

Distributed Inference in Resource-Constrained IoT for Real-Time Video Surveillance

Muhammad Asif Khan^{ID}, *Senior Member, IEEE*, Ridha Hamila^{ID}, *Senior Member, IEEE*,
Aiman Erbad^{ID}, *Senior Member, IEEE*, and Moncef Gabbouj^{ID}, *Fellow, IEEE*

Abstract—Advances in communication technologies and computational capabilities of Internet of Things (IoT) devices enable a range of complex applications that require ever increasing processing of sensors' data. An illustrative example is real-time video surveillance that captures videos of target scenes and process them to detect anomalies using deep learning (DL). Running deep learning models requires huge processing and incurs high computation delay and energy consumption on resource-constrained IoT devices. In this article, we introduce methods for distributed inference over IoT devices and edge server. Two distinct algorithms are proposed to split the deep neural network layers computation between IoT device and an edge server; the early split strategy (ESS) for battery powered IoT devices and the late split strategy (LSS) for IoT devices connected to regular power source. The evaluation shows that both the ESS and LSS schemes achieve the target inference delay deadline when tested over VGG16 and MobileNet_V2 CNN models. In terms of computational load, the ESS scheme achieves nearly 15–20% reduction whereas LSS scheme achieves up to 60% reduction. The gains in energy saving of IoT devices for both the ESS and LSS schemes are nearly 18% and 52%, respectively.

Index Terms—Computation, distributed machine learning, edge computing, inference, neural networks, split strategy, video IoT.

I. INTRODUCTION

THE Internet of Things (IoT) is an emerging paradigm, giving rise to a world of interconnected devices with sensing, computing, and communication capabilities. According to the study [1], the number of IoT devices will reach 41.6 billion by 2025 with data generation up to 79.4 ZB. IoT networks incorporate multimedia (such as images and videos), computer vision, and communication technologies to deploy smarter and more user-friendly applications (such as video surveillance, behavior analysis, and driving assistance) [2]. Deep learning has been proven as an effective method for processing images and videos captured by IoT devices to enhance several applications such

as object detection and tracking, face recognition, and activity recognition.

There are two conventional approaches to process videos using DNN in IoT applications, i.e., 1) process videos at the IoT device, or 2) transmit video to the server (local edge or remote cloud) for processing. Both approaches have advantages and disadvantages. For instance, using on-device local processing, data remain local and does not need to be transferred to the remote server, thus respects users' privacy and saves bandwidth. However, processing huge amount of multimedia data on the resource constrained IoT device can cause higher processing delay. Furthermore, as mostly IoT devices are battery powered, on-device processing increases the energy consumption of the device. In the second approach, the IoT device does not process the videos locally and instead offload them to the remote server for processing. The server which hosts the DNN model, process the videos (e.g., detect objects, faces, anomalies) and sends the result back to the IoT device. This approach has the benefits to reduce the energy consumption of the device and provides faster processing at the server. However, it requires huge amount of communication bandwidth and incurs high latency to transfer the videos [3]. Although, local computing of DL models may not be efficient due to limited computational resources, edge-based processing may be inevitable in some applications, e.g., in image retrieval problem in which the data resides at the edge server [4].

Recently, researchers are investigating distributed learning and inference computation in which the computational tasks are distributed over end devices and cloud/edge servers. In distributed inference, the DNN model is processed partially on the IoT devices and partially on the edge and/or cloud server. Distributed DNN (DDNN) [3], DNN Surgery [5], Neurosurgeon [6], Fused Tile partitioning (FTP) [7], and Distributed INference Acceleration (DINA) [8] are recent proposals on distributed DL inference. These techniques propose splitting the DNN models across two or more network entities (i.e., end device, edge/fog server, and cloud server). Among these works, authors in [3] and [6] proposed horizontal splitting (layers-based), whereas works in [5], [7], and [8] proposed vertical splitting of DNN models across multiple layers. Despite the performance gains in terms of computational load, energy or latency achieved using these methods, none of these works provide sufficient guarantees to be adopted in real-time inference applications such as video surveillance. In video surveillance applications employing DL, frames are captured at a fixed predefined interval (aka frame rate). Thus, each frame must be processed by the DNN model

Manuscript received 27 November 2021; revised 11 May 2022 and 21 June 2022; accepted 7 August 2022. This work was supported in part by the Qatar University Internal under Grant IRCC-2020-001 and in part by the NPRP under Grant NPRP13S-0205-200265 from the Qatar National Research Fund (a member of Qatar Foundation). (Corresponding author: Muhammad Asif Khan.)

Muhammad Asif Khan and Ridha Hamila are with the Qatar University, Doha, Qatar (e-mail: mkhan@qu.edu.qa; hamila@qu.edu.qa).

Aiman Erbad is with the Hamad Bin Khalifa University, Doha, Qatar (e-mail: aerbada@hbku.edu.qa).

Moncef Gabbouj is with the Tampere University, 33100 Tampere, Finland (e-mail: moncef.gabbouj@tuni.fi).

Digital Object Identifier 10.1109/JSYST.2022.3198711

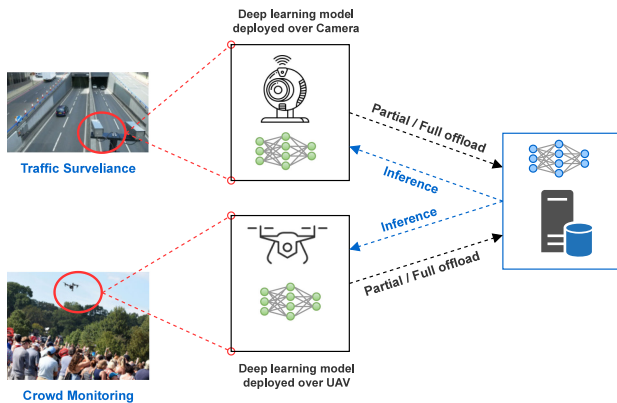


Fig. 1. Edge-enabled IoT-based video processing using deep neural network.

to infer (e.g., classify frame, detect anomaly, etc.) before the next frame arrives. In fact, the aforementioned works focus on the model splitting with different constraints and performance goals. For instance, Jankowski et al. [3] improved the model accuracy, Hu et al. [5] reduced inference latency but ignores energy consumption, and Zhao et al. [7] reduced the memory footprint. Kang et al. [6] although reduced the latency and energy consumption of the target DNN model, but incurs additional overhead and energy consumption.

In this article, we propose novel lightweight strategies based on horizontal layer-based splitting of inference computational tasks between IoT devices and edge server to improve real-time applications such as video surveillance with stringent delay constraints as illustrated in Fig. 1. More specifically,

- 1) We propose an *Early Split Strategy (ESS)* to split the DNN model in the lower layers and offload the maximum portion of the DNN to the edge server. The ESS scheme opt for maximum energy efficiency and is more suitable for battery-powered IoT devices such as unmanned aerial vehicles (UAVs), smartphones, etc.
- 2) We propose a *Late Split Strategy (LSS)* to exploit more local resources and offload only the DNN layers that involve heavy computation. The LSS scheme also respects the delay deadlines of the application. It is more suitable for IoT devices with regular power supply such as CCTV cameras, etc. It is also a preferred method for IoT networks with higher probability of communication bottlenecks.
- 3) We evaluate the proposed strategies using well-known deep learning models for image and video processing for performance in real-world video surveillance applications. Finally, we provide future insights on the particular applications of each scheme.

A. Article Organization

This article is organized as follows: Section I describes the problem of running deep learning models on the edge devices with real-time video surveillance as example. It includes a strong motivation for distributed inference computing requirement in IoT networks running deep learning and video services. Section II provides a comprehensive overview of the distributed

inference tasks and layers-based splitting of DNN models. The detailed system model of our article is presented in Section IV. Section V explains the proposed splitting techniques including the ESS and LSS strategies. Section VI presents the experimental set up, and presents the acquired results of our evaluation. Finally, Section VII concludes this article.

II. DISTRIBUTED INFERENCE COMPUTATION: AN OVERVIEW

This section provides a brief overview of how DNN inference can be performed in distributed manner over IoT devices and edge server.

