EFFECT-*DNN*: Energy-efficient Edge Computing 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 environments. In this paper, we address the trade-off between end-to-end latency of DNN inference and IoT device energy preservation by proposing 'EFFECT-*DNN*', an energy efficient edge computing framework.The EFFECT-*DNN* framework aims to strike such balance by performing an optimized 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 that is solved by Lyapunov optimization and a game-like heuristic algorithm. We evaluate the performance of EFFECT-*DNN* framework on a hardware testbed and via extensive simulations with real-world DNNs. The results demonstrate that under realistic conditions, the proposed framework can ensure DNN inference deadline satisfaction with significant ($\sim$ 20-30%) IoT 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 computation applications used in mission-critical use cases are Deep Neural Networks (DNNs) where they 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 this 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 servers 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 size, the remote only strategy might also lead to latency violation due to high transmission time of data upload.

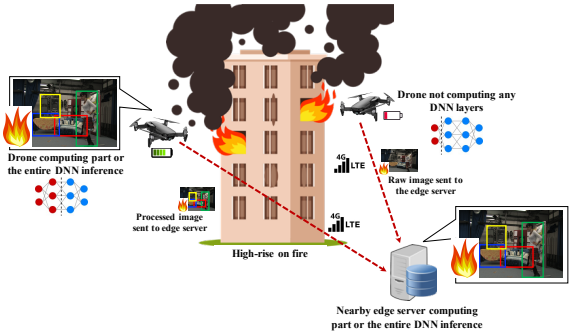

Fig. 1: An exemplary objection detection and classification related DNN partitioning scenario between drones and edge server for fire rescue use case

One common sense strategy to perform such DNN inference in a balanced manner can be 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 are sent to the edge server where it continues the DNN inference 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 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 expenditure (by the device) for offloading the output data after the cut point layer is a function of the size of that data. Cut point selection where the output data size is large would result in considerable energy consumption and transmission latency resulting in longer end-to-end latency. This problem is even more pronounced (i.e., considerably more energy expenditure by the device and longer latency) for low data rate scenarios which is often the case for cellular uplink connections used under mission critical use cases. Thus, choice of cut point layer is of paramount importance in order to strike a balance between devices'

energy efficiency and end-to-end DNN inference latency. At the same time, compute resource allocation from the device and choice of wireless link bandwidth 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 devices, the edge servers, and the available cellular network bandwidth.

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 energy expenditure of IoT devices running heterogeneous DNNs and 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 network resource allocation between the IoT devices and edge servers. 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 guarantee the long-term average end-to-end latency 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. Here, we apply *Lyapunov* optimization by creating a virtual queue to express the long-term end-to-end latency requirements, followed by decoupling the cut point selection and resource allocation problems. Finally, a game-like heuristic algorithm is proposed as part of the EFFECT-*DNN* framework to jointly compute the optimal cut points and resource allocations in run-time 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 on DNN partitioning decisions by simulating a wide range of edge environments with different network and compute resource availability. The simulation 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 also 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 elaborates problem evidence analysis. Section IV formulates the problem and proposes our system model. Section V discusses the solution strategy and algorithms. Section VI discusses testbed and simulation results. Section VII concludes the paper.

II. Related works

In literature, different approaches are used to tackle task offloading in edge architectures. Task offloading was first formulated as a deterministic problem that decides between local-only and remote-only computation plans [4], [5], [6]. Das et al. [7] deployed a similar approach for task-offloading between edge servers and the cloud. The problem can also be formulated in a *computing while transmitting* paradigm i.e. partial task offloading that starts the computation on the IoT device [8], [9]. Expressing tasks as directed acyclic graphs (DAG) with different computation components is another alternative [10], [11], [12]. In this approach, the task components can be placed across the edge servers. EFFECT [13] extends this approach to a more complex environment with multiple servers and multi-stage computations.

Partitioning in DNNs is conceptually similar to task offloading in sequential DAGs as most of the popular DNN models are consist of a sequence of layers each triggering the next layer by sending the intermediate data. Due to the variations in data size and computational requirements of DNN layers, the choice of cut point affects the end-to-end latency and energy consumption of IoT devices. There is extensive work in the literature regarding layer-based prediction models. A combination of a regression model predicting the layer-wise performance and a dynamic cut-point selection mechanism is deployed 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. To increase the overall parallelism, layers can be further partitioned horizontally. MoDNN [17] first partitions the DNN by layers and then maps different parts of a layer onto mobile devices to accelerate the computations. A scalable Fused Tile Partitioning (FTP) of convolutional layers along with a novel work scheduling is proposed in [18] to minimize memory footprint and reduce overall execution latency.

