEH's resource availability. Also, empirical models of real NVIDIA® edge boards energy consumption for Convolutional Neural Networks (CNNs) based inference have been recently developed [12], but no previous works have exploited them.

The work of M. Merluzzi has been partly funded by the European Commission through the H2020 project Hexa-X (Grant Agreement no. 101015956).

Our contribution. We propose an algorithm that dynamically optimizes SP selection and communication resources, to minimize the average energy consumption at the end device while meeting a deadline set for the overall delay, under uncertain availability of resources at the MEH. Also, for the numerical evaluation, we exploit a recently developed empirical model for the energy consumption of CNNs on real NVIDIA® edge boards, proposed in [12]. To the best of our knowledge, this has never been done before in the literature.

II. System Model

In this work, focusing on edge inference, an end device executes part of a CNN model locally and offloads the remaining part to an MEH. The system under investigation is dynamic, and the following parameters can vary across time according to a priori unknown distributions: *i)* wireless channels; *ii)* data arrivals (number of input images to classify); *iii)* CPU availability at the MEH. To handle the system dynamics through observations and control actions, we organize time in instants $t = 1, 2, 3, \ldots$, at the beginning of which a resource orchestrator observes the current system state, and accordingly optimizes, jointly, radio resources and splitting point selection. In the sequel, for a variable X, we denote $\overline{X} = \lim_{T\to\infty} \frac{1}{T} \sum_{t=0}^{T-1} \mathbb{E}\{X(t)\}$.

A. Inference performance on NVIDIA® Edge boards

A convolutional layer can be represented as a multidimensional matrix of input values, with size $w \times h \times d_i$, and a set of Kernel functions, which are square matrices of size $K \times K \times d_i$. Then, denoting by $w = h$ the input feature map size, by d_i the number of input feature maps, by d_o the number of output feature maps, by K the Kernel size parameter, and by s the stride parameter [12], the total number of Multiply–accumulate (MAC) operations [13], needed to process an input at convolutional layer c is approximated as

$$M_c = \left((w_c - K_c)s_c^{-1} + 1\right)^2 \times d_{i,c} \times K_c^2 \times d_{o,c} \quad (1)$$

Then, given M_c, the average energy consumed to process over a convolutional layer c reads as follows [12]:

$$E_c = M_c \left(a d_{o,c}^{-1} + b\right), \quad (2)$$

where a and b are specific parameters that depend on the edge board. Specific values, available in [12, Table III], will be reported in Section IV. Also, since the MAC and the number of FLOPs W_c are linked through a linear relation, i.e. $W_c = 2M_c$, we can write the computing latency of convolutional layer c:

$$D_c = \frac{W_c}{f_x} = \frac{2\left((w_c - K_c)s_c^{-1} + 1\right)^2 \times d_{i,c} \times K_c^2 \times d_{o,c}}{f_x}, \quad (3)$$

with f_x the performance of the edge board x in terms of FLOPs/s. Similarly, denoting by u_i and u_o the input and output size of a fully connected layer u, respectively, the energy consumption can be written as follows:

$$E_u = M_u \times a', \quad (4)$$

where M_u is the number of MAC operations, computed as

$$M_u = u_i \times u_o, \quad (5)$$

and a' is an hardware dependent parameter, specified in [12, Table III]. Also, the computing latency can be simply written as $D_u = \frac{W_u}{f_x} = \frac{2M_u}{f_x}$. In the sequel, we will use $W_{c(u)}$ (i.e., the number of FLOPs) to measure the computing load.

B. Communication parameters

To complement the computing performance and parameters, it is important to characterize the output size of a layer, which represents the amount of data to be transmitted when part of the computation is transferred to an MEH. In particular, denoting by P_c the padding value, the output size of a convolutional layer can be written as follows:

$$L_c = d_{o,c}\left((w_c - K_c + 2P_c)s_c^{-1}\right)^2. \quad (6)$$

C. Computation model

Considering the described model, we denote by $\mathcal{J} = \{0, 1, \ldots, J\}$ the set of all splitting points, with $j = 0$ the full offloading case, and $j = J$ the full local inference.

