ScissionLite: Accelerating Distributed Deep Learning with Lightweight Data Compression for IIoT

Hyunho Ahn, Munkyu Lee, Sihoon Seong, Gap-Joo Na, In-Geol Chun, Blesson Varghese, and Cheol-Ho Hong

***Abstract*— Industrial Internet of Things (IIoT) applications can benefit from leveraging edge computing. For example, applications relying on deep neural network (DNN) models can be sliced and distributed across the IIoT device and the edge of the network for decreasing the latency of inference. However, low network performance between IIoT devices and the edge is often a bottleneck. In this study, we propose ScissionLite, a holistic framework for accelerating distributed DNN inference using lightweight data compression. For the compression method, we implement a new lightweight down/upsampling network for performance-limited IIoT devices, which is inserted at the slicing point of a DNN model in order to decrease the outbound network traffic without a significant accuracy drop. We also develop a benchmarking tool to accurately find the optimal slicing point of the DNN for the best inference latency. ScissionLite improves the inference latency by up to 15.7x with a minimum accuracy degradation.**

***Index Terms*— Edge computing; IIoT; deep neural networks; model slicing; inference**

I. Introduction

EDGE computing is gaining a growing interest in Industrial Internet of Things (IIoT) applications [1]. Recent advances in edge computing for IIoT are best exemplified by deep neural network (DNN) model slicing for product surface inspection [2]. This research employs a surface inspection camera on an automated assembly line and captures the surface of products one by one for DNN inference. However, as IIoT devices have limited computational capability, the inspection time can be continuously delayed, resulting in a significant decline of outcomes. To address this issue, the entire DNN model can be split to assign the initial layers onto the IIoT device and the later layers to an edge server. Leveraging both device and edge resources for DNN inference has the benefit of reducing inference time by leveraging the computation power of the edge [3].

The benefit of multi-access edge computing (MEC) for IIoT is clear - enabling cloud computing at the edge of the cellular network [4]. However, the overhead of data transfer from the local to the edge is problematic for IIoT applications. 5G cellular networks provide high download bandwidth ranging from 600 to 1,700 Mbps [5], whereas the upload speed is comparatively low (30 – 60 Mbps) owing to weak cellular radio power on local devices. Therefore, when DNN model slicing is applied in an MEC environment, transferring intermediate DNN data of an IIoT device to the multi-access edge can be a significant bottleneck.

To overcome this limitation, existing research presents methods to reduce the volume of data transferred between the local device and the edge by applying data encoding at the split point of the DNN [6]–[9]. However, they are limited in the following ways: (1) The adopted compression techniques are not integrated into the original DNN [6] and incur high computational costs [7]. These compression techniques are a further bottleneck on IIoT devices that are already performance-limited. (2) The encoding techniques are validated on lightweight DNN models with relatively few layers and small datasets [6]–[8], such as CIFAR[1]. These data encoding methods may result in a significant accuracy drop when used with large models and high resolution images typical for IIoT. (3) The existing methods do not find the optimal split point of the DNN, and therefore the overall performance improvement cannot be maximized. The methods find the split point arbitrarily or based on estimations rather than actual measurements [8], [9].

In this paper, we propose *ScissionLite*, a framework to accelerate edge-based distributed deep learning inference based on a down/upsampling (DU) neural network for IIoT. The primary design goal of ScissionLite is to develop a lightweight compression method for performance-limited IIoT devices that incurs a negligible accuracy drop. Our method employs a simple DU neural network seamlessly integrated into the target DNN. Therefore, it does not require specialized hardware owing to low computational requirements. In addition, this

Hyunho Ahn and Munkyu Lee contributed equally to this work. Corresponding author: Cheol-Ho Hong

Hyunho Ahn is with the School of Electrical and Electronics Engineering, Chung-Ang University, Seoul, Korea. e-mail: hanid842@cau.ac.kr

Munkyu Lee, Sihoon Seong, and Cheol-Ho Hong are with the Department of Intelligent Semiconductor Engineering, Chung-Ang University, Seoul, Korea. e-mail: dse112@cau.ac.kr; seongsihoon7@cau.ac.kr; cheolhohong@cau.ac.kr

Gap-Joo Na and In-Geol Chun are with Electronics and Telecommunications Research Institute, Daejeon, Korea. e-mail: funkygap@etri.re.kr; igchun@etri.re.kr

Blesson Varghese is with the School of Computer Science, University of St Andrews, United Kingdom. e-mail: bv6@st-andrews.ac.uk

[1] https://www.cs.toronto.edu/~kriz/cifar.html

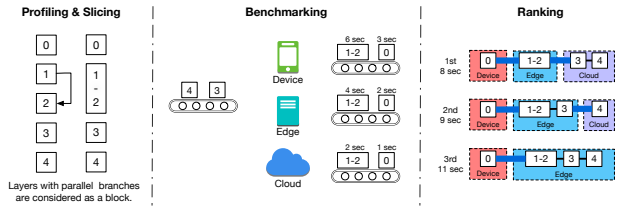


Fig. 1: Profiling, slicing, benchmarking, and ranking in Scission.

simple network does not lose a significant amount of information during data compression and inherently results in a small accuracy drop even for large DNN models. The second goal is to decide the best slicing point of the target DNN for achieving optimal performance. For this purpose, we develop an automated benchmarking tool based on previous research, Scission [3]. ScissionLite accurately finds the optimal slicing point based on empirical data whereas other studies employ an approximate solution with estimations [9].

We evaluate ScissionLite in an emulated MEC environment using large-scale production DNNs with two high-precision datasets. The datasets include a subset of ImageNet [10] called ILSVRC[2] proposed for large-scale visual recognition and the Xsteel Surface Defect Dataset (X-SDD) [11] for product quality control in steel industry. ScissionLite can increase the performance of deep learning inference by up to 15.7x and 3.29x compared to the local device execution and ScissionLite without data compression, respectively. In addition, ScissionLite only incurs 0.51 – 1.36% of an accuracy drop with ImageNet and 0 – 1.82% with X-SDD.

The rest of this paper is organized as follows: Section II presents the background and related work. Section III proposes ScissionLite. Section IV highlights the performance evaluation results. Section V presents discussion points. Finally, the paper is concluded in Section VI.

II. Background and Related Work

In this section, we present the background research on which ScissionLite's benchmarking tool is based and then present an overview of the related research.

A. Scission

In ScissionLite, the automated benchmarking tool is based on Scission[3] [3], a framework that determines the optimal slicing point for distributed deep learning inference. ScissionLite extends the capabilities of Scission by considering the time spent on data compression, data serialization, and communication between the device and the edge.

The following design choices drove the development of Scission: (1) DNN slicing must identify optimally performing slices that can be distributed across the device, edge, and cloud resources, (2) DNN slicing must be based on empirical data rather than estimations because layers are extremely performance sensitive and cannot be easily predicted with estimations, and (3) DNN slicing should account for user-defined constraints, such as the target overall latency.

Scission is underpinned by a methodology involving profiling, slicing, benchmarking, and ranking as shown in Figure 1 and described below:

Profiling: The DNN is profiled to find suitable slicing points. Each layer in the sequential DNN is considered a potential slicing point. However, for a DNN model in which there are parallel branches, the layers within the parallel branch are considered as a block and not sliced individually.

Slicing: The DNN is sliced into distinct sub-models consisting of individual layers or blocks for benchmarking.

