DNN Surgery: Accelerating DNN Inference on the Edge Through Layer Partitioning

Huanghuang Liang, Qianlong Sang, Chuang Hu, Dazhao Cheng, *Senior Member, IEEE*, Xiaobo Zhou, *Senior Member, IEEE*, Dan Wang, *Senior Member, IEEE*, Wei Bao, *Senior Member, IEEE*, and Yu Wang, *Fellow, IEEE*

***Abstract*—Recent advances in deep neural networks have substantially improved the accuracy and speed of various intelligent applications. Nevertheless, one obstacle is that DNN inference imposes a heavy computation burden on end devices, but offloading inference tasks to the cloud causes a large volume of data transmission. Motivated by the fact that the data size of some intermediate DNN layers is significantly smaller than that of raw input data, we designed the DNN surgery, which allows partitioned DNN to be processed at both the edge and cloud while limiting the data transmission. The challenge is twofold: (1) Network dynamics substantially influence the performance of DNN partition, and (2) State-of-the-art DNNs are characterized by a directed acyclic graph rather than a chain, so that partition is incredibly complicated. To solve the issues, We design a Dynamic Adaptive DNN Surgery(DADS) scheme, which optimally partitions the DNN under different network conditions. We also study the partition problem under the cost-constrained system, where the resource of the cloud for inference is limited. Then, a real-world prototype based on the selif-driving car video dataset is implemented, showing that compared with current approaches, DNN surgery can improve latency up to 6.45 times and improve throughput up to 8.31 times. We further evaluate DNN surgery through two case studies where we use DNN surgery to support an indoor intrusion detection application and a campus traffic monitor application, and DNN surgery shows consistently high throughput and low latency.**

***Index Terms*—Computation offloading, deep neural networks, edge computing, inference acceleration, layer partitioning.**

I. Introduction

RECENT advancements in deep neural networks (DNN) have significantly enhanced the accuracy and speed of computer vision and video analytics, paving the way for a new generation of intelligent applications [1]. The maturation of cloud computing, equipped with powerful hardware such as TPU and GPU, has become a typical choice for computation-intensive DNN tasks [2]. In a self-driving car application, for instance, cameras continuously monitor and stream surrounding scenes to servers, which subsequently conduct video analytics and feedback control signals to the pedals and steering wheel [3]. In an augmented reality application, a smart glass continuously records its current view and streams the information to the cloud servers. In contrast, cloud servers perform object recognition and send back contextual augmentation labels to be seamlessly displayed, overlaying the actual scenery [4].

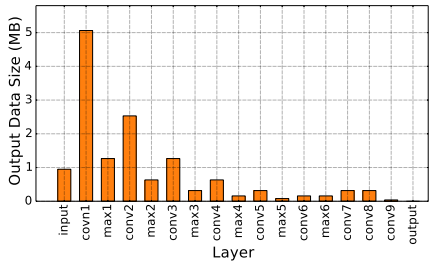


Fig. 1. The output data size of each layer of YOLOv2.

The enormous data volume of video streaming is an obstacle to developing intelligent applications. For example, Google's self-driving car can generate up to 750 megabytes of sensor data per second [5]. However, the average uplink rate of 4G, the fastest existing solution, is only 5.85Mbps [6]. Moreover, the data rate substantially decreases when the user is fast-moving, or the network is heavily loaded. In order to avoid the effect of the network and put the computing in the proximity of the data source, edge computing emerges [7]. As a network-free approach, it provides anywhere and anytime available computing resources. For example, the AWS DeepLens camera can run deep convolutional neural networks (CNNs) to analyze visual imageryte [8]. Nevertheless, edge computing is limited by its computing capacity and energy constraints, which cannot fully replace cloud computing.

For DNNs, the amount of some intermediate results (the output of intermediate layers) is significantly smaller than that of raw input data, as shown in Fig. 1. For example, the input data size of tiny YOLOv2 [9] is 0.95MB, while the output data size of intermediate layer max5 is 0.08MB with a reduction of 93%. This allows us to take advantage of the powerful

Manuscript received 3 December 2022; revised 7 February 2023; accepted 12 March 2023. Date of publication 20 March 2023; date of current version 6 September 2023. This work was supported by the Special Fund of Hubei Luojia Laboratory under Grant 220100016. Recommended for acceptance by D. Wu. *(Corresponding authors: Chuang Hu; Dazhao Cheng.)*

Huanghuang Liang, Qianlong Sang, Chuang Hu, and Dazhao Cheng are with the School of Computer Science, Wuhan University, Wuhan, Hubei 430072, China (e-mail: hhliang@whu.edu.cn; qlsang@whu.edu.cn; handc@whu.edu.cn; dcheng@whu.edu.cn).

Xiaobo Zhou is with the State Key Laboratory of Internet of Things for Smart City, Department of Computer and Information Sciences, University of Macau, Macau 999078, China (e-mail: waynexzhou@um.edu.mo).

Dan Wang is with the Department of Computing, The Hong Kong Polytechnic University, Hong Kong (e-mail: csdwang@comp.polyu.cn).

Wei Bao is with the School of Computer Science, The University of Sydney, Sydney, NSW 2006, Australia (e-mail: wei.bao@sydney.edu.cn).

Yu Wang is with the Department of Computer and Information Sciences, Temple University, Philadelphia, PA 19122 USA (e-mail: wangyu@temple.edu).

Digital Object Identifier 10.1109/TCC.2023.3258982

2168-7161 © 2023 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.
See https://www.ieee.org/publications/rights/index.html for more information.

Authorized licensed use limited to: KAUST. Downloaded on February 19,2024 at 07:16:23 UTC from IEEE Xplore. Restrictions apply.

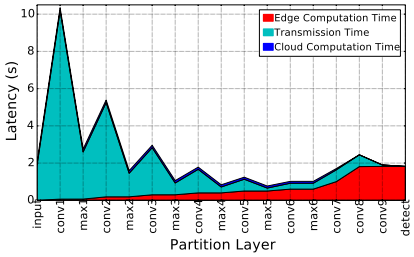


Fig. 2. Latency constitution when partition at the different layers of tiny YOLOv2.

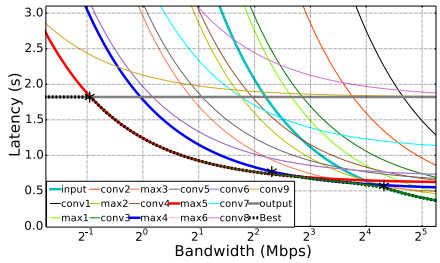


Fig. 3. The latency of partition at different layers of YOLOv2 as a function of bandwidth.

computation capacity of cloud computing and the proximity of edge computing. More specifically, we can compute a part of DNN on the edge side, transfer a small number of intermediate results to the cloud, and compute the left part on the cloud side. The partition of DNN constitutes a tradeoff between computation and transmission [10]. As shown in Fig. 2, partition at different layers will cause different computation and transmission times. Bandwidth is 4Mbps. So, an optimal partition is desirable. The latencies are progressively cumulative from left to right. The red section relates to the cumulative computation time of all layers at the edge side prior to layer division, and the individual computation time method for each layer at the edge side is the computation time of the later layer in the figure minus the time of the previous layer. Furthermore, the computation time of conv8 of yolov2 in the figure is longer, which is due to yolov2's unique structure.

Unfortunately, the decision on how to split the DNN layers heavily depends on the network conditions. For example, in a LTE network, the throughput can decrease by 10.33 times during peak hours [11], and this value could reach 18.65 for a WiFi hotspot [12]. Under a high-throughput network condition where computational latency dominates, it is more desirable to offload the DNNs as early as possible. However, if the network condition degrades severely, we should prudently determine the DNN cut to decrease the data transmission volume. For example, Fig. 3 shows that when the network capacity is as high as 18Mbps, the optimal cut is at input layer and the overall processing delay is 0.59s. However, when the network capacity is lowered to 4Mbps, cutting at input layer is no longer valid as the communication delay increases substantially. Under this scenario, cutting at max5 is optimal, with a delay reduction of 62%. nother challenge in the partition is that the recent advances of DNN show that DNNs are no longer limited to a chain topology, and DAG topologies are gaining popularity. For example, GoogleNet [13] and ResNet [14], the champion of ImageNet Challenge 2014 and 2015 respectively, are DAGs. Obviously, partitioning DAG instead of a chain involves much more complicated graph theoretic analysis, which may lead to NP-hardness in performance optimization.

To this end, in this paper, we investigate the DNN partition problem. In order to find the optimal DNN partitioning in an integrated edge and cloud computing environment with dynamic network conditions.

First, we design a Dynamic Adaptive DNN Surgery scheme, which optimally partitions the DNN network by continually monitoring the network condition. The critical design of DNN Surgery is as follows. DNN Surgery monitors the network condition and determines if the system is operated in the lightly loaded or heavily loaded condition. Under the lightly loaded condition, DNN Surgery Light (DSL) is developed, which minimizes the overall delay to process one frame. In this part, to solve the delay minimization problem, we convert the original problem to an equivalent min-cut problem to find the globally optimal solution. On the other hand, DNN Surgery Heavy (DSH) is developed in the heavily loaded condition, which maximizes the throughput, i.e., the number of frames that can be handled per unit of time. However, we prove such an optimization problem is NP-hard, which cannot be solved within polynomial computational complexity. Therefore, DSH resorts to an approximation approach, which achieves an approximation ratio of 3.

