EFFECT-*DNN*: Energy-efficient Edge Framework for Real-time DNN Inference

Xiaojie Zhang, Motahare Mounesan, Saptarshi Debroy
Computer Science, City University of New York, New York, NY
Emails: {*xzhang6,mmounesan*}*@gradcenter.cuny.edu, saptarshi.debroy@hunter.cuny.edu*

***Abstract*—Real-time visual computing applications running Deep Neural Networks (DNN) are becoming popular for mission-critical use cases such as, disaster response, tactical scenarios, and medical triage that require establishing ad-hoc edge environments. However, strict latency deadlines of such applications require real-time processing of pre-trained DNN layers (i.e., DNN inference) involving image/video data which is highly challenging to achieve under such resource- constrained edge environments. In this paper, we address the trade-off between end-to-end latency of DNN inference and IoT devices' energy consumption by proposing 'EFFECT-*DNN*', an energy efficient edge computing framework. The EFFECT-*DNN* framework aims to strike such balance by employing a collaborative DNN partitioning and task offloading strategy. Such strategy also involves resource allocation from IoT devices and edge servers to satisfy DNN inference deadline requirement even when the network bandwidth is on the lower end, which is often the case for critical use cases. The underlying optimization is formulated as a dynamic Mixed-Integer Nonlinear Programming (MINLP) problem is decoupled and solved by convex optimization and a game-like heuristic algorithm. We evaluate the performance of EFFECT-*DNN* framework on a hardware testbed and using extensive simulations with real-world DNNs. The results demonstrate that the proposed framework can ensure DNN inference deadline satisfaction with significant ($\sim$ 20-30%) device energy savings.**

***Index Terms*—Deep neural networks, edge computing, task partitioning and offloading, resource allocation, energy efficiency.**

I. INTRODUCTION

Due to the proliferation of camera-enabled IoT devices over the past two decades, real-time visual computing has become popular, especially for mission-critical use cases that deploy ad-hoc edge computing environments. Such environments typically comprise of energy-constrained IoT devices, moderately powerful edge servers, wireless connectivity (often cellular 4G/5G) for data transfer between the devices and edge servers, and sometimes connectivity to distant cloud data centers. The key component for such visual computing applications are Deep Neural Networks (DNNs) that are pre-trained offline in cloud data centers with real-time/online inference within the edge environment during missions. The inherent complexity of the DNNs makes the computation for such inference time-consuming and energy-draining. Thus, on-device (i.e., local-only) computation of the entire DNN inference process (i.e., all DNN layers) is not always feasible as it would result in longer latency and violation of real-time latency requirements. At the same time such intensive computation might severely drain the energy-constrained IoT devices. Alternatively, offloading all DNN layers to the edge server for computation (i.e., remote-only) means sending all the DNN input images to the edge servers over low data rate uplink 4G/5G cellular connections that are commonly used for such ad-hoc edge systems. As such images are often considerable in their sizes, the remote only strategy might also lead to latency violation due to high transmission time of data upload.

This material is based upon work supported by the National Science Foundation under Award Number: CNS-1943338. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of the National Science Foundation.

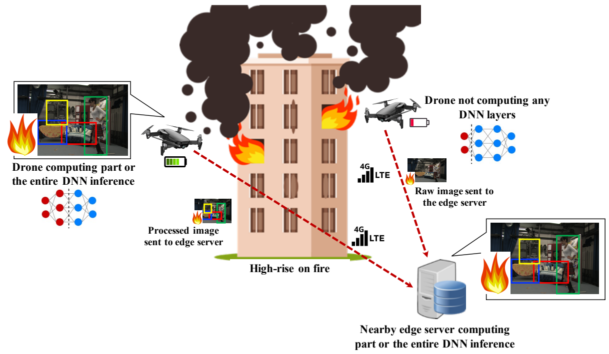

Fig. 1: An exemplary objection detection and classification related DNN partitioning scenario between drones and edge server for fire rescue use case

One common sense strategy to perform such DNN inference in a balanced manner is to implement partial task offloading between devices and edge servers, viz., DNN partitioning. In DNN partitioning, all DNN layers upto and including a certain *cut point* layer are processed at the IoT device. Next, the intermediate results generated by the cut point layer and the cut point layer information are sent to the edge server where DNN inference continues from the layer after the cut point. Under most naive approaches to balance end-to-end inference latency and device energy preservation, the number of DNN layers computed at the IoT device will depend on the device's available energy, i.e., devices with higher available energy will compute more layers. Fig. 1 demonstrates one exemplary fire rescue use case where a typical DNN inference for object detection and classification is partitioned between drones and a nearby edge server. The figure shows that one of the drones (with higher available energy) is computing all or part of the DNN layers and sending the processed data to the edge server over LTE links. Whereas, the other drone with lower available energy decides to offload all DNN layers to the edge servers. However, finding the optimal cut point layer for such DNN partitioning is not that straightforward.

Typically, DNNs for visual computing vary in terms of the number of layers, computational complexity of each layer, and the size of the output data after each layer. Thus, for any cut point layer, the transmission energy consumption (by the device) for offloading the output data after the cut point layer is a function of the size of that data. Cut points where the output data size is larger would result in considerable energy consumption and latency resulting in longer end-to-end latency. This problem is even more pronounced (i.e., considerably more energy expenditure and longer latency) for low data rate scenarios which is often the case for cellular uplink connections used under mission critical use cases. Thus, the choice of optimal cut point layer is of paramount importance in order to strike a balance between devices' energy consumption and end-to-end DNN inference latency. At the same time, computation resource allocation and bandwidth allocation between the device and the edge server also considerably impact the device energy expenditure, as well as end-to-end inference latency. Therefore, optimal DNN partitioning that seeks to compute such trade-off must be cognizant of the resource availability of the IoT device and the edge server, and the available cellular network bandwidth between the devices and the edge server.

In this paper, we strike the aforementioned balance by proposing EFFECT-*DNN*, an Energy-eFFicient Edge CompuTing framework for real-time DNN inference. The EFFECT-*DNN* framework balances the energy expenditure of IoT devices running heterogeneous DNNs and the end-to-end inference latency of such DNNs by: i) optimizing DNN partitioning, i.e., computing the optimal cut points of each DNN and ii) optimizing compute resource allocation at IoT devices and the edge server, and network resource allocation between the devices and edge server. The DNN partitioning, along with computation and network resource allocations are formulated as a long-term optimization problem that aims to minimize the long-term average device energy consumption and to satisfy the long-term average end-to-end latency requirement of DNN inference. The joint optimization problem to select the cut point and resource allocation is formulated as a complex dynamic Mixed-Integer Nonlinear Programming (MINLP) problem. We apply *Lyapunov* optimization by creating a virtual queue to express the long-term end-to-end latency requirements, followed by decoupling the resource allocation and cut point selection problems in runtime. The resource allocation problem is solved using convex optimization and a game-like heuristic algorithm is proposed for cut point selection based on observations made under extensive benchmarking experiments.

We evaluate the performance of the proposed EFFECT-*DNN* framework with three of the most popular visual computing DNNs, viz., AlexNet [1], VGG [2], and ResNet [3] using both hardware edge testbed based experiments and extensive simulations. Edge testbed based results demonstrate that under very low to barely acceptable data rate conditions between the devices and edge servers, EFFECT-*DNN* can jointly improve the end-to-end latency of DNN inference and energy preservation of IoT devices by 20.7% and 9.2% respectively. We further evaluate the impact of resource availability and number of IoT devices on DNN partitioning decisions through simulations. The results demonstrate that DNNs like AlexNet are more likely to choose partial offloading by enabling DNN partitioning, while others, such as ResNet and VGG prefer remote-only inference strategy due to their layer structures. In addition, the simulation results show that under moderate to high data rate conditions, EFFECT-*DNN* can achieve upto 41% of latency reduction and upto 32.5% improvement on device energy savings. Overall, such results validate the benefits of EFFECT-*DNN* framework in balancing energy efficiency and inference latency by optimizing DNN partitioning and resource allocation.

The remainder of this paper is organized as follows. Section II discusses the related work. Section III introduces problem evidence analysis. Section IV proposes our system model and formulates the problem. Section V discusses the solution strategy and algorithms. Section VI discusses testbed and simulation results. Section VII concludes the paper.

II. Related works

Task offloading as a solution for end-to-end latency minimization is popular within edge environments. Task offloading is first formulated as a deterministic problem that decides between local-only and remote-only computation plans [4], [5], [6]. Authors in [7] deploy a similar approach for task-offloading between edge servers and the cloud. Task offloading can also be formulated as a *computing while transmitting* problem (i.e., partial task offloading) that considers IoT devices as computational units [8], [9]. Expressing tasks as directed acyclic graphs (DAG) for computation resource allocation is another approach being adopted in the recent literature [10], [11], [12]. In this approach, the task components can be placed across the edge servers with some works, such as EFFECT [13] extend this to a more complex environment with multiple edge servers and multi-stage computations.