Local computation delay: let us assume that, at time t, the end device generates $N(t)$ new input images to be classified, as part of a random process whose statistics are unknown in advance. Also, let us suppose that, at time t, the user selects a generic SP k, i.e., it performs computations up to splitting point k and offloads the remaining part (i.e., from SP $k+1$ up to SP J) to the MEH. Then, assuming a local CPU processing performance f_l, and denoting by $W_{0\to k}$ the number of FLOPs needed to perform processing from the input up to SP k, the local computing delay reads as follows:

$$D_l(t) = \frac{W_{0\to k} \cdot N(t)}{f_l}, \quad (7)$$

where, obviously, $W_{0\to 0} = 0$.

Local computation energy consumption: recalling the energy consumption of each layer of a CNN ((2), (4)), we denote by E_j the energy spent for processing between SP $j-1$ and SP j ($E_{-1} = 0$ for consistency, and $E_0 = 0$). Then, if a generic SP k is selected at time t, the end device spends

$$E_l(t) = \sum\nolimits_{j=0}^{k} E_j. \quad (8)$$

Remote computation delay: in this paper, we assume that remote computation resources are provided by the MEH if available, with a priori unknown statistics of such availability, due to, e.g., higher priotity traffic. Then, denoting by f_r the MEH's FLOP frequency, and assuming that a generic SP k is selected at time t, the remote computation delay is

$$D_r(t) = \frac{W_{k+1\to J} \cdot N(t)}{\alpha_r(t) f_r}, \quad (9)$$

where $W_{k+1\to J}$ denotes the number of FLOPs to be performed to process from SP k up to SP J (i.e., the output of the architecture), and $\alpha_r(t) \in (0, 1]$ is a random variable denoting the availability of the remote CPU processing power. Obviously, $W_{J+1\to J} = 0$ for consistency.

D. Communication Model

Let us denote by $R(t)$ the uplink data rate (in bits/s) at time t, used to upload data (either raw data or intermediate output

feature maps). The latter depends on the bandwidth, the noise power spectral density, and the time-varying wireless channel. Then, denoting by L_k the output feature map size (in bits) of a generic layer k at which the splitting is performed, the wireless communication delay can be written as follows:

$$D_{\text{tx}}(t) = \frac{L_{k(t)} \cdot N(t)}{R(t)}. \tag{10}$$

At the same time, by inverting the well known *Shannon formula*, we can write the device energy consumption as:

$$E_{\text{tx}}(t) = \frac{N_0 B(t)}{h(t)} \left(\exp\left(\frac{R(t)\ln(2)}{B(t)} \right) - 1 \right) D_{\text{tx}}(t), \tag{11}$$

where $B(t)$ is the available bandwidth at time t, N_0 is the noise power spectral density, and $h(t)$ is the instantaneous realization of the channel power gain, whose statistics are assumed to be unknown in advance. Finally, recalling (7), (9), and (10), we can write the total inference time, which entails local computation, communication, and remote computation phases, as $D_{\text{tot}}(t) = D_l(t) + D_{\text{tx}}(t) + D_r(t)$. Also, recalling (8) and (11), the total energy spent by the device for (partial) local processing and transmission is $E_{\text{tot}}(t) = E_l(t) + E_{\text{tx}}(t)$.

III. Problem formulation

As already mentioned, our aim is to guarantee inference service continuity with the minimum cost in terms of device energy consumption. As such, we consider two performance: *i)* the average end-to-end delay, and *ii)* the outage probability. The latter is defined as the probability that D_{tot} exceeds a predefined threshold $D_{\max}$, and can be formally written as $\overline{\Pr\{D_{\text{tot}}(t) > D_{\max}\}} = \overline{u\{D_{\text{tot}}(t) - D_{\max}\}}$, where $u\{\cdot\}$ denotes the unitary step function, and the equality holds due to the fact that the RHS represents the expectation of a Bernoulli random variable, i.e. the probability of the event. In particular, we require the average end-to-end delay to not exceed a predefined threshold D_{avg}, and the outage probability to not exceed a threshold ϵ, formulating the following problem:

$$\min_{\{k(t),R(t)\}_t} \ \overline{E_{\text{tot}}(t)} \tag{12}$$

$$\text{subject to } (a)\ \overline{D_{\text{tot}}(t)} \le D_{\text{avg}}; \quad (b)\ \overline{u\{D_{\text{tot}}(t) - D_{\max}\}} \le \epsilon;$$
$$(c)\ R_{\min}(t) \le R(t) \le R_{\max}(t), \quad \forall t; \quad (d)\ k(t) \in \mathcal{J}, \quad \forall t,$$