Secondly, we break the assumption that the resource of the cloud used for DNN inference is unlimited and study DNN surgery under cost-constrained system, where the resource of the cloud can be used for inference is limited. We formulate the DNN partition problem under cost-constrained system, which is also NP-hard. DNN Surgery Cost-Constrained System (DSCCS) algorithm is developed, which minimizes the overall delay to process one frame using the limited cloud resource.

Then, we develop a real-world testbed to validate our proposed DNN surgery scheme. The testbed is based on the self-driving car video dataset and real traces of the wireless network. We test 5 DNN models. We observe that compared with executing entire DNNs on the cloud and edge, DNN surgery can reduce execution latency up to 6.45 times and 8.08 times, respectively, and improve throughput up to 8.31 times and 14.01 times, respectively. We also show that, under Cost-constrained Systems, DNN surgery can reduce execution latency by up to 4.16 times and improve throughput by up to 8.51 times, respectively.

Finally, we developed various case studies on whether cloud resources are constrained. Our DNN surgery prototype supports an indoor intrusion detection application running in the lab for more than six hours. We further integrate DNN surgery into a campus traffic monitor application (e.g., parking and speeding violations). These case studies demonstrate that DNN surgery can optimally segment DNNs under various network conditions with guaranteed accuracy and speed.

The rest of this paper is organized as follows. Section II gives background and motivation for the Edge-Cloud DNN inference model. Section III explains the Partitioning optimization of the Edge-Cloud DNN inference model in depth. Section IV investigates Edge Cloud DNN Inference under Cost-Constrained System issue. Section V implements a DNN surgery prototype system. Section VI employs real-trace driven simulations to

Authorized licensed use limited to: KAUST. Downloaded on February 19,2024 at 07:16:23 UTC from IEEE Xplore. Restrictions apply.

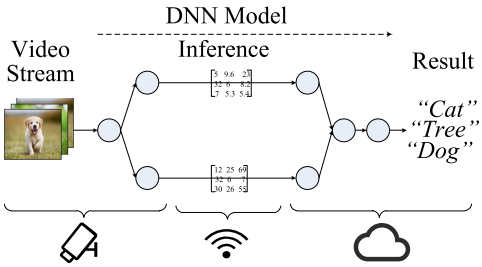


Fig. 4. A 7-layer DNN model classifies frames of video.

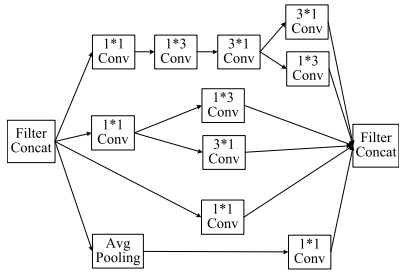


Fig. 5. The inception v4 network represented in layer form.

assess the DNN surgery prototype. Section VII developed various case studies based on whether cloud resources were limited. Section VIII examines related work. Finally, Section IX is the paper's conclusion.

II. An Edge-Cloud DNN Inference (ECDI) Model

A. Background

Video analytics is the core to realizing a wide range of exciting applications ranging from surveillance and self-driving cars, to personal digital assistants and automatic drone controls. The current state-of-the-art approach is to use a deep neural network where the video frames are processed by a well-trained Convolutional neural network or recurrent neural network. Video analytics use DNNs to extract features from input frames of the video and classify the objects in the frames into one of the predefined classes.

DNN network consists of quite a few layers which can be organized in a directed acyclic graph (DAG). Fig. 4 shows a 7-layer DNN model. Inference for video is performed with a DNN using a feed-forward algorithm that operates on each frame separately. The algorithm begins at the input layer and progressively moves forward layer by layer. Each layer receives the output of prior layers as the input, performs a series of computations on the input data to get the output, and feeds its output to the successor layers. This process terminates once the computation of output layer is finished.

The video is generated at the edge side, and the video frames are fed into the DNN as input. The computation of each layer in DNN can be performed at the edge or cloud. Computing layers at edge devices do not require to transmit data to the cloud but incur more computation due to resource-constrained devices. Conversely, computing layers in the cloud lead to less computation but incurs transmission latency for transmitting data from edge devices to the cloud.

B. The ECDI Model

In this subsection. We formally present the ECDI model.

1) Video Frame: A video consists of a sequence of frames (pictures) to be processed, with a sampling rate Q frames/second. Each sampled frame is fed to a predetermined DNN for inference. Please note that the sampling rate is not the frame rate of the video. Instead, it indicates how many frames/pictures are processed each unit time [15].

2) DNN as a Graph: A DNN is modeled as a DAG. Each vertex represents one layer of the neural network. A layer is indivisible and must be processed on either the edge or the cloud. We add a virtual entry vertex and an exit vertex to represent the starting and ending points of DNN, respectively. The links[1] represent communication and dependency among layers.

Let $\mathcal{G} = (\mathcal{V} \bigcup \{e, c\}, \mathcal{L})$ denote the DAG of DNN, where $\mathcal{V} = \{v_1, v_2, \cdots, v_n\}$ is the set of vertices representing the layers of the DNN (specially, v_1 and v_n represent the input layer and output layer respectively). e and c denote virtual entry and exit vertices (to facilitate the subsequent analysis). $\mathcal{L}$ is the set of links. A link $(v_i, v_j) \in \mathcal{L}$ represents that v_i has to be processed before v_j, and v_i feeds its output to v_j. Fig. 6 shows the DAG of the pure inception v4 network [16] in Fig. 5.

Since each layer can be processed on either the edge or cloud side, its processing time depends on where it is processed (i.e., on the edge or on the cloud). Let t_i^e and t_i^c be the time needed to process v_i one edge and cloud respectively. Let d_i and t_i^t denote the output data size and the transmission time of v_i. We define $\mathbf{D}_t \triangleq [d_1, d_2, \cdots, d_n]$. Let B be the network bandwidth, we have $t_i^t = \frac{d_i}{B}$. Please note that B can be dynamically changed and we need to adapt such changes. We define $\mathbf{F}_e \triangleq [t_1^e, t_2^e, \cdots, t_n^e]$, $\mathbf{F}_c \triangleq [t_1^c, t_2^c, \cdots, t_n^c]$, $\mathbf{F}_t \triangleq [t_1^t, t_2^t, \cdots, t_n^t]$. They denote the *three key delays*: processing delay at the edge, transmission delay, and processing delay at the cloud of each layer.

3) DNN Partitioning: Our objective is to partition DNN into two parts so the one part is processed at the edge and the other is processed at the cloud. Mathematically, we should find a set of vertices $\mathcal{V}_S$ as a subset of $\mathcal{V}$ such that removing $\mathcal{V}_S$ causes that the rest of $\mathcal{G}$ becomes two disconnected components. One component contains e, denoted by $\mathcal{V}'_E$ and the other component contains c, denoted by $\mathcal{V}_C$. $\mathcal{V}_S$ is the cut so that all downstreaming layers are processed at the cloud. $\mathcal{V}'_E$ and $\mathcal{V}_S$ are processed at the edge and $\mathcal{V}_C$ are processed at the cloud. We define $\mathcal{V}_E = \mathcal{V}'_E \bigcup \mathcal{V}_S$. The output data of vertices in $\mathcal{V}_S$ will be transmitted from the edge side to the cloud. $\mathcal{V}_E$, including $\mathcal{V}'_E$ and $\mathcal{V}_S$ will generate processing delay at the edge. $\mathcal{V}_S$ will generate transmission delay. $\mathcal{V}_C$ will generate processing delay at the cloud. Our aim is to determine best cut $\mathcal{V}_S$ so that the overall delay is minimized.

As shown in Fig. 6, we cut at $\mathcal{V}_S = \{v_3, v_5, v_9, v_{12}\}$ so that the $\mathcal{V}'_E = \{e, v_1, v_2, v_4\}$, $\mathcal{V}_E = \{e, v_1, v_2, v_3, v_4, v_5, v_9, v_{12}\}$, and $\mathcal{V}_C = \{v_6, v_7, v_8, v_{10}, v_{11}, v_{13}, c\}$. The overall delay is the processing delay of $\mathcal{V}_E$ on the edge and $\mathcal{V}_C$ on the cloud plus the communication delay of the output data of layer in $\mathcal{V}_S$.

[1]Please note that to avoid misunderstanding, throughout this paper, we use the term "link" to represent the "edge of a graph." This is because "edge" in this paper already represents "edge computing."

Authorized licensed use limited to: KAUST. Downloaded on February 19,2024 at 07:16:23 UTC from IEEE Xplore. Restrictions apply.

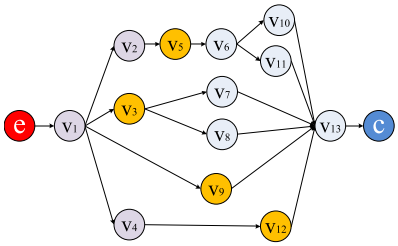


Fig. 6. Graph representation of inception v4 network.

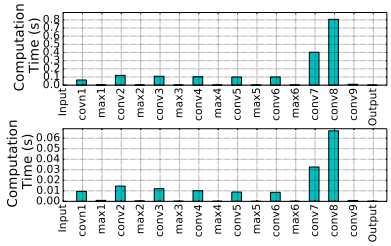


Fig. 7. The computation latency of YOLOv2's layers on the edge (top) and cloud (bottom) respectively.

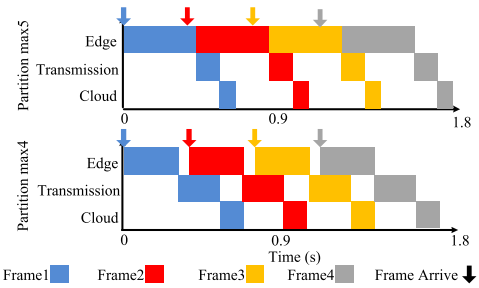


Fig. 8. Gantt charts for three stages.