Partitioning of DNN layers is conceptually similar to task offloading of sequential DAGs as most of the popular DNN models consist of a sequence of layers with each layer triggering the next layer. Due to the variations in data size and computational requirements of each layer, the choice of cut point affects the end-to-end latency and energy consumption of IoT devices. There is extensive work in the literature for such cut-point prediction. A combination of regression models for predicting layer-wise performance and dynamic cut-points are proposed in [14], [15]. SPINN [16] uses the ratio between actual time and offline latency estimation to create a 2-staged linear model which predicts inference latency. MoDNN [17] partitions the DNN in layers and then maps different parts of a layer onto mobile devices to accelerate the computations. A scalable partitioning of convolutional layers along with a novel work scheduling are proposed in [18] to minimize memory footprint and reduce overall execution latency.

As an alternative to partitioning, a different approach of DNN compression is being proposed in recent times. Here, two dedicated DNN models with different accuracy levels are considered; a heavy-weight high accuracy DNN for servers and a compressed light-weight version for resource-constrained devices. A joint offloading and resource management problem is typically solved to decide between the version with subsequent resource allocation. Task-specific applications cache the compressed DNNs in the IoT devices and store the heavy-weight model in the servers [19][20]. However, in applications with a variety of tasks, an extra DNN placement step may be required to ensure that the light-weight DNN is available in the IoT device [21]. Authors in [22] categorize the DNN compression techniques into 4 major groups with different extents of accuracy compromises, viz., network pruning, sparse representation, bits precision, and knowledge distillation. *Unlike these works, we focus on applications that cannot tolerate the accuracy compromise.*

More recent frameworks brought the importance of resource allocation under constrained environments, such as edge into attention.Authors in [23] apply an iterative alternating optimization algorithm in a realistic multi-user resource-constrained environment. An optimization-based joint self-adaptive DNN partitioning and cost-aware resource allocation is proposed in [24]. In [25], authors pair a multi-exit DNN inference acceleration framework with a Deep Reinforcement Learning (DRL) based policy to make joint decisions about DNN partitioning and resource allocation. *However, to the best of our knowledge, the energy constraints of IoT devices have not been taken into account in such works.*

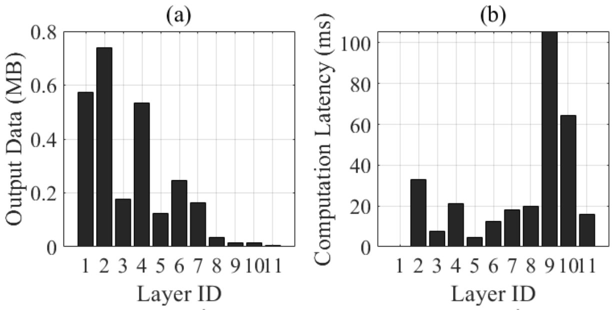

Fig. 2: Layer-wise data size and inference time for AlexNet

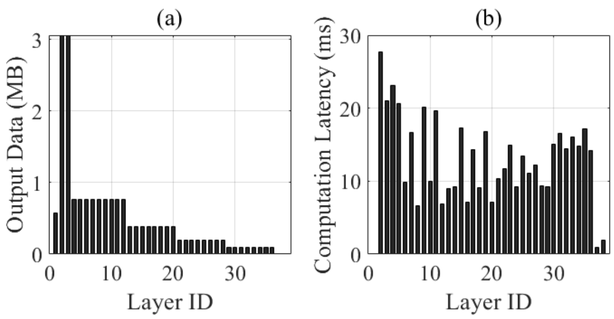

Fig. 3: Layer-wise data size and inference time of ResNet

III. Problem evidence analysis

In order to capture the layer-specific characteristics of real-world DNN inference and data transmission, we perform benchmarking experiments on popular DNNs. The experiments are performed on a hardware device-server testbed with NVIDIA Jetson TX2's mimicking computationally-capable IoT devices and a Dell PoweEdge desktop with 16 cores and CPU frequency 3.2 GHz mimicking the edge server. It is to be noted that NVIDIA Jetson TX2 is a fast embedded AI computing device that is the most power-efficient among its similar models. Similar measurements with a less powerful computing device will only increase the end-to-end latency and energy consumption. for the experiments, we select and analyze three well-known neural networks, viz., AlexNet [1], ResNet [3], and VGG [2]. These neural networks are widely used in mission-critical emergency response and public safety use cases for applications, such as object detection, image recognition, and image classification and thus are good representatives for real-time DNN inference. We also extend the such experiments on collaborative DNN inference to ascertain the potential benefits of DNN petitioning. Here, by collaborative inference, we mean the IoT device and edge server computing parts of the DNN layers in a collaborative manner.

A. Layer Characteristics

Before measuring the characteristics of individual DNN layers, we integrate lightweight layers (e.g., activation function) with their adjacent layers (e.g., convolutional) which are compute-intensive. By doing such integration, we can reduce the number of layers and thus the complexity of the optimization problem (later proposed in Section IV). The results of output data size and computation latency of each layer when running the DNN on IoT devices (i.e., NVIDIA TX2) are shown in Figs. 2, 3, and 4 respectively for different DNNs. The results offer certain useful insights on layer specific DNN characteristics. First, different DNNs have different number of layers with the output data size and inference time of each layer varying greatly. Capturing such layer differences is important for deciding how many layers should be executed locally for latency and energy consumption minimization. Secondly, for the purpose of reducing transmission cost (in terms of time, energy, or both), the DNNs can have fixed number of options for layer partitioning. As shown in Fig. 2, AlexNet can have considerable reduction in output data after the execution of layers 3, 5, and 8. Considering that the local computation cost of layers before such layers is very limited, AlexNet is a potential candidate to obtain benefit from DNN partitioning. However, for ResNet and VGG, the output data is not reduced until the 13th layer (shown in Figs. 3 and 4) respectively. Thus, we can surmise that in most cases, IoT devices running VGG and ResNet will be unwilling to execute any of the layers locally as the local computation cost of its first 13 layers is very high. Therefore, they will require considerable network and computational resources at the server-side.

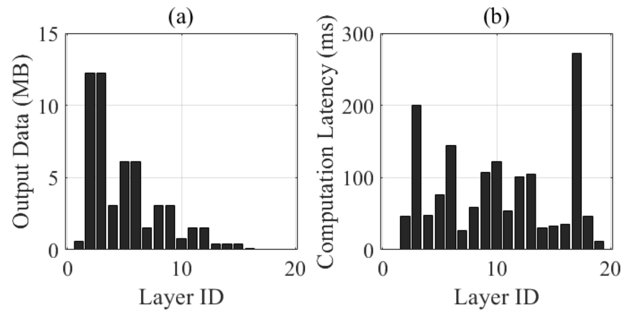

Fig. 4: Layer-wise data size and inference time of VGG

B. Potential Benefits of DNN Partitioning

Here, we seek to examine the potential improvement in the end-to-end latency and energy consumption of collaborative DNN inference enabled by layer partitioning. As mentioned before, the layers are divided into local-layers and remote-layers based on a given partition location, called *cut point*. Through these experiments, we intend to observe how the data rate obtained by IoT devices affects such collaborations. In these measurements, upload data rates between the devices and server are tuned to 8 Mbps and 20 Mbps, simulating different network scenarios, e.g., low data rate LTE connectivity and acceptable data rate WiFi connectivity.

Figure 5 shows the end-to-end latency and energy consumption of an IoT device running the AlexNet using different cut points. In Fig. 5(a) and Fig. 5(b), the best cut points in terms of end-to-end latency are layers 8 and 3, respectively. However, Fig. 5(c) and Fig. 5(d) show that if energy consumption is the concern, then the optimal cut points are probably layers 3 and 1 (for data rates 8 Mbps and 20 Mbps respectively). However, for both ResNet and VGG, the computational requirements are much higher than AlexNet, especially for VGG. Based on the results shown in Fig. 6, all layers of ResNet should be executed either locally (Fig. 6(a)) or remotely (Fig. 6(b-d)). While VGG layers should be executed fully remotely in all cases, both in terms of end-to-end latency and energy consumption (as shown in Fig 7). *From these observations, we argue that for energy-constrained IoT devices running latency-sensitive applications, there is often a trade-off between end-to-end latency and energy consumption. Additionally, although the cut point selection great impacts the end-to-end latency and energy consumption, such selection should also be cognizant of availability of resources in order to be practical.*

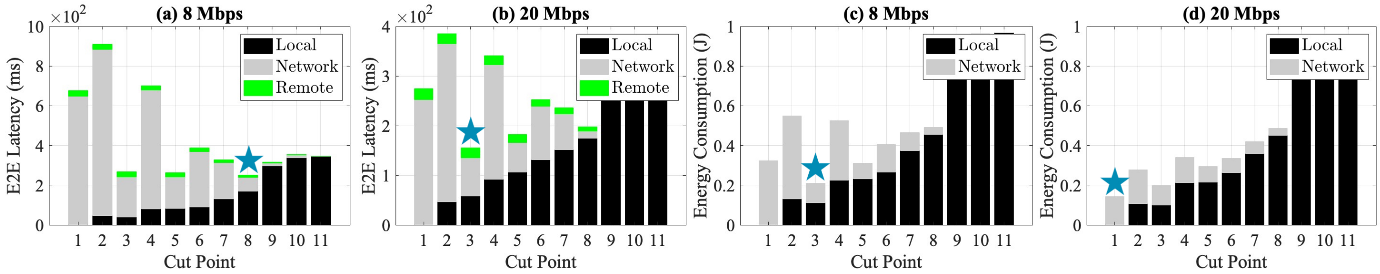

Fig. 5: The end-to-end latency and energy consumption of AlexNet running collaboratively between TX2 and edge server under a given cut point.