Benchmarking: Each layer or block is benchmarked against hardware resources for obtaining the average execution time between the layers and blocks.

Ranking: The DNN slices are ranked based on user-defined constraints, and a suitable DNN slice configuration can be chosen for deployment across the edge-cloud environment.

While Scission supports the device-edge-cloud continuum, this article primarily focuses on the device and edge tiers. As modern edge servers provide powerful computational resources such as GPUs, the research presented in this article leverages these resources at the edge without transmitting data to the cloud, resulting in improved inference performance.

B. Related Work

Slicing and distributing DNNs across a combination of the device, edge, or cloud resources have performance benefits [12]. Slicing approaches rely on identifying a sequence of layers and mapping them onto resources so that the distributed DNN is optimized against the overall latency, ingress bandwidth, or a combination of these.

Existing research has proposed methods to reduce the inter-layer traffic in distributed deep learning inference. BottleNet [6] compresses the output layer of a mobile device by using conventional compressors such as JPEG before sending the data to the cloud. The research was evaluated with ResNet50 and VGG19 in 3G, 4G, and WiFi environments that offer 1.1, 5.85, and 18.88 Mbps for the upload bandwidth, respectively. Data compressors based on an autoencoder are presented [7], [8]. An autoencoder is an artificial neural network that duplicates its inputs to its outputs through several convolutional layers and incurs additional computational costs compared to the original DNN. Lee et al. [13] proposed a reconfigurable neural architecture for real-time object detection and suggested an autoencoder to reduce network traffic. The study introduces a splittable architecture enabling model division at different extractor layers by consolidating diverse splittable models into a single-weight reconfigurable model. However, the study's autoencoder is somewhat heavy for execution on IIoT devices, and there are constraints on the split points where the reconfigurable neural architecture can divide.

Slicing requires the identification of the optimal slicing point, which in the literature is based on two different approaches.

The first is an estimation-based approach in which the slicing point is determined based on an estimation of the

[2] https://image-net.org/challenges/LSVRC/

[3] Scission is available at https://github.com/qub-blesson/Scission

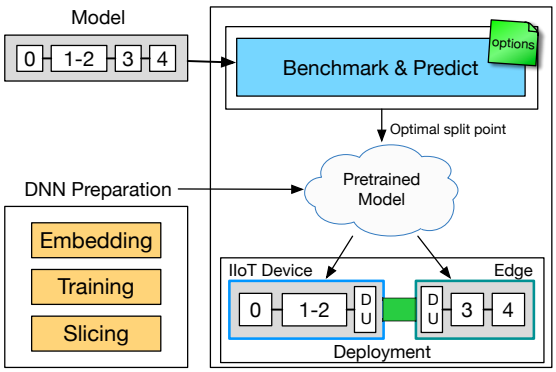


Fig. 2: Overall design of ScissionLite.

performance of the layers on a target hardware platform [9], [14]–[16]. This approach does not require running the model directly on hardware for profiling, offering the benefit of faster initial execution times. However, it is not trivial to identify the optimal slicing point using estimation-based approaches as DNN models become deeper and more complex. Gao et al. [17] present an estimation-based model designed to reduce time and energy expenses during task offloading. However, the cost estimation model relies solely on profiling data from AlexNet. This raises concerns as it may significantly differ for non-vision models or heavier models. Liao et al. [18] developed the Optimal Partition Point (OPP) algorithm, which uses an estimation model to predict the computational load of convolution and dense layers in each layer of a DNN to determine slice points. This research can identify split points where inference time and energy are optimized. However, if the DNN's layers are composed of elements other than just convolution and dense layers, the estimation may not be accurate.

The second is a benchmarking based approach for slicing DNNs. This measurement-based approach gathers empirical data on the performance of each layer from target hardware resources. Scission, an underlying approach of the benchmarking tool presented in this paper, adopts this approach [3]. The advantage of the benchmark-based approach is its ability to precisely predict the execution time for each layer, allowing for optimal determination of split points. However, its downside is the need for offline profiling for each model, which involves considerable profiling time before the model can be executed. For example, profiling about 10 models with ScissionLite takes about an hour.

Compared to existing research, ScissionLite offers a lightweight compression technique using a down/upsampling network and a method to obtain the optimal slicing point in response to the network traffic change. In addition, it is evaluated by large-scale DNN models such as DenseNet169 and ResNet101 with a large visual image database as well as a realistic industrial dataset in an MEC environment.

III. Design

In this section, we present the architecture of ScissionLite, which is a framework to accelerate edge-based distributed deep learning inference for IIoT.

A. Overall Design

ScissionLite inserts an additional data compression layer at the optimal split point of the DNN for decreasing the latency of inference as shown in Figure 2. For this purpose, ScissionLite develops two components including a new neural network layer for data compression and a Scission-based benchmarking tool. The new layer decreases the amount of data transfer between the device and the edge by adopting a down/upsampling (DU) neural network layer. The Scission-based tool automatically determines the optimal model slicing point when the DU layer is applied. After the optimal slicing point is identified, ScissionLite prepares a new DNN model that employs the DU layer with embedding, training, and slicing phases.

B. Down/Upsampling Neural Network Layer

The down/upsampling (DU) layer is a small neural network that is embedded at the split point of the DNN for reducing the amount of data exchanged during communication. The DU layer is composed of the *Down* and *Up* layers for the device and edge resources, respectively. The Down layer compresses the feature maps of the end layer of the sliced DNN on the IIoT device. The compressed data is then transferred through the cellular network connection. The Up layer expands the received data and passes the data to the starting layer of the remaining DNN on the edge. In this manner, the DU layer can decrease the amount of data transferred.

For a compression method, previous research utilizes an existing raw data encoding method such as a traditional image compression technique [19] (e.g., JPEG compression) or an autoencoder-based coding scheme [20]. These methods are a good candidate for realizing communication-efficient inference in edge-based machine learning systems. However, we identify that the former generally requires specialized hardware based on an application-specific integrated circuit (ASIC) to boost the compression and expansion process [21], and the latter demands significant training time for adapting the autoencoder and computation resources during inference due to the complexity of the autoencoder [20]. Since IIoT devices may have relatively performance-limited resources, a more efficient method is required for a compression method.

To address this issue, we develop a new lightweight compression method for the Down and Up layers. We focus on the utilization of down/upsampling neural networks in recent DNN models to reduce the number of parameters and computational load without significantly impacting accuracy [22]. By applying the same scheme to the DU layer, we expect that the data transfer amount will be decreased due to reduced parameters, and the DU layer will only cause a minimum accuracy drop. In addition, as down/upsampling neural networks have no weights that need to be trained, training time for adapting the DU layer itself would not be required. Partial re-training of the DNN that embeds the DU layer is needed for better accuracy, though. This will be explained in Section III-D.

Our implementations of the Down and Up layers are as follows:

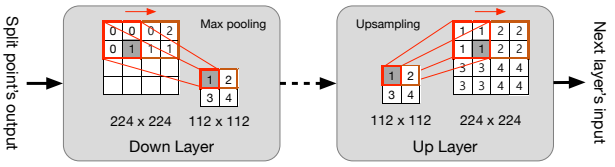


Fig. 3: Max pooling and upsampling of a neural network.

TABLE I: Latency and top-5 accuracy of the ResNet101 model at split point 49 with varying kernel sizes in max pooling.