A. DNN Computation—The AlexNet Example

An illustrative example of processing videos is the image classification using convolution neural network (CNN). However, most CNN models tend to be huge in size, i.e., these are comprised of several hundreds of layers and millions of parameters. For instance, the AlexNet [9] model is an efficient CNN model for image classification. The model as shown in Fig. 2 is comprised of five convolutional (Conv) layers and three fully connected (FC) layers. It has around 60 million parameters and 650 000 neurons. It takes around five to six days to train on two GTX 580 3 GB GPUs.

Running huge DNN models entirely on the IoT devices is not efficient and it requires large computational resources and memory footprints [10]. Hence, IoT (or any client) devices offload the DNN computational tasks to the edge server and the server run the DNN model on behalf of the requesting device. However, if the edge server does not have the preinstalled DNN model, offloading the entire DNN models to the server causes overhead. For instance, uploading the entire AlexNet model to the server takes around 24 s on 80 Mb/s wireless link [11]. A more practical method is to upload the DNN model incrementally (i.e., layer by layer) [11].

B. Splitting DNN Models

To minimize the computational load and energy consumption of the IoT device, several works have proposed splitting the DNN model computation in a way that heavy computational tasks in the DNN model are offloaded to a remote cloud or edge server. Among the splitting strategies, the layer-based splitting strategy [5], [12], [13], [14] is more intuitive and attractive. In layer-based DNN splitting, the DNN model that consists of several layers is cut into two parts such as the model is computed partially at the IoT device whereas the output of the last layer of this part is offloaded to the server for the remaining computation. Finally, the end result or inference (i.e., predicted class or anomaly) is sent back to the IoT device. Fig. 3 illustrates the layer-based splitting approach in DL models.

For instance, Table I presents the per-layer details of the AlexNet [9] model explained earlier. It can be observed that the output of some intermediate layers is significantly smaller than those of other layers. Hence, this provides an opportunity to partially compute the model (i.e., few layers) at the IoT device,

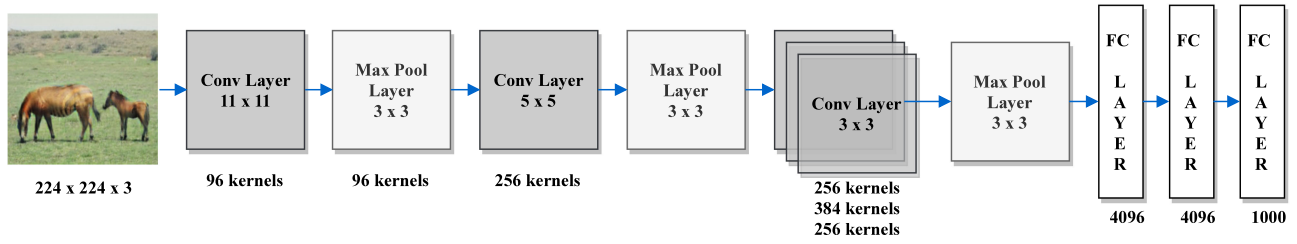


Fig. 2. AlexNet architecture.

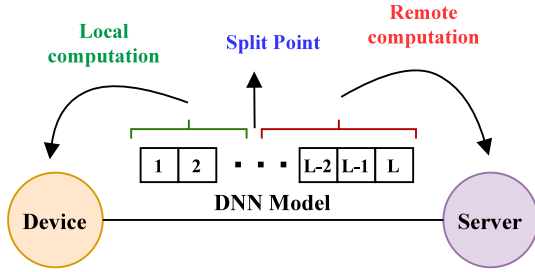


Fig. 3. Model splitting technique for distributed inference.

TABLE I
ALEXNET MODEL SUMMARY

Layer	Type	Parameters (M)	Size (MB)
1	Conv	0.03485	0.139
2	Conv	0.30720	1.229
3	Conv	0.88474	3.539
4	Conv	0.66355	2.654
5	Conv	0.44237	1.769
6	FC	37.74874	150.995
7	FC	16.77722	67.109
8	FC	4.09600	16.384
Total		60.95466	243.819

whereas offload the output of the last layer of the first partition to the server for the remaining computation.

The partitioning or splitting of the DNN model however should generally consider several tradeoffs among parameters such as computation load of the IoT device and the edge server, energy consumption of the IoT device, inference latency required by the application, and the wireless channel strength to transmit the output of the particular layer. When the IoT device has low processing power or heavy computational load, the splitting should be done such as to execute only few or low computational layers at the IoT device, whereas we offload the remaining heavy computational layers to the server. However, the splitting should also consider the communication delay to transmit the output of the particular layer to the server. If the channel data rate degrades severely, the splitting should be done such as the size of the data that is to be transmitted is small.

C. Sequential Versus Nonsequential DNN Models

DNN model can be sequential or nonsequential. In sequential DNNs, the input of one layer is directly connected to the output of the previous layer. In nonsequential DNNs, there can be some

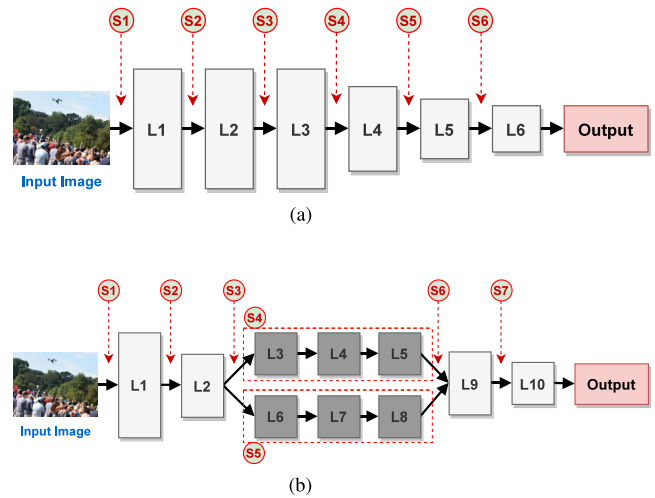


Fig. 4. Sequential and nonsequential DNN models.

layers that are connected to two or more layers, thus creating multiple paths between the first input layer and the last output layer. Fig. 4 illustrates the between the two types of DNNs.

Splitting a sequential DNN is simple and straightforward. In a sequential DNN that consists of L layers, there can be a maximum of $L - 1$ splitting points (e.g., the AlexNet model consists of 12 layers and thus has 11 splitting points). However, in nonsequential DNNs, the splitting should be done carefully. When splitting a nonsequential DNN model between two devices (e.g., an IoT device and a server) is required, it is usually done at the layers along the linear paths (e.g., at L_1 , L_2 , L_9 , and L_{10}). Splitting at the parallel paths (e.g., $L_3 - L_8$) can cause synchronization issues [13]. Thus, all parallel layers (e.g., $L_3 - L_8$) are usually grouped as a single block and computed at a single device. However, a mechanism that guarantees the execution of the last layers on both parallel paths (e.g., L_3 , and L_8) completes at the same time, further splitting points over the parallel paths can be considered.

D. Computation Load and Energy Efficiency in DNN

In DNN such as CNN, each convolution (Conv) layer convolves the input feature map (ifmap) with the kernel to generate the output feature map (ofmap). This process of convolution involves elementwise multiplication and accumulating (MAC) these products. Hence, the number of MACs is an accurate metric to estimate the computation load/latency of CNN models. The

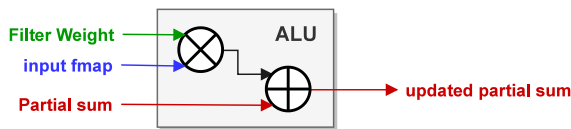


Fig. 5. MAC operation in arithmetic logic unit (ALU): Each MAC operation requires at least two memory reads and one memory write operation.

MAC operations in a CNN model involve around 99% of the computational load [15], whereas almost 90% of the execution time of a CNN model is due to Conv layers during inference task [16].