More recent frameworks brought the importance of resource allocation in an environment with constrained resources on the edge side into attention [19], [20], [21]. Tang et al. [19] 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 [20]. Dong et al. [21] paired 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.

III. Application profiling and problem evidence analysis

In order to capture actual characteristics of real-world DNN inference and data transmission, we perform a case study of collaborative DNN inference with NVIDIA Jetson TX2 (mimics real-world IoT devices) and a Dell desktop (working as an edge server with 16 cores and CPU frequency 3.2 GHz). NVIDIA Jetson TX2 is a fast embedded AI computing device that is the most power-efficient among its similar models. Doing the measurements with a less powerful computing

device will increase the end-to-end latency and energy consumption. In this case study, we select and analyze three well-known neural networks, namely AlexNet [1], ResNet [3] and VGG [2]. These neural networks are widely used in many applications such as object detection, image recognition, and image classification.

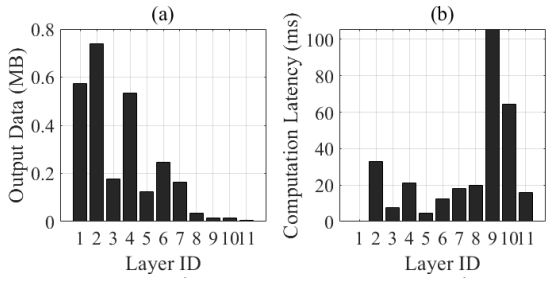

Fig. 2: The layer-wise data size and inference time of AlexNet

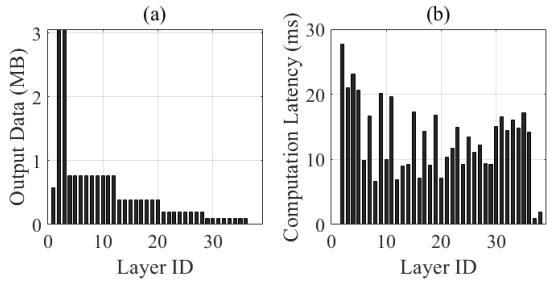

Fig. 3: The layer-wise data size and inference time of ResNet

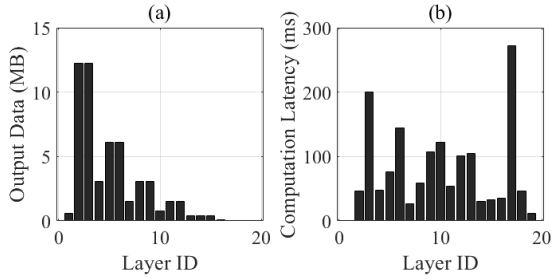

Fig. 4: The layer-wise data size and inference time of VGG

A. Layer Characteristics

Before we measure the characteristics of individual layers, we first integrate lightweight layers (e.g., activation function) with their adjacent layers which are compute-intensive (e.g., convolutional layer). By doing this integration, we can reduce the number of layers and thus the complexity of the optimization problem (will be proposed soon). The results of size of output data and computation latency of each layer when running these layers on IoT devices (we use TX2) are shown in Fig. 2 (AlexNet), Fig. 3 (ResNet) and Fig. 4 (VGG), respectively. Through these results, we obtained some useful information. First, DNNs have different number of layers, and the size of output data and inference time of each layer can have great difference. Capturing such layer differences is important for deciding where and how layers should be executed. Secondly, with the purpose of reducing transmission cost (time, energy or both), the DNNs may have fixed number of options for layer partitioning. As shown in Fig. 2, the AlexNet can have considerable reduction in output data after the execution of layer 3, 5 and 8. Considering the local computation cost of layers before these locations is very limted, AlexNet is a potential candidate who can obtain benefit from DNN partitioning. However, for ResNet and VGG, the output data is not reduced until the 13-th layer (shown in Fig. 3 and Fig. 4). We can foresee that, in most cases, IoT devices running VGG and ResNet are unwilling to execute any part of the layers locally because the local computation cost of its first 13 layers is very high. Therefore, they require a lot of network and computational resources at the server-side.

B. Benefits of DNN Partitioning

In this subsection, we seek to study the improvement in the end-to-end latency and energy consumption of collaborative DNN inference enabled by layer partitioning (*the layers are divided into local-layers and remote-layers based on a given location, called cut point*). We also look at how the data rate obtained by IoT devices will affect such collaborations. In this measurement, data rates are tuned to 8 Mbps and 20 Mbps, simulating different network scenarios (i.e., LTE - low data rate case and WiFi - acceptable data rate case).