Kernel size	Latency (s) (60 Mbps)	Latency (s) (30 Mbps)	Top-5 accuracy
Original	0.242	0.456	0.928
2x2	0.082	0.135	0.923
3x3	0.050	0.072	0.909
4x4	0.041	0.055	0.9

The Down layer compresses the output of the local IIoT device by adopting a downsampling network. The Down layer implements a max pooling layer [22] for downsampling, which sub-samples (or shrinks) each feature map of the last layer of the IIoT device. In max pooling, a 2×2 kernel moves back and forth across the feature map and only selects the maximum value in the kernel for an output neuron value as shown in Figure 3. Other neurons lower than the maximum value in the kernel are dropped. The Down layer thus obtains sub-sampled feature maps whose size is a quarter of the original layer. Max pooling drops 75% of the input values, but this information loss offers some level of position invariance [23], which is useful for image classification tasks.

A larger kernel size in max pooling can improve the compression ratio but may incur a significant accuracy drop owing to information loss. Table I demonstrates the network transmission latency and top-5 accuracy in the ResNet101 model with a split point at 49 in the layer, based on different pooling kernel sizes, at network bandwidths of 60Mbps and 30Mbps. As shown in the table, increasing the kernel size leads to a reduction in latency; however, it also results in a decrease in top-5 accuracy. In consideration of this tradeoff relationship, ScissoinLite opted to utilize a 2x2 pooling kernel size for each slice point to minimize accuracy drops.

The Up layer expands the received data from the IIoT device by an upsampling neural network. The Up layer implements a nearest-neighbor interpolation method layer [22] for upsampling. This interpolation method selects the value from the nearest pixel and copies the selected value into the dropped neurons, as shown in Figure 3. This algorithm is computationally efficient and suitable for decreasing the latency of inference. We have also considered other interpolation methods including bilinear and bicubic interpolation, but they resulted in greater accuracy degradation than the nearest-neighbor interpolation.

C. Benchmarking Tool

As described in Section II-A, Scission automatically benchmarks DNN models on a target set of the device, edge, and cloud resources for determining the optimal slicing point. In this study, we develop a new benchmarking tool[4] based on Scission for automated benchmarking of DNNs that embed a data compression layer including the DU layer. Our benchmarking tool adopts a general approach and can be also applied to DNNs employing other data encoding methods.

We first present a comparative analysis on Scission and our benchmarking tool and later show how our tool finds the optimal slicing point. The target DNN model consisting of n layers is denoted by $L = \{L_1, L_2, \cdots, L_n\}$. Then, t_i^{Device} represents the latency of executing layer L_i of the DNN in the local device, and t_i^{Edge} denotes the latency in the edge server.

The DU layer in ScissionLite is an additional neural network that demands computation resources such as CPUs and GPUs during inference. The execution time for processing all layers of the distributed DNN including the DU layer in ScissionLite is denoted by E_{SL}. The DU layer is composed of the Down and Up layers. When the DNN slicing point is layer L_j, the latency values of executing the Down and Up layers are indicated by t_j^{Down} and t_j^{Up}, respectively. Then, E_{SL} is calculated as follows:

$$E_{SL} = \sum_{k=1}^{j} t_k^{Device} + t_j^{Down} + t_j^{Up} + \sum_{k=j+1}^{n} t_k^{Edge} \quad (1)$$

As explained in Section III-B, we apply a lightweight max pooling and upsampling mechanism for the DU layer. Then, the computation cost of the DU layer, which is $t_j^{Down} + t_j^{Up}$, is expected to be low. However, the computation burden would be high when a compute-intensive data encoding method such as an autoencoder is adopted.

The original execution time for processing all layers of the distributed DNN in Scission is denoted by E_{Orig}. When the DNN slicing point is layer L_l in Scission, we obtain E_{Orig} as follows:

$$E_{Orig} = \sum_{k=1}^{l} t_k^{Device} + \sum_{k=l+1}^{n} t_k^{Edge} \quad (2)$$

Before network transmission, data or neurons compressed by the Down layer should be serialized to an appropriate data format such as Protocol Buffers (Protobuf) or JavaScript Object Notation (JSON). At the edge, the received data will be deserialized. This also requires CPU resources. The execution time regarding (de)serialization in ScissionLite is indicated by S_{SL}. Then, $t_j^{SerialDU}$ and $t_j^{DeserialDU}$ represent the latency values of serializing the data, which is compressed by the Down layer at split point j, on the device and deserializing the data on the edge, respectively. S_{SL} is then calculated as follows:

$$S_{SL} = t_j^{SerialDU} + t_j^{DeserialDU} \quad (3)$$

Original Scission's (de)serialization time, S_{Orig}, is calculated in the same way. t_l^{Serial} and $t_l^{Deserial}$ denote the latency values of serializing the output data at layer l on the device and deserializing the data on the edge, respectively. We then

[4]The benchmarking tool will be made available as open source in an accepted version, but is not presented in this version due to double-blind review.

obtain S_{Orig} as follows:

$$S_{Orig} = t_l^{Serial} + t_l^{Deserial} \quad (4)$$

The communication time between the device and the edge in ScissionLite is denoted by C_{SL}, when the DU layer is applied. $s_j^{SerialDU}$ denotes the size of the serialized data from the output of the Down layer at split point j. C_{SL} is then calculated as follows:

$$C_{SL} = Latency + \frac{s_j^{SerialDU}}{Bandwidth} \quad (5)$$

where the network latency is $Latency$, and the network bandwidth is $Bandwidth$.

The communication time in Scission, C_{Orig}, is then calculated in the same way. When s_l^{Serial} denotes the size of the serialized data at layer L_l, C_{Orig} is then obtained as follows:

$$C_{Orig} = Latency + \frac{s_l^{Serial}}{Bandwidth} \quad (6)$$

Both communication times are affected by inherent network latency and the data transfer time, which can be obtained by dividing the size of the data by the network bandwidth. In the 5G cellular network, the upload bandwidth is as low as between 30 and 60 Mbps. Therefore, to decrease communication time, it is essential to reduce the amount of data transfer.

Consequently, Δt, which is the benefit of adopting the DU layer, is calculated as follows:

$$\Delta t = (E_{Orig} + S_{Orig} + C_{Orig}) - (E_{SL} + S_{SL} + C_{SL}) \quad (7)$$

The performance improvement that will be achieved is shown in Section IV.

Our benchmarking tool executes all possible split points individually for deciding the optimal slicing point as with Scission, which performs the profiling, slicing, benchmarking, and ranking phases as presented in Figure 1. The profiling and slicing phases of our benchmarking tool are identical to those of Scission. In the benchmarking phase, our tool benchmarks each layer, the Down layer, the Up layer, and the (de)serialization function, and obtains the values of E_{SL}, S_{SL}, and C_{SL} for each slicing point. In the ranking phase, our tool grades each DNN slice based on the sum of E_{SL}, S_{SL}, and C_{SL} and finds the optimal slicing point.

For IIoT, our benchmarking tool takes a privacy enhancement mechanism into account. The user can indicate that the slicing point needs to be above a certain layer so that the original industrial image cannot be easily inferred at the edge side. A recent study proposing a privacy-preserving medical platform reports that when a medical image passes through a small number of DNN layers, it is difficult to distinguish a specific data item in the processed image [24], because the convolutional and max pooling operations apply non-linear and non-reversible effect to the image. As data privacy in IIoT is also crucial, our benchmarking tool supports this privacy enhancement, and its performance implication will be presented in Section IV-B.