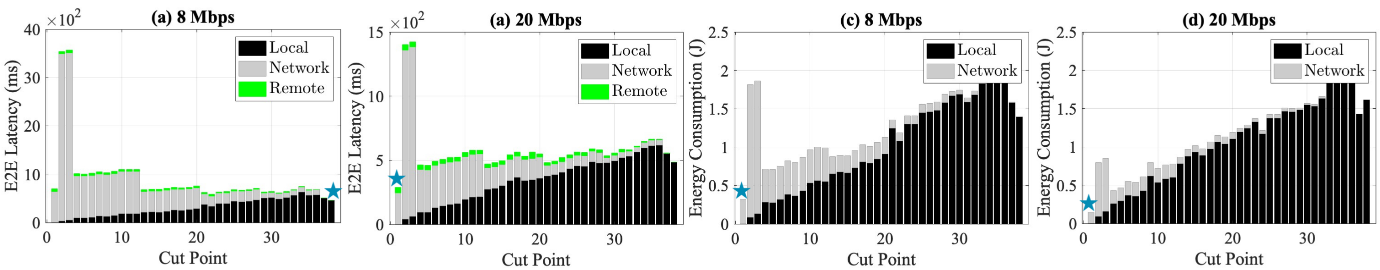

Fig. 6: The end-to-end latency and energy consumption of ResNet running collaboratively between TX2 and edge server under a given cut point.

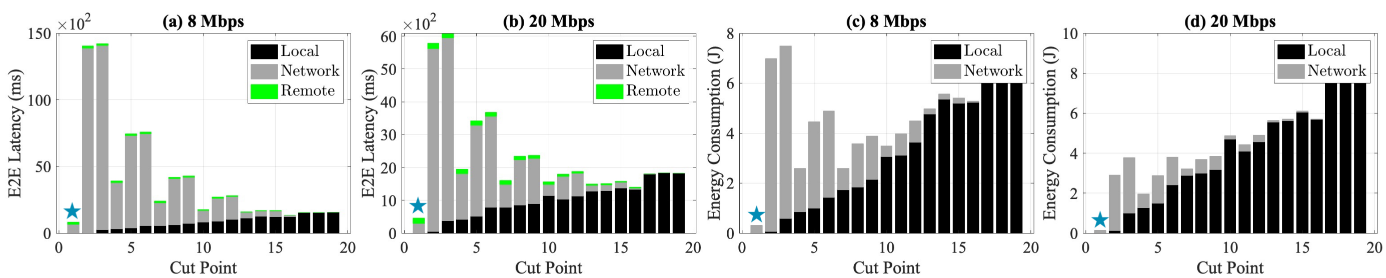

Fig. 7: The end-to-end latency and energy consumption of VGG running collaboratively between TX2 and edge server under a given cut point.

IV. System Model and Problem Formulation

Here, we describe the system model for our edge-native and real-time DNN inference, along with optimization problem formulation for cut point selection and resource allocation. First, we define $\mathcal{M} = \{1, 2, ..., M\}$ as a set of M IoT devices with each device m running a specific type of DNN (let us call it type m). The DNN of type m consists of a set of I_m layers, denoted by $\mathcal{I}_m = \{1, 2, ..., I_m\}$ with the first layer being the input. We assume that the DNN (or the related application) requested by m-th IoT device has a strict performance requirement, i.e., its end-to-end inference latency can not be greater than τ_m (measured in seconds for our analysis). As explained earlier, for our model, the DNN layers are divided into local-layers and remote-layer under a given cut point, and the two set of layers are executed sequentially by the IoT device and edge server respectively. Figure 8 describes EFFECT-*DNN* framework's proposed collaborative DNN inference model where parts of the DNN within the blue and red boxes signify layers computed in-device and at edge server respectively. In order to accommodate multiple heterogeneous IoT devices in terms of the DNN models being run on them and device resource capacity, the overall resource provisioning for DNN model computation should be optimized. The resource provisioning is in terms of on-demand: i) computation resource allocation from the devices to execute local layers, ii) network resource allocation for transmission of intermediate data generated by local-layers, and iii) computation resource allocation from the edge server for execution of remote-layers.

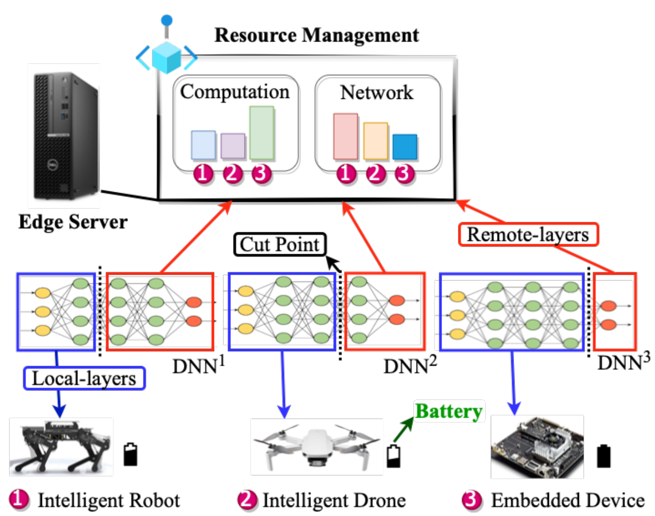

Fig. 8: EFFECT-*DNN* system architecture with collaborative DNN inference

A. DNN Model

For each DNN, we denote the size of output data of layer $i \in \mathcal{I}_m$ as $d_{m,i}$ (measured in bits) and the computational complexity of that layer as $x_{m,i}$ (measured in terms of the number of CPU cycles or simply the CPU time). As layers before a given cut point (i.e., layers 1 to i) are executed locally on IoT devices (i.e., local-layers)) and layers after (i.e., layers i to I_m) are executed remotely on the edge server (i.e., remote-layers), the accumulated computational complexity of local-layers and remote-layers are defined as:

$$X_{m,i}^{IoT} = \sum_{j=1}^{i} x_{m,j}, \; X_{m,i}^{Edge} = \sum_{j=i+1}^{I_m} x_{m,j}$$

Important to mention that there exist two extreme cases where the layers are executed either entirely on IoT device or entirely on the edge server, called local-only and remote-only strategies respectively. Correspondingly, $i = 1$ refers to remote-only inference and $i = I_m$ refers to local-only inference. Given I_m possible cut points on the DNN, we define $p_{m,i} =< X_{m,i}^{IoT}, d_{m,i}, X_{m,i}^{Edge} >$ as the DNN execution profile for the collaborative inference of the m-th IoT device selecting cut point i with $\mathcal{P}_m = \{p_{m,i} | i \in \mathcal{I}_m\}$ being the set of available execution profiles. Intuitively, the IoT devices should select the cut point that minimizes their transmission cost compared to remote-only inference, i.e., $\arg\max_i (d_{m,1} - d_{m,i})$. However, in practice, they also need to pay attention to the cost of the local-layers (local latency and energy consumption). Therefore, an optimal cut point is the outcome of the ideal trade-off between the local and transmission costs.

In this work, we allow the IoT devices (i.e., applications running on them) to select the optimal cut point based on their own requirements. Let us denote $o_{m,i} \in \{0, 1\}$ as the profile selection indicator. If $o_{m,i} = 1$, profile $p_{m,i}$ is chosen; otherwise $o_{m,i} = 0$. Since each IoT device can only choose one profile, we have the following constraint

$$\sum_{i=1}^{I_m} o_{m,i} = 1, \; \forall m \in \mathbf{M} \tag{1}$$

B. Communication Model

We assume that IoT devices are connected to the edge server using access point (AP) or base stations (BS) depending on the type of wireless communication. The AP has a fixed amount of network resources, denoted by B. In this paper, we do not limit the format of resources and thus B can be manifested through either bandwidth (measured in MHz), or number of channels/sub-channels, or simply point to the data rate (in Mbps). Irrespective of such manifestation, in our model the network resource can be partitioned and allocated to IoT devices. We define b_m as the ratio of network resource allocated to the m-th IoT device with the constraint:

$$\sum_{m=1}^{M} b_m = 1 \tag{2}$$

Due to the heterogeneity in channel noise, signal fading, transmission power, and other radio resource characteristics, our model acknowledges the reality that different IoT devices can have different network delay between the device and the edge server even when the devices are allocated the same amount of network resources. For a given DNN execution profile $p_{m,i} =< X_{m,i}^{IoT}, d_{m,i}, X_{m,i}^{Edge} >$, the network delay is modeled as a function of b_m and data size $d_{m,i}$, such that,

$$T_{m,i}^{Data} = \frac{d_{m,i}}{\alpha_m b_{m,i} \times B}$$

where α_m is a coefficient related to channel noise and the transmit power of IoT devices. The energy consumption caused by data transmission can be computed as:

$$E_{m,i}^{Data} = \beta_{m,i} \times T_{m,i}^{Data}$$

where $\beta_{m,i}$ is the transmission power used in the transmission period. The transmission power may vary based on the edge system's adoption of the underlying communication technologies, such as LTE and WiFi.