where the expectation is taken with respect to random wireless channels, data arrivals, and MEH's computing availability, assumed to be time-varying according to non-controllable exogenous events (e.g., higher priority traffic - cf. (9)). Constraint (a) imposes the average E2E delay not to exceed D_{avg}; (b) imposes the outage not to exceed ϵ; whereas, the instantaneous constraints (c)-(d) have the following meaning: (c) the instantaneous data rate is selected between a minimum and a maximum value, both depending on the instantaneous realization of the wireless channel, and the minimum and maximum transmit power, also set a priori; (d) the SP is selected among the set of possible SPs of the CNN architecture.

A. Proposed solution

Problem (12) is challenging due to the unknown statistics of context parameters, which make the objective function and constraints (a)-(b) generally unknown. To solve it effectively, we first apply the tools of *Lyapunov stochastic optimization* to transform (12) into a pure stability problem to be solved in a per slot fashion. In particular, following [14], for constraint (a), we can define a *virtual queue* $Z(t)$, which evolves as follows across subsequent time instants:

$$Z(t+1) = \max\left(0, Z(t) + D_{\text{tot}}(t) - D_{\text{avg}}\right). \tag{13}$$

As defined in the above equation, this virtual queue increases at all time instants in which $D_{\text{tot}}(t)$ exceeds the predefined average threshold, and decreases otherwise, i.e., it keeps track of the system's constraint violations. Similarly, we can define a virtual queue for constraint (b):

$$H(t+1) = \max\left(0, H(t) + u\{D_{\text{tot}}(t) - D_{\max}\} - \epsilon\right). \tag{14}$$

The aim is to drive the network towards low virtual queue backlog states (congestion), while minimizing the objective function of the original problem (i.e., the energy). More specifically, the mean-rate stability of the virtual queues[1], guarantees constraint (a)-(b) to be met. To this end, we define a Lyapunov function (LF) capturing the virtual queue congestion state as $L(\mathbf{Q}(t)) = \frac{1}{2}(Z^2(t) + H^2(t))$, with $\mathbf{Q}(t) = [Z(t), H(t)]$. Furthermore, from the LF, we can define the *drift-plus-penalty* (DPP) function, which is the conditional expected change of the LF over successive slots, with an additional term weighting the objective function of (12), i.e., the energy consumption [14]. The DPP reads as follows [14]:

$$\Delta_p(t) = \mathbb{E}\left\{ L(\mathbf{Q}(t+1)) - L(\mathbf{Q}(t)) + V E_{\text{tot}}(t) | \mathbf{Q}(t) \right\}, \tag{15}$$

where V denotes the trade-off parameter weighting the energy consumption. The higher is V, the more importance is devoted to the objective function but less to the virtual queues' backlog. Intuitively, a higher V leads to lower energy consumption but also higher queue backlogs and thus convergence time. Due to the unknown statistics of context parameters, we proceed by defining a suitable upper bound of the DPP that is then opportunistically minimized in a per-slot fashion, thus removing the expectation, following the theoretical arguments in [14]. Exploiting [14, Eqn. (4.47)], the upper bound reads as follows in our case:

$$\Delta_p(t) \le S + \mathbb{E}\{ Z(t)(D_{\text{tot}}(t) - D_{\text{avg}}) + H(t)(u\{D_{\text{tot}}(t) - D_{\max}\} - \epsilon) + V E_{\text{tot}}(t) | \mathbf{Q}(t) \}, \tag{16}$$

where $S = \frac{(D'_{\max} - D_{\text{avg}})^2 + (1-\epsilon)^2}{2}$, with $D'_{\max}$ a maximum delay, which is finite due to constraint (c) of (12). Now, hinging on the concept of *opportunistically minimizing expectations*, we formulate the following per-slot deterministic problem by removing the expectation from (16):

$$\min_{k(t),R(t)} \ Z(t) D_{\text{tot}}(t) + H(t) u\{D_{\text{tot}}(t) - D_{\max}\} + V E_{\text{tot}}(t)$$
$$\text{subject to } (c)\text{-}(d) \text{ of problem (12)} \tag{17}$$

where $D_{\text{tot}}(t)$ is defined through (7), (9), and (10), while E_{tot} is defined through (8) and (11). Under feasibility assumption of (12), and i.i.d. assumption of context parameters realizations across different time instants, by optimally solving (17), the mean rate stability of the virtual queues is ensured [14] and,