Energy consumption of DNN constitutes two portions, i.e., computation energy and data-flow or memory access energy. The total energy consumed to process a single CNN layer is the sum of the computation energy and memory access energy [17]. The computation energy is directly calculated from the number of multiplication-and-accumulation (MAC) operations performed at the layer. The data flow energy is computed from the energy consumed to access every bit at the hierarchical memory. Each MAC operation involves three memory reads access and one memory write access, as depicted in Fig. 5. The amount of energy consumption for memory access is different for each memory level. Usually, accessing dynamic random access memory (DRAM) requires 1 or 2 order of energy higher than accessing on-chip memory [18]. Yang et al. [15] showed that the amount of data in MAC operations in CNN does not directly increases with the increase in number of MAC operations due to the data reuse. Due to high reuse probability, the weight and activation in Conv layer has a high impact on energy than in FC layer.

III. RELATED WORK

Eshratifar et al. [19] demonstrated that local or remote computing alone is not an optimal approach while considering inference delay and energy consumption of the mobile device, and motivate splitting the inference task for optimal performance. To achieve the maximum performance gain from splitting DL models, optimal splitting strategies are required. Matsubara et al. [20] discussed the various challenges of split computing in complex neural networks focusing on the inference time reduction.

A layer-based model splitting technique, i.e., distributed DNN (DDNN) using hierarchical computation is proposed in [3]. In the DDNN scheme, the candidate split points are predefined based on the model accuracy. A local aggregator determines if the combined model (sent by all end devices) at given split points provides sufficient accuracy, it infers the sample, otherwise it send the model to the edge server. Eventually, the edge determines first if it can correctly infer the sample using the received model, it does; otherwise it sends the model to the cloud server. The limitation of the DDNN scheme [3] is that it targets the model accuracy and does not consider the inference time, thus making it less suitable for real-time application with strict delay requirements.

In [5], Hu et al. formulated the DNN partitioning problem by representing DNN as a directed acyclic graph (DAG). Each vertex in the DAG represents a layer of the DNN model, whereas

the edges (the link between vertices) represent communication. The method splits the DNN model into two parts (for processing at the edge and cloud, respectively) by considering the actual network condition with the objective to minimize the overall delay of processing a single frame. This method although tries to achieve the delay performance of the network, it does not consider the energy consumption which is a critical challenge in IoT networks. Moreover, the proposed work [5] evaluates the case of a single edge server and a single cloud server. DAG-based split inference is also proposed in [21]. In contrast to layer-based splitting, Zhao et al. [7] proposed a fused tile partitioning (FTP) method to split the CNN layers into independently distributable tasks. The layers are split in grid fashion to form several independent tasks by vertically fusing and are then distributed to IoT devices to minimize the memory footprint and communication overhead in the cluster. The article was motivated by the fact that layers based splitting may leads to aggressive or complete offloading when the IoT devices are severely constrained. However, the FTP method can easily fail in smaller clusters of severely constrained IoT devices. The failure becomes more evident when the number of IoT devices receiving the DNN tiles (parts of the layers fused vertically) are unable to compute the tasks within the available time frame in real-time applications. Mohammed et al. [8] proposed Distributed Inference Acceleration (DINA), i.e., a vertical segment-based method of splitting DNN to allow for more than one partition of a DNN model. Multiple partitions allow us to distribute the computations among multiple edge servers in the network. DINA [8] has two limitations. First, multiple partition splitting has no significance in the case of a single edge server in the network. Second, the vertical splitting in DINA has a lower bound on latency equal to the longer processing time of one of the segments. Thus, the performance gains in DINA can be less significant as compared to the complexity of the model. A more detailed insight into layer-based splitting of DNN in mobile devices is presented in Kang et al. [6]. The proposed algorithm *Neurosurgeon* [6] splits the DNN horizontally using prediction models. The prediction models estimates the latency and energy consumption for executing each layer on the mobile and cloud and dynamically decide the splitting of the DNN model accordingly. Although this technique may achieve better performance on the DNN computation, it incurs additional overhead to store, run, and update two additional prediction models. It is also inevitable as the prediction models will recur for every splitting decision thus supplementing the actual DNN overhead on the IoT device.

Tuli [22] proposed SplitPlace, a distributed placement of neural networks on mobile hosts. The placement decision is taken using multiarmed bandit (MAB) to meet the workload deadline. The work provided experimental evaluation using Raspberry-Pi devices on ResNet50-V2 [23], MobileNetV2 [24], and InceptionV3 [25] networks. A layer-based splitting strategy in CNN in proposed in [26]. However, the goal of splitting is to balance the computational load instead of inference time reduction. Li et al. [27] investigated the benefits of quantization in reducing the communication delay in transmitting the models over the wireless links in a cloud-edge collaborative inference. A

TABLE II
SYMBOLS AND MEANINGS

Parameter	Description
L	The total number of layers in the model
L_{conv}	The total number of conv layers in the model
L_{fc}	The total number of FC layers in the model
S_l	The output size of the l th layer
S_L	Total Size of the model consisting of L layers
F_e	CPU frequency of edge server (cycles/seconds)
f_n	CPU frequency of device n (cycles/seconds)
C_L	Total Computational load of the DNN model with L layers as number of multiplications
C_l	Computational load of the l th layer as number of multiplications
τ_l	Time required to process the l th layer at the device.
τ_e	Time required to process the l th layer at the edge.
P_i^t	Transmit power of device i
P_e^t	Transmit power of edge server
W	Communication bandwidth
$R(\gamma)$	Data rate at a given SNR value (γ)
\mathcal{T}	Delay deadline to process a whole frame

learning algorithm to solve the distributed inference is presented in [28]. The work presents a learning rule based on adaptive quantization and event-triggering.

The aforementioned relevant works propose different approaches to address the distributed inference problem, none of these provides a realistic solution that can be adopted in real-time IoT applications.

IV. SYSTEM MODEL

Consider a network that consists of N number of resource constrained devices. We assume that all devices are homogeneous in processing capabilities. More specifically, the processing power of all devices is constant ($P_n = P, \forall n \in N$). Each device is running a DNN model to perform common tasks such as image classification, etc. We assume each device has a pretrained generic DNN model that consists of L number of sequential layers. The size of each layer is denoted as S_l . In our system model, we use a single edge server with processing power P_e ($P_e \gg P_n \forall n \in N$). We assume that all devices in the network can connect to the edge via wireless link. Furthermore, the devices can exploit the edge computational power by offloading any task to it via the wireless link.

A. Computational Load

Let the computational load (the DNN task) is defined as the number of multiplications per second denoted as C . The computational load of a single layer is measured as the sum of multiplications performed to compute the output. The total computational load can be calculated as

$$C_L = \sum_{l=1}^L C_l. \quad (1)$$

Consider a neural network with N inputs at input layer x_i , the output layer a_j is calculated as

$$a_j = f \left(b_j + \sum_{i=1}^N W_{ij} x_i \right) \quad (2)$$

where f is the activation function, W_{ij} is the weights, and b_j is the bias vector. The total computations involved are the $N \times M$ multiplications, M additions, and M applications of activation function $f(\cdot)$.

If $X[C, H, W]$ is the input feature map (ifmap) with C as no. of channels, H height, and W width of the ifmap, $K \times K$ is the kernel size, the output feature map (ofmap) of Conv layer is calculated as

$$Y[b, n, u, v]^{conv} = \beta[n] + \sum_{c=1}^C \sum_{j=1}^J \sum_{k=1}^K (X^{conv}[b, c, v + j, u + k] \cdot \Theta[n, c, j, k]).$$

The pooling layer subsamples each channel of the input by taking the average or maximum to reduce the dimensionality of the CNN. Pooling operation can be formulated as

$$Y[b, n, u, v]^{pool} = \max_{p, q \in [1:K]} (X^{pool}[b, c, v + p, u + q]). \quad (3)$$

The computation workload of DL models is represented in terms of floating point operations (FLOPs) [29], [30]. FLOPs are the number of floating point operations carried out by a DL model. The number of FLOPs for Conv layer is computed as

$$C^{conv} = 2H \times W(C_{in}K^2 + 1)C_{out} \quad (4)$$

where H , W , and C_{in} are height, width, and number of features of ifmap, K is the kernel size, and C_{out} is the number of output channels. For FC layer, the number of FLOPs is calculated as

$$C^{fc} = (2I - 1)O \quad (5)$$

where I is the input dimensionality and O is the output dimensionality.