D. DNN Preparation

After the benchmarking tool identifies the optimal slicing point, ScissionLite prepares a new DNN model that employs the DU layer with the following three steps: embedding, training, and slicing.

Embedding: This step injects the DU layer in the split point of the pre-trained DNN model (e.g., production DNNs from Keras Applications[5]). ScissionLite divides the target model into two and inserts the DU layer between them. Afterward, it joins the sub-models for re-training, which is explained in the next sub-section.

Training: We use a pre-trained DNN model for generating the new DNN, but the pre-trained weights are not aware of the DU layer, which can lead to a significant accuracy drop. Existing neurons then need to learn how to cooperate with new neurons in the DU layer. This step retrains the new DNN for reducing the drop in accuracy. As the DU layer is lightweight, existing layers can be adapted to the DU layer fast during re-training. The top-k accuracy and re-training time will be presented in Section IV-C.

Slicing: Slicing divides the newly trained model into two for the IIoT device and the edge. For the device-side model, this step piles up the layers until the Down layer and exports them as a single Keras model. Similarly, the edge-side model starts from the Up layer and ends at the last fully-connected layer.

E. Deployment

In the IIoT device, we use the TensorFlow runtime to execute the sliced DNN. On the edge side, we utilize the NVIDIA Triton inference server[6], which is open-source serving software and optimized for edge deployment, for executing the remaining DNN.

For communication between the IIoT device and the edge, both REST and gRPC protocols can be utilized as a communication method. We adopted gRPC because it is based on HTTP/2 and is faster than REST with HTTP/1.1. gRPC uses Protocol Buffers (Protobuf) as the message interchange format. ScissionLite implements the Down and Up converters in order to serialize the output of the Down layer to Protobuf and deserialize the Protobuf to the input data of the Up layer, respectively.

IV. Evaluation

We present the performance evaluation results of ScissionLite in this section.

A. Experimental Environment

We implemented ScissionLite with two IIoT devices and an edge server for distributed inference. For the first IIoT device, we utilized the NVIDIA Jetson TX2 that features an NVIDIA Pascal-family GPU with 8 GB of memory. As for the second IIoT device, we employed the NVIDIA Jetson AGX Xavier

[5]https://keras.io/api/applications/

[6]https://developer.nvidia.com/nvidia-triton-inference-server

TABLE II: Test bed configurations.

Configuration name	Local device	Edge
LocalTX2_CPU	NVIDIA Jetson TX2 CPU	N/A
LocalTX2_GPU	NVIDIA Jetson TX2 GPU	N/A
EdgeTX2_GPU	NVIDIA Jetson TX2 GPU	NVIDIA RTX 3090 GPU
EdgeAGX_CPU	NVIDIA Jetson AGX CPU	NVIDIA RTX 3090 GPU
EdgeAGX_GPU	NVIDIA Jetson AGX GPU	NVIDIA RTX 3090 GPU

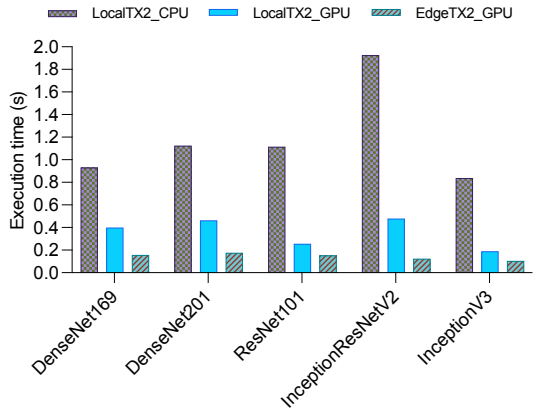


Fig. 4: Best performance of various DNN models obtained from the LocalTX2_CPU, LocalTX2_GPU, and EdgeTX2_GPU configurations.

that is equipped with an NVIDIA Volta-family GPU with 32 GB of memory. The edge server is an Intel Xeon E5-2698 CPU platform with twenty 2.2 GHz cores and an NVIDIA GeForce RTX 3090 GPU having 24 GB of memory. Each IIoT device is directly connected to the edge server via 1 GbE link. We adjusted the upload bandwidth of the network connection as 60 Mbps for 5G environments and 30 Mbps for 4G environments. We set the latency as 28 ms by exploiting the Linux Traffic Control tool. These values emulate the commercial 5G and 4G network environment based on the latest research [5].

We employed the TensorFlow runtime version 2.3 in the IIoT devices and NVIDIA Triton as an inference server in the edge. We obtained deep learning models with pre-trained weights from Keras Applications[7]. We used 10 pre-trained production DNNs for evaluation, namely DenseNet169, DenseNet201, ResNet101, InceptionResNetV2, InceptionV3, VGG16, VGG19, MobileNetV2, RegNetX004, and EfficientNetB0v2. We also employed a subset of ImageNet [10] called ILSVRC and the Xsteel Surface Defect Dataset (X-SDD) [11] as image databases. They are utilized for checking the inference accuracy of ScissionLite with a large high-quality image database and a realistic industrial dataset, respectively. The test bed configurations reflecting an operational IIoT environment are presented in Table II. We combined possible hardware resource types in the IIoT devices and the edge server.

B. Inference Latency

ScissionLite accelerates distributed deep learning inference by inserting the DU layer at the split point. This section highlights the performance of ScissionLite in terms of inference latency.

[7]https://keras.io/api/applications/

Feasibility of distributed inference: Figure 4 shows the best inference latency of the five DNN models executed in the LocalTX2_CPU, LocalTX2_GPU, and EdgeTX2_GPU configurations in Table II. The first two configurations do not employ DNN model slicing so that the DNN models were run in the local IIoT device using its embedded CPU and GPU. The last configuration utilizes DNN model slicing with ScissionLite. As shown in the figure, ScissionLite improves the performance of the DNN models up to 15.7 and 3.9 times compared to the LocalTX2_CPU and LocalTX2_GPU configurations, respectively. This result shows that DNN model slicing is effective in edge-based IIoT.

Slice-by-slice analysis: In this experiment, we present a slice-by-slice analysis of VGG16, VGG19, DenseNet169, and InceptionResNetV2 on the Jetson TX2 GPU (EdgeTX2_GPU). The analysis was conducted with a 60 Mbps connection as depicted in Figure 5, and with a 30 Mbps connection as illustrated in Figure 6. Additionally, we provide a slice-by-slice analysis of MobileNetV2, VGG16, RegNetX004, and EfficientNetV2B0 on the Jetson AGX CPU (EdgeAGX_CPU) for MobileNetV2 and VGG16, and on the GPU (EdgeAGX_GPU) for RegNetX004 and EfficientNetV2B0. This analysis was performed with a 60 Mbps connection, as shown in Figure 7, and with a 30 Mbps connection, as shown in Figure 8. In the figures, we denote ScissionLite without compression as baseline 1, which is identical to Scission. We also introduce another baseline, denoted as baseline 2, which naively sets the split point to Layer 1 and offloads the entire model to the edge. This is done to demonstrate the importance of selecting the optimal slice point for reducing inference latency. Each sub-figure presents the total end-to-end latency when the split point or layer changes from the first to the last layer; less important split points are not shown in the figure. In the case of ScissionLite, the profiling result of each split point is shown as a stacked column chart. The optimal split point decided by ScissionLite is denoted by a blue circle. In addition, we exclude Layer 1 as a valid split point if it performs zero padding. This is because the feature maps after zero padding are almost identical to the input image, and transmitting them to the edge would compromise privacy, particularly in IIoT scenarios where commercially sensitive input data is involved.