[1] defined as $\lim_{T\to\infty} \mathbb{E}\{G(T)\}/T = 0$ for a generic virtual queue G

as a consequence, constraints (a)-(b) are guaranteed. Also, as V increases, the distance between the global optimal solution of (12) and the one obtained through (17) reduces at the cost of increased virtual queues backlog, i.e., convergence time. Despite the strong complexity reduction induced by the above stochastic optimization framework, problem (17) is a mixed integer non-convex problem. However, let us notice that the set $\mathcal{J}$ is generally of low cardinality, as typical CNN architectures do not include a large number of possible SPs. Therefore, from the point of view of the SP choice k, it is feasible to perform an *exhaustive search* in each time slot. More specifically, at time t, it is sufficient to solve the problem with respect to R, and finally compare the obtained solutions for all possible choices of SP $k(t)$, to select the one (i.e., involving k, and R) that minimizes the objective function in (17). Now, the last task is to solve (17) for a fixed k, with respect to R. To efficiently perform this step, we hinge on the upper bound $u\{D_{\text{tot}} - D_{\max}\} \leq D_{\text{tot}}/D_{\max}$, which holds on the feasible set $(D_{\text{tot}}(t) > 0, \forall t)$. This approximation hinges on the concept of a *C-additive approximation*, which allows inexact solutions of the per-slot problem without preventing convergence, provided that such approximations are within a finite constant C from the optimal one. Of course, this is paid by the need of a higher value of V to asymptotically approach the minimum, and thus with a longer convergence time. Nevertheless, numerical results will show the effectiveness of the above approximation, which in turn dramatically simplifies the solution of the per slot problem. Given this approximation, the objective function in (17) is replaced by its surrogate $(Z(t) + H(t)/D_{\max})D_{\text{tot}}(t) + VE_{\text{tot}}$. Interestingly, once k is fixed, it is easy to prove that, with this approximation, (17) reduces to a *convex problem* with respect to the data rate R, and thus it can be *optimally solved*. In particular, considering only the terms that depend on $R(t)$ in (17), the optimal solution can be found in closed-form through the Karush-Kuhn-Tucker (KKT) conditions [15], which in this case reduce to finding a stationary point of the objective function, and bounding the result according to the linear constraint (c) in (12). In particular, defining $\Omega(t) = Z(t) + H(t)/D_{\max}$, let $\mathcal{G}$ be the derivative of the surrogate objective function with respect to R (we omit the temporal index t to ease the notation):

$$\mathcal{G} := -\frac{L_{k'}N}{R^2}\left[V\frac{N_0 B}{h}\left(\exp\left(\frac{R\ln(2)}{B}\right)-1\right)+\Omega\right] + \frac{L_{k'}NVN_0\ln(2)}{Rh}\exp\left(\frac{R\ln(2)}{B}\right) = 0. \quad (18)$$

Due to the fact that $R \geq R_{\min} > 0$, by multiplying both sides of (18) by $\frac{R^2 h}{L_{k'}NVN_0 B}$, we have ($e = \exp(1)$):

$$1 - \frac{\Omega h}{VN_0 B} + \left(\frac{\ln(2)R}{B} - 1\right)\exp\left(\frac{R\ln(2)}{B} - 1\right)e = 0,$$

where we multiplied the second term by e/e, to obtain a convenient structure of the function, exploited to derive the solution in closed form, as clarified here below. This leads to:

$$\left(\frac{\ln(2)R}{B} - 1\right)\exp\left(\frac{\ln(2)R}{B} - 1\right) = \frac{1}{e}\left(\frac{\Omega h}{VN_0 B} - 1\right).$$