B. Latency

If C_l is the number of computations in the l th layer of a DL model to be processed, the time to process the layer locally at device n is calculated as

$$t_l^{local} = \frac{C_l}{f_n} \quad (6)$$

where f_n is the CPU-cycle frequency of device n .

The total time required to process the complete DL model that comprised of L layers, locally at the device is calculated as

$$T^{local} = \frac{\sum_{l=1}^L C_l}{f_n}. \quad (7)$$

Similarly, the computational latency to execute a single layer remotely at the server is calculated as

$$t_l^{edge} = \frac{C_l}{F_e} \quad (8)$$

where F_e is the CPU-cycle frequency of edge server. The total time required to process the complete DL model that comprised of L layers, remotely at the server is calculated as

$$T^{edge} = \frac{\sum_{l=1}^L C_l}{F_e}. \quad (9)$$

If the task to be executed at the server, there can be extra delay associated with the two-way data transfer, i.e., the communication delay to transmit the data (i.e., feature maps) from the IoT device to the server, and the communication delay to transmit the processing results (i.e., predictions) from the server back to the device.

If the total available channel bandwidth is W , and we use orthogonal multiple access scheme (such as OFDMA) in which the devices can offload their tasks to the server via orthogonal subbands simultaneously, the achieved data rate by a single device is calculated as

$$R(\gamma) = \frac{W}{n} \times \log_2(1 + \gamma_{(n,e)}) \quad (10)$$

where $\gamma_{(n,e)}$ is the signal-to-noise ratio (SNR) of the wireless channel between the server and the n th device. The channel data rate is used to calculate the communication delay for transmitting a layer output to the server as $S_l/R(\gamma)$.

When multiple IoT devices offload their tasks to the same edge server, the tasks (i.e., DNN layers to be executed) are queued in the server buffer. To avoid complexity, we use first-in first-out (FIFO) scheme to execute tasks in the server queue and calculate this extra delay analytically as T^{queue} .

C. Energy Consumption

Running DNN model fully or partially consumes energy at end devices. If the inference task is completed locally at the device, the energy consumed is mainly for executing the DNN model. However, if the DNN model is partially executed at the device and partially offloaded to the server, the energy consumption is the sum of energy consumed for executing the partial model (few layers) and the energy consumed for transferring the remaining DNN layers. Recall that the number of computations to process the l th layer is C_l , the energy consumption of processing a layer locally at the device is

$$E_l^{comp} = \xi_n C_l (f_n)^2 \quad (11)$$

where ξ_n is the switched capacitance of the device processor.

The energy consumption of offloading a layer to the edge server is the sum of energy consumed in transmitting to the

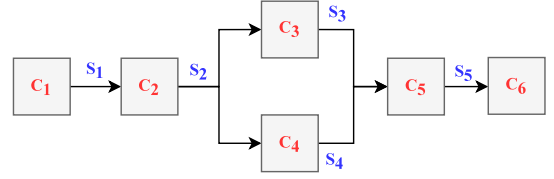


Fig. 6. Representation of nonsequential models.

edge and the energy consumed in receiving the results back

$$E_l^{comm} = E_{tx}^l + E_{rx}^l. \quad (12)$$

When a server completes the model execution, it transmits the final output which can be a predicted class in image recognition application. As the size in bits of the output is very small, we assume the energy consumed by the IoT device in receiving data (i.e., model output) is negligible, i.e., $E_{rx} \approx 0$. Thus, the total device energy consumption in offloading a single layer becomes

$$E_l^{comm} = E_{tx}^l = P_n^t \frac{S_l}{R(\gamma)} \quad (13)$$

where P_n^t is the transmit power of device n . Finally, in distributed inference, the total energy consumption is computed as the sum of computational energy of processing few layers locally and the communication energy of offloading the remaining layers to the server.

V. PROPOSED SCHEME

We consider resource-constrained IoT devices connected to an edge server as illustrated in Fig. 3. Each IoT device is running a DL-based video processing application to perform a common inference task (e.g., anomaly detection). The DL application requires the inference task to be completed within a fixed delay deadline (constrained by the frame capturing rate of IoT devices). The IoT devices must complete the inference task within the delay deadline. If the local processing power is insufficient, it offloads the DL model fully or partially to the edge server for faster computation. The objective is thus to find the optimal splitting of the DNN model such that task can be completed within the delay deadline. Furthermore, the IoT devices may be battery powered or may be connected to direct power source. If the IoT device is battery powered, energy consumption is an additional constraint and must be reduced.

As illustrated in Section I, the DNN model can be sequential or nonsequential. For sequential models, the layer splitting technique is just to find the index of the layer in the layers sequence. For nonsequential models, we propose to represent the model first in an equivalent (in terms of computational/memory load) sequential model. Consider a generic nonsequential DNN model in Fig. 6, where the letter C represents the computational load of the layer and S represents the size of the ofmap of the respective layer.

Let C^* is the matrix representing the computation load of the model:

$$C = \begin{bmatrix} C_1 & C_2 & C_3 & C_5 & C_6 \\ 0 & 0 & C_4 & 0 & 0 \end{bmatrix}. \quad (14)$$

where each element of the matrix is the number of computations at each single layer, the corresponding equivalent sequential matrix C is written such that each

$$C = [C_1 \ C_2 \ \text{sum}(C_3, C_4) \ C_5 \ C_6]. \quad (15)$$

In the sum way, the layer's parameter size matrix is also computed to represent the memory load of the respective nonsequential model.

In the following, we propose two distinct techniques for layer-based splitting of the DNN model for distributed inference computation.

A. Early Split Strategy (ESS)

In this article, we assume that the IoT device has scarce energy resource, whereas the inference task must be completed within a delay deadline for optimum performance of the system. The IoT device thus offloads the maximum number of layers to the edge server. The layers are offloaded as soon as the communication channel provides sufficient data rate to transmit the layer parameters, while meeting the delay constraint. The problem is formulated as follows: Let x_i is a binary variable that represents the layer splitting

$$x_i = \begin{cases} 1, & \text{If } i \text{ is the split point} \\ 0, & \text{otherwise} \end{cases}$$

$$\min \sum_{i=1}^L x_i \quad \forall i \in L \quad (16)$$

subject to

$$\sum_{i=1}^L T^{\text{comp}(n)}[1:i] + T^{\text{comm}}[i] + T^{\text{queue}} \\ T^{\text{comp}(e)}[i:L] \leq \mathcal{T} \quad \forall n \in N, i = 0, 1, 2, \dots, L \quad (16a)$$

$$\sum_{i=1}^L x_i = 1 \quad \forall i = 0, 1, 2, \dots, L. \quad (16b)$$

Equation (16) ensures early splitting of the model. Equation (16a) enforces the task completion deadline. Equation (16b) is to ensure that the model must be split at exactly single point.

We propose the *Early Split Strategy (ESS)* heuristic in Algorithm 1 to solve the optimization problem in (16).

The ESS algorithm works as follow: The IoT devices compute the input layer locally and update the value of remaining delay deadline $\mathcal{T}^* = \mathcal{T} - T^{\text{local}}$. Then at each layer, it first checks whether it can offload the remaining layers while meeting the inference delay deadline. If yes, it offloads the remaining layers to the edge server, otherwise; it continues local processing for the next layer. The main decisive factor here is the communication channel. If the channel gain is high enough to support the offloading of the layer, the model is offloaded immediately. The ESS algorithm ensures the inference task to be completed within the delay deadline and perform the model splitting at early stage by limiting the number of layers for local processing and offload the maximum portion of the model to the edge server. In this

Algorithm 1: Early Split Strategy (ESS).

input : $L, C_L, S_L, F_e, f_n, R(\gamma), \mathcal{T}, T^{\text{queue}}$

```

1 Initialize
2 Process input layer (i.e.,  $C_0$ ) locally and calculate
  processing delay.
3  $T^{\text{local}} = \frac{C[0]}{f_n}$ 
4  $i = 1$ 
5 while  $i < L$  do
6   Calculate remaining time after processing input
    layer.
7    $\mathcal{T}^* = \mathcal{T} - T^{\text{local}}$ 
8   Calculate time for edge processing [i:L] layers.
9    $T^{\text{comp}(e)} = \frac{C[i:L]}{F_n}$ 
10  Calculate time for transmitting [i:L] layers.
11   $T^{\text{comm}} = \frac{S[i]}{R(\gamma)_l}$ 
12  if  $T^{\text{queue}} + T^{\text{comp}(e)} + T^{\text{comm}} \leq \mathcal{T}^*$  then
13    Offload remaining layers to the edge.
14     $i = L$ 
15  else
16    Continue processing the next layer.
17     $i = i + 1$  //increment s
18  end
19 end
20 end

```

way, the energy consumption of the IoT device can be greatly reduced.