C. Local Computation Model

In our model, we do not limit the format of IoT devices' computation resources, as the implementation of computation resource allocation can be manifested in different ways, such as, CPU frequency allocation [13], or CPU core allocation [23], or GPU time allocation [26]. As the IoT devices perform the computation of local-layers (i.e., from layer 1 to i_m), the local computation time with $f_m^{IoT} \in (0, F_m^{IoT}]$ being the available CPU speed (i.e., CPU cycles per second) of device m, can be expressed as:

$$T_{m,i}^{IoT} = \frac{X_{m,i}^{IoT}}{f_m^{IoT}} \tag{3}$$

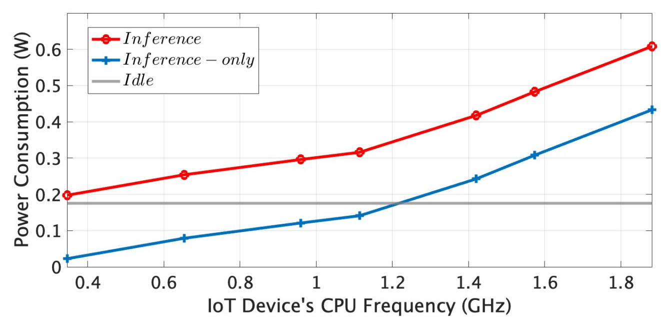

Fig. 9: Power consumption characteristics of Jetson TX2

Typically for IoT devices, the energy consumption is closely related to their computation speed. In general, energy consumption increases with CPU frequency and the magnitude of such increase may vary between different types of IoT devices. As an exemplary scenario, in Fig. 9, we show the power consumption characteristics of NVIDIA Jetson TX2 (in Watts or Joules/s) when it executes the DNN and when it remains idle albeit being power on. With these two power characteristics, we can compute the power consumption that is caused by only the inference part (shown in blue in Fig. 9). For TX2, the inference-only power consumption characteristics thus can be fitted to $\epsilon_{m,i} = \kappa (f_m^{IoT})^2$, where κ is a constant related to the chip architecture. Consequently, the actual energy spent by IoT computation can be computed as:

$$E_{m,i}^{IoT} = \epsilon_{m,i} \times T_{m,i}^{IoT}$$

where $E_{m,i}^{IoT}$ is measured in Joules.

D. Server Computation Model

We assume that the edge server has a fixed amount of computational resources, denoted by C. Similar to the case of IoT devices, our model is agnostic of edge server's computation resource format. We define c_m as the ratio of computational resources that is allocated to the IoT device m. Thus, computational resource allocation follows this constraint:

$$\sum_{m=1}^{M} c_m = 1 \tag{4}$$

Therefore, the server computation time is computed by:

$$T_{m,i}^{Edge} = \frac{X_{m,i}^{Edge}}{c_m \times C} \tag{5}$$

It is important to mention that in our work we assume the edge servers to be connected to continuous power source and thus server's energy consumption is ignored for the resource allocation optimization problem.

E. Problem Formulation

Since real-world DNN inference and data transfer can generate some randomness in the end-to-end latency and devices' energy consumption, we formulate a long-term optimization problem that aims to minimize the long-term average energy consumption of all connected IoT devices in the service area and to ensure that the long-term average end-to-end latency is no greater than threshold τ_m. We define time epochs as $\mathcal{T} = \{1, 2, ..., T\}$ and add symbol t as one of the subscripts of the previously introduced parameters to denote their actual value at a certain time epoch. With such model parameters and assumptions, the end-to-end latency of m-th IoT device in epoch t can be expressed as:

$$L_{m,t} = \sum_{i=1}^{I_m} o_{m,i,t} \times (T_{m,i,t}^{IoT} + T_{m,i,t}^{Data} + T_{m,i,t}^{Edge}) \tag{6}$$

and the average energy consumption of all IoT devices as:

$$E_t = \frac{1}{M} \sum_{m=1}^{M} \sum_{i=1}^{I_m} o_{m,i,t} \times (E_{m,i,t}^{Data} + E_{m,i,t}^{IoT}) \tag{7}$$

Thus, the proposed problem is stated as follows:

$$\begin{aligned}
&\min_{c,b,o,f^{IoT}} \frac{1}{T} \sum_{t=1}^{T} E_t \\
&\text{s.t. } \mathbf{C1:} \sum_{m=1}^{M} b_{m,t} = 1 \\
&\quad \mathbf{C2:} \sum_{m=1}^{M} c_{m,t} = 1 \\
&\quad \mathbf{C3:} \sum_{i=1}^{I_m} o_{m,i,t} = 1, \ \forall m \in \mathbf{M} \\
&\quad \mathbf{C4:} f_{m,t}^{IoT} \in (0, F_m^{IoT}] \\
&\quad \mathbf{C5:} \frac{1}{T} \sum_{t=1}^{T} L_{m,t} \leq \tau_m, \forall m \in \mathcal{M} \\
&\quad \mathbf{C6:} o_{m,i,t} \in \{0, 1\}
\end{aligned} \tag{\textbf{P1}}$$

V. Solution Strategy

Problem (**P1**) is a dynamic Mixed-Integer Nonlinear Programming (MINLP) problem which is NP-hard. In this paper, we first use Lyapunov optimization to address the dynamic nature of problem (**P1**). Afterwards, we apply decomposition that decouples the problem of cut point selection (i.e., integer variable selection) from the problem of resource allocation (i.e., continuous variable selection). The solution to problem (**P1**) is obtained by alternatively solving the two individual sub-problems.

A. Lyapunov optimization

For the Lyapunov optimization, we create a virtual queue to alternatively express long-term constraint **C5** and the long-term objective function of **P1**. Since during T optimization epochs, the long-term average end-to-end latency is no greater than threshold τ_m, we consider τ_m as the expected departure and arrival (i.e., virtual) in each epoch. Based on the difference between these two, a virtual queue is constructed. We define $Q_{m,t} \geq 0$ as the length of the virtual queue at epoch t, therefore, the queue dynamics is given by:

$$Q_{m,t+1} = \Big[Q_{m,t} + (L_{m,t} - \tau_m)\Big]^+ \tag{8}$$

Based on the evolution of the virtual queue, the queue length increases if the current end-to-end latency is greater than threshold τ_m; otherwise, the queue decreases. Therefore, the queue length specifies the distance to the latency τ_m, which indicates how strictly the long-term constraint **C5** in problem **P1** is satisfied. Now, an equivalent expression for **C5** is needed to shorten and stabilize the virtual queue for each IoT device.

In order to stabilize the queue, we need to find a policy that can balance the stability of the queue and the energy consumption of IoT devices. We introduce the quadratic Lyapunov function and the Lyapunov drift function as:

$$\mathcal{L}(Q_{m,t}) = \frac{1}{2}(Q_{m,t})^2$$

and

$$\Delta(Q_{m,t}) = \mathbb{E}\Big[\mathcal{L}\big(Q_{m,t+1}\big) - \mathcal{L}\big(Q_{m,t}\big)|Q_{m,t}\Big]$$

where $\Delta(Q^t)$ measures the difference in $\mathcal{L}(\cdot)$ between two adjacent epochs, i.e., t and $t+1$. Subsequently, such difference can be computed as:

$$\begin{aligned}
\mathcal{L}\big(Q_{m,t+1}\big) - \mathcal{L}\big(Q_{m,t}\big) &= \frac{1}{2}(Q_{m,t+1})^2 - \frac{1}{2}(Q_{m,t})^2 \\
&\leq C + Q_{m,t} \times \Big(L_{m,t} - \tau_m\Big)
\end{aligned}$$

where $C = \frac{1}{2}\Big(L_{m,t}\Big)^2 + \frac{1}{2}\Big(\tau_m\Big)^2$ and therefore:

$$\Delta(Q_{m,t}) \leq C + \mathbb{E}\Big[Q_{m,t} \times \Big(L_{m,t} - \tau_m\Big)|Q_{m,t}\Big]$$

where C is bounded to $\Big(L_{m,t}\Big)^2$.