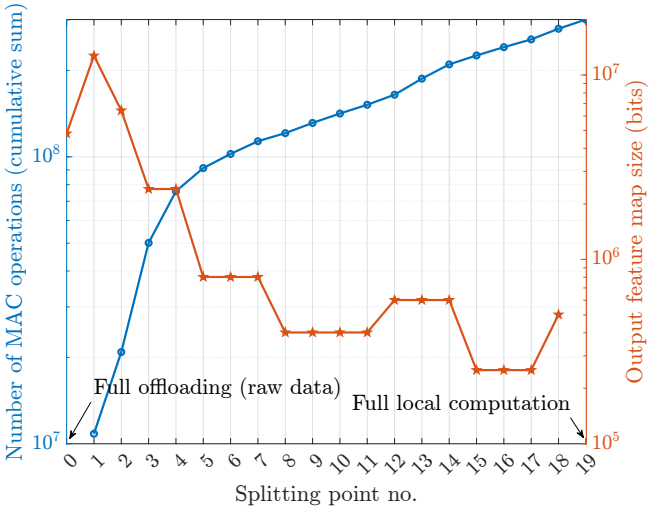


Fig. 1: Number of MAC operations and output feature map size

Exploiting the principal branch of the *Lambert function* $\mathcal{W}\{\cdot\}$ [16], we have:

$$R_* = \left[\frac{B}{\ln(2)}\left(\mathcal{W}\left\{\frac{\Omega h}{VN_0 Be} - \frac{1}{e}\right\} + 1\right)\right]_{R_{\min}}^{R_{\max}}. \quad (19)$$

Finally, denoting by $\mathbf{R}^*$ the vector of optimal rates for each SP, the optimal SP is found as follows:

$$\arg\min_k \; Z(t)\mathbf{D}_{\text{tot}}(t) + H(t)u\{\mathbf{D}_{\text{tot}}(t) - D_{\max}\} + V\mathbf{E}_{\text{tot}}(t),$$

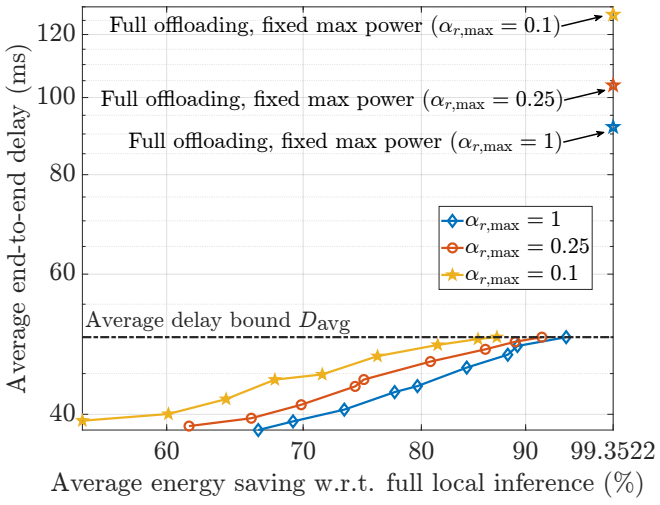
where $\mathbf{D}_{\text{tot}}(t)$ and $\mathbf{E}_{\text{tot}}(t)$ are the vectors containing delay and energy consumption for each k, respectively, given R_*. in (19).

IV. Numerical results

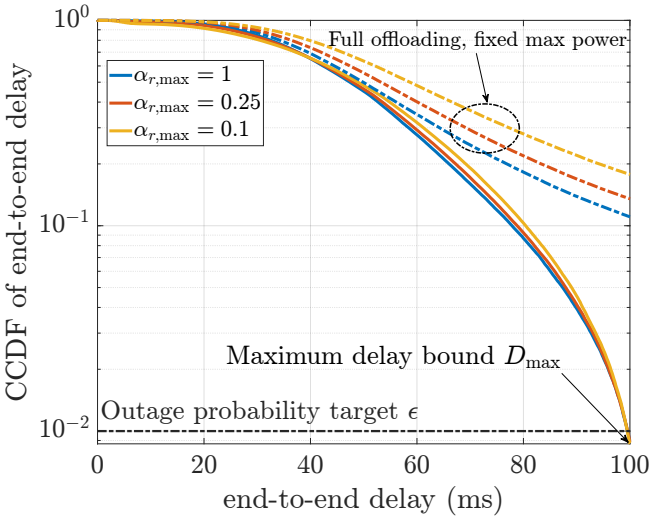
In this section, we provide a numerical assessment of the proposed method, considering the NVIDIA® edge boards characteristics reported in [12, Tables I, II, III].