Figure 5 plots the end-to-end latency and energy consumption of IoT device running the AlexNet using different cut points. In Figure 5 (a) and Figure 5 (b), the best cut points in terms of end-to-end latency are 8 and 3, respectively. However, as shown in Figure 5 (c) and Figure 5 (d), if the IoT device is more concerned about its energy consumption, the optimal cut points become 3 and 1 (when the data rate is 8 Mbps and 20 Mbps). However, for ResNet and VGG, the computational requirement of both is much higher than AlexNet, especially VGG. Based on the results shown in Fig. 6, the ResNet should execute all layers either locally (Fig. 6 (a)) or remotely (Fig. 6 (b), (c) and (d)). While shown in Fig 7 (a)-(d), VGG layers should execute fully remotely in all cases, both in terms of end-to-end latency and energy consumption.

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. By enabling DNN partitioning, it provides IoT devices with the flexibility to address such trade-offs by choosing different cut points. More specifically, we know that different cut points have a great impact on end-to-end latency and energy consumption. On the other hand, the selection of cut point needs to depend on the availability of resources.

IV. System Model

Define $\mathcal{M} = \{1, 2, ..., M\}$ as a set of M IoT devices in the service area, each IoT device m is running a specific type of DNN (let say type m). The type m DNN consists of a set of I_m layers, denoted by $\mathcal{I}_m = \{1, 2, ..., I_m\}$ (the first layer is input). We assume that the DNN (or related application) requested by m-th IoT device has a strict performance requirement, that is, its end-to-end inference latency should be no greater than τ_m (measured in seconds). On the hand, real-world IoT devices are usually energy-constrained, which makes energy saving indispensable. With the purpose of energy saving, we seek to implement a collaborative DNN inference architecture which

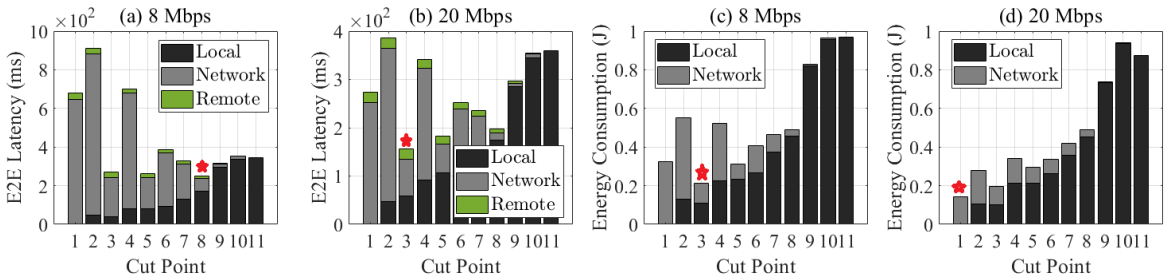

Fig. 5: The end-to-end latency and energy consumption of AlexNet running collaboratively between TX2 and edge server under a given cut point.

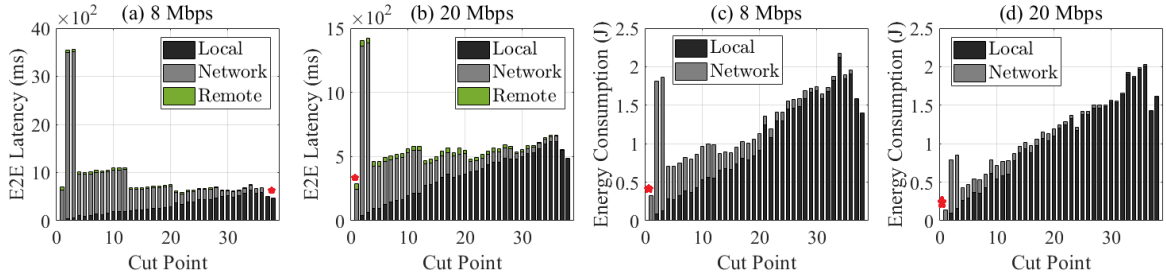

Fig. 6: The end-to-end latency and energy consumption of ResNet running collaboratively between TX2 and edge server under a given cut point.