Thus, problem (**P1**) is transformed into multiple deterministic and identical per-epoch problems that opportunistically minimize expected values. This transformed problem can be stated as:

$$
\begin{aligned}
&\min_{c,b,o,f^{IoT}} V \times E_t + \frac{1}{M}\sum_{m=1}^{M} Q_{m,t} \times \Big(L_{m,t} - \tau_m\Big) \\
&\text{s.t. } \textbf{C1:} \sum_{m=1}^{M} b_{m,t} = 1 \\
&\quad \textbf{C2:} \sum_{m=1}^{M} c_{m,t} = 1 \\
&\quad \textbf{C3:} \sum_{i=1}^{I_m} o_{m,i,t} = 1 \\
&\quad \textbf{C4:} f_{m,t}^{IoT} \in (0, F_m^{IoT}] \\
&\quad \textbf{C5:} o_{m,i,t} \in \{0,1\}
\end{aligned} \qquad (\textbf{P2})
$$

where V is a factor that balances the trade-off between energy consumption and end-to-end latency e.g., $([O(1/V), O(V)])$ [9]. The value of V needs to be properly selected based on the system objective/preference (i.e., energy-sensitivity, latency-sensitivity, or a combination of the two). However, problem (**P2**) is still non-trivial to solve because of its MINLP nature. It is well known that solving such NP-hard problems with closed-form expressions is very challenging. To solve problem (**P2**) efficiently, we decompose problem (**P1**) into two sub-problems: 'Resource allocation' and 'Cut point selection', respectively. The descriptions of the two sub-problems are stated as follows:

1) Resource allocation: For a given set of selected execution profiles i.e., $o_{m,i,t}, \ \forall m \in \mathcal{M}$, the resource allocation problem is reduced to a convex optimization problem, which is solved by a Lagrangian method.
2) Cut point selection: When resource allocation strategies are given, the selection of cut point (i.e., execution profile) is a 0-1 integer linear programming problem. In this paper, a heuristic algorithm is proposed to solve the selection of cut point.

B. Resource Allocation

Once the execution profile $o_{m,i,t}$ is selected, problem (**P2**) is reduced to the following resource allocation problem:

$$
\begin{aligned}
&\min_{c,b,f^{IoT}} V \times E_t + \frac{1}{M}\sum_{m=1}^{M} Q_{m,t} \times \Big(L_{m,t} - \tau_m\Big) \\
&\text{s.t. } \textbf{C1:} \sum_{m=1}^{M} b_{m,t} = 1 \\
&\quad \textbf{C2:} \sum_{m=1}^{M} c_{m,t} = 1 \\
&\quad \textbf{C4:} f_{m,t}^{IoT} \in (0, F_m^{IoT}]
\end{aligned} \qquad (\textbf{P3})
$$

We first seek to prove that problem (**P3**) is strictly convex $w.r.t$ resource variables (i.e., $c_{m,t}, b_{m,t}, f_{m,t}^{IoT}$). To this end, we define the following Lagrange function of problem (**P3**):

$$
\begin{aligned}
\mathcal{L} = {} & V \times \frac{1}{M}\sum_{m=1}^{M}(E_{m,i,t}^{Data} + E_{m,i,t}^{IoT}) \\
& + \frac{1}{M}\sum_{m=1}^{M} Q_{m,t} \times \Big(L_{m,t} - \tau_m\Big) \\
& + w_b \times (\sum_{m=1}^{M} b_{m,t} - B) \\
& + w_c \times (\sum_{m=1}^{M} c_{m,t} - C)
\end{aligned} \qquad (9)
$$

where variables w_b and w_c are Lagrangian multiplier that are associated with network and computational resource allocation constraints, respectively. With the help of $\mathcal{L}$, the following optimal allocation strategies are derived.

1) Edge Server Computational Resource Allocation: For edge server's computational resource allocation $c_{m,t}$, we have:

$$
\frac{\nabla \mathcal{L}^2}{\nabla c_{m,t} c_{n,t}} = \begin{cases} \frac{1}{M} \times Q_{m,t} \times \frac{X_{m,i}^{Edge}}{C} \times \frac{2}{(c_{m,t})^3} & m = n \\ 0 & m \neq n \end{cases}
$$

Here, the Hessian matrix $\mathbf{H} = (\frac{\nabla \mathcal{L}^2}{\nabla c_{m,t} c_{n,t}})_{M \times M}$ is symmetric and positive definite [27], making problem (**P3**) strictly convex $w.r.t$ variable $c_{m,t}$. Based on KKT conditions [13], the optimal computational resource allocation $c_{m,t}^*$ can be obtained by solving the following expression:

$$
\frac{\nabla \mathcal{L}}{\nabla c_{m,t}} = -\frac{1}{M} \times Q_{m,t} \times \frac{X_{m,i}^{Edge}}{C} \times \frac{1}{(c_{m,t})^2} + w_c = 0
$$

The optimal $c_{m,t}^*$ is:

$$
c_{m,t}^* = \left[\frac{\frac{1}{M} \times Q_{m,t} \times X_{m,i}^{Edge}}{C \times w_c}\right]^{\frac{1}{2}}
$$

Since $\sum_{m=1}^{M} c_{m,t} = 1$ and w_c is a shared variable among all IoT devices, we have:

$$
\frac{c_{m,t}^*}{c_{n,t}^*} = \left[\frac{Q_{m,t} \times X_{m,i}^{Edge}}{Q_{n,t} \times X_{n,i}^{Edge}}\right]^{\frac{1}{2}} \qquad (10)
$$

which signifies that the allocation of computational resources can only be determined by the computation requirement of remote-layers and the current queue length. More specifically, $c_{m,t}^*$ is proportional to the square root of the product of these two parameters. Therefore, we can express:

$$
c_{m,t}^* = \begin{cases} \frac{\sqrt{Q_{m,t} \times X_{m,i}^{Edge}}}{\sum_{n=1}^{M}\sqrt{Q_{n,t} \times X_{n,i}^{Edge}}} & 1 \leq i < I_m \\ 0 & i = I_m \end{cases} \qquad (11)
$$

2) Device-Edge Network Resource Allocation: Similarly, for network resource allocation $b_{m,t}$ between the IoT device and edge server, we have:

$$
\frac{\nabla \mathcal{L}^2}{\nabla b_{m,t} b_{n,t}} = \begin{cases} \frac{1}{M} \times \frac{d_{m,i}}{\alpha_m} \times \frac{2}{(b_{m,t})^3} \times (Q_{m,t} + V\beta_{m,i}) & m = n \\ 0 & m \neq n \end{cases}
$$

Since this Hessian matrix $\mathbf{H} = (\frac{\nabla \mathcal{L}^2}{\nabla b_{m,t} b_{n,t}})_{M \times M}$ is also symmetric and positive definite, problem (**P3**) is also strictly convex $w.r.t$ allocated network resource ratio $b_{m,t}$. Similar to the previous analysis, the optimal network resource allocation $b^*_{m,t}$ can be obtained (based on the KKT condition) by solving the following expression:

$$\frac{\nabla \mathcal{L}}{\nabla b_{m,t}} = -\frac{1}{M} \times \frac{d_{m,i}}{\alpha_m \times (b_{m,t})^2} \times (Q_{m,t} + V\beta_{m,i}) + w_b = 0$$

with optimal $b^*_{m,t}$ being:

$$b^*_{m,t} = \left[\frac{1}{M} \times \frac{d_{m,i} \times (Q_{m,t} + V\beta_{m,i})}{\alpha_m \times w_b} \right]^{\frac{1}{2}}$$

Since $\sum_{m=1}^{M} b_{m,t} = 1$ and w_c is a shared variable among all IoT devices, the allocation of network resource ratio is determined by the data transmission requirement $d_{m,i}$, current queue length $Q_{m,t}$, power consumption $\beta_{m,i}$, and the network coefficient a_m of IoT devices. Therefore, we can express:

$$b^*_{m,t} = \begin{cases} \frac{\sqrt{\frac{1}{\alpha_m} \times d_{m,i} \times (Q_{m,t} + V\beta_{m,i})}}{\sum\limits_{n=1}^{M} \sqrt{\frac{1}{\alpha_n} \times d_{n,i} \times (Q_{n,t} + V\beta_{n,i})}} & 1 \le i < I_m \\ 0 & i = I_m \end{cases} \tag{12}$$