Computing model: we assume the device to be equipped with the NVIDIA board *Jetson TX2*®, with $f_l = 1.3$ TFLOPs (cf. (7)), and the MEH with the NVIDIA board *Jetson Xavier NX*®, with $f_r = 21$ TFLOPs (cf. (9)). The parameters in (2) and (4) are taken from [12, Table III], and are $a = 2.6727 \times 10^{-8}$, $b = 1.21334 \times 10^{-10}$, and $a' = 6.2454 \times 10^{-9}$. The available CPU FLOPs are computed, across time slots as $f_{r,a}(t) = \alpha_r(t) f_r$, where α_r is assumed to be uniformly distributed in $(0, \alpha_{r,\max}]$, with different values of $\alpha_{r,\max} \leq 1$, as specified in the respective figures.

Inference Model - MobileNetv2: although the method proposed in this paper is independent from the CNN architecture, in this paper we assume that both the device and the MEH are equipped with *MobileNetv2* [17], one of the most popular lightweight models for computer vision tasks. *MobileNetv2* is a CNN with 53 convolutional layers, and the pretrained version is trained on more than a million images from the *ImageNet* dataset [18]. In this paper, we assume 20 splitting points, including $j = 0$, i.e. the offloading of the whole computation task through the transmission of raw data. In Fig.1, we show the number of MAC operations to be executed locally (cf. (1), (5)), and the intermediate data size (cf. (6)), that needs to be transmitted to the MEH to catch up the processing



(a) Average E2E delay vs. energy saving w.r.t. full local inference



(b) CCDF of end-to-end delay

Fig. 2: Energy-delay trade-off and SP selection

remotely for each SP (in logarithmic scale), for the Imagenet data set, assuming 32 bits encoding each element of the output feature maps. More specifically, the blue curve represents the number of MAC operations to be executed locally, for each SP choice (i.e., the cumulative sum, up to the SPs in the abscissa), computed using (1) and (5). The heaviest local workload is obviously reached for the full local inference case, with around 3×10^8 MAC operations[2]. Indeed, SP 0 and 19 represent the extreme cases, namely the full offloading case, and the full local inference, respectively. For $j = 0$, no MAC operations are needed locally, while for $j = 19$, no data need to be transmitted. Note that, for some of the convolutional layers, *MobileNetv2* makes use of depthwise convolution, a computationally efficient variant, whose number of operations can be found in [13, Eqn. (4)]. Notably, the output feature map size strongly depends on the splitting point choice.

Wireless communication model: we consider an end device

[2]https://towardsdatascience.com/review-mobilenetv2-light-weight-model-image-classification-8febb490e61c

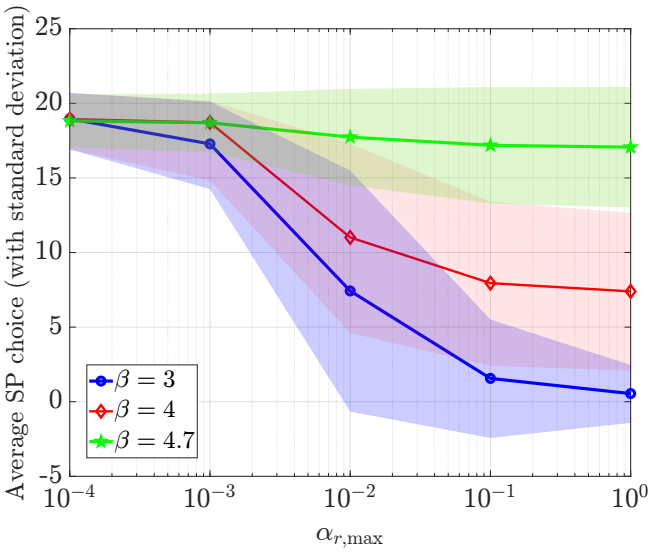


Fig. 3: Average SP selection vs. MEH's availability

placed at distance $d = 50$ m from the AP, operating at a carrier frequency $f_c = 3.5$ GHz, with $B = 200$ MHz bandwidth. The noise power spectral density is set to $N_0 = -174$ dBm/Hz, and an exponent $\beta = 3$ is assumed to generate the path loss. Also, Rayleigh fading with unit variance is considered. The minimum and maximum transmit powers are set to $p_{\min} = 10$ mW and $p_{\max} = 100$ mW, respectively. Also, at each slot t, the end device generates $N(t)$ new inferencing requests, from a Poisson process with parameter $\lambda = 10$.