As shown in Figure 5 to 8, ScissionLite generally outperforms baseline 1 (Scission) due to its implementation of the DU layer based on down/upsampling at the split point. Compared to baseline 1, ScissionLite achieves a speedup of up to 3.48 in InceptionResNetV2 on EdgeTX2_GPU with a 60 Mbps connection. Please note that the optimal split point of VGG19 in both baseline 1 and ScissionLite is the same, and this layer does not involve significant data transfer. As a result, speedups are not achieved for VGG19. In addition, ScissionLite outperforms baseline 2, which naively configures the split point as Layer 1. The speedup values achieved by ScissionLite compared to baseline 1 and 2 are summarized in Table III.

Furthermore, we show a comparison between the estimation-based methods, Neurosurgeon [14], and the Optimal Partition Point (OPP) algorithm [18], along with ScissionLite, in Figure9. For an accurate comparison,

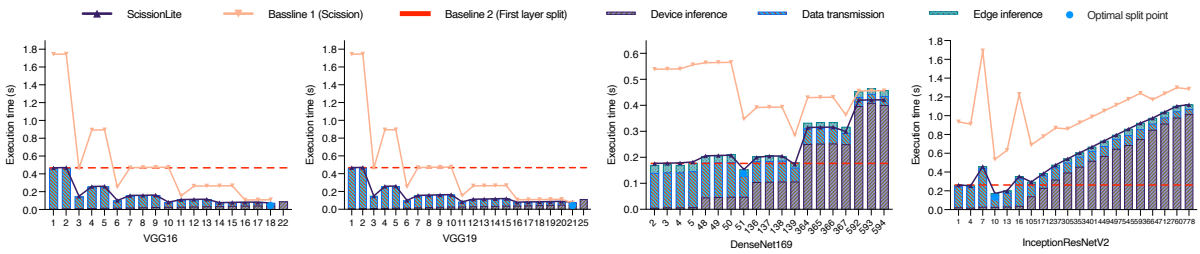


Fig. 5: Slice-by-slice analysis of four models using the Jetson TX2 GPU, with a 60 Mbps connection.

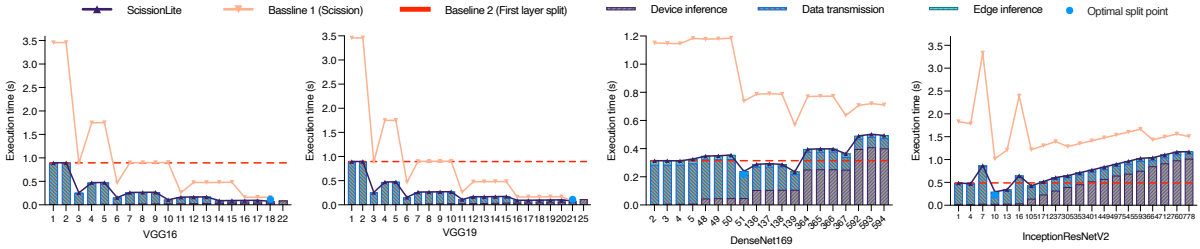


Fig. 6: Slice-by-slice analysis of four models using the Jetson TX2 GPU, with a 30 Mbps connection.

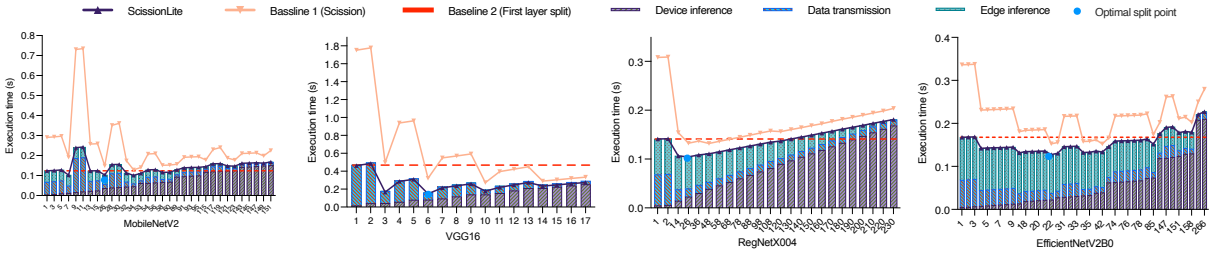


Fig. 7: Slice-by-slice analysis of four models using the Jetson AGX CPU for MobileNetV2 and VGG16, and GPU for RegNetX004 and EfficientNetV2B0, with a 60 Mbps connection.

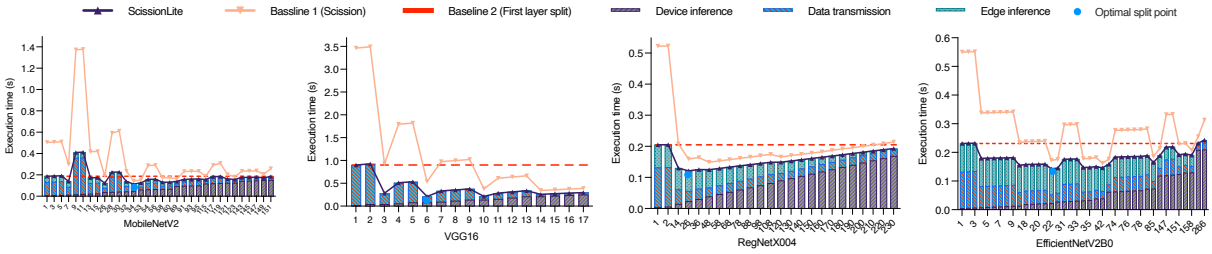


Fig. 8: Slice-by-slice analysis of four models using the Jetson AGX CPU for MobileNetV2 and VGG16, and GPU for RegNetX004 and EfficientNetV2B0, with a 30 Mbps connection.

ScissionLite's DU layer compression technique was applied to Neurosurgeon and OPP. This figure demonstrates the difference between the actual inference execution time on real hardware and the benchmarked or estimated time for the optimal split points chosen differently in VGG16, VGG19, and DenseNet169 models. OPP showed discrepancies between actual execution time and estimated time of 67%, 62.4%, and 98.9% in each model. OPP exhibits severe discrepancies with deeper models, as the calculation of the computation amount for execution time prediction does not consider the execution times of layers other than convolution and dense layers. Neurosurgeon showed discrepancies between actual execution time and estimated time of 33.2%, 33.4%, and 13.2% in each model. Neurosurgeon designs a regression model that infers latency based on the size and computational load of the input tensor, thus accurately predicting the overall model latency. However, dealing with complex layers associated with numerous parameters and blocks of layers with skip connections is challenging. Compared to ScissionLite, each baseline tends to choose a later layer as the optimal split point. This difference in split point decision, due to inaccurate estimation values, means that ScissionLite can reduce latency by an average of 21.9% in the case of OPP, and by an average of 8.8% for Neurosurgeon.

As explained in Section III-C, data privacy can be enhanced if the slicing point is a later layer than Layer 1. To formally

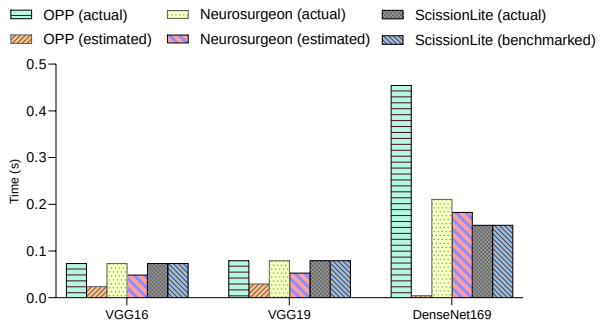


Fig. 9: Latency comparison at the optimal split points selected by benchmark and estimation-based baselines.