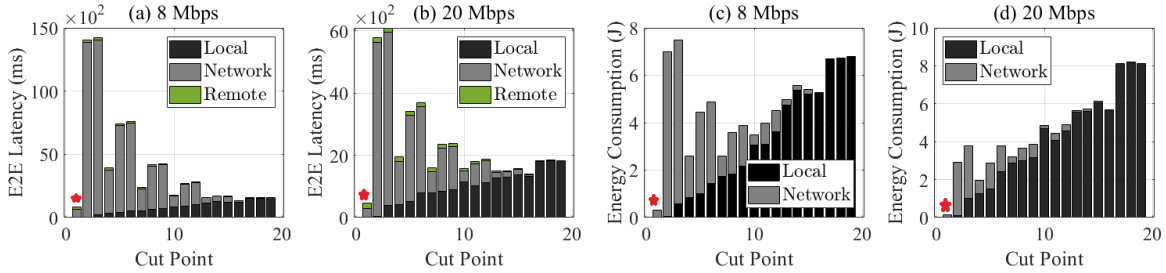

Fig. 7: The end-to-end latency and energy consumption of VGG running collaboratively between TX2 and edge server under a given cut point.

acrosses the IoT device and edge server (depicted in Figure 8). In this architecture, the layers of DNN 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 (the blue and red box in Figure 8). To accommodate multiple IoT devices, resource provisioning is optimized. In this work, we provide IoT devices with on-demand (i.e., based on end-to-end latency and energy consumption) network and computational resources for the transmission of intermediate data (generated by local-layers) and execution of remote-layers.

A. DNN Profile

Denote the size of output data of layer $i \in \mathcal{I}_m$ as $d_{m,i}$ (measured in bits), and denote the computational complexity of this layer as $x_{m,i}$ (such as the number of CPU cycles or simply the CPU time). Given a cut point after the layer i s.t. layers 1 to i are executed locally on IoT devices (local-layers) and layers i to I_m are executed remotely on the edge server (remote-layers). Then, the accumulated computational complexity of local-layers and remote-layers are defined as

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

There are two extreme cases where the layers are only executed by IoT device or by the edge server, referring to local-only and remote-only strategy respectively. When $i = 1$, it refers to remote-only inference; $i = I_m$ indicates local-only inference. Given I_m possible cut points, 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. And $\mathcal{P}_m = \{p_{m,i} | i \in \mathcal{I}_m\}$ is set of available execution profiles. Intuitively, the IoT devices should select the cut point that minimizes their transmission cost compared to remote-only inference, that is, $\arg\max_i (d_{m,1} - d_{m,i})$. 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 best balance between the trade-off of local cost and transmission cost.

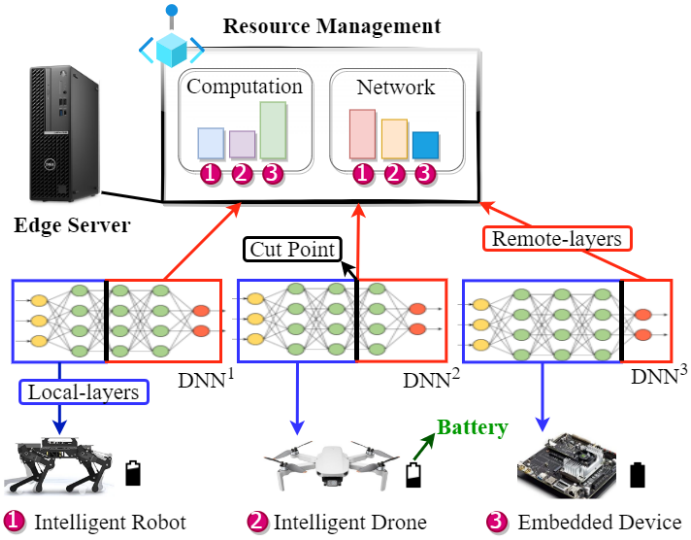

Fig. 8: The system architecture of collaborative DNN inference.

In this work, we allow the IoT devices select the optimal cut point based on their own requirement. Let us denote $o_{m,i} \in \{0, 1\}$ as the profile selection indicator. If $o_{m,i} = 1$, the 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 connecting to the access point (AP) of the edge server. 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 the value of B can refer to bandwidth (measured in MHz), number of channels/sub-channels or simply point to the data rate (in Mbps). Based on whichever is available to be partitioned and allocated to IoT devices at this AP, we defined b_m as the ratio of network resource allocated to the m-th IoT device. The constraint of network resource allocation follows

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

Due to the heterogeneous in channel noise, signal fading and transmit power, we obey the reality that different IoT devices can have different network delay even they are using 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 a function of b_m and data size $d_{m,i}$ e.g.,

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

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

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

where $\beta_{m,i}$ is the transmit power used in the transmission period. The transmit power may change when the edge architecture uses different communication technologies such as LTE and WiFi.