As a first result, in Fig. 2a, we show the trade-off between the average end-to-end delay, and the average energy saving, with respect to the case in which the end device runs the entire inference task locally, i.e., the case in which no computation is offloaded to the MEH. Therefore, the results intrinsically show the comparison between our proposed solution, and a full local computation setting. We consider an average delay threshold $D_{\text{avg}} = 50$ ms, a maximum delay threshold $D_{\text{avg}} = 100$ ms, and a target outage probability below $\epsilon = 10^{-2}$. The trade-off is obtained by increasing the parameter V (cf. (15)) from left to right, and the simulation is run 2×10^5 slots (the first 10^4 are not considered when averaging to avoid any transient interval). Besides the full local inference case, the proposed method is compared with a non optimized full offloading case, with fixed transmission rate $R(t) = R_{\max}(t)$, i.e., with fixed maximum transmit power $p_{\max}, \forall t$, whose resulting delays are represented by the pentagrams, and whose energy saving is 99.35%. Also, results are obtained for three different conditions of MEH's CPU availability $\alpha_{r,\max}$ (as shown in the figure). First, let us notice how the full offloading case achieves the best performance in terms of energy saving (99.35%), however without guaranteeing the end-to-end delay, represented by the horizontal black dashed line, in any of the cases. Obviously, as the MEH's CPU availability decreases, the delay of this benchmark solution increases without control, and gets further from the predefined threshold, due to the absence of a delay-aware SP selection and transmit power optimization. Also, as visible from Fig. 2b, the benchmark strategy does not guarantee the outage delay, incurring in outages more than 10 times higher than the requirement, differently from the proposed method. On the other hand, our method achieves

lower but large energy savings, without the cost of violating the delay constraints, a fundamental aspect of edge inference services. This is thanks to its capability of selecting the best SP, i.e., moving more computations locally whenever the MEH's CPU suffers from sever drops in terms of FLOP frequency. Obviously, a very severe drop also deteriorates the performance of the energy-aware splitting decision and transmit power optimization, due to the unavailability of the needed resources. For example, even with $\alpha_{r,\max} = 0.1$, the maximum energy saving is around 85% in the proposed system setup. Nevertheless, the delay constraint is always guaranteed and never sacrificed, both in average and probabilistic sense (see Fig. 2b). Then, our method is able to find the lowest energy solution that guarantees the constraints. This is due to the fact that, as the MEH's CPU availability decreases, more computations are pushed locally, in a dynamic and adaptive way. The latter conjecture is numerically proven in Fig. 3, in which we show, for the same simulation, the average SP decision, as a function of the MEH's CPU availability, for the highest value of V, i.e., the one corresponding to the rightest points of Fig. 2a. The latter can be interpreted as the average depth of local computations, directly relating to the number of layers executed locally. This is shown for different path loss exponents β. As expected, the method autonomously increases the number of locally executed layers, as the MEH's CPU availability decreases, at the cost of lower energy savings (see Fig. 2a). Obviously, the higher is β, the higher the number of local computation is, even in the case of high availability of computing resources. Interestingly, for $\beta = 3$ (i.e., the best path loss conditions), the standard deviation shows different behaviors. Namely, for low availability, full local inference is preferred most of the time (high average and low standard deviation). Then, increasing MEH's computing availability, the mean decreases (i.e., more computations are pushed remotely on average), while the standard deviation increases, as the method has more degrees of freedom in selecting the best SP, based on current connect-compute resources. By further increasing the MEH's availability, the mean and the standard deviation decrease again, due to the fact that full offloading decisions become more frequent and mostly preferred, although not always possible due to channel fluctuations. For worse channel conditions (i.e., $\beta = 4$ and $\beta = 4.7$), the mean reaches larger values even in the case of high MEH's availability, with lower standard deviation for $\beta = 4.7$, since the method has less freedom to offload due to bad channel conditions.

Overall, the simulations show the inherent coupling between wireless and computing resources. First, partial offloading based on adaptive DNN splitting is the most promising solution for edge inference, thanks to its higher degrees of freedom in choosing the offloaded workload, based on current connect-compute network conditions. Also, as the MEH's CPU availability decreases, more computations are automatically pushed locally at the device, incurring in lower energy savings. This is even more noticed in case of severe propagation conditions at the wireless access.