B. Late Split Strategy (LSS)

In this article, we assume that the IoT device has no energy constraint (e.g., it is connected to a regular power source) and the only constraint on the inference task is the inference delay deadline. The IoT device will thus attempt to execute more tasks locally and offloads only when the local computational resource is insufficient to complete the task. It is worth to notice that the inference delay deadline also captures the computational resource of the IoT device as low computational power of IoT device will cause higher inference delay when local execution is adopted. The problem is formulated as follows: Given x_i the decision variable, we aim to maximize the number of layers processed locally

$$\max \sum_{i=1}^L x_i \quad \forall i \in L \quad (17)$$

subject to:

$$\sum_{s=i}^L T^{\text{comp}(n)}[1:i] + T^{\text{comm}}[i] T^{\text{queue}} + \\ T^{\text{comp}(e)}[i:L] \leq \mathcal{T} \quad \forall n \in N \quad (17a)$$

$$\sum_{i=1}^L x_i = 1 \quad \forall i = 0, 1, 2, \dots, L. \quad (17b)$$

Algorithm 2: Late Split Strategy (LSS).

input : $L, C_L, S_L, F_e, f_n, R(\gamma), \mathcal{T}$
output: g

```

1 Initialize
2 Process input layer locally
3  $T^{local} = \frac{C[0]}{f_n}$ 
4  $i = 1$ 
5 while  $i < L$  do
6   Calculate processing delay of first  $[0 : i]$  layers.
7    $T^{local} = \frac{C[0:i]}{f_n}$ 
8   Calculate remaining time.
9    $\mathcal{T}^* = \mathcal{T} - T^{local}$ 
10  Calculate processing time of remaining layers at
    the edge using ComputeDelay() function.
11   $T^{dist} \leftarrow \text{ComputeDelay}(i)$ 
12  if  $T^{dist} > \mathcal{T}^*$  then
13    If remote execution time is greater than
    remaining time, process another layer locally.
14     $i = i + 1$ 
15     $g = 0$ 
16  else
17    If remote execution time is less than
    remaining time, offload remaining (L-i)
    layers to the edge.  $g = L - i$ 
18     $i = L$ 
19  end
20 end
21 end

22 Function ComputeDelay( $i$ ):
23   // Function to compute distributed processing delay
    given a split point (i.e., sum of time to transmit
     $i^{th}$  layer, and time to process last (L-s) layers.
     $T^{comp} = \frac{C[1:i]}{F_e}$ 
24    $T^{comm} = \frac{S[i:L]}{R(\delta)}$ 
25    $T^{dist} = T^{queue} + T^{comp}(e) + T^{comm}$ 
26   return  $T^{dist}$ 

```

Equation (17) ensures late splitting of the model. Equation (17a) enforces the task completion deadline. Equation (17b) is to ensure that the model must be split at exactly single point. We propose the *Late Split Strategy (LSS)* heuristic in Algorithm 2 to solve the optimization problem in (17).

The algorithm works as follow: The IoT device starts computing the input layer locally. Then, at each layer, it first check whether it can compute the next layer locally while meeting the inference delay deadline. If yes, it continues local processing for the next layer. If the delay deadline cannot be met, it then offloads the remaining layers to the edge server. At each layer, the algorithm updates the delay deadline (\mathcal{T}) by calculating the remaining time (\mathcal{T}^*) to process the entire model. Then, the distributed delay (T^{dist}) for the remaining layers is calculated using the equations in Section IV. If the distributed delay (T^{dist}) is greater than the updated delay deadline (\mathcal{T}^*), it must process the next layer locally, otherwise it decides the split point and

TABLE III
SIMULATION PARAMETERS

Parameter	Value
No. of devices (N)	10
Edge computational power (F_e)	$[1, 4] \times 10^9$
IoT device computational power (f_n)	$[0.4, 1] \times 10^9$
Inference delay deadline (τ)	$[1, 2]$ s
No. of layers of the DNN model (L)	23 (VGG-16)
Effective capacitance (k)	1×10^{-28}

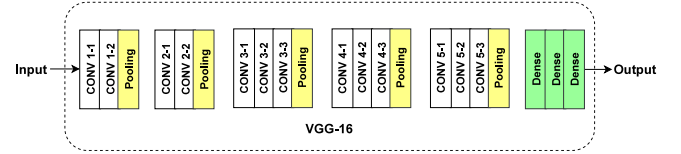


Fig. 7. VGG16 CNN architecture.

offloads the remaining layers to the edge server to complete the inference task.

The LSS algorithm offload the computation only when the inference delay cannot be met using the local processing. Processing more layers locally results in higher energy consumption of the IoT device. However, as the LSS algorithm is proposed for IoT devices with regular power sources, the energy consumption is not a constraint. On the other hand, the inference delay deadline is inherited in the LSS strategy and is always ensured.

VI. RESULTS AND DISCUSSION

A. Simulation Settings

We consider N IoT devices and a single edge server in our analysis. We use the VGG16 [31] and MobileNet_V2 [32] CNN models. Both VGG16 and MobileNet_V2 are popular and widely used CNN model. The VGG16 architecture provides the basis for other object recognition models. They also surpasses baselines on many tasks and datasets beyond ImageNet. Due to their single single-column architecture, these are more suitable for layer-based splitting schemes. Furthermore, both these models have sufficient layers to test our algorithms. For instance, its predecessor, the AlexNet model would be too shallow. AlexNet has only eight layers and has large size of the parameters set (60.9 M parameters). Hence, a model with few number of layers (and more than 57 M parameters in the last three layers) would not be a reasonable choice to evaluate the performance of splitting techniques. The performance of splitting is more evident in large but dense models [33].

The simulation parameters are listed in Table III.

B. Simulation Results

We first evaluated the performance of the proposed algorithms in Section IV by considering a single IoT device and an edge server. Both VGG16 (Fig. 7) and MobileNet_V2 models are used to analyze the performance of ESS and LSS schemes. Both VGG16 and MobileNet_V2 have input size of 224×224 .

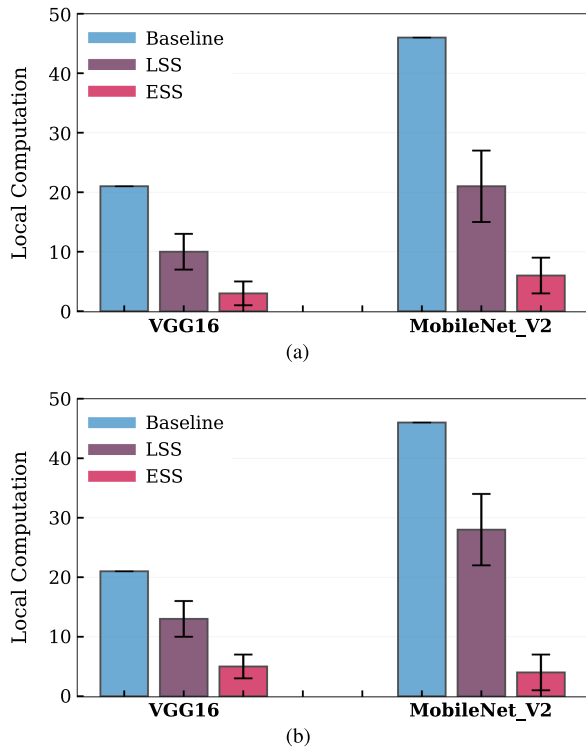


Fig. 8. Distributed inference over VGG16 CNN model: An analysis of the computational load. (a) No. of Locally Processed layers (using inference delay deadline 1s). (b) No. of Locally Processed layers (using inference delay deadline 2s).