3) Local Computation Resource Allocation: When IoT device m selects $o_{m,I_m,t} = 1$ (local-only inference), it should use a computation speed that would just meet its end-to-end latency requirement due to the underlying objective of energy saving. In this case, the optimal computation speed (as a manifestation of compute resource) which is also the minimum computation speed can be expressed as:

$$f^{IoT*}_{m,t} = \min\{\frac{X^{IoT}_{m,i}}{\tau_m}, F^{IoT}_m\}$$

Therefore, the entire DNN can be executed locally if and only if the computation requirement of the entire DNN satisfies $X^{IoT}_{m,i} \le F^{IoT}_m \times \tau_m$. Based on aforementioned discussions, the minimum energy consumption $w.r.t.$ the end-to-end latency constraint (**C5**) can be computed by:

$$E_{m,I_m,t}{}^* = \kappa \times (\frac{{X^{IoT}_{m,i}}^2}{\tau_m}) \tag{13}$$

For this analysis, Eq. (13) gives the baseline for energy consumption, i.e., the IoT device would like to select $o_{m,i,t} = 1$ and $i < I_m$ (remote-only or DNN partition) if and only if its energy consumption can be less than $E_{m,I_m,t}{}^*$ or the it cannot finish executing the entire DNN within τ_m amount of time even using its maximum computation speed. In the latter case, its energy consumption would inevitably be greater than the baseline $E_{m,I_m,t}{}^*$. As for the optimal computation speed with DNN partitioning decisions ($o_{m,i,t} = 1$ and $1 < i < I_m$), we have:

$$\frac{\nabla \mathcal{L}^2}{\nabla f^{IoT}_{m,t} f^{IoT}_{n,t}} = \begin{cases} \frac{1}{M} \times Q_{m,t} \times X^{IoT}_{m,i} \times \frac{2}{(f^{IoT}_{m,t})^3} & m = n \\ 0 & m \ne n \end{cases}$$

Since this Hessian matrix $\mathbf{H} = (\frac{\nabla \mathcal{L}^2}{\nabla f^{IoT}_{m,t} f^{IoT}_{n,t}})_{M \times M}$ is symmetric and positive definite, problem (**P3**) is also strictly convex $w.r.t.$ local computation speed $f^{IoT}_{m,t}$. The optimal $f^{IoT*}_{m,t}$ can be obtain at $\frac{\nabla \mathcal{L}}{\nabla f^{IoT}_{m,t}} = 0$, which is stated as:

$$\frac{\nabla}{\nabla f^{IoT}_{m,t}} = -Q_{m,t} \times X^{IoT}_{m,i} \times \frac{1}{(f^{IoT}_{m,t})^2} + V\kappa X^{IoT}_{m,i} = 0 \tag{14}$$

Algorithm 1: Cut Point Selection Algorithm

1 Initial a set of execution profiles:
2 $\mathcal{O}(t=0) = \{o_{m,I_m,t=0} = 1, o_{m,i,t=0} = 0 | \forall m \in \mathcal{M}, \forall i \in \mathcal{I}_m \backslash \{I_m\}\}$
3 **while** $t \in \mathcal{T}$ **do**
4   Randomly select an IoT device $m \in \mathcal{M}$
5   **for** $i \in \mathcal{I}_m$ **do**
6   Set $o_{m,i,t} = 1$, and $o_{m,j,t} = 0, \forall j \in \mathcal{I}_m \backslash \{i\}$
7   Update resource allocation based on Eq. (11), Eq. (12) and Eq. (15)
8   Compute utility $U_{m,i,t}$
9   $i = \arg\max_{i \in \mathcal{I}_m} U_{m,i,t}$
10   Update $\mathcal{O}(t)$:
11   set $o_{m,i,t} = 1$, and $o_{m,j,t} = 0, \forall j \in \mathcal{I}_m \backslash \{i\}$
12   Update resource allocation based on Eq. (11), Eq. (12) and Eq. (15)
13   Run DNN inference and observe end-to-end latency $L_{m,t}, \forall m \in \mathcal{M}$
14   Update queue length based on Eq. (8)
15   $t \leftarrow t + 1$

By solving Eq. (14), we get:

$$f^{IoT*}_{m,t} = \begin{cases} \min\{\left[\frac{Q_{m,t}}{V\kappa}\right]^{\frac{1}{2}}, F^{IoT}_m\} & 1 < i < I_m \\ 0 & i = 1 \\ \min\{\frac{X^{IoT}_{m,i}}{\tau_m}, F^{IoT}_m\} & i = I_m \end{cases} \tag{15}$$

C. Cut Point Selection

Once the optimal resource allocation strategies are obtained for a given set of selected execution profiles, we next aim to address the problem of cut point selection that determines the execution profile. Since (**P1**) is a long-term optimization problem, the solution should be adapted based on observation. Moreover, the solution needs to be generated quickly so that the execution of actual DNN inference can commence. Based on these two requirements, we propose a heuristic algorithm to solve the selection of cut point during the run time. The proposed algorithm is explained in Alg. (1).

In Algo. 1, we first initialize a set of execution profiles $\mathcal{O}(t = 0)$, where all IoT devices select local-only inference strategy, i.e., $i = I_m$ (line 2). Afterward, for each optimization epoch, we randomly select an IoT device $m \in \mathcal{M}$ (Line 4). Then, by fixing the cut points of other IoT devices, we let this IoT device select its cut point that maximizes its utility $U_{m,i,t}$ (Line 5-9). Finally, with the updated set of execution profiles $\mathcal{O}(t)$, the edge server performs resource allocation based on Eq. (11), Eq. (12), and Eq. (15) when all DNN inferences start executing using the allocated resources. Based on the observed end-to-end latency $L_{m,t}, \forall m \in \mathcal{M}$, the queue length of IoT devices is updated according to Eq. (8) (Line 10-14). This process of cut point selection is motivated by game-based solutions proposed in [28], [29], and the utility function (line 8) is defined as follows:

$$U_{m,i,t} = -\Big(V \times E_t + \frac{1}{M} \sum_{m=1}^{M} Q_{m,t} \times \Big(L_{m,t} - \tau_m\Big)\Big)$$

which reflects the objective function problem (**P2**).

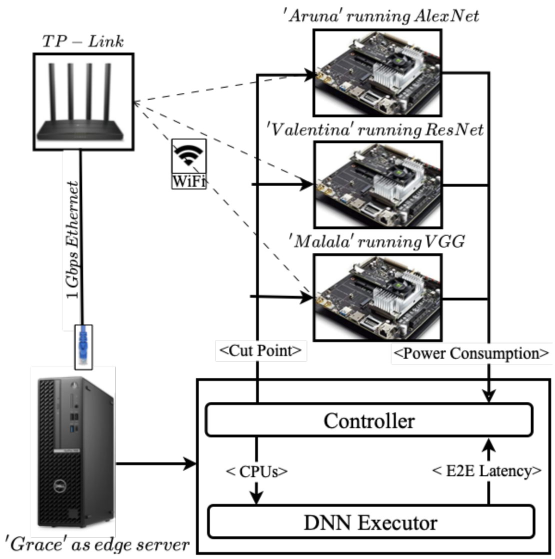

Fig. 10: Hardware testbed setup for EFFECT-*DNN* evaluation

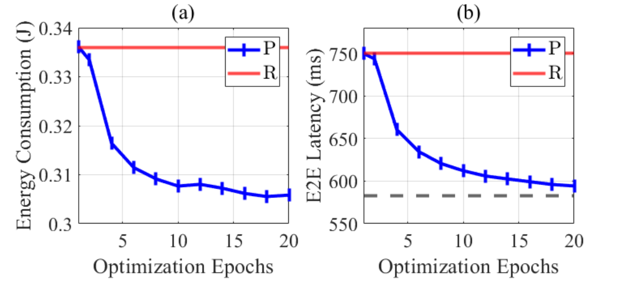

Fig. 11: Energy consumption and end-to-end latency of IoT devices with acceptable data rate (24 Mbps)

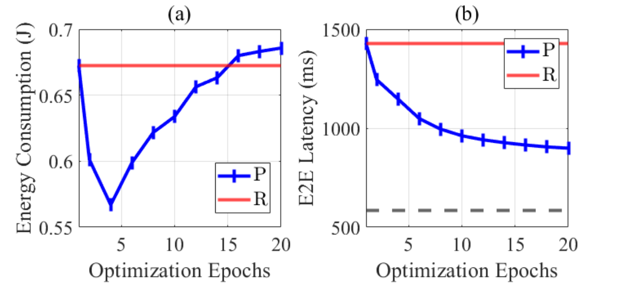

Fig. 12: Energy consumption and end-to-end latency of IoT devices with low data rate (12 Mbps)

VI. Evaluation

In this section, we discuss EFFECT-*DNN*'s performance using both hardware testbed based experimental evaluation and simulation evaluation.