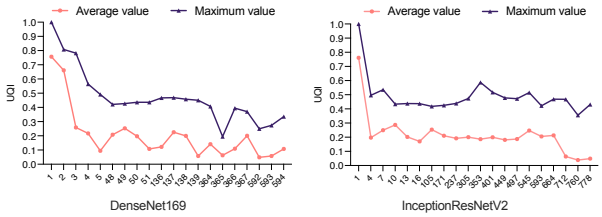
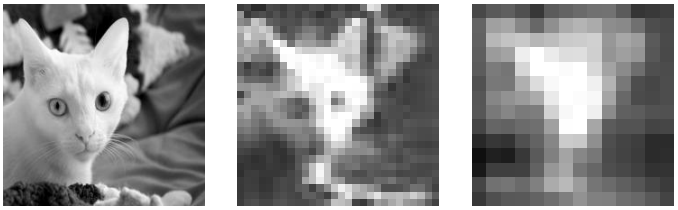


Fig. 10: Slice-by-slice average and maximum universal image quality index (UQI) values of DenseNet169 and InceptionResNetV2.



(a) UQI: 1.0 (b) UQI: 0.48 (c) UQI: 0.24

Fig. 11: Changes in image distortion according to UQI values.

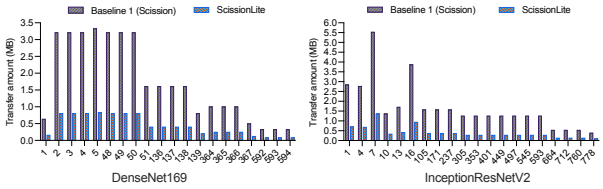


Fig. 12: Volume of data transferred in DenseNet169 and InceptionResNetV2 at each split point in baseline 1 (Scission) and ScissionLite.

define the privacy constraint, we employ the universal image quality index (UQI) metric [25], which is widely used in image processing studies. UQI measures any image distortion using three factors including loss of correlation, luminance distortion, and contrast distortion. When the UQI value is 1, the original and target images are the same. When the two images become more dissimilar, the UQI value converges to 0. We measure the average and maximum UQI values of the feature maps at each split point in DenseNet169 and InceptionResNetV2 and provide this information in Figure 10. In ScissionLite, a user can demand a split point that has

TABLE III: Speedup achieved by ScissionLite compared to baseline 1 (Scission) and baseline 2 (first layer split).

Model	Configuration	Bandwidth (Mbps)	Speedup (baseline 1)	Speedup (baseline 2)
VGG16	EdgeTX2_GPU	60	1.50	6.37
VGG19	EdgeTX2_GPU	60	1.02	5.89
DenseNet169	EdgeTX2_GPU	60	2.61	1.13
InceptionResNetV2	EdgeTX2_GPU	60	3.11	1.21
VGG16	EdgeTX2_GPU	30	2.15	11.81
VGG19	EdgeTX2_GPU	30	1.05	10.94
DenseNet169	EdgeTX2_GPU	30	3.09	1.32
InceptionResNetV2	EdgeTX2_GPU	30	3.48	1.38
MobileNetV2	EdgeAGX_CPU	60	1.27	1.33
VGG16	EdgeAGX_CPU	60	2.16	3.16
RegNetX004	EdgeAGX_GPU	60	1.27	1.34
EfficientNetV2B0	EdgeAGX_GPU	60	1.20	1.31
MobileNetV2	EdgeAGX_CPU	30	1.24	1.66
VGG16	EdgeAGX_CPU	30	2.53	4.29
RegNetX004	EdgeAGX_GPU	30	1.31	1.68
EfficientNetV2B0	EdgeAGX_GPU	30	1.21	1.63

average and maximum UQI values under a certain threshold. In enforcing privacy constraints on ScissionLite, it should be done according to the following procedure. The first step is to create a candidate list of split points in order of low inference latency. The next step involves traversing the candidate list in order of low inference latency and selecting the optimal split point where both the average and maximum UQI values are below a certain threshold.

We configure the average Universal Quality Index (UQI) value to be below 0.25 and the maximum value to be below 0.5. As shown in Figure 11, when the UQI is around 0.5, the image is distorted to an extent that the original image cannot be accurately inferred. Furthermore, when the UQI is about 0.25, the distortion is so severe that it becomes impossible to guess the shape of the original image. Therefore, when a layer contains multiple feature maps, setting the average and maximum UQI to 0.25 and 0.5 ensures that, on average, the images are distorted to the point where the shape of the original image cannot be guessed, and even the image with the highest UQI cannot accurately determine the original image. Then, considering the privacy constraint, the optimal slicing points are determined to be 51 for DenseNet169 and 13 for InceptionResNetV2 at 30 Mbps and 60 Mbps on the Jetson TX2 GPU.

C. Inference Accuracy

In order to prevent an accuracy drop, ScissionLite retrains the split model embedding the DU layer, as explained in Section III-D. ScissionLite employs the stochastic gradient descent (SGD) optimizer with 0.001 of a learning rate for re-training. The existing neurons in the pre-trained model are adapted to the new neurons in the DU layer during re-training and converge to minimum loss rapidly. This takes between 5 and 10 hours for ImageNet and less than one hour for X-SDD with a single NVIDIA GeForce RTX 3090 GPU. The training speed can be boosted by using multiple GPUs in a cloud data center.

Accuracy with ImageNet: The subset of ImageNet called ILSVRC has a very large image database with more than 1.2 million high-quality images and one thousand categories. Such a huge dataset is suitable for training and evaluating advanced

TABLE IV: Top-1 and top-5 accuracy of the original Keras model and the retrained DNN at the top 5 split points suggested by ScissionLite with the ImageNet dataset.

Model	Original Top-1 accuracy	Original Top-5 accuracy	ScissionLite (1st)			ScissionLite (2nd)			ScissionLite (3rd)			ScissionLite (4th)			ScissionLite (5th)		
			Split point	Top-1 accuracy (drop)	Top-5 accuracy (drop)	Split point	Top-1 accuracy (drop)	Top-5 accuracy (drop)	Split point	Top-1 accuracy (drop)	Top-5 accuracy (drop)	Split point	Top-1 accuracy (drop)	Top-5 accuracy (drop)	Split point	Top-1 accuracy (drop)	Top-5 accuracy (drop)
DenseNet169	0.762	0.932	51	0.741 (2.08)	0.923 (0.90)	2	0.739 (2.32)	0.922 (0.97)	3	0.740 (2.19)	0.922 (1.00)	4	0.739 (2.31)	0.921 (1.14)	5	0.739 (2.32)	0.921 (1.11)
DenseNet201	0.773	0.936	51	0.752 (2.15)	0.928 (0.79)	2	0.747 (2.60)	0.927 (0.89)	3	0.749 (2.39)	0.926 (1.01)	4	0.747 (2.61)	0.926 (1.05)	5	0.747 (2.59)	0.926 (0.99)
ResNet101	0.764	0.928	49	0.735 (2.92)	0.923 (0.51)	91	0.730 (3.42)	0.919 (0.90)	59	0.740 (2.40)	0.921 (0.73)	101	0.728 (3.60)	0.916 (1.18)	69	0.743 (2.07)	0.924 (0.45)
InceptionResNetV2	0.803	0.953	10	0.775 (2.83)	0.939 (1.36)	11	0.782 (2.14)	0.942 (1.13)	12	0.778 (2.55)	0.941 (1.13)	13	0.777 (2.56)	0.942 (1.06)	61	0.776 (2.74)	0.941 (1.16)
InceptionV3	0.779	0.937	10	0.753 (2.60)	0.928 (0.89)	11	0.755 (2.42)	0.930 (0.73)	12	0.755 (2.42)	0.930 (0.71)	13	0.755 (2.43)	0.930 (0.69)	1	0.754 (2.55)	0.928 (0.88)

TABLE V: Accuracy of the original model and the retrained DNN at the top 5 split points suggested by ScissionLite with the X-SDD dataset.