C. Local Computation Model

The IoT devices are asked to finish the computation requriement of local-layers (from layer 1 to i_m). Define $f_m^{IoT} \in (0, F_m^{IoT}]$ as the available CPU speed (CPU cycles per second) that IoT device m can take, the local computation time is measured by

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

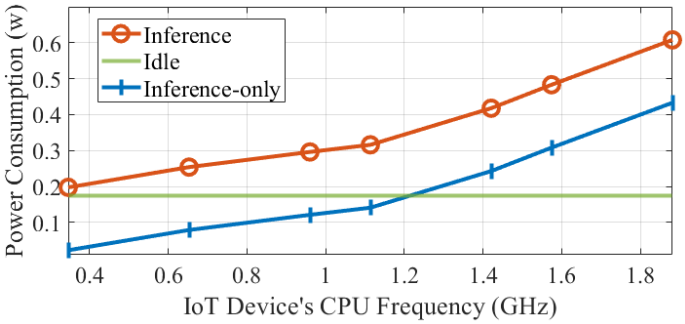

Fig. 9: The power consumption of Nvidia TX2.

For IoT devices, their energy consumption is closely related to their computing speed. In general, energy consumption increases with CPU frequency, and the magnitude of the increase may vary for different types of IoT devices. In Fig. 9, we show the power consumption of NVIDIA Jetson TX2 (in watts or joules/s) when the machine is running the DNN (inference state) or remains idle. With these two power values, we can compute the power consumption that is caused by inference-only (the blue line in Fig. 9). For TX2, its inference-only power consumption is fitted into $\epsilon_{m,i} = \kappa(f_m^{IoT})^2$, where κ is a constant related to the chip architecture. Then, the actual energy spent by IoT computation is computed by

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

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. Again, we do not limit the format of edge resource as the implementation of resource allocation (computational) can have different manners, such as CPU frequency allocation [13], CPU core allocation [19] or GPU time allocation [22]. Let us define c_m as the ratio of computational resources that is allocated to the IoT device m. The constraint of computational resource allocation follows

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

Then the server computation time is computed by

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

In this paper, the server's energy consumption is ignored.

E. Problem Formulation

Since real-world DNN inference and data transfer can generate some randomness in end-to-end latency and energy consumption. We formulate a long-term optimization problem, which aims to minimize the long-term average energy consumption of all connected IoT devices in the service area and ensures that the long-term average end-to-end latency is no greater threshold τ_m. Define time epochs as $\mathcal{T} = \{1, 2, ..., T\}$, We add symbol t as one of the subscripts of the above parameters to denote their actual value for a certain time period. Then, the end-to-end latency of m-th IoT device in epoch t is

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

and the average energy consumption of all IoT devices is

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

The proposed problem is stated as follows:

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

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

V. Solution Strategy

We apply Lyapunov optimization and create a virtual queue to alternatively express long-term constraint **C5** and the long-term objective function of **P1**.

A. Lyapunov optimization

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 (virtual) in each epoch. Based on the difference between these two, a virtual queue is constructed. Let us define $Q_{m,t} \geq 0$ as the length of the virtual queue at epoch t, 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. On the contrary, 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 to shorten and stabilize the virtual queue for each IoT device.

To stabilize the queue, we need to find a policy to 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 $\Delta(Q^t)$ is used to measure the difference in $\mathcal{L}(\cdot)$ between two adjacent epochs, i.e., t and $t+1$. 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 we can easily get

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

while C is bounded to $\Big(L_{m,t}\Big)^2$. Then problem (**P1**) is transformed to multiple deterministic and identical per-epoch problems with opportunistically minimizing an expectation, which is stated as follows:

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

where V is a factor that addresses 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 preference (energy-sensitive or latency-sensitive). However, problem (**P2**) is still non-trivial to solve because of its MINLP nature, which is usually NP-hard. 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 for the selection of cut point and resource allocation, 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 Lagrange method.
2) Cut point selection: when resource allocation strategies are given, the selection of cut point (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:

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

We first want to prove 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**):

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

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 optimal allocation strategies are derived.

1) Server-side Computational Resource Allocation: For computational resource allocation $c_{m,t}$, we have

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

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 [23], so problem (**P3**) is 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

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

where we get that the allocation of computational resources is only determined by the computation requirement of remote-layers and current queue length. More specifically, $c_{m,t}^*$ is proportional to the square root of the product of these two. Finally, we have

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

2) Server-side Network Resource Allocation: Similarly, for network resource allocation $b_{m,t}$, 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 the Hessian matrix $\mathbf{H} = (\frac{\nabla \mathcal{L}^2}{\nabla b_{m,t} b_{n,t}})_{M \times M}$ is symmetric and positive definite, problem (**P3**) is strictly convex $w.r.t$ allocated network resource ratio $b_{m,t}$. Based on the KKT condition, the optimal network resource allocation $b_{m,t}^*$ can be obtained by solving the following expression