A. Testbed Setup and Experimental Results

The overall hardware testbed setup for the experimental evaluation is shown in Fig. 10. For this experiment, we have 3 NVIDIA Jetson TX2 devices (viz., 'Aruna Ali', 'Valentina Tereshkova', and 'Malala Yousafzai') mimicking IoT devices and 1 Dell Poweredge desktop with 16 cores and CPU frequency 3.2 GHz (viz., 'Grace Hopper') mimicking edge server. A TP-link router is used to create the network connections. Specifically, 'Grace' is connected to the router via high speed Ethernet cable, while the TX2s are wirelessly connected to the router. Due to the high speed connectivity between 'Grace' and the router, the network delay between them is ignored and only wireless delay for the TX2s are considered. To implement network resource allocation, we utilize *Wondershaper* [30] to tune the data rate between each TX2 and 'Grace'. On 'Grace', we implement the controller that runs Algo. (1) and deploys a DNN executor for 'Aruna', 'Valentina', and 'Malala'. In this experiment, we consider CPU cores as manifestation of computational resources, i.e., based on the value of $c_{m,t}$, a discrete and non-overlapping set of CPU cores are assigned to the corresponding DNN executor. To configure EFFECT-*DNN* online resource allocation, we use *taskset* [31] to set and retrieve the CPU affinity of a running executor. In each optimization epoch (starting with $t = 0$), we let the TX2s (i.e., their executors) run a specified DNN collaboratively 20 times based on selected cut point (given by Algo. (1)). In particular, 'Aruna' runs AlexNet (with $\tau = 400$ ms), 'Valentina' runs ResNet (with $\tau = 600$ ms), and 'Malala' runs VGG (with $\tau = 750$ ms). At the same time, we collect the energy consumption and end-to-end latency of the TX2s during this period, which in turn is sent to the controller to update the resource allocation and cut point selection for the next optimization epoch (i.e., $t+1$). We evaluate and compare the EFFECT-*DNN*'s partitioning strategy's performance (denoted by P) against remote-only based inference (denoted by R) under different network resource availability conditions.

i) Under low data rate (24 Mbps) conditions: We first examine the average energy consumption and end-to-end latency characteristics of the TX2s when the total network upload speed is fixed at 24 Mbps (for 4G like scenarios) in Figs. 11(a) and 11(b). We observe that both performance metrics become stable after few optimization epochs. Algo. (1) running on the controller tells 'Aruna' (running AlexNet) to select the cut point $i = 3$ and lets the 'Valentina' and 'Malala' run all their DNN layers remotely (i.e., ResNet and VGG, respectively). In Figs. 11(a) and 11(b), EFFECT-*DNN* partitioning (P) achieves 9.2% energy savings and 20.7% lower end-to-end latency compared to remote-only based solution (R). Such results clearly highlight the benefits of EFFECT-*DNN*'s collaborative DNN inference. On the other hand, the results also show the benefit of EFFECT-*DNN*'s resource allocation in stabilizing the end-to-end latency (i.e., the length of the virtual queue). This is evident from Fig. 11(b), where the average end-to-end latency of all TX2 converge close to their latency requirements (denoted the dashed line) after a few optimization epochs.

ii) Under very low data rate (12 Mbps) conditions: Figs. 12(a) and 12(b) demonstrate the TX2s' average energy consumption and end-to-end latency characteristics when the total network upload speed is fixed at even lower 12 Mbps. In this case, all TX2s are unlikely to meet their end-to-end latency requirements as shown in Fig. 12(b) due to high network delay and limited on-board computation abilities. By running Algo. (1), controller tells 'Aruna' (running AlexNet) to select the cut point $i = 8$ and let 'Valentina' and 'Malala' run all their DNN layers remotely (ResNet and VGG, respectively). These results are consistent with our previous intuitions about the characteristics of DNN layers in Section III-A. For AlexNet, by choosing the cut point $i = 8$, the data is reduced by more than 90%. This justifies such cut point selection when the TX2s are faced with very low data rates. In the results, although the average energy consumption is increased by 2.0%, the average end-to-end latency is reduced by 37.1%. Such preference over energy consumption and end-to-end latency is carried out by the weighted objective function

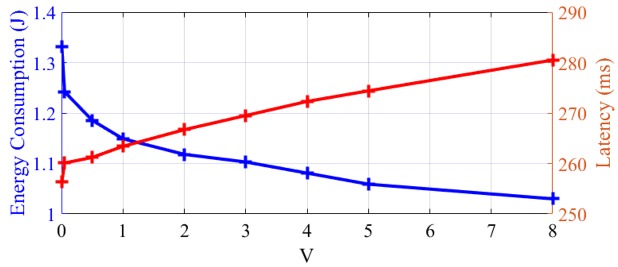

Fig. 13: The role of balance factor V for the trade-off between energy consumption and end-to-end inference latency

that we proposed in problem (**P2**). Overall, the above results demonstrate the effect of available network resource on the overall system performance as it acts as a bottleneck for the end-to-end inference latency. Also, for DNNs, such as VGG and ResNet, partitioning is not a good option as these DNNs are mo are computationally intensive and consequently require considerable local computation in order to reduce data transfer requirements. This in turn incurs significant energy consumption from the device and thus the entire DNNs are offloaded to the server with partitioning.

B. Simulation Results

In this subsection, we evaluate EFFECT-*DNN*'s performance using a realistic simulation with large scale IoT devices running DNN inference. The devices' simulation setup mimics the hardware configurations of TX2, e.g., maximum CPU frequency and energy consumption parameters. The server's computational resources are measured by the number of CPU cores with the CPU frequency is set to 3.2 GHz. As for DNN types and inference latency requirements, we assume the same three DNNs, viz., AlexNet ($\tau = 200$ ms), ResNet ($\tau = 300$ ms) and VGG ($\tau = 350$ ms) and their overall ratios to be 50%, 30%, and 20% of all workflows running in the simulation, respectively. Through this simulation, we seek to observe: i) the impact of balance factor proposed in problem (**P2**), ii) the impact of data rate, and iii) the impact of number of IoT devices. The overall objective of this simulation is to test the schedulability and scalability of EFFECT-*DNN*.

i) Impact of balance factor V: Previously in problem (**P2**), we defined a balance factor V (> 0) that seeks to strike a trade-off between average device energy consumption and end-to-end inference latency. The impact of V can be visualized in Fig. 13 where we use 10 IoT devices with edge resources of 200 Mbps and 32 CPU cores. As shown in Fig. 13, larger V leads to better energy saving but also results in higher inference latency. Here, for the given the distribution of DNNs mentined earlier, the average inference latency requirement is calculated to be around 260 ms. Based on Fig. 13, we can can observe that, the balance factor needs to be set at $V \leq 0.05$ in order to satisfy such latency constraint (i.e., **C5** in problem (**P1**)).

ii) Impact of number of IoT devices: The impact of number of IoT devices on energy consumption and end-to-end inference latency is shown in Fig. 14. In the case of a fixed amount of edge resources, as more IoT devices run collaborative DNN inference, resource competition among IoT devices becomes more intense. The comparisons between EFFECT-*DNN*'s partitioning (P) and remote-only (R) strategies show that with a few IoT devices ($\leq$ 20), the two strategies perform similarly as there are sufficient edge resources. However, with more IoT devices, the performance

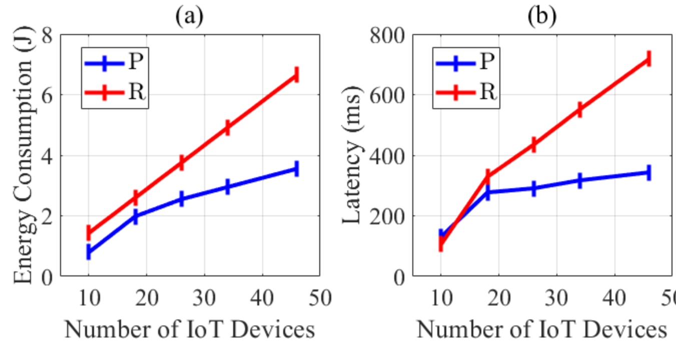

Fig. 14: Device energy consumption and end-to-end inference latency performance against different number of IoT devices

gap between the two gradually widens with EFFECT-*DNN* (P) achieving considerably better performance for both reduction in device energy consumption (23% to 46%) and decreased end-to-end latency (15% to 52%).

iii) Impact of available data rate: Finally, Fig. 15 compares the performance of EFFECT-*DNN*'s collaborative inference (P) strategy against two baseline DNN inference strategies, viz., remote-only (R) and local-only (L), in terms of offloading decision, average energy consumption, and inference latency with varying available data rate. Intuitively, IoT devices are more likely to choose DNN partitioning when more network resources are available, i.e., when total data rate increases. In Fig. 15(a), the ratio of partitioning decisions increases from 11% to 36% when available upload data rate increases from 80 Mbps to 200 Mbps (e.g., under WiFi or 5G LTE conditions). At the same time, with higher data rate, fewer IoT devices are willing to execute DNNs locally due to low transmission delay as we can see that local-only decisions reducing from 41% to 14%. On the other hand, EFFECT-*DNN* saves considerable energy consumption compared to remote-only inference. In Fig. 15(b), DNN partitioning saves more energy in low data rate scenarios, particularly 32.5% and 7% energy saving when total data rate is set to 80 Mbps and 200 Mbps, respectively. Also, as shown in Fig. 15(c), our collaborative DNN inference ensures that all DNN inference can be finished within the latency requirement constraint of 260 ms. Compared to remote-only inference, EFFECT-*DNN* saves 8% to 21% on end-to-end latency.