Hence, the ESS and LSS schemes produces the same results for any real-world dataset. Moreover, we repeated the analysis for two different delay deadlines of 1 s and 2 s, respectively. Fig. 8(a) (for application delay deadline = 1 s) and Fig. 8(b) (for application delay deadline = 2 s) present the results in terms of local computational load (i.e., average number of layers that are processed locally).

It is worth to note that the split point changes over time depending upon the local and remote processing resources, and the quality of the communication channel at the respective instance of time. In such case, the number of layers to be processed locally or offloaded to the edge server always vary. Hence, we present the mean of the locally processed layers for each frame of the video sequence. It can be observed in Fig. 8(a) that with a delay deadline of 1 s, the ESS scheme encounters minimum computational load for both VGG16 and MobileNet_V2 models. On average, the ESS scheme allows us to process three layers and five layers locally for VGG16 and MobileNet_V2, and offload the remaining to the edge. Similarly, in the LSS scheme, ten layers are offloaded to the edge server for VGG16 model, and 21 layers are offloaded for MobileNet_V2 model. The intuition behind such behavior of the ESS and LSS scheme is the design principle. ESS is designed to ensure maximum saving of the local resources and exploiting the maximum possible edge resources, provided that the communication channel supports the transfer of the respective layer data and respecting the total delay deadline of processing the frame. On the other hand, the

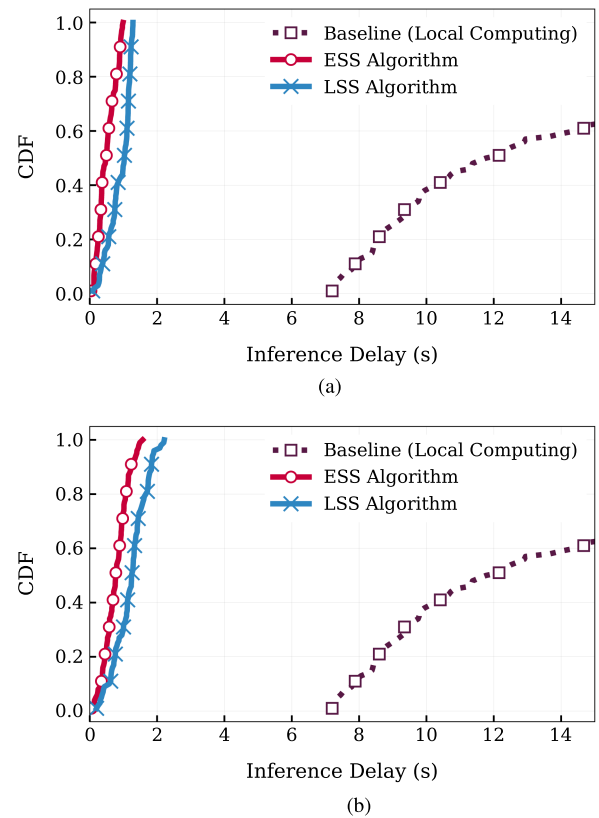


Fig. 9. Distributed inference over VGG16 CNN model: An analysis of the inference delay. (a) Inference delay (with deadline = 1s). (b) Inference delay (with deadline = 2s).

LSS scheme is least opportunistic and use the local resources to the maximum limit. It offloads the layers only when the local resources become insufficient to process the remaining layers in the given delay deadline. Fig. 8(b) further signifies the acquired results. When the delay deadline is 2 s, the LSS scheme allows us to process 13 for VGG16 and 28 for MobileNet_V2 models. The ESS does not depend upon the delay deadline and thus allows almost similar performance as with deadline of 1 s.

Next, we evaluate the inference delay for the ESS and LSS schemes. It is worth to note that the inference delay depends upon the splitting strategy and more specifically on the splitting point. Hence, we hereby provide an analysis for the VGG16 model. The performance of both schemes on MobileNet_V2 or any other DNN models would depend upon the model depth and size of the respective layers. However, the relative benefits of the proposed schemes should be similar to the results presented here for the VGG16 model. As depicted in Fig. 9, the inference delay for local computation is very huge and much greater than the delay deadlines. Hence, distributed inference is needed. In Fig. 9(a) and (b), the inference delay for both ESS and LSS schemes is compared for two different delay deadlines (1 s and 2 s, respectively) over the VGG16 model. As ESS scheme is more opportunistic and attempts to split the model at the earliest, offloading more number of layers to the edge server, the inference task is always completed quickly that results in less inference delay. However, this benefit is at the cost of more

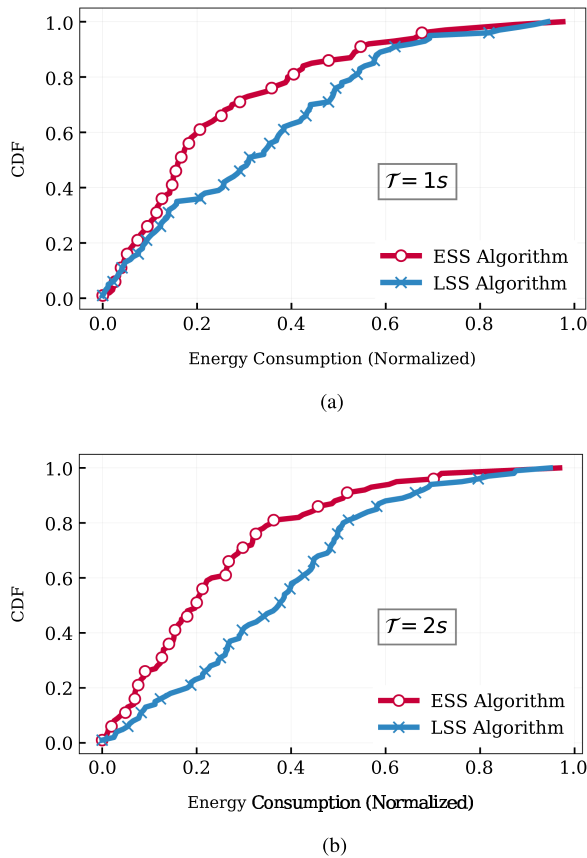


Fig. 10. Distributed inference over VGG16 CNN model: An analysis of the energy saving. (a) Energy consumption (with deadline = 1s). (b) Energy consumption (with deadline = 2s).

edge resources, which may not be available for free. On the other hand, in LSS scheme, the inference delay is relatively larger than ESS. This is because LSS allows us to utilize maximum use of the local resources and offload only when necessary to meet the delay deadline. One may observe that the inference delay in LSS is sometimes higher than the delay deadline. This is due to the fact the estimated edge resources vary over time and sometimes can lead to slower processing of the offload tasks than the estimated time.

Reducing the local computation and increasing the amount of tasks offloaded to the edge server results in reduced energy consumption of IoT devices. The comparison of energy consumption using is depicted in Fig. 10. Fig. 10(a) and (b) shows the CDF of normalized energy consumption using both ESS and LSS schemes under delay deadlines of 1 s and 2 s, respectively. It is evident from the figures that the ESS scheme incurs less energy consumption of IoT devices than the LSS scheme. The difference is even more when the delay deadline increases to 2 s. On average, the ESS and LSS schemes offers energy saving of up to 18% and 52%, respectively, for VGG16 CNN model.