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

The optimal $b_{m,t}^*$ is

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

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. Finally, we have

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

3) Local Computation Speed: When IoT device m selects $o_{m,I_m,t} = 1$ (local-only inference), the IoT device should use the computation speed which just meets its end-to-end latency requirement with the purpose of energy saving. In this case, the optimal computation speed (also the minimum computation speed) is

$$f_{m,t}^{IoT^*} = \min\{\frac{X_{m,i}^{IoT}}{\tau_m}, F_m^{IoT}\}$$

Therefore, the entire DNN can be executed locally if and only if the computation requirement of the entire DNN satisfies $X_{m,i}^{IoT} \leq F_m^{IoT} \times \tau_m$. Based on previous discussion, the minimum energy consumption $w.r.t.$ the end-to-end latency constraint (**C5**) can be computed by

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

Eq. (13) gives the baseline of energy consumption, that is, the IoT device would like to select $o_{m,i,t} = 1$ and $i < I_m$ (remote-only or DNN partitioning) if and only if its energy consumption can be less than $E_{m,I_m,t}{}^*$ or the IoT device cannot finish the entire DNN within τ_m amount of time even using its maximum computation speed. In the latter case, its energy consumption is inevitably 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

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

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

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

By solving the above equation, we get

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

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 needed to be generated quickly so the actual DNN inference can start soon. 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 depicted in Alg. (1).

At first, we initial 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, in 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 which maximizes its utility $U_{m,i,t}$ (line 5-9). Finally, with updated set of execution profiles $\mathcal{O}(t)$, the edge server performs resource allocation based on Eq. (11), Eq. (12) and Eq. (15) and all DNN inference starts using the allocated resources. Based on observe 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 [24], [25], and the utility function (line 8) is defiend as follows

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

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

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 $t \leftarrow t + 1$

VI. Evaluation

In this section, we preform both experimental evaluation and simulation evaluation.

A. Testbed Setup and Experimental Results

The testbed setup for experimental evaluation is shown in Fig. 10. In this experiment, we have 3 NVIDIA TX2 and one edge server. A TP-link router is used to create the network connections. The edge server is connected to the router via high speed Ethernet cable (thereby the network delay between router and server machine is ignored), while the TX2 is wirelessly connected to the router. To implement network resource allocation, we utilize *Wondershaper* [26] to tune the data rate between each TX2 and the server. On the server-side, the controller (where Alg. (1) is running) is going to deploy a DNN executor for each connected TX2. In this experiment, we consider CPU cores as the virtualization of computational resources. That is, according to the value of $c_{m,t}$, a discrete and non-overlapping set of CPU cores are assigned to the corresponding DNN executor. To configure online resource allocation, we use *taskset* [27] to set and retrieve the CPU

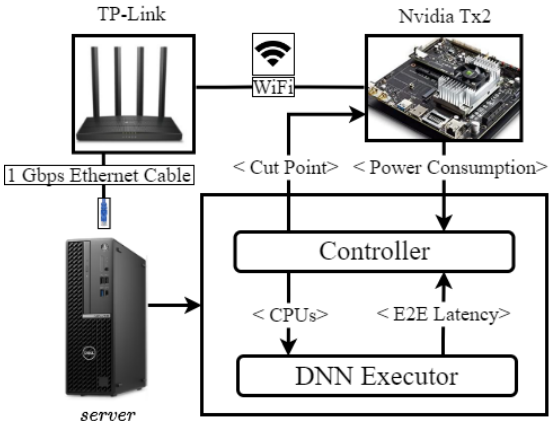

Fig. 10: The testbed setup.

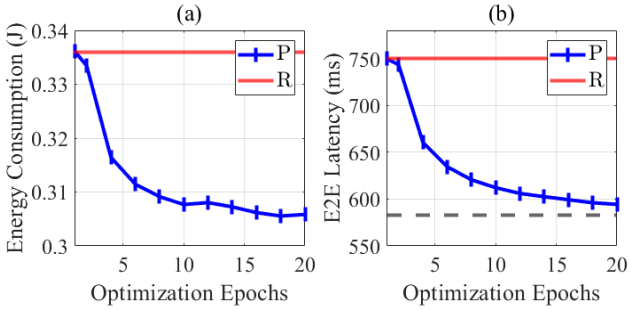

Fig. 11: The energy consumption and end-to-end latency of IoT devices with acceptable data rate (24 Mbps).