4) Delay Components: Once the partition is made, each frame is processed at the edge, and then sent from the edge to the cloud, and then processed at the cloud. Since there are multiple frames to be processed, we assume that the three stages are conducted in *pipeline*. In order words, when frame 1 is being processed at the cloud, frame 2 can be transmitted and frame 3 can be processed at the edge.

The delays of the three stages are characterized as follows. In the edge-computing stage

$$T_e = \sum_{v_i \in \mathcal{V}_E} t_i^e. \tag{1}$$

In the cloud-computing stage

$$T_c = \sum_{v_i \in \mathcal{V}_C} t_i^c. \tag{2}$$

In the communication stage

$$T_t = \sum_{v_i \in \mathcal{V}_S} t_i^t. \tag{3}$$

For each frame, T_e, T_c, and T_t are spent for each stage. Frames are processed in pipeline every $\frac{1}{Q}$. As a consequence, the Gantt chart (scheduling chart) of frames can be shown in Fig. 8. T_e, T_c, and T_l cannot exceed $\frac{1}{Q}$. Otherwise, the incoming rate is greater than the completion rate, leading to system congestion. *Our aim is to smartly partition the DNN so that the overall delay to process frames is minimized and the system is not congested.*

C. Parameter Estimation for ECDI

In this subsection, we discuss how to derive the input parameters. The first class of parameters is called DNN profile, including DNN topology $\mathcal{G}$, processing delays of each layer at the edge and the cloud $\mathbf{F}_e$, $\mathbf{F}_c$, data size of each layer $\mathbf{D}_t$. These parameters can be well derived in advance. $\mathcal{G}$ and $\mathbf{D}_t$ can be directly derived given the DNN definition. $\mathbf{F}_e$ and $\mathbf{F}_c$ can be measured beforehand. For example, we derive $\mathbf{D}_t$ of tiny YOLOv2 model and measure $\mathbf{F}_e$ of tiny YOLOv2 model processed on Raspberry Pi 3 model B and Ali Cloud respectively. We show the results in Figs. 1 and 7 respectively.

The value B is dynamic and should be measured during the process of DNN inference. This can be realized by a method similar to HTTP DASH [17]. We use the tool "ping" at edge to send two different size data consecutively to the cloud, and measure the response times. The bandwidth equals to the ratio between the difference of data size and the difference of response times.

The value Q is user-specific. The user lets the system know Q when the inference starts. The system does nothing unless Q is too large for the system to handle (See Section III-D).

III. ECDI Partitioning Optimization

A. The Impact of DNN Inference Workloads

Our first objective is to minimize the overall delay to process each frame. This is true under the *light workload*: for each stage, the current frame is completed before the next frame arrives. Mathematically $\max\{T_e, T_t, T_c\} < \frac{1}{Q}$ so that the Gantt chart is shown as the bottom one of Fig. 8. In this case, we just need to complete every frame *as soon as possible*, i.e., minimize $T_c + T_t + T_e$.

However, if the system is heavily loaded, minimizing $T_e + T_t + T_c$ may lead to system congestion as $\max\{T_e, T_t, T_c\} \geq \frac{1}{Q}$. For example, in Fig. 8 (top), $T_e > \frac{1}{Q}$ so that the next frame arrives before the current frame is completed at the edge. Therefore, under this situation, we need to maximize the throughput of the system, i.e., how many frames at most the system can handle per unit time. Our objective is to minimize $\max\{T_e, T_t, T_c\}$ as the system throughput is $\frac{1}{\max\{T_e, T_t, T_c\}}$. For presentation convenience, $\max\{T_e, T_t, T_c\}$ is referred to as the *max stage time*.

Please note that in Section III-D, we will further discuss how to judge if the system is lightly loaded or heavily loaded. There, we also need to consider that if the sampling rate is greater than $\frac{1}{\min\max\{T_e, T_t, T_c\}}$ so that the system will be congested eventually. The system has to force the sender/user to reduce sampling rate.

B. The Light Workload Partitioning Algorithm

In this subsection, we study *Edge Cloud DNN Inference for Light Workload (ECDI-L)* problem. Our goal is to minimize the overall delay of one frame, under a given the network condition B. In summary, we have the following optimization problem:

Problem 1. (ECDI-L) Given $\mathcal{G}$, $[\mathbf{F}_e, \mathbf{F}_c, \mathbf{D}_t]$, and B, determine $\mathcal{V}_E$, $\mathcal{V}_S$ and $\mathcal{V}_C$, to minimize $T_{\text{inf}} = T_e + T_t + T_c$.

Authorized licensed use limited to: KAUST. Downloaded on February 19,2024 at 07:16:23 UTC from IEEE Xplore. Restrictions apply.

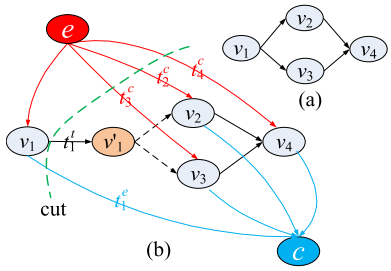


Fig. 9. Illustration of conversion to the minimum s-t cut problem.

Proposition 1. Problem ECDI-L can be solved in polynomial time.

One challenge to solve ECDI-L problem directly is that each vertex in $\mathcal{G}$ contains three delay values $t_i^e, t_i^c, t_i^t = \frac{d_i}{B}$. The delay that contributes to the overall delay depends on where the vertex is processed. To this end, we construct a new graph $\mathcal{G}'$ so that each edge only captures *a single* delay value. By doing so, we convert ECDI-L problem to the minimum weighted s-t cut problem of $\mathcal{G}'$.

We first illustrate how to construct $\mathcal{G}'$ based on $\mathcal{G}$:

1) *Cloud Computing Delay.* Based on $\mathcal{G}$, we add links between e and each vertex $v \in \mathcal{V}$, referred to as "red links," to capture the cloud-computing delay of v.
2) *Edge Computing Delay.* Similarly, we add links between vertex $v \in \mathcal{V}$ and c, referred to as "blue links," to capture the edge-computing delay of v.
3) *Communication Delay.* All the other links correspond to communication delays. A link from v to u should capture the communication delay of v.

However, this is insufficient as one vertex may have multiple successors and its communication delay is counted multiple times. For example, v_1 in Fig. 6 has 4 outgoing links but the communication delay of v_1 has to be counted at most once. To this end, we introduce *auxiliary vertices* into graph $\mathcal{G}'$. That is, for any vertex $v_k \in \mathcal{V}$ whose outdegree is greater than one, we add an auxiliary vertex v'_k and link (v_k, v'_k). The links from v_k to successors of v_k are now re-placed from v'_k to successors of v_k. For example, a 4-layer DNN is shown in Fig. 9(a). The outdegree of vertex v_1 is greater than one, we thus add an auxiliary vertex v'_1 and link (v_1, v'_1) shown in Fig. 9(b). The links (v_1, v_2) and (v_1, v_3) are re-placed by links (v'_1, v_2) and (v'_1, v_3) respectively. We define $\mathcal{V}_D$ to be the set of auxiliary vertices.

Now, without considering e and c, if a vertex v has one successor, the link starting from v corresponds to its communication delay, which is referred to as "black link." If v has multiple successors, then all the links starting from v are referred to as "dashed links" and should not be considered since the communication delay has already been considered from v to v'.

Links are assigned costs. The costs assigned to red, blue, black links are cloud-computing, edge-computing, and communication delays. Dashed links are assigned infinity.

$$c(v_i, v_j) = \begin{cases} t_i^e, & \text{if } v_i \in \mathcal{V}, v_j = c. \\ t_i^t, & \text{if } v_i \in \mathcal{V}, v_j \in \mathcal{V} \bigcup \mathcal{V}_D. \\ t_i^c, & \text{if } v_i = e, v_j \in \mathcal{V}. \\ +\infty, & \text{others.} \end{cases} \tag{4}$$

Algorithm 1: DSL Algorithm `DSL()`.

Input: $\mathcal{G}, \mathbf{F}_e, \mathbf{F}_c \mathbf{D}_t, B$
Output: $\mathcal{V}_E, \mathcal{V}_S, \mathcal{V}_C, T_e, T_t, T_c$
1 $\mathbf{F}_t \leftarrow$ `compute-net(`$\mathbf{D}_t, B$`);`
2 $\mathcal{G}' \leftarrow$ `graph-construct(`$\mathcal{G}, \mathbf{F}_e, \mathbf{F}_c, \mathbf{F}_t$`);`
3 $[\mathcal{V}_E, \mathcal{V}_S, \mathcal{V}_C, T_e, T_t, T_c] \leftarrow$ `min-cut(`$\mathcal{G}'$`);`
4 return $\mathcal{V}_E, \mathcal{V}_S, \mathcal{V}_C, T_e, T_t, T_c$;

At this stage, we can convert ECDI-L problem to the minimum weighted s–t cut problem of $\mathcal{G}'$.