V. Conclusions and future directions

We proposed a method to dynamically and jointly select the best SP of a CNN model, and communication resources, to perform low energy edge inference with controlled E2E delay. We exploited tools from Lyapunov stochastic optimization to devise our solution. Numerical results on empirical models for the energy consumption show that our proposed solution outperforms two benchmarks (full offloading and full local inference) by effectively reducing energy consumption while guaranteeing delay constraints. Future works include multi-user scenarios, the investigation of energy fluctuations of the edge boards, and the impact of wireless errors on the SP selection, towards goal-oriented cooperative edge inference.

References

[1] S. Kekki *et al.*, "ETSI White Paper: MEC in 5G networks," *The European Telecommunications Standards Institute (ETSI), Tech. Rep. ETSI White Paper No. 28*, 2018.

[2] M. Merluzzi, P. Di Lorenzo, S. Barbarossa, and V. Frascolla, "Dynamic Computation Offloading in Multi-Access Edge Computing via Ultra-Reliable and Low-Latency Communications," *IEEE Transactions on Signal and Information Processing over Networks*, pp. 1–1, 2020.

[3] M. Merluzzi, P. Di Lorenzo, and S. Barbarossa, "Wireless Edge Machine Learning: Resource Allocation and Trade-Offs," *IEEE Access*, vol. 9, pp. 45 377–45 398, 2021.

[4] Y. Matsubara, M. Levorato, and F. Restuccia, "Split computing and early exiting for deep learning applications: Survey and research challenges," *ACM Comput. Surv.*, mar 2022, just Accepted. [Online]. Available: https://doi.org/10.1145/3527155

[5] J. Shao and J. Zhang, "Communication-computation trade-off in resource-constrained edge inference," *IEEE Communications Magazine*, vol. 58, no. 12, pp. 20–26, 2020.

[6] M. Nakahara *et al.*, "Retransmission edge computing system conducting adaptive image compression based on image recognition accuracy," in *2021 IEEE 94th Vehicular Technology Conference (VTC2021-Fall)*. IEEE, 2021, pp. 1–5.

[7] C. Hu, W. Bao, D. Wang, and F. Liu, "Dynamic adaptive dnn surgery for inference acceleration on the edge," in *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*, 2019, pp. 1423–1431.

[8] S. D. Manasi, F. S. Snigdha, and S. S. Sapatnekar, "Neupart: Using analytical models to drive energy-efficient partitioning of cnn computations on cloud-connected mobile clients," *IEEE Transactions on Very Large Scale Integration Systems*, vol. 28, no. 8, pp. 1844–1857, 2020.

[9] Y. Mao *et al.*, "A privacy-preserving deep learning approach for face recognition with edge computing," in *Proc. USENIX Workshop Hot Topics Edge Comput.(HotEdge)*, 2018, pp. 1–6.

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

[11] A. Banitalebi-Dehkordi *et al.*, "Auto-Split: A General Framework of Collaborative Edge-Cloud AI," *Available online: https://arxiv.org/abs/2108.13041*, 2021.

[12] S. Lahmer, A. Khoshsirat, M. Rossi, and A. Zanella, "Energy consumption of neural networks on nvidia edge boards: an empirical model," in *2022 20th International Symposium on Modeling and Optimization in Mobile, Ad hoc, and Wireless Networks (WiOpt)*, 2022, pp. 365–371.

[13] Y. Huang, C. Qiu, X. Wang, S. Wang, and K. Yuan, "A compact convolutional neural network for surface defect inspection," *Sensors*, vol. 20, no. 7, 2020. [Online]. Available: https://www.mdpi.com/1424-8220/20/7/1974

[14] M. J. Neely, "Stochastic network optimization with application to communication and queueing systems," *Synthesis Lectures on Communication Networks*, vol. 3, no. 1, pp. 1–211, 2010.

[15] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge university press, 2004.

[16] P. Brito, F. Fabião, and A. Staubyn, "Euler, Lambert, and the Lambert W-function today," *The Mathematical Scientist*, vol. 33, January 2008.

[17] M. Sandler *et al.*, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 4510–4520.

[18] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 2009, pp. 248–255.