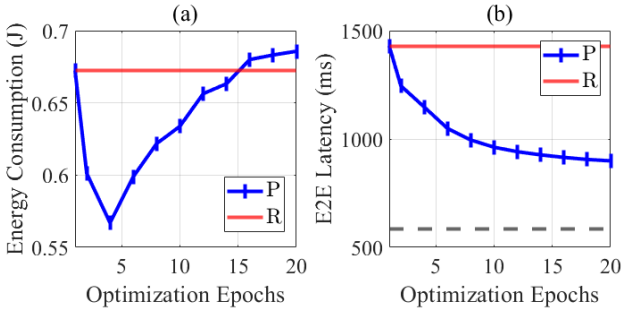

Fig. 12: The energy consumption and end-to-end latency of IoT devices with low data rate (12 Mbps).

affinity of a running executor. In each optimization epoch (start with $t = 0$), we let IoT devices and their executors run a specified DNN collaboratively 20 times based on selected cut point (given by Alg. (1)). At the same time, we collect the energy consumption and end-to-end latency of IoT devices during this period, which will be sent to the controller to update the resource allocation and selection of cut point for the next optimization epoch $(t+1)$. As for DNN applications, the three NVIDIA TX2 are running AlexNet ($\tau = 400$ ms), ResNet ($\tau = 600$ ms) and VGG ($\tau = 750$ ms), respectively. We evaluate and compare the system performance of DNN partition (P)-based inference with remote-only (R) inference under different network resource availability.

i) Acceptable data rate case (24 Mbps): we first show the energy consumption and end-to-end latency of IoT devices when the total network resource is fixed at 24 Mbps (for uploading). The results are shown in Fig. 11 (a) and (b). Overall, both performance metrics the IoT devices become stable after few optimization epochs. By running Alg. (1), controller tells the IoT device who is running AlexNet to select the cut point $i = 3$ and let the other two IoT devices to run all their DNN layers remotely (VGG and ResNet). In Fig. 11 (a) and (b), our DNN partitioning based solution (P) achieves 9.2% energy savings and 20.7% lower end-to-end latency compared to remote-only based solution (R). Such results highlight the benefits of collaborative inference by enabling DNN partitioning. On the other hand, it also shows the ability of our Lyapunov optimization based solution in stabilizing the end-to-end latency (or the virtual queue length). In Fig. 11 (b), when the total data rate is 24 Mbps, the end-to-end latency of the three IoT devices are very close to their latency requirements (the dashed line).

ii) Low data rate case (12 Mbps): we also show a low data rate case where the total data rate is limited at 12 Mbps. The results are shown in Fig. 12 (a) and (b). In this case, the IoT devices 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 ability of TX2. By running Alg. (1), controller tells the IoT device who is running AlexNet to select the cut point $i = 8$ and let the other two IoT devices to run all their DNN layers remotely (VGG and ResNet). These results are consistent with our previous discussion about the characteristics of DNN layers in subsection III-A. For AlexNet, by choosing the cut point $i = 8$, the data is reduced by more than 90%, which makes this cut point a good option for DNN partitioning when IoT devices are faced with very low data rates. In the results, although the average energy consumption is increased by 2.0%, but 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 we proposed in problem (**P2**).

Above results indicate the importance of network resource to the overall system performance as it adds a bottleneck to end-to-end latency. Also, for DNNs like VGG and ResNet, neither of them are willing to do any DNN partitioning because they are more computationally intensive and require a lot of local computation to reduce data transfer requirements, which incurs a lot of energy consumption.

B. Simulation Results

In this subsection, we perform a realistic emulation by considering an use case where there are a large number of IoT devices whose hardware setup mimics the configurations of TX2 (e.g., maximum CPU frequency and energy consumption parameters). Our server's computational resources are measured by the number of CPU cores (the individual CPU frequency is set to 3.2 GHz). As for DNN types and inference latency requirements, we assume that the ratios of AlexNet ($\tau = 200$ ms), ResNet ($\tau = 300$ ms) and VGG ($\tau = 350$ ms) are 50%, 30% and 20%, respectively. In this simulation, we seek to find i) Impact of balance factor proposed in problem (**P2**), ii) Impact of data rate and iii) Impact of number of IoT devices. The objective of this simulation is to test the schedulability and scalability of proposed solution (Alg. (1)).

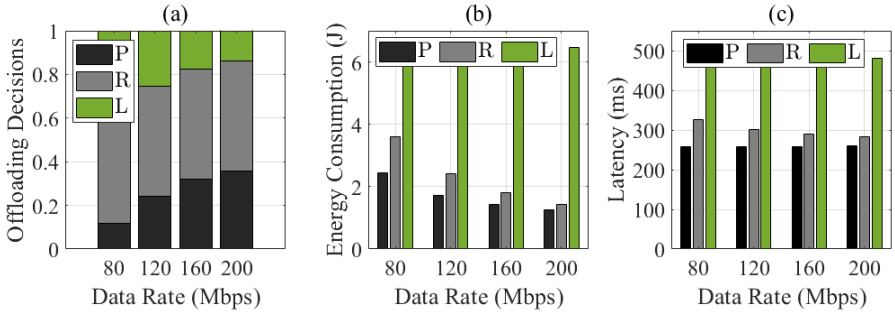

Fig. 13: The impact of total data rate.