A cut is a partition of the vertices of a DAG into two disjoint subsets. The s–t cut of $\mathcal{G}'$ is a cut that requires source s and sink t to be in different subsets, and its *cut-set only consists of links going from the source's side to sink's side*. The value of a cut is defined as the sum of the cost of each link in the cut-set. Problem ECDI-L is equivalent to the minimum e–c cut of $\mathcal{G}'$. If cutting on link from e to $v_i \in \mathcal{V}$ (red link shown in Fig. 9(b)), then v_i will be processed on the cloud, i.e $v_i \in \mathcal{V}_C$. If cutting on link from $v_j \in \mathcal{V}$ to c (blue link show in Fig. 9(b)), then v_j will be processed on the edge, i.e., $v_j \in \mathcal{V}_E$. If cutting on link from $v_i \in \mathcal{V}$ to $v_j \in \mathcal{V} \bigcup \mathcal{V}_D$ (black link show in Fig. 9(b)), then the data of v_i will be transmitted to the cloud, i.e $v_i \in \mathcal{V}_S$. It is impossible to cut on link from $v_i \in \mathcal{V}_D$ to $v_j \in \mathcal{V}$ (dashed links), because otherwise it will lead to infinite cost (but finite cost exists). The total cost of cut on red links equals to cloud computation time T_c. The total cost of cut on blue links equals to edge computation time T_e. The total cost of cut on black links equals to transmission time without network latency T_t. If the e–c cut of $\mathcal{G}'$ is minimum, then the inference latency on a single frame is minimum. For example, in Fig 9(b), the cut is at (e, v_2), (e, v_3), (e, v_4), (v_1, v'_1) and (v_1, c). v_1 is processed at the edge so that t_1^e is counted in the blue link. v_2, v_3 and v_4 are processed at the cloud so that t_2^c t_3^c and t_4^c are counted in the red links. The communication delay t_1^t is counted in the black link.

We develop DNN Surgery Light (denoted as DSL) algorithm for ECDI-L problem. The overall algorithm `DSL()` is shown in Algorithm 1. The algorithm first calls `compute-net()` to compute $\mathbf{F}_t$. Then it calls `graph-construct()` (line 2) to construct $\mathcal{G}'$ based on $\mathcal{G}$ with the computation complexity of $\mathcal{O}(n+m)$, where n is the number of layers $|\mathcal{V}|$, m is the number of links $|\mathcal{L}|$, and then it calls `min-cut()` (line 3) to find minimum e–c cut of $\mathcal{G}'$ which outputs the partition strategy (i.e., $\mathcal{V}_E, \mathcal{V}_S$ and $\mathcal{V}_C$). Boykov's algorithm [18] is used in `min-cut()` to solve the minimum e–c cut problem with the computational complexity of $\mathcal{O}((m+n)n^2)$. `DSL()` is a polynomial-time algorithm with the computational complexity of $\mathcal{O}((m+n)n^2)$.

C. The Heavy Workload Partitioning Algorithms

As discussed in Section III-A, we formulate the *Edge Cloud DNN Inference for Heavy Workload (ECDI-H)* problem, to minimize $\max\{T_e, T_t, T_c\}$. The decision variables are $\mathcal{V}_E$, $\mathcal{V}_S$ and $\mathcal{V}_C$. In summary, we have the following optimization problem:

Problem 2. (ECDI-H) Given $\mathcal{G}$, $[\mathbf{F}_e, \mathbf{F}_c, \mathbf{D}_t]$, and B, determine $\mathcal{V}_E$, $\mathcal{V}_S$ and $\mathcal{V}_C$, to minimize $\max\{T_e, T_t, T_c\}$ (i.e., maximize throughput).

Theorem 1. Problem ECDI-H is NP-hard.

Authorized licensed use limited to: KAUST. Downloaded on February 19,2024 at 07:16:23 UTC from IEEE Xplore. Restrictions apply.

Proof. We prove this theorem by reducing from the minimum bisection problem (MSP), which is known to be NP-complete. We consider the following the minimum bisection problem on $\mathcal{G}_A = (\mathcal{V}_A, \mathcal{L}_A)$ with n_A vertices, where n_A is even, the goal is to find a link cut set to partition the graph into two disjoin components (V_1, V_2), for which $|V_1| = |V_2| = \frac{n}{2}$ and the link cut size is at most $\frac{n}{2}$. We show that any instance of the above problem is equivalent to an instance in ECDI-H problem.

From the instance of MSP, we can construct an instance of ECDI-H as follows: Construct a DNN network the same as G_A. Let w_i denote the out degree of vertex $v_i \in \mathcal{V}_A$. For each $v_i \in \mathcal{V}_A$ we set $t_i^e = 1$, $t_i^c = 1$, and $t_i^t = w_i$. We aim to judge if the min of max of the three delays can reach $\frac{n}{2}$. The instance of MSP is solved if and only if the min max delay in the instance of DSH is $\frac{n}{2}$. (1) If MSP solved, we let $\mathbf{V}_1$ and $\mathbf{V}_2$ be processed at the edge and cloud respectively and the min max delay reaches $\frac{n}{2}$. (2) When min max delay of ECDI-H is minimized as $\frac{n}{2}$, we can find $\frac{n}{2}$ vertices in the edge and $\frac{n}{2}$ vertices in the cloud, and the total delay t_i^t of vertices in the vertex cut is at most $\frac{n}{2}$. Note that t_i^t is set as the out degree of vertex v_i, thus the number of links going from the vertices in the vertex cut is smaller than $\frac{n}{2}$. The links going from the vertices in the vertex cut form a link cut set and partition the graph into two disjoin components with the same vertex size, i.e., the vertices in edge and vertices in the cloud, so that we find the solution to MSP.

ECDI-H is NP-hard. It is unrealistic to find a globally optimal solution within polynomial time. We design DNN Surgery Heavy (denoted as DSH) algorithm which achieves a locally optimal solution. In addition, its approximation ratio is 3.

The rationale to develop DSH is as follows. We modify $\mathcal{G}'$ by changing the costs of links as follow:

$$c(v_i, v_j) = \begin{cases} \alpha t_i^e, & \text{if} \quad v_i \in \mathcal{V}, v_j = c. \\ \beta t_i^t, & \text{if} \quad v_i \in \mathcal{V}, v_j \in \mathcal{V} \bigcup \mathcal{V}_D. \\ \gamma t_i^c, & \text{if} \quad v_i = e, v_j \in \mathcal{V}. \\ +\infty, & \text{others.} \end{cases} \tag{5}$$

Here α, β and γ are non-negative variables. The approach is to run `DSL()` with several different α, β and γ values. By this way, a solution is generated to optimize ECDI-L with a specific α, β, γ tuple. Then we test if this solution is also good enough for ECDI-H. If it is better than all existing solutions, it is regarded as a new solution to ECDI-H. We repeat the above procedure for a wide range of α, β, γ tuples.

Here, the result of `DSL()` is determined by the ratio of the three parameters, instead of their absolute values. Therefore, we can fix one of the three, for example, $\beta = 1$, and only vary the other two. Thus, we have a two-dimensional search space for α and γ. We first search in the two-dimensional plane with a coarse granularity to find the best solution. Then we use a finer granularity search in the neighborhood of the best solution for further improvement. We repeat the steps until the improved performance is smaller than a threshold ϵ.

The overall algorithm `DSH()` is shown in Algorithm 2. A function `search()` (line 11–19) is designed to search for the best solution in a given space $\mathbf{S} \triangleq [\alpha_l, \gamma_l, \alpha_h, \gamma_h]$, meaning that $\alpha_l \leq \alpha \leq \alpha_h, \gamma_l \leq \gamma \leq \gamma_h$, and a granularity δ (line 13–14), i.e., the step size of changing α and γ is δ each time. For each α and γ, `search()` calls `DSL()` to compute the vertex cut and calls `max-time()` to compute the $\max[T_c, T_e, T_t]$.

Algorithm 2: DSH Algorithm `DSH()`.

Input: $\mathcal{G}, \mathbf{F}_e, \mathbf{F}_c, \mathbf{D}_t, B, \epsilon, K$
Output: $\mathcal{V}_E, \mathcal{V}_S, \mathcal{V}_C, T_{max}$
1 $\mathbf{F}_t \leftarrow$`compute-net(`$\mathbf{D}_t, B$`)`;
2 $T_{max} \leftarrow +\infty$; $T'_{max} \leftarrow 0$; $\delta \leftarrow 1$;
3 $\alpha_l \leftarrow 0$; $\gamma_l \leftarrow 0$; $\alpha_u \leftarrow \frac{\sum(\mathbf{F}_e)}{\min(\mathbf{F}_t)}$; $\gamma_u \leftarrow \frac{\sum(\mathbf{F}_c)}{\min(\mathbf{F}_t)}$;
4 $\mathbf{S} \leftarrow [\alpha_l, \gamma_l, \alpha_u, \gamma_u]$;
5 **while** $|T'_{max} - T_{max}| \geq \epsilon$ **do**
6 $\quad T'_{max} \leftarrow T_{max}$;
7 $\quad [\alpha, \gamma, \mathcal{V}_E, \mathcal{V}_S, \mathcal{V}_C, T_{max}] \leftarrow$`Search(`$\mathbf{S}, \delta, T_{max}$`)`;
8 $\quad \alpha_l \leftarrow \alpha - \delta$; $\alpha_u \leftarrow \alpha + \delta$;$\gamma_l \leftarrow \gamma - \delta$; $\gamma_u \leftarrow \gamma + \delta$;
9 $\quad \delta \leftarrow \delta / K$;
10 return $\mathcal{V}_E, \mathcal{V}_S, \mathcal{V}_C, T_{max}$;
11 **function** *Search*$([\alpha_l, \gamma_l, \alpha_u, \gamma_u], \delta, T^*_{max})$
12 $\quad T_{max} \leftarrow +\infty$;
13 $\quad$ **for** $\alpha \leftarrow \alpha_l; \alpha \leq \alpha_u; \alpha \leftarrow \alpha + \delta$ **do**
14 $\quad\quad$ **for** $\gamma \leftarrow \gamma_l; \gamma \leq \gamma_u; \gamma \leftarrow \gamma + \delta$ **do**
15 $\quad\quad\quad [\mathcal{V}_E, \mathcal{V}_S, \mathcal{V}_C, T_e, T_t, T_c] \leftarrow$`DSL(`$\mathcal{G}, \alpha\mathbf{F}_e, \gamma\mathbf{F}_c, \mathbf{D}_t, B$`)`
16 $\quad\quad\quad T_{max} \leftarrow$`max-time(`$T_e, T_t, T_c$`)`;
17 $\quad\quad\quad$ **if** $T_{max} \leq T^*_{max}$ **then**
18 $\quad\quad\quad\quad \alpha^* \leftarrow \alpha$; $\gamma^* \leftarrow \gamma$; $T^*_{max} \leftarrow T_{max}$;
19 $\quad$ return $\alpha^*, \gamma^*, \mathcal{V}_E, \mathcal{V}_S, \mathcal{V}_C, T^*_{max}$;

Lines 17–18 guarantee $\max[T_c, T_e, T_t]$ derived is non-increasing.