Figs. 8–10 evaluate the performance of ESS and LSS scheme for a single IoT device. The algorithms are further evaluated using multiple IoT devices to validate the performance gains. Fig. 11 shows the performance of both schemes using ten IoT devices over VGG16 model. When considering multiple devices,

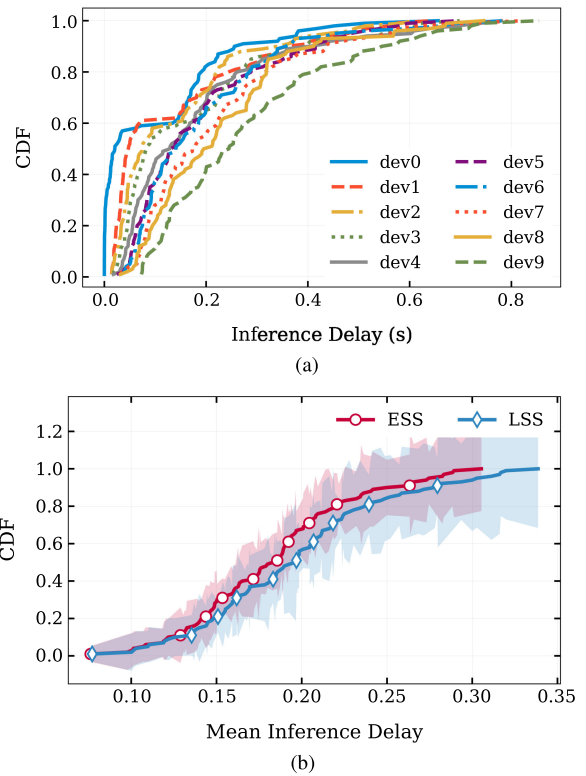


Fig. 11. Distributed inference over VGG16 CNN model for ten devices. (a) Inference delay of distributed inference. (b) Inference delay of distributed inference.

the algorithms take into account the queuing delay. The queuing delay can produce unpredictable results if it is not taken into account. Thus, both the proposed schemes take the offloading decisions based on the overall delay (i.e., computation delay + communication delay + queuing delay) to avoid exceeding the delay threshold.

In Fig. 11(a), the inference delay for each device using ESS scheme is plotted, which clearly shows that the delay deadline is met by each device. To compare the performance gain in terms of inference delay, we plotted the mean delay (solid line) with standard deviation (filled area) incurred by all devices to process each frame. It can be seen that the maximum mean delay using ESS scheme is around 0.31 s and 0.34 s for LSS scheme. Also, the ESS scheme always incurs less delay than the LSS scheme. As from the single device analysis presented earlier in this section, it is very clear that the lower inference delay is due to the less number of layers processed locally, which results in lower energy consumption of IoT devices.

Finally, we extend our analysis to measure the effect of increasing number of IoT devices served by a single edge server. In this analysis, the number of IoT devices are increased from 1 to 20 with a step size of 5, and the mean inference delay of the network is computed. The results are exhibited in Fig. 12.

It can be observed in Fig. 12 that the average inference delay of all IoT devices increases when more devices are offloading the computations simultaneously to the single edge server. Both algorithms are still able to meet the delay deadlines up to a certain

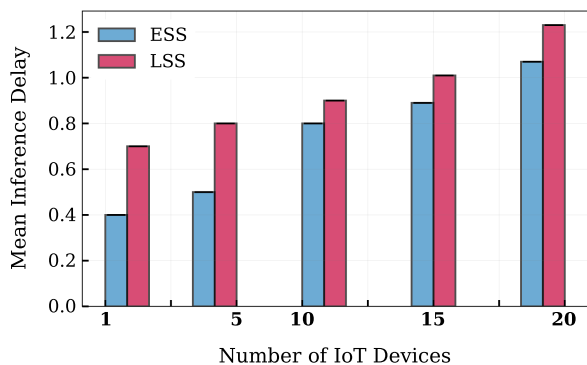


Fig. 12. Mean inference delay versus the number of IoT devices in the network.

network size (15 devices). However, when the number of devices further increases, the inference delay increases slightly due to the queuing delay at the edge server. The analysis shows that the inference delay increases by an average rate of 30% for every increase of five additional devices that are offloading tasks to the server. However, we believe that the statistic can vary depending upon various factors such as wireless channel quality of each device, the DNN model running over the devices, the processing capabilities of IoT devices, and the edge server. Nevertheless, the performance gains of both ESS and LSS schemes over other metrics such as energy consumption and computational gain achieved at the cost increase in the inference delay is nontrivial. We highly recommend that accurate tradeoffs among the required inference delay and the desired computational load and power saving should be carried out to implement the proposed strategies in practical deployments.

VII. CONCLUSION

This article presents two lightweight and efficient algorithms (i.e., ESS and LSS) to implement distributed inference on resource constrained IoT devices. Both techniques effectively split the DL models in a way that the inference delay constraint of the application is respected. While attaining the minimum delay deadline in both schemes, the ESS scheme additionally achieves higher energy efficiency by maximum utilization of the edge resources and is thus a more suitable technique for battery powered IoT devices (e.g., drones, smartphones, smart watches, etc.). Conversely, the LSS scheme is more suitable for IoT devices with a regular power source (e.g., surveillance cameras with Power over Ethernet (PoE) feature). Simulation results show that the ESS scheme typically achieves 15–20% reduction in the computational load of IoT device whereas the LSS scheme achieves up to 60% reduction. The amount of gain achieved for computational load provides a direct indication of the potential energy saving in IoT devices, which is also confirmed by the simulations. The results are significant and provide basis for the practical use cases of both schemes as stated earlier. While both the ESS and LSS schemes always achieve the target delay deadline and the reduction in the inference delay may be seen as trivial in small networks, it is highly desired in large IoT networks with limited server resources. As future

work, we suggest to extend the proposed schemes to arbitrary CNN architectures and other types of deep learning models. We also intend to test the performance of the proposed schemes in real-world IoT networks with mix of IoT devices running inference over CPUs and GPUs.

ACKNOWLEDGMENT

The statements made herein are solely the responsibility of the authors.

REFERENCES

- [1] D. R.-J. G.-J. Rydning, J. Reinsel, and J. Gantz, "The digitization of the world from edge to core," Framingham: International Data Corporation, vol. 16, 2018.
- [2] C. W. Chen, "Internet of video things: Next-generation IoT with visual sensors," *IEEE Internet Things J.*, vol. 7, pp. 6676–6685, Aug. 2020.
- [3] S. Teerapittayanon, B. McDanel, and H. T. Kung, "Distributed deep neural networks over the cloud, the edge and end devices," in *Proc. IEEE 37th Int. Conf. Distrib. Comput. Syst.*, 2017, pp. 328–339.
- [4] M. Jankowski, D. Gündüz, and K. Mikolajczyk, "Wireless image retrieval at the edge," *IEEE J. Sel. Areas Commun.*, vol. 39, no. 1, pp. 89–100, Jan. 2021.
- [5] C. Hu, W. Bao, D. Wang, and F. Liu, "Dynamic adaptive DNN surgery for inference acceleration on the edge," in *Proc. IEEE Conf. Comput. Commun.*, 2015, pp. 1423–1431.
- [6] Y. Kang et al., "Neurosurgeon: Collaborative intelligence between the cloud and mobile edge," *SIGARCH Comput. Archit. News*, vol. 45, pp. 615–629, Apr. 2017.
- [7] Z. Zhao, K. M. Barijough, and A. Gerstlauer, "DeepThings: Distributed adaptive deep learning inference on resource-constrained IoT edge clusters," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 37, no. 11, pp. 2348–2359, Nov. 2018.
- [8] T. Mohammed, C. Joe-Wong, R. Babbar, and M. D. Francesco, "Distributed inference acceleration with adaptive DNN partitioning and offloading," in *Proc. IEEE Conf. Comput. Commun.*, 2020, pp. 854–863.
- [9] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2012, pp. 1097–1105.
- [10] N. D. Lane, S. Bhattacharya, P. Georgiev, C. Forlivesi, and F. Kawsar, "An early resource characterization of deep learning on wearables, smartphones and Internet-of-Things devices," in *Proc. Int. Workshop Internet of Things Towards Appl.*, 2015, pp. 7–12.
- [11] H.-J. Lee, "IONN: Incremental offloading of neural network computations from mobile devices to edge servers," in *Proc. ACM Symp. Cloud Comput.*, 2018, pp. 401–411.
- [12] C. Alippi, S. Disabato, and M. Roveri, "Moving convolutional neural networks to embedded systems: The AlexNet and VGG-16 case," in *Proc. 17th ACM/IEEE Int. Conf. Inf. Process. Sensor Netw.*, 2018, pp. 212–223.
- [13] F. McNamee, S. Dustadar, P. Kilpatrick, W. Shi, I. Spence, and B. Varghese, "A case for adaptive deep neural networks in edge computing," in *Proc. IEEE 14th Int. Conf. Cloud Comput.*, 2021, pp. 43–52, doi: [10.1109/CLOUD53861.2021.00017](https://doi.org/10.1109/CLOUD53861.2021.00017).
- [14] M. Samragh, H. Hosseini, K. Azarian, and J. Soriaga, "Private split inference of deep networks," 2020.
- [15] T.-J. Yang, Y.-H. Chen, and V. Sze, "Designing energy-efficient convolutional neural networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2017, pp. 6071–6079.
- [16] J. Cong and B. Xiao, "Minimizing computation in convolutional neural networks," in *Proc. Int. Conf. Artif. Neural Netw. Mach. Learn.*, 2014, pp. 281–290.
- [17] T.-J. Yang, Y.-H. Chen, J. Emer, and V. Sze, "A method to estimate the energy consumption of deep neural networks," in *Proc. 51st Asilomar Conf. Signals Syst. Comput.*, 2018, pp. 1916–1920.
- [18] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, "Efficient processing of deep neural networks: A tutorial and survey," *Proc. IEEE*, vol. 105, no. 12, pp. 2295–2329, Dec. 2017.
- [19] A. E. Eshratifar, M. S. Abrishami, and M. Pedram, "JointDNN: An efficient training and inference engine for intelligent mobile cloud computing services," *IEEE Trans. Mobile Comput.*, vol. 20, no. 2, pp. 565–576, Feb. 2021.