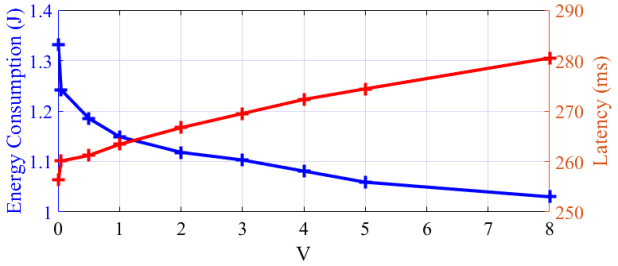

Fig. 14: The trade-off between energy consumption and end-to-end inference latency.

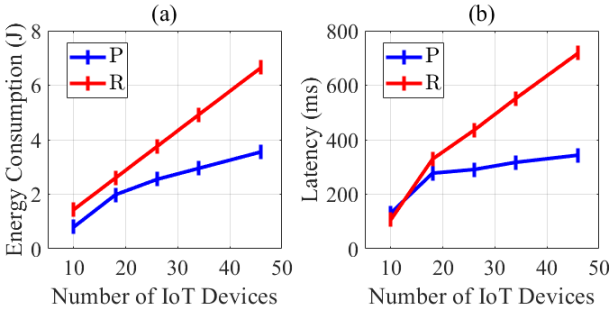

Fig. 15: The energy consumption and end-to-end inference latency against different number of IoT devices.

i). Impact of balance factor V: In problem (**P2**), we define a balance factor V (> 0) that strikes the trade-off between energy consumption and end-to-end inference latency (in average). The impact of V is described in Fig. 14. In the system setup, we use 10 IoT devices with edge resources of 200 Mbps and 32 cpu cores. As shown in Fig. 14, larger V leads to better energy saving but also results in higher end-to-end latency. Given the distribution of DNNs, the average end-to-end latency requirement is around 260 ms. Based on the results, to satisfy the long-term latency requirement constraint **C5** in problem (**P1**), we need to set $V \leq 0.05$.

ii). Impact of data rate: In this simulation, we compare three DNN inference decisions, namely Local-only (L), Remote-only (R), and DNN Partitioning (P). The results of offloading decision-making, energy consumption and end-to-end latency are summarized in Fig. 13. Overall, IoT devices are more likely to choose DNN partitioning when total data rate increases. In Fig. 13 (a), the ratio of partitioning decisions increases from 11% to 36% when available data rate increases from 80 Mbps to 200 Mbps. At the same time, with higher data rate, fewer IoT devices are willing to execute DNNs locally due to high computational energy costs (local-only decisions reduce from 41% to 14%). On the other hand (compared with remote-only inference), DNN partitioning saves considerable energy consumption. In Fig. 13 (b), DNN partitioning saves more energy in low data rate use cases (32.5% and 7% energy saving when total data rate is set to 80 Mbps and 200 Mbps, respectively). Also, as shown in Fig. 13 (c), our DNN partitioning based solutions ensure that all DNN inference can be finished within the end-to-end latency requirement (260 ms). Compared with remote-only inference, our solution saves 8% to 21% on end-to-end latency.

iii). Impact of number of IoT devices: We also show the impact of number of IoT devices on energy consumption and end-to-end inference latency. In the case of a fixed number of edge resources, as more IoT devices enter the service area, resource competition among IoT devices becomes more intense. The comparisons of DNN partitioning (P) and remote-only (R) solutions are shown in Fig. 15. With few IoT devices ($\leq$ 20), the two offloading decisions perform similar as the edge resources are sufficient. However, when more IoT devices enter the service area, the performance gap between these two offloading decisions (i.e., P and R) gradually widens. Our DNN partitioning based solution (P) achieves better performance on both energy consumption (23% to 46%) and end-to-end latency (15% to 52%).

VII. Conclusions

In this paper, we study and analyze the trade-off between end-to-end latency and energy consumption by proposing a collaborative DNN inference architecture. A long-term optimization framework is developed to jointly optimize DNN partitioning and resource allocation. Using real-world DNNs and testbed, we validate the benefits of the proposed collaborative DNN inference architecture in both energy saving and end-to-end latency reduction.

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] 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.

[20] 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.

[21] 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.

[22] 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.

[23] 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.

[24] 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.

[25] 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.

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

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