The overall algorithm first initializes the search granularity δ to be 1 (line 2) and the search space large enough (line 3–4). It calls `search()` (line 8) to search on the given space $\mathbf{S}$ with a granularity δ, and returns the best α and γ found currently. Then `DSH()` narrows down the search space $\mathbf{S}$ (line 8) to the neighborhood of the best α and γ for the current iteration, and adjusts δ to a finer granularity (line 9). Such space $\mathbf{S}$ and granularity δ is returned to `search()`. The termination condition for the loop is that the improved performance is smaller than a threshold ϵ (line 5). Finally, it returns the vertex cut with the best-found performance (line 10). Obviously, we can achieve a local optimal result with respect to the neighborhood of the final α and γ.

Theorem 2. The approximation ratio of the algorithm DSH for ECDI-H is 3.

Proof. Let the max stage time of DHL be t_{DSH}. Let the optimal max stage time of ECDI-H be t^*. We prove $\frac{t_{DSH}}{t^*} \leq 3$. Let T^* denote the minimum inference latency for one frame. Let T_o denote the inference latency of a single frame when achieving the optimal max stage time. We have $T^* \leq T_o$. Because there are three stages, we have $T_o \leq 3t^*$, thus $T^* \leq 3t_o$.

As shown in Algorithm 2, when $\delta = 1$, `Search()` will calls `DSL()` using $\alpha = 1$ and $\gamma = 1$ as the parameter. When $\alpha = 1$ and $\gamma = 1$, `DSL()` achieves the minimum inference time T^* for one frame. Let t_1, t_2 and t_3 be the edge computation time, the transmission time and the cloud computation time respectively when achieving the minimum inference time. We have $T^* = t_1 + t_2 + t_3$. `DSH()` guarantees the searched max stage time is non-increasing, thus $t_m \leq \max\{t_1, t_2, t_3\}$, combined with $T^* = t_1 + t_2 + t_3$, we have $t_m \leq T_{min}$. As $t_m \leq T^*$ and $T^* \leq 3t^*$, we prove $\frac{t_{DSH}}{t^*} \leq 3$.

D. The Dynamic Partitioning Algorithm

We now consider network dynamics. In practice, the network status B varies. This will affect the workload mode selection

Authorized licensed use limited to: KAUST. Downloaded on February 19,2024 at 07:16:23 UTC from IEEE Xplore. Restrictions apply.

Algorithm 3: DNN Surgery Algorithm `DADS()`.

```
1 while true do
2     if monitor-task()==true then
3         B ← monitor-net();
4         [V_E, V_S, V_C, T_e, T_t, T_c] ← DSL(G, F_e, F_c, D_t, B);
5         if max{T_e, T_t, T_c} > 1/Q then
6             [V_E, V_S, V_C, T_max] ← DSH(G, F_e, F_c, D_t, B, ε, K);
7             if T_max > 1/Q then
8                 inform-decrease();
9         execute(V_E, V_S, V_C);
```

and the partition decision dynamically. We design Dynamic Adaptive DNN Surgery scheme to adapt network dynamics.

It is shown in Algorithm 3. `monitor-task()` monitors whether the video is active (line 2). This can be realized by tool "iperf." Detailed implementation can be found in Section V. The real-time network bandwidth is derived by `monitor-net()` (line 3). Then `DSL()` is called to compute the partition strategy (line 4). In this case, if it satisfies the sampling rate $\frac{1}{Q}$, i.e.$\max\{T_e, T_t, T_c\} < \frac{1}{Q}$, we can confirm that the system is in the light workload mode and the partition by DSL is accepted.

Otherwise, the system is in the heavy workload mode and calls `DSH()` to adjust the partition strategy to minimize the max delay (line 6). However, if the completing rate is still smaller than the sampling rate, it means that the sampling rate if too large so that even `DSH()` still cannot satisfy the sampling rate. The system will be congested. It calls the user to decreases the sampling rate (line 7–8).

IV. Partitioning Optimization Under Cost-Constrained System

We now specifically consider partitioning under cost-constrained system. This is because, in practice, the cloud have to deal with video analytics tasks from a variety of edge devices. From the cloud perspective, the computational and communicational resource allocated for each task of each device is limited, i.e., the cloud has a hard real-time constraints for processing a frame. In this section, we study *Edge Cloud DNN Inference under Cost-constrained System (ECDI-CCS)* problem.

A. Problem

For cost-constrained cloud, the time (including communication time and computation time) allocated for processing a frame is limited. Let T_0 denote the constrained running time of the cloud. We have the following cost constraint:

$$T_t + T_c \leq T_0 \tag{6}$$

The objective is to minimize the overall delay of one frame, which is equivalent to minimize the edge-computing delay T_e in this scenario. Thus, we arrive the following problem:

Problem 3. (ECDI-CCS) Given $\mathcal{G}$, $[\mathbf{F}_e, \mathbf{F}_c, \mathbf{D}_t]$, and B, determine $\mathcal{V}_E$, $\mathcal{V}_S$ and $\mathcal{V}_C$, subject to constraint (6), to minimize T_e.

B. Problem Analysis

In this subsection, we analyze the complexity of problem ECDI-CCS. We prove problem ECDI-CCS is NP-hard by reducing it to an *introduced decision (IntroD)* problem which is NP-complete.

The following decision problem IntroD can be defined:

Problem 4. (IntroD) Given $\mathcal{G}$, $[\mathbf{F}_e, \mathbf{F}_c, \mathbf{D}_t]$, B, T_0 and T_1, is there a partition $[\mathcal{V}_E, \mathcal{V}_S, \mathcal{V}_C]$ so that $T_e \leq T_1$ and $T_t + T_c \leq T_0$.

Theorem 3. Problem IntroD is NP-complete even if only graphs with no links are considered.

Proof. To prove the NP-hardness, we reduce the Knapsack problem to problem IntroD. Let an instance of Knapsack problem be given, i.e., there are n objects, the weights of the objects are denoted by w_i, the price of the objects by p_i, the weight limit by W and the price limit by K. The task is to decide whether there is a subset X of objects, so that $\sum_{i\in X} w_i \leq W$ and $\sum_{i\in X} p_i \geq K$. Based on this, we define an instance of problem IntroD as follows: $\mathcal{V} = \{v_1, v_2, ..., v_n\}$, $\mathcal{L} = \emptyset$. Let $t_i^c = p_i$, $t_i^e = w_i$ (since $\mathcal{L} = \emptyset$, we can define $t_i^t = 0$). Introducing $A = \sum_{v_i\in\mathcal{V}} p_i$. Let $T_1 = W$ and $T_0 = A - K$.

We state that this instance of IntroD is solvable iff the original Knapsack problem has a solution. Assuming that problem IntroD has a solution $(\mathcal{V}_E, \mathcal{V}_S, \mathcal{V}_C)$, where $\mathcal{V}_S = \emptyset, \mathcal{V}_E \bigcup \mathcal{V}_C = \mathcal{V}$ and $\mathcal{V}_E \bigcap \mathcal{V}_C = \emptyset$. This means that $T_e \leq W$ and $T_c + T_t = \sum_{v_i\in\mathcal{V}_C} \leq A - K = \sum_{v_i\in\mathcal{V}} p_i - K$. The latter one can be formulated as $K \leq \sum_{v_i\in\mathcal{V}} p_i - \sum_{v_i\in\mathcal{V}_C} p_i = \sum_{v_i\in\mathcal{V}_E} p_i$. This proves that $X = \mathcal{V}_E$ is a solution of the original Knapsack problem.

Now let assume that X solves the Knapsack problem. Therefore $\sum_{v_i\in X} t_i^e = \sum_{v_i\in X} w_i \leq W = T_1$ and $\sum_{v_i\in X} p_i \geq K = A - T_0 = \sum_{v_i\in\mathcal{V}} p_i - T_0$. The latter one can be formulated as $T_0 \geq \sum_{v_i\in\mathcal{V}} p_i - \sum_{v_i\in X} p_i = \sum_{v_i\in\mathcal{V}/X} p_i = \sum_{v_i\in\mathcal{V}/X} t_i^c$. This verifies that $(\mathcal{V}/X, \emptyset, X)$ solves problem IntroD.

The above proof shows that the special case of problem IntroD which the graph has no link is equivalent with Knapsack problem.

Theorem 4. Problem ECDI-CCS is NP-hard.

Proof. Problem IntroD can be reduced to problem ECDI-CCS: ECDI-CCS provides a solution where $T_t + T_c \leq T_0$ and T_e is minimal; let this value be T_e^*. Clearly IntroD is solvable iff $T_e^* \leq T_1$.