- [20] Y. Matsubara and M. Levorato, "Split computing for complex object detectors: Challenges and preliminary results," in *Proc. 4th Int. Workshop Embedded Mobile Deep Learn.*, 2020, pp. 7–12.
- [21] A. Parthasarathy and B. Krishnamachari, "DEFER: Distributed edge inference for deep neural networks," in *Proc. 14th Int. Conf. Commun. Syst. Netw.*, 2022, pp. 749–753.
- [22] S. Tuli, "SplitPlace: Intelligent placement of split neural nets in mobile edge environments," *SIGMETRICS Perform. Eval. Rev.*, New York, NY, USA: Association for Computing Machinery, vol. 49, no. 2, pp. 63–65, Jan. 2022, doi: [10.1145/3512798.3512821](https://doi.org/10.1145/3512798.3512821).
- [23] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 770–778.
- [24] A. G. Howard et al., "MobileNets: Efficient convolutional neural networks for mobile vision applications," 2017, *arXiv:1704.04861*.
- [25] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 2818–2826.
- [26] D. Hu and B. Krishnamachari, "Fast and accurate streaming CNN inference via communication compression on the edge," in *Proc. IEEE/ACM 5th Int. Conf. Internet-of-Things Des. Implementation*, 2020, pp. 157–163.
- [27] G. Li, L. Liu, X. Wang, X.-J. Dong, P. Zhao, and X. Feng, "Auto-tuning neural network quantization framework for collaborative inference between the cloud and edge," in *Proc. Int. Conf. Artif. Neural Netw.*, 2018, pp. 402–411.
- [28] A. Mitra, J. A. Richards, S. Bagchi, and S. Sundaram, "Distributed inference with sparse and quantized communication," *IEEE Trans. Signal Process.*, vol. 69, pp. 3906–3921, 2021.
- [29] X. Dong, J. Huang, Y. Yang, and S. Yan, "More is less: A more complicated network with less inference complexity," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2017, pp. 1895–1903.
- [30] T. Vu, E. Wen, and R. Nehoran, "How not to give a FLOP: Combining regularization and pruning for efficient inference," 2020, *arXiv:2003.13593*.
- [31] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *Proc. Int. Conf. Learn. Representations*, 2015.
- [32] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "MobileNetV2: Inverted residuals and linear bottlenecks," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 4510–4520.
- [33] A. Singh, P. Vepakomma, O. Gupta, and R. Raskar, "Detailed comparison of communication efficiency of split learning and federated learning," 2019, *arXiv:1909.09145*.



Muhammad Asif Khan (Senior Member, IEEE) received the B.Sc. degree in telecommunication engineering from the University of Engineering and Technology Peshawar, Peshawar, Pakistan, in 2009, the M.Sc. degree in telecommunication engineering from the University of Engineering and Technology Taxila, Taxila, Pakistan, in 2013, and the Ph.D. degree in electrical engineering from Qatar University, Doha, Qatar, in 2020.

He was a Researcher Assistant with Qatar University from 2014 to 2015 and at Qatar Mobility Innovation Center from 2016 to 2017. He is currently working as a postdoctoral researcher at Qatar University. His current research interests include wireless communication, mobile edge computing, and distributed machine learning. For more detailed information, please visit his homepage: <http://www.asifk.me>.



Ridha Hamila (Senior Member, IEEE) received the M.Sc., LicTech with distinction, and Ph.D. degrees in telecommunications from Tampere University of Technology (TUT), Tampere, Finland, in 1996, 1999, and 2002, respectively.

He is currently a Full Professor with the Department of Electrical Engineering, Qatar University, Qatar. From 1994 to 2002, he held various research and teaching positions at TUT within the Department of Information Technology, Finland. His current research interests include mobile and broadband wireless communication systems, mobile edge computing, Internet of Everything, and machine learning. In these areas, he has published more than 200 journal and conference papers most of them in the peer reviewed IEEE publications, filed seven U.S. patents, and wrote numerous confidential industrial research reports. He has been involved in several past and current industrial projects, Ooreedo, Qatar National Research Fund, Finnish Academy projects, EU research, and education programs. He supervised a large number of under/graduate students and postdoctoral fellows. He organized many international workshops and conferences.



Aiman Erbad (Senior Member, IEEE) received the M.Sc. degree from the University of Essex, Colchester, U.K., in 2005, and the Ph.D. degree in computer science from the University of British Columbia, Vancouver, Canada, in 2012.

He is currently an Associate Professor with the College of Science and Engineering, Hamad Bin Khalifa University, Ar-Rayyan, Qatar. His research interests include cloud computing, edge computing, the IoT, private and secure networks, and multimedia systems.

Dr. Erbad received the Platinum Award from H. H. Emir Sheikh Tamim bin Hamad Al Thani at the Education Excellence Day 2013 (Ph.D. category). He also received the 2020 Best Research Paper Award from Computer Communications, the IWCMC 2019 Best Paper Award, and the IEEE CCWC 2017 Best Paper Award. His research interest spans cloud computing, edge computing, IoT, distributed AI algorithms, and private/secure networks. He is currently an Editor in the International Journal of Sensor Networks (IJSNet), an Editor in KSII Transactions on Internet and Information Systems, and served as a Guest Editor in IEEE Networks.



Moncef Gabbouj (Fellow, IEEE) received the Ph.D. degree in electrical engineering from Purdue University, West Lafayette, Indiana, in 1989.

He is a well-established world expert in the field of image processing, and held the prestigious post of Academy of Finland Professor during 2011–2015. He has been leading the Multimedia Research Group for nearly 25 years and managed successfully a large number of projects in excess of 18 M Euro. He has supervised 45 Ph.D. theses and over 50 M.Sc. theses. He is the author of several books and over 700 papers.

His research interests include Big Data analytics, multimedia content-based analysis, indexing and retrieval, artificial intelligence, machine learning, pattern recognition, nonlinear signal and image processing and analysis, voice conversion, and video processing and coding. He served as an Associate Editor and Guest Editor of many IEEE, and international journals.

Dr. Gabbouj is a fellow of the IEEE and a member of the Academia Europaea and the Finnish Academy of Science and Letters. He is the past Chairman of the IEEE CAS TC on DSP and committee member of the IEEE Fourier Award for Signal Processing.