Model	Original accuracy	ScissionLite (1st)		ScissionLite (2nd)		ScissionLite (3rd)		ScissionLite (4th)		ScissionLite (5th)	
		Split point	Accuracy (drop)	Split point	Accuracy (drop)	Split point	Accuracy (drop)	Split point	Accuracy (drop)	Split point	Accuracy (drop)
DenseNet169	0.971	51	0.960 (1.09)	2	0.971 (0.00)	3	0.967 (0.36)	4	0.953 (1.82)	5	0.971 (0.00)
DenseNet201	0.975	51	0.960 (1.45)	2	0.975 (0.00)	3	0.975 (0.00)	4	0.975 (0.00)	5	0.975 (0.00)
ResNet101	0.971	49	0.971 (0.00)	91	0.967 (0.36)	59	0.971 (0.00)	101	0.964 (0.73)	69	0.960 (1.09)
InceptionResNetV2	0.986	10	0.967 (1.82)	11	0.971 (1.46)	12	0.967 (1.82)	13	0.975 (1.10)	61	0.971 (1.46)
InceptionV3	0.982	10	0.971 (1.09)	11	0.967 (1.45)	12	0.982 (0.00)	13	0.960 (2.18)	1	0.982 (0.00)

large-scale DNN algorithms used in this study. Table IV compares the top-1 and top-5 accuracy for the top 5 split points suggested by ScissionLite when the ImageNet dataset is employed. In Table IV, different split points result in a uniform and stable accuracy pattern in ImageNet, showing little preference regarding the position of the DU layer. The result shows that the retrained DNN provides an acceptable inference accuracy with 0.51 – 1.36% of a top-5 accuracy drop at the optimal split point (i.e., 1st). This highlights that the DU layer does not deteriorate the overall accuracy significantly, as discussed in Section III-B. For the top 5 split points, there was an accuracy drop of 2.0% to 3.6% in top-1 accuracy, which is not significant.

Accuracy with X-SDD: X-SDD is a surface defect image database consisting of 1,360 images of hot-rolled steel strip defects in seven types of categories. This dataset is employed in order to identify whether ScissionLite incurs an accuracy drop with a realistic industrial dataset. Table V compares the top-1 accuracy of the original DNN model and the retrained DNN in ScissionLite when the X-SDD dataset is employed. In Table V, the X-SDD dataset shows a wider range of an accuracy pattern than that of ImageNet. As the number of images in X-SDD is small (i.e., 1,360 images), it seems that each DNN model with X-SDD overfits the training data during re-training. However, the result shows that the retrained DNN provides an acceptable inference accuracy with 0 – 1.82% of a drop at the optimal split point (i.e., 1st). This result shows that ScissionLite is feasible for an industrial environment in which the accuracy of a DNN model is crucial.

D. Comparison with Other Compression Methods

In this section, we compare the JPEG-based compression method and the autoencoder approach, identified as motivation examples for ScissionLite in Section III-B, based on compression time and accuracy. The JPEG-based compression employs TensorFlow's JPEG encode/decode API, while the autoencoder uses Bottlenet++ [8]. Bottlenet++ implements a simple and fast autoencoder, which consists of a convolution and a transposed convolution layers. For an accurate comparison, at the three split points—49, 91, and 59—selected by ScissionLite for the ResNet101 model, we compared each compression method at the same fourfold compression ratio as ScissionLite.

Compression time: Figure 13 shows the latency measured during the encoding and decoding processes of each compression method, performed respectively on the device and at the edge. According to the measured results, at all split points, the compression time of the DU layer used in ScissionLite is the shortest. In the case of Bottlenet++, the autoencoder structure demands more computation compared to the max pooling and upsampling structure of the DU layer, resulting in longer encoding latencies on the device of 5.16ms, 6.17ms, and 5.37ms. Moreover, the JPEG compression technique requires more computation than the lightweight DU layer, leading to significantly longer times of 22.33s, 44.83s, and 22.75s. The encoding and decoding of JPEG, which must be performed separately for each channel of the feature map, can greatly increase the latency when repeatedly performed on a device with limited computational capacity.

Accuracy with Compression: Figure 13 shows the results of measuring the top-1 and top-5 accuracy drops for the entire ResNet101 model when each compression method was applied at three split points. For Bottlenet++, the accuracy drop was higher by 0.55%, 0.6%, and 0.06% for top-1 and 1.08%, 1.1%, and 0.22% for top-5 compared to ScissionLite. This higher accuracy loss could be due to the use of the ImageNet dataset instead of the CIFAR-100 dataset previously used by Bottlenet++. Furthermore, Bottlenet++ requires a separate training process for the autoencoder and a fine-tuning process for the entire model, which is considerably time-consuming [8]. Notably, training ResNet101 on Bottlenet++ with an NVIDIA RTX 3090 GPU takes approximately 45 hours per split point. JPEG experiences a greater accuracy drop with top-1 accuracy drops of 3.93%, 1.49%, and 2.98% and top-5 accuracy drops of 3.18%, 1.52%, and 2.02% higher than those of ScissionLite.

V. Discussion

This section discusses considerations for deploying ScissionLite in a real IIoT environment.

Re-training of the DNN for the new slicing point: When the device or network environment changes, leading to online alterations in the slice point, ScissionLite has the drawback of requiring re-training for that specific slice point. To address this concern, we conducted an analysis of slice point rankings based on inference latency for each device and network environment. Through this assessment, we noticed instances where

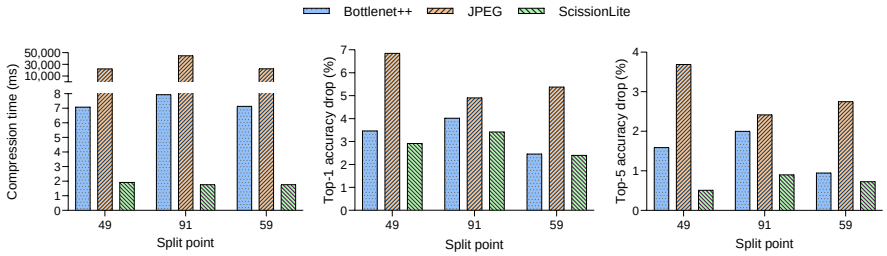


Fig. 13: Compression latency and accuracy drop for the top-3 split points in ResNet101.

a slice point ranked second or third in a specific environment becomes the top-ranked slice point in a different environment. This implies that one of the high-ranking split points in one environment has a probability of becoming an optimal split point in a different environment. Based on this observation, we intend to retain a few retrained models for the predominant split points consistently appearing in the top rankings across various environmental configurations, thereby mitigating the re-training overhead.