C. Algorithm

We develop an integer linear programming algorithm to solve ECDI-CCS problem.

Let f_i denote the indicator, i.e., $f_i = 0$ if v_i is processed at cloud; $f_i = 1$ if v_i is processed at edge. We define $\mathbf{f} \triangleq [f_1, f_2, ..., f_n]$. $\mathbf{f}$ is a decision variable to be optimized.

Let $\mathbf{H} \in \{-1, 0, 1\}^{m\times n}$ (where m is the number of links and n is the number of DNN layers) is the transposed incidence matrix of graph $\mathcal{G}'' = \{\mathcal{V}, \mathcal{L}\}$, that is

$$h_{ij} = \begin{cases} -1, & \text{if } l_{ji} \in \mathcal{L}. \\ 1, & \text{if } l_{ij} \in \mathcal{L}. \\ 0, & \text{others.} \end{cases} \tag{7}$$

It can be seen that the components of the vector $|\mathbf{Hf}|$ indicate which vertices incur communication cost. The problem can be

Authorized licensed use limited to: KAUST. Downloaded on February 19,2024 at 07:16:23 UTC from IEEE Xplore. Restrictions apply.

formulated as follows:

$$\min T_e = \mathbf{F}_e\mathbf{f} \tag{8}$$

$$\mathbf{F}_c(\mathbf{1}-\mathbf{f}) + \mathbf{F}_t|\mathbf{H}\mathbf{f}| \leq T_0 \tag{9}$$

$$\mathbf{f} \in \{0,1\}^n \tag{10}$$

This problem can be transformed to an integer linear programming (ILP) equivalent by introducing the variables $\mathbf{y} \in \mathbf{R}^m$ to eliminate the $|\mathbf{H}\mathbf{f}|$:

$$\mathbf{F}_c(\mathbf{1}-\mathbf{f}) + \mathbf{F}_t\mathbf{y} \leq T_0 \tag{11}$$

$$\mathbf{H}\mathbf{f} \leq \mathbf{y} \tag{12}$$

$$-\mathbf{H}\mathbf{f} \leq \mathbf{y} \tag{13}$$

Constraints (11), (12), and (13) are equivalent to constraint (9). More specifically, if $\mathbf{f}$ solves (9), then $(\mathbf{f}, |\mathbf{H}\mathbf{f}|)$ will solve (11)- (13); if $(\mathbf{f}, \mathbf{y}$ solves (11)- (13), then $\mathbf{f}$ will solve (9) too, since $|\mathbf{H}\mathbf{f}| \leq \mathbf{y}$ and $\mathbf{F}_t \geq \mathbf{0}$.

We design DNN Surgery Cost-Constrained System (denoted as DSCCS) algorithm to solve this ILP using LP-relaxation and branch-and-bound [19].

V. Implementation

We implement a DNN surgery prototype system. We use the Raspberry Pi 3 model B as the edge device, integrated with a Logitech BRIO camera. We rent a server in Cloud Ali with eight cores of 2.5 GHz and a total memory of 128 GB. We employ WiFi as the communication link between the edge device and the cloud. The wired link from the edge router and the cloud is sufficiently large. We implement our client-server interface using GRPC, an open source flexible remote procedure call (RPC) interface for inter-process communication.

The Edge Device. The duty of the edge device is to 1) extract video from the camera and to sample frames from video, 2) make partition decision, 3) process the layers allocated to the edge device, and 4) inform the cloud the partition decision and transfer the intermediate results to the cloud.

For video extraction, we extract videos from camera logitech BRIO using the provided API `video_capture()`. The camera transfers the captured video to Raspberry Pi through the USB-to-serial cable.

For partition decision making, we implement a process that monitors the generated frame by the camera, and runs DNN surgery scheme. DNN surgery requires to estimate the real-time network bandwidth. We use the command "iperf" provided by the operation system Raspbian on Raspberry Pi. This command feeds back the real-time network bandwidth between the Raspberry Pi and the cloud.

For processing allocated layers on the edge, we install a modified instance of Caffe and store a full DNN model on the edge device. The challenge is to control Caffe to stop execution at partitioned layers (e.g., $\mathcal{V}_S$). In Caffe, there is a "prototxt" file recording the DNN structure. Layers are processed according to this file. To solve the challenge, we modify the model structure file "prototxt" by inserting a "stop layer" after each partitioned layer. The instance of Caffe will stop processing at the desired places. We modify the execution flow of Caffe by setting a breakpoint at the stop layer, and the modified Caffe system automatically recognizes the breakpoint and stops execution.

TABLE I
DNN Benchmark Specifications

	CAT1	3G	4G	WiFi
Uplink rate (Mpbs)	0.13	1.1	5.85	18.88

We modified the execution flow of the Caffe system, redesigned the API, and added parameters describing the stop layers. Our program adds the stop layer when initializing the DNN model, and when Caffe executes the inference, it parses the parameters and identifies the stop layer. This method can initialize once and dynamically stop inference by changing the stop layers parameters.

For the intermediate results and partition decision transmission, the edge device calls the RPC function `receiveRPC()` provided by the cloud to transmit the data to the cloud.

The Cloud. The duty of the cloud is to execute the DNN layers allocated to the cloud. There are two jobs: 1) to receive the partition decision and the intermediate results from the edge device, and 2) to execute the layers allocated to the cloud.

For the first job, we expose an API `receiveRPC()` to the edge device. After completing processing layers allocated to the edge, the edge device calls this RPC function to transmit the intermediate results packed with the partition decision to the cloud.

For the second job, we implement a modified instance of Caffe and store a full DNN model. The challenge is to execute only the layers allocated to the cloud. To this end, after receiving the partition decision and intermediate results, the layers allocated to the edge are deleted before the marked place in "prototxt," and the intermediate results are forwarded to the corresponding layers as input. By this way, only layers allocated to the cloud will be executed.

VI. Performance Evaluation

We evaluate the DNN surgery prototype (Section V) using real-trace driven simulations.

A. Setup

Video Datasets. We employ the publicly available BDD100K self-driving dataset [20]. The videos of this dataset are obtained from the camera on the self-driving car. Each video is about 40 seconds long and is viewed in 720p at 30 FPS.

Workload Setting. We divide the inference task into low workload mode and heavy workload mode. Accordingly, We transform the video into different sampling rates to produce different workload. We set a low sampling rate to 0.1 frame per second when evaluating light workload mode, and 20 frames per second for heavy workload mode. The default resolution is 224p. Each inference task consists of processing 100 frames using the given DNN benchmarks.

Communication Network Parameters. To model the communication between edge and cloud, we used the average uplink rate of mobile Internet for different wireless networks, i.e., CAT1, 3G, 4G and WiFi as shown in Table I.

DNN Benchmarks. DNN surgery can make partition not only on chain topology DNN but also on the DAG topology as shown in Table II. We evaluate the performance of DNN surgery for

Authorized licensed use limited to: KAUST. Downloaded on February 19,2024 at 07:16:23 UTC from IEEE Xplore. Restrictions apply.

TABLE II
DNN BENCHMARK SPECIFICATIONS

Type	Chain			DAG	
Model	NiN	YOLOv2	VGG16	Alexnet	ResNet18
Layers	9	17	24	23	20

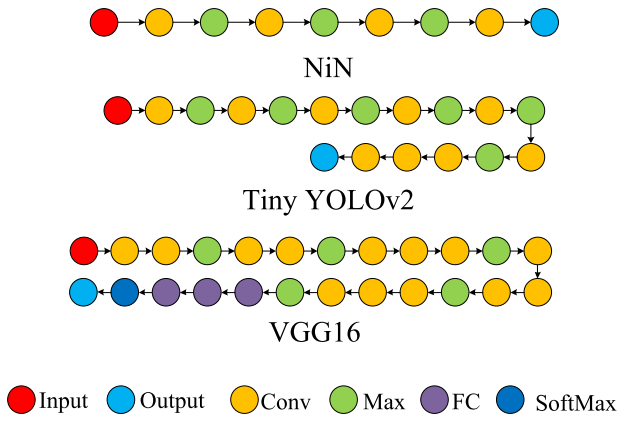


Fig. 10. The chain-topology DNN models.

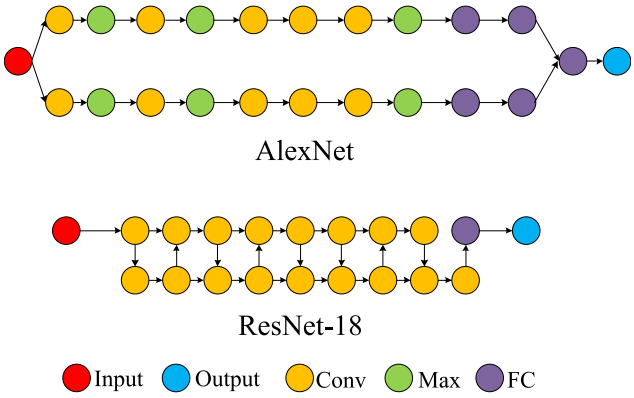


Fig. 11. The DAG-topology DNN models.

both topologies. For the chain topology, NiN, tiny YOlOv2 and VGG16, are well-known models used as benchmarks in this evaluation shown in Fig. 10. For the DAG topology, we employ AlexNet and ResNet-18 as the benchmarks shown in Fig. 11.