VII. Conclusions and Future Work

In this paper, we analyzed the trade-off between end-to-end latency of DNN inference and IoT device energy consumption and proposed EFFECT-*DNN* framework that employs a novel collaborative DNN inference model. EFFECT-*DNN* framework performs cut point selection, computation, and network resource allocation that jointly optimize latency and energy consumption. The framework decouples the long term optimization problem in run-time where the resource allocation is solved using convex optimization and cut point selection is carried out using a game-like heuristic. Using real-world DNNs and a hardware testbed, we evaluated the benefits of EFFECT-*DNN* in terms of both energy saving and end-to-end latency reduction. A simulation is based evaluation is also conducted to measure the benefit at scale. In future, we seek to explore other DNN latency optimization techniques, such as DNN compression and layer pruning and further analyze the trade-off between inference latency, energy consumption, and model accuracy.

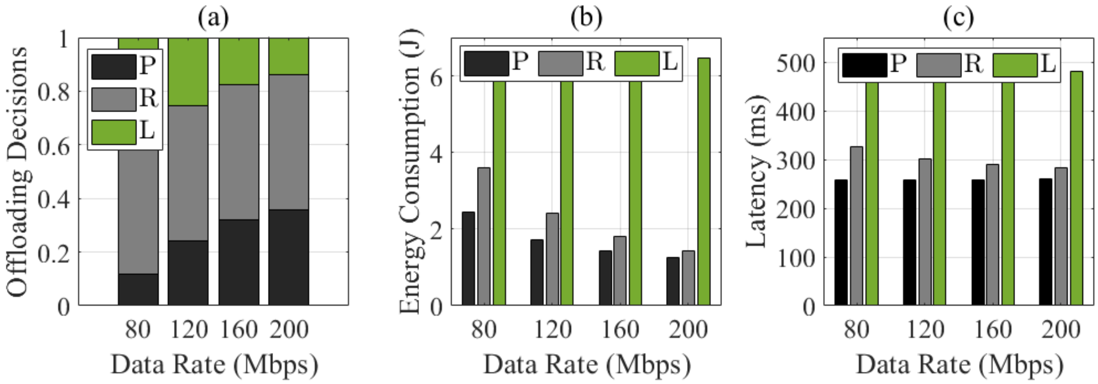

Fig. 15: Offloading decision, device energy consumption, and end-to-end inference latency performance against available data rate

References

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Communications of the ACM*, vol. 60, no. 6, pp. 84–90, 2017.

[2] A. Sengupta, Y. Ye, R. Wang, C. Liu, and K. Roy, "Going deeper in spiking neural networks: Vgg and residual architectures," *Frontiers in neuroscience*, vol. 13, p. 95, 2019.

[3] S. Targ, D. Almeida, and K. Lyman, "Resnet in resnet: Generalizing residual architectures," *arXiv preprint arXiv:1603.08029*, 2016.

[4] M. Chen and Y. Hao, "Task offloading for mobile edge computing in software defined ultra-dense network," *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 3, pp. 587–597, 2018.

[5] K. Zhang, Y. Mao, S. Leng, Q. Zhao, L. Li, X. Peng, L. Pan, S. Maharjan, and Y. Zhang, "Energy-efficient offloading for mobile edge computing in 5g heterogeneous networks," *IEEE access*, vol. 4, pp. 5896–5907, 2016.

[6] J. Ren, G. Yu, Y. Cai, and Y. He, "Latency optimization for resource allocation in mobile-edge computation offloading," *IEEE Transactions on Wireless Communications*, vol. 17, no. 8, pp. 5506–5519, 2018.

[7] A. Das, S. Imai, S. Patterson, and M. P. Wittie, "Performance optimization for edge-cloud serverless platforms via dynamic task placement," in *2020 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID)*. IEEE, 2020, pp. 41–50.

[8] F. Wang, J. Xu, X. Wang, and S. Cui, "Joint offloading and computing optimization in wireless powered mobile-edge computing systems," *IEEE Transactions on Wireless Communications*, vol. 17, no. 3, pp. 1784–1797, 2017.

[9] Y. Mao, J. Zhang, S. Song, and K. B. Letaief, "Stochastic joint radio and computational resource management for multi-user mobile-edge computing systems," *IEEE Transactions on Wireless Communications*, vol. 16, no. 9, pp. 5994–6009, 2017.

[10] T. Bahreini and D. Grosu, "Efficient placement of multi-component applications in edge computing systems," in *Proceedings of the Second ACM/IEEE Symposium on Edge Computing*, 2017, pp. 1–11.

[11] S. Khare, H. Sun, J. Gascon-Samson, K. Zhang, A. Gokhale, Y. Barve, A. Bhattacharjee, and X. Koutsoukos, "Linearize, predict and place: minimizing the makespan for edge-based stream processing of directed acyclic graphs," in *Proceedings of the 4th ACM/IEEE Symposium on Edge Computing*, 2019, pp. 1–14.

[12] S. Yi, Z. Hao, Q. Zhang, Q. Zhang, W. Shi, and Q. Li, "Lavea: Latency-aware video analytics on edge computing platform," in *Proceedings of the Second ACM/IEEE Symposium on Edge Computing*, 2017, pp. 1–13.

[13] X. Zhang, A. Pal, and S. Debroy, "Effect: Energy-efficient fog computing framework for real-time video processing," in *2021 IEEE/ACM 21st International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*. IEEE, 2021, pp. 493–503.

[14] Y. Kang, J. Hauswald, C. Gao, A. Rovinski, T. Mudge, J. Mars, and L. Tang, "Neurosurgeon: Collaborative intelligence between the cloud and mobile edge," *ACM SIGARCH Computer Architecture News*, vol. 45, no. 1, pp. 615–629, 2017.

[15] E. Li, L. Zeng, Z. Zhou, and X. Chen, "Edge ai: On-demand accelerating deep neural network inference via edge computing," *IEEE Transactions on Wireless Communications*, vol. 19, no. 1, pp. 447–457, 2019.

[16] S. Laskaridis, S. I. Venieris, M. Almeida, I. Leontiadis, and N. D. Lane, "Spinn: synergistic progressive inference of neural networks over device and cloud," in *Proceedings of the 26th annual international conference on mobile computing and networking*, 2020, pp. 1–15.

[17] J. Mao, X. Chen, K. W. Nixon, C. Krieger, and Y. Chen, "Modnn: Local distributed mobile computing system for deep neural network," in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*. IEEE, 2017, pp. 1396–1401.

[18] Z. Zhao, K. M. Barijough, and A. Gerstlauer, "Deepthings: Distributed adaptive deep learning inference on resource-constrained iot edge clusters," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 11, pp. 2348–2359, 2018.

[19] H. Ma, R. Li, X. Zhang, Z. Zhou, and X. Chen, "Reliability-aware online scheduling for dnn inference tasks in mobile edge computing," *IEEE Internet of Things Journal*, 2023.

[20] E. Cui, D. Yang, H. Wang, and W. Zhang, "Learning-based deep neural network inference task offloading in multi-device and multi-server collaborative edge computing," *Transactions on Emerging Telecommunications Technologies*, vol. 33, 07 2022.

[21] W. Zhang, D. Yang, H. Peng, W. Wu, W. Quan, H. Zhang, and X. Shen, "Deep reinforcement learning based resource management for dnn inference in industrial iot," *IEEE Transactions on Vehicular Technology*, vol. 70, no. 8, pp. 7605–7618, 2021.

[22] R. Mishra, H. P. Gupta, and T. Dutta, "A survey on deep neural network compression: Challenges, overview, and solutions," *arXiv preprint arXiv:2010.03954*, 2020.

[23] X. Tang, X. Chen, L. Zeng, S. Yu, and L. Chen, "Joint multiuser dnn partitioning and computational resource allocation for collaborative edge intelligence," *IEEE Internet of Things Journal*, vol. 8, no. 12, pp. 9511–9522, 2020.

[24] C. Dong, S. Hu, X. Chen, and W. Wen, "Joint optimization with dnn partitioning and resource allocation in mobile edge computing," *IEEE Transactions on Network and Service Management*, vol. 18, no. 4, pp. 3973–3986, 2021.

[25] F. Dong, H. Wang, D. Shen, Z. Huang, Q. He, J. Zhang, L. Wen, and T. Zhang, "Multi-exit dnn inference acceleration based on multi-dimensional optimization for edge intelligence," *IEEE Transactions on Mobile Computing*, 2022.

[26] W.-J. Kim and C.-H. Youn, "Lightweight online profiling-based configuration adaptation for video analytics system in edge computing," *IEEE Access*, vol. 8, pp. 116 881–116 899, 2020.

[27] Q. Liu, S. Huang, J. Opadere, and T. Han, "An edge network orchestrator for mobile augmented reality," in *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*. IEEE, 2018, pp. 756–764.

[28] X. Chen, "Decentralized computation offloading game for mobile cloud computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 26, no. 4, pp. 974–983, 2015.

[29] X. Chen, L. Jiao, W. Li, and X. Fu, "Efficient multi-user computation offloading for mobile-edge cloud computing," *IEEE/ACM Transactions on Networking*, vol. 24, no. 5, pp. 2795–2808, 2016.

[30] *Wondershaper - https://github.com/magnific0/wondershaper*.

[31] *taskset - https://man7.org/linux/man-pages/man1/taskset.1.html*.