Potential constraints: Additionally, there are potential constraints to consider for deploying ScissionLite in a real IIoT environment. Firstly, when applying ScissionLite to models with lengthy skip connections like the YOLO model, the options for split point candidates are limited. Studies on optimal split points, including ScissionLite, generally identify only layers without skip connections to reduce data transmission. YOLO has long skip connections that traverse many layers, and the sections without skip connections are concentrated in the earlier layers, thus reducing the number of potential candidates for split points. Secondly, ScissionLite currently lacks consideration for energy efficiency. It is focused solely on reducing inference latency; therefore, consideration of energy consumption on local devices should be added in future developments.

VI. Conclusions

This paper presents ScissionLite, a framework to accelerate edge-based distributed deep learning inference. ScissionLite reduces the amount of data transferred between the DNN slices across the device and the edge, but at the same time finds the optimal DNN slicing point. A novel method that uses a lightweight compression method using a down/up-sampling network and an automated benchmarking tool for performance-limited IIoT devices are developed. ScissionLite demonstrates the feasibility of the edge-based distributed deep learning approach for accelerating DNN inference for IIoT applications.

References

[1] E. Sisinni, A. Saifullah, S. Han, U. Jennehag, and M. Gidlund, "Industrial internet of things: Challenges, opportunities, and directions," *IEEE transactions on industrial informatics*, vol. 14, no. 11, pp. 4724–4734, 2018.

[2] L. Zeng, E. Li, Z. Zhou, and X. Chen, "Boomerang: On-demand cooperative deep neural network inference for edge intelligence on the industrial internet of things," *IEEE Network*, vol. 33, no. 5, pp. 96–103, 2019.

[3] L. Lockhart, P. Harvey, P. Imai, P. Willis, and B. Varghese, "Scission: Performance-driven and context-aware cloud-edge distribution of deep neural networks," in *Proceedings of the 13th IEEE/ACM 13th International Conference on Utility and Cloud Computing*, 2020, pp. 257–268.

[4] T. Qiu, J. Chi, X. Zhou, Z. Ning, M. Atiquzzaman, and D. O. Wu, "Edge computing in industrial internet of things: Architecture, advances and challenges," *IEEE Communications Surveys & Tutorials*, vol. 22, no. 4, pp. 2462–2488, 2020.

[5] A. Narayanan, E. Ramadan, J. Carpenter, Q. Liu, Y. Liu, F. Qian, and Z.-L. Zhang, "A first look at commercial 5g performance on smartphones," in *Proceedings of the Web Conference 2020*, 2020, pp. 894–905.

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

[7] D. Hu and B. Krishnamachari, "Fast and accurate streaming cnn inference via communication compression on the edge," in *Proceedings of the 5th IEEE/ACM International Conference on Internet-of-Things Design and Implementation*, 2020, pp. 157–163.

[8] J. Shao and J. Zhang, "Bottlenet++: An end-to-end approach for feature compression in device-edge co-inference systems," in *Proceedings of the IEEE International Conference on Communications Workshops*, 2020, pp. 1–6.

[9] S. Yao, J. Li, D. Liu, T. Wang, S. Liu, H. Shao, and T. Abdelzaher, "Deep compressive offloading: Speeding up neural network inference by trading edge computation for network latency," in *Proceedings of the 18th Conference on Embedded Networked Sensor Systems*, 2020, pp. 476–488.

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

[11] X. Feng, X. Gao, and L. Luo, "X-sdd: A new benchmark for hot rolled steel strip surface defects detection," *Symmetry*, vol. 13, no. 4, p. 706, 2021.

[12] Z. Zhou, X. Chen, E. Li, L. Zeng, K. Luo, and J. Zhang, "Edge Intelligence: Paving the Last Mile of Artificial Intelligence With Edge Computing," *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1738–1762, 2019.

[13] J. C. Lee, Y. Kim, S. Moon, and J. H. Ko, "A reconfigurable neural architecture for edge–cloud collaborative real-time object detection," *IEEE Internet of Things Journal*, vol. 9, no. 23, pp. 23 390–23 404, 2022.

[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," in *Proceedings of the 22nd International Conference on Architectural Support for Programming Languages and Operating Systems*, 2017, pp. 615–629.

[15] M. Xu, F. Qian, M. Zhu, F. Huang, S. Pushp, and X. Liu, "DeepWear: Adaptive Local Offloading for On-Wearable Deep Learning," *IEEE Transactions on Mobile Computing*, vol. 19, no. 2, pp. 314–330, 2020.

[16] R. Hadidi, J. Cao, M. Woodward, M. S. Ryoo, and H. Kim, "Musical chair: Efficient real-time recognition using collaborative iot devices," *arXiv preprint arXiv:1802.02138*, 2018.

[17] M. Gao, R. Shen, L. Shi, W. Qi, J. Li, and Y. Li, "Task partitioning and offloading in dnn-task enabled mobile edge computing networks," *IEEE Transactions on Mobile Computing*, 2021.

[18] Z. Liao, W. Hu, J. Huang, and J. Wang, "Joint multi-user dnn partitioning and task offloading in mobile edge computing," *Ad Hoc Networks*, vol. 144, p. 103156, 2023.

[19] Z. Liu, T. Liu, W. Wen, L. Jiang, J. Xu, Y. Wang, and G. Quan, "Deepn-jpeg: A deep neural network favorable jpeg-based image compression framework," in *Proceedings of the 55th annual design automation conference*, 2018, pp. 1–6.

[20] S. Yao, Y. Zhao, H. Shao, S. Liu, D. Liu, L. Su, and T. Abdelzaher, "Fastdeepiot: Towards understanding and optimizing neural network execution time on mobile and embedded devices," in *Proceedings of the 16th ACM Conference on Embedded Networked Sensor Systems*, 2018, pp. 278–291.

[21] J. H. Ko, T. Na, M. F. Amir, and S. Mukhopadhyay, "Edge-host partitioning of deep neural networks with feature space encoding for resource-constrained internet-of-things platforms," in *Proceedings of the 15th IEEE International Conference on Advanced Video and Signal Based Surveillance*, 2018, pp. 1–6.

[22] A. Giusti, D. C. Cireşan, J. Masci, L. M. Gambardella, and J. Schmidhuber, "Fast image scanning with deep max-pooling convolutional neural networks," in *Proceedings of the IEEE International Conference on Image Processing*, 2013, pp. 4034–4038.

[23] J. Nagi, F. Ducatelle, G. A. Di Caro, D. Cireşan, U. Meier, A. Giusti, F. Nagi, J. Schmidhuber, and L. M. Gambardella, "Max-pooling convolutional neural networks for vision-based hand gesture recognition," in *2011 IEEE international conference on signal and image processing applications (ICSIPA)*. IEEE, 2011, pp. 342–347.

[24] Y. J. Ha, M. Yoo, G. Lee, S. Jung, S. W. Choi, J. Kim, and S. Yoo, "Spatio-temporal split learning for privacy-preserving medical platforms: Case studies with covid-19 ct, x-ray, and cholesterol data," *IEEE Access*, vol. 9, pp. 121 046–121 059, 2021.

[25] Z. Wang and A. C. Bovik, "A universal image quality index," *IEEE signal processing letters*, vol. 9, no. 3, pp. 81–84, 2002.