Evaluation Criteria. We compare DNN surgery against Edge-Only (i.e., executing the entire DNN on the edge), Cloud-Only (i.e., executing the entire DNN on the cloud), and a variant Neurosurgeon which is a partition strategy for chain-topology DNN. To evaluation Neurosurgeon's performance for DAG, we consider a variant Neurosurgeon, which first employs topological sorting method to transform the DAG topology to the chain topology, and then uses the original partition method. We use the Edge-Only method as the baseline, i.e., the performance is normalized to Edge-Only method.

We evaluate the latency and throughput of DNN surgery compared with Edge-Only, Cloud-Only and Neurosurgeon in Section VI-B. We also evaluate the impact of different types of wireless network to DNN surgery, and the impact of bandwidth on the selection of workload mode in Section VI-C. Finally, we evaluate the performance of DNN surgery under cost-constrained system.

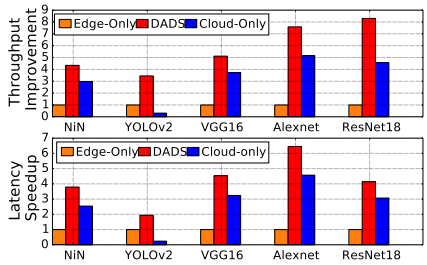


Fig. 12. Latency speedup and throughput gain achieved by DNN Surgery under light workload.

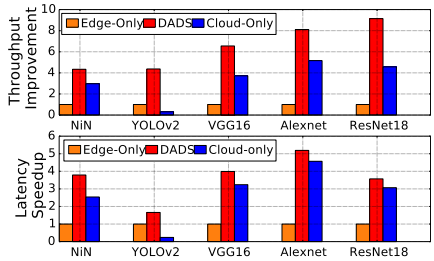


Fig. 13. Latency speedup and throughput gain achieved by DNN Surgery under heavy workload.

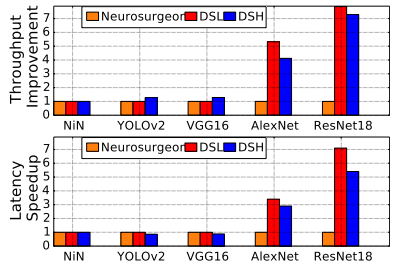


Fig. 14. Latency and throughput speedup achieved by DNN Surgery vs Neurosurgeon.

B. Performance Comparison

We first compare our DNN Surgery with Edge-Only, Cloud-Only and Neurosurgeon under light workload mode and heavy workload mode across the 5 DNN benchmarks in Figs. 12, 13, and 14. The results are normalized to Edge-Only method. We see that DNN Surgery achieves a higher latency speedup and throughput gain compared with other methods.

Comparing DNN Surgery With Edge-Only and Cloud-Only: DNN Surgery (90 ms) has a latency speedup of 1.91–6.45 times, 1.35–8.08 times compared with Edge-Only and Cloud-Only methods respectively under the light workload mode shown in the bottom graph of Fig. 12. DNN Surgery (11.11 FPS) has a throughput gain of 3.45–8.31 times, 1.46–11.13 times compared with Edge-Only and Cloud-Only methods respectively under the light workload mode shown in the upper graph of Fig. 12. This is because, Edge-Only method executes the entire DNN on the edge side, it avoids data transmission and ignores the weak computation capacity of edge side. Cloud-Only method ignores the effect of the transmission time. DNN Surgery considers both computation and transmission, and it makes a good tradeoff between them.

From Fig. 16, we can see that, for the heavy workload mode, DNN Surgery (110 ms) outperforms Edge-Only and Cloud-Only

Authorized licensed use limited to: KAUST. Downloaded on February 19,2024 at 07:16:23 UTC from IEEE Xplore. Restrictions apply.

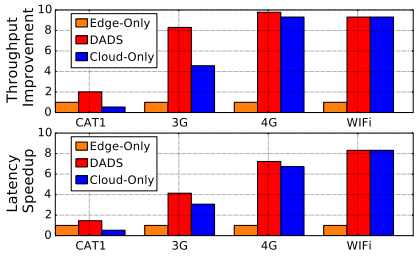


Fig. 15. Latency speedup and throughput gain achieved by DNN Surgery of different networks under light workload.

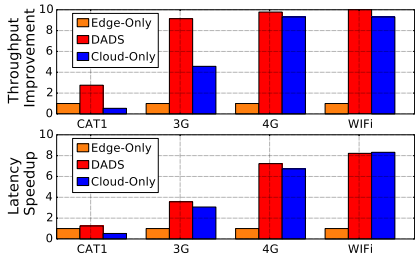


Fig. 16. Latency speedup and throughput gain achieved by DNN Surgery of different networks under heavy workload.

1.66–5.19 times and 1.07–6.92 times respectively in latency reduction, and DNN Surgery (9.09 FPS) outperforms Edge-Only and Cloud-Only 4.34–9.14 times and 1.46–14.10 times respectively in throughput gain. This further confirms that DNN Surgery significantly outperforms Edge-Only and Cloud-Only methods.

Comparing DNN Surgery With Neurosurgeon: Neurosurgeon can automatically partition DNN between the edge device and cloud at granularity of neural network layers, but it is only effective for chain topology.

From Fig. 14, we can see that, for the chain topology models, DNN Surgery and Neurosurgeon have the similar performance in latency and throughput for the light workload. While for the heavy workload, Neurosurgeon has a latency reduction of 16.28% and 13.64% than that of DNN Surgery for YOLOv2 and VGG16, however the throughput gain of DNN Surgery is 1.26 times and 1.27 times than that of Neurosurgeon under these two DNN models. This is because, for the heavy workload, the higher throughput is prior for DNN Surgery. We also can see that, for the heavy workload and NiN model, the latency and the throughput of Neurosurgeon and DNN Surgery are both the same. This is because for NiN model, DNN Surgery achieves the minimum max stage time when the latency is minimum.

For the DAG topology, we can observe that DNN Surgery outperforms Neurosurgeon significantly. For DAG topology models, DNN Surgery has a latency speedup 66%–86% and throughput gain of 76%–87% compared with Neurosurgeon. This observation validates the usefulness of DNN Surgery for DAG topology.

C. Network Variation

In this section, we evaluate how transmission network affects the performance of DNN Surgery using ResNet18 model. The sampling rate is 1 frame per second.

The Impact of Transmission Network Type: We first evaluate the performance of DNN Surgery, Edge-Only and Cloud-Only for ResNet18 model when using Cat1, 3G, 4G and WiFi as the communication network.

In Figs. 15 and 16, we show the latency speedup and the throughput gain achieved by DNN Surgery and Cloud-Only normalized to Edge-Only when using Cat1, 3G, 4G and WiFi for light and heavy workload respectively.

As shown in Fig. 15, when the workload is light and the edge device communicates with the cloud through Cat1, DNN Surgery achieves 1.46 times latency reduction and 2.03 times throughput gain compared with Edge-Only. When the network

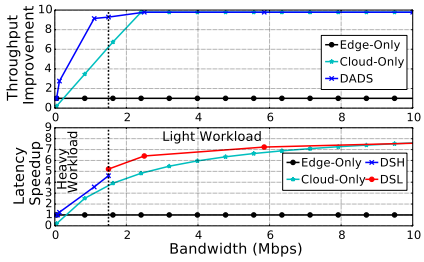


Fig. 17. Latency speedup and throughput gain achieved by DNN Surgery as a function of bandwidth.

changes to 3G, 4G and 5G the latency reduction and the throughput gain becomes more significant: 4.14 times and 8.3 times for 3G, 7.23 times and 9.78 times for 4G, 8.32 times and 9.31 times for WiFi respectively. When the communication link provides more bandwidth, DNN Surgery pushes larger portions of layers to the cloud to achieve better performance. We can also see that, compared with Cloud-Only, DNN Surgery achieves latency reduction of 64% for CAT1, 26% for 3G and 7% for 4G receptively, and throughput gain of 73% for CAT1, 45% for 3G and 4% for 4G. For WiFi, the performance of Cloud-Only is good enough, it has the same performance with DNN Surgery.

Edge-Only is only good for low data rate. Cloud-Only is only good for high data rate, DNN Surgery can be adaptive to a wide range of network setting.

The Impact of Bandwidth on Workload Mode Selection: In Fig. 17, we show the workload mode switch of DNN Surgery under different network bandwidth. We can see that when the available bandwidth is smaller than 1.51Mbps, DNN Surgery works at heavy workload mode, and the achieved latency speedup and throughput gain increase compared with Edge-Only. When the bandwidth is greater than 1.51Mbps, DNN Surgery works at light workload mode.

We also evaluate DNN Surgery's resilience to real-world measured wireless network variations. In Fig. 18, the top graph shows measured wireless bandwidth over a period of time. The bottom graph shows the latency speedup of DNN Surgery normalized to Edge-Only for ResNet18 model. We can see that DNN Surgery adjusts the partition strategy according to the bandwidth variance successfully. For example, when the bandwidth drops from 3.41Mbps to 2.15Mbps, DNN Surgery changes the partition from conv2 layer to conv3 layer. DNN Surgery changes the partition from conv3 layer to conv7 layer when bandwidth is smaller than 1.72Mbps.

Authorized licensed use limited to: KAUST. Downloaded on February 19,2024 at 07:16:23 UTC from IEEE Xplore. Restrictions apply.

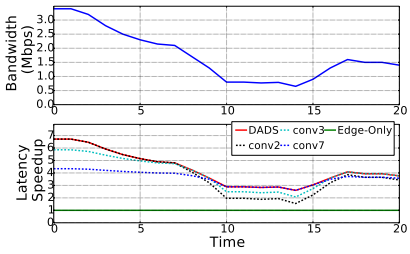


Fig. 18. The impact of network variance on DNN Surgery partition decision using Edge-Only as the baseline.

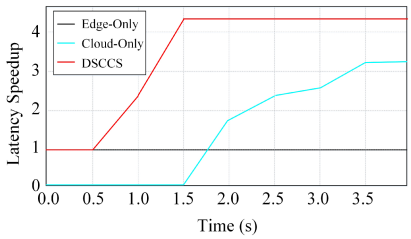


Fig. 19. Latency speedup achieved by DSCCS under the constrained resource of the cloud.

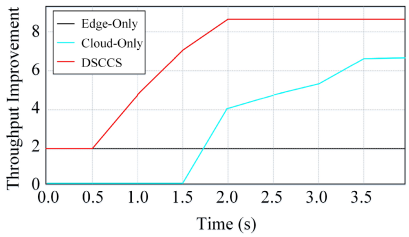


Fig. 20. Throughput gain achieved by DSCCS under the constrained resource of the cloud.

D. Performance Under Cost-Constrained System

In this section, we evaluate how the constrained resource of the cloud allocated for DNN inference affect the performance of DNN surgery using ResNet18 model under 3G network.

We show the latency speedup achieved by DSCCS under the constrained resource of the cloud in Fig. 19. We note that, when the processing time including communication time and cloud-processing time is smaller than 0.5 s, DSCCS has the same performance with Edge-Only approach. This is because DSCCS process the whole DNN inference on the edge as the allocated time from the cloud is too smaller. While the latency speedup can reach to 4.16 times when the allocated time from the cloud is bigger than 1.5s. We also can see that DSCCS can outperform Cloud-Only 1.36 times even when existing enough allocated resource from the cloud.

We also show the throughput gain achieved by our proposed DSCCS in Fig. 20. Compared with Edge-Only, the throughput improvement of DSCCS can reach up to 8.51 times. Compared with Cloud-Only, the throughput improvement of DSCCS is at least 1.33 times.

VII. Case Studies

In this section, we focus on applying our DNN surgery approaches to the real-world video analytics applications. We first present a case study that uses DNN Surgery algorithm of DNN surgery to support an indoor intrusion detection application. The indoor intrusion detection application has exclusive access to the resource of the edge device and the cloud. We then present a case study that uses DSCCS algorithm of DNN surgery to support an campus traffic monitor application. The campus traffic monitor application consists of multiple smart cameras sharing a server. These case studies show the end-to-end operations of DNN surgery in field.

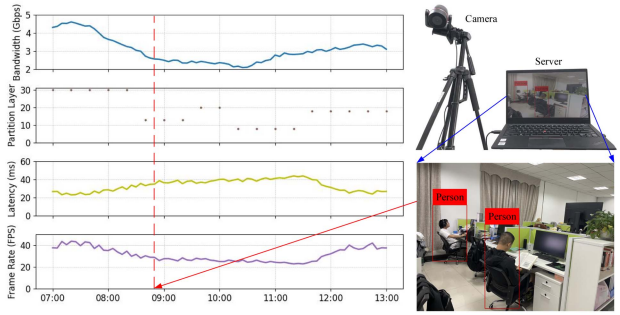


Fig. 21. The end-to-end operations of DNN surgery supporting an indoor intrusion detection application. (The photo has been informed and approved by the people in it and erased private information.)

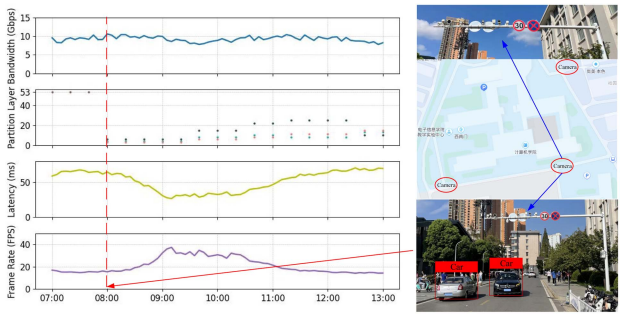


Fig. 22. The end-to-end operations of DNN surgery supporting an campus traffic monitor application. (The photo has been informed and approved by the vehicle owners in it and erased private information.)

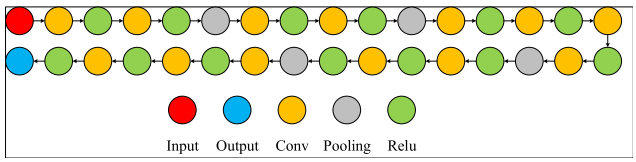


Fig. 23. Faster RCNN model.

A. Case Study: Indoor Intrusion Detection Application

We present a case study where we use our DNN Surgery algorithm of DNN surgery to support an indoor intrusion detection application that has been deployed in an intelligent computing laboratory in our department as shown in Fig. 21. The application applies a Hikvision intrusion detection application indoor. It has a pre-trained model Faster RCNN [21] to detect people in the monitroing area, as shown in the Fig. 23. The Hikvision camera has a customized SoC with 1 GB memory running on an embedded system. The server has an NVIDIA GeForce RTX 3080 Ti with 12GB memory running on Ubuntu Server 18.04. The camera connect to the server through the

Authorized licensed use limited to: KAUST. Downloaded on February 19,2024 at 07:16:23 UTC from IEEE Xplore. Restrictions apply.

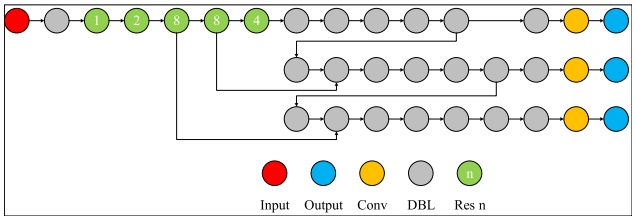


Fig. 24. YOLOv3 model.

campus WiFi with average 3 Gbps bandwidth. We ran the application supported by DNN Surgery algorithm for over 6 hours.

The bottom two graphs of Fig. 21 show the latency and the throughput over a period. We can see the throughput is holding at high throughput (above 20 FPS). We can also observe that the latency is stable low ranging from 25 ms to 40 ms. It illustrates that DNN surgery can provide consistent high service quality.

Fig. 21 also shows the end-to-end operations in the field of DNN surgery between 7:00 am to 1:00 pm. When there are successively more researchers in the lab, such at 9:00 am, the system changes from a light workload condition to a high workload state. In Fig. 21, the top graph shows the bandwidth over a period. The second graph shows the partition layer computed by DNN Surgery over a period of time. We can see that DNN surgery adjusts partition layer in runtime successfully. For example, at the time 8:50 am (9:00 am is the start time of office hours), as shown in the green dash line in Fig. 21, a burst of persons appear in the laboratory and connect their devices to the sharing campus WiFi, which reduces the bandwidth allocated to the surveillance camera from 4.3 Gbps at 7:00 am to 2.5 Gbps at 9:00 am. To adapt to the decreasing bandwidth, DNN surgery changes partition layer from the 30th layer to the 13th layer. When it comes to lunchtime, around noon, the bandwidth for the camera increases to 3.1 Gbps. DNN surgery detects such bandwidth change and adjusts the partition layer to the 18th layer to keep low latency and high throughput.

We further estimate the DNN surgery overhead by computing the number of floating-point operations (FLOPs) in this case. We find that DNN surgery has the computation of 15.8 MFLOPs, which is only 11.3% of MobileNet (140 MFLOPs) [22]. It takes only 1.7 ms to compute the dual computing resource allocation strategy, which occupied 4.8% of the total CPU time of the camera. In short, we believe that DNN surgery can successfully be deployed on laptops, mobile phones, or even on low-capacity cameras.

B. Case Study: Campus Traffic Monitor Application

We further integrate DNN surgery into a campus traffic monitor application. The campus traffic monitor application deploys 30 Hikvision radar cameras on important roads and intersections over a 2 km^2 campus to recognize the vehicles and detect their moving speed. The DNN model is a 53-layer YOLOv3 [23], as shown in the Fig. 24. The server has an Intel Core i9 11900K CPU with 120 GB memory running Ubuntu Server 18.04. Each camera has an ARM Cortex-A9 MPCore Soc with 8 GB memory running on an embedded system, connects to the server with a dedicated network to maintain the system's security and stability. All cameras share the resource of the server, thus resource allocated to each camera is limited. Thus, we uses DSCCS algorithm of DNN surgery to support such application. We also ran the application over 6 hours.

Fig. 22 shows the end-to-end operations in field of DNN surgery between 7:00 am to 13:00 pm. The system also quickly switches between light and heavy workload states during peak commuter periods on campus. In Fig. 22, the top graph shows the bandwidth over a period. The bandwidth is stable due to dedicated communication channel. The bottom three graphs show the partition layer computed by DSCCS algorithm, the acheived latency and thorughput, respectively, which validates DNN surgery adjusts partition layer in runtime successfully. Specifically, during the commuting time from 7:00 am to 9:00 am, the partition layer is 53th layer. The latency ranges from 67 ms to 30 ms. The throughput reaches up to 38 FPS. When it comes to working hours 9:00 am to 12:00 am, the partition layer changes to 29th layer, 40th layer and 44th layer. The changes are incurred by the dynamics in the number of vehicles at different periods, which leads to varied amount of DNN inference tasks resulting in various resource of the server allocated to the cameras.

In this scenario, we also calculate the FLOPs of DNN surgery to assess its overhead. We discover that DNN surgery computes 18.7 MFLOPs and the computational resource allocation approach takes just 1.9 ms, which is slight higher than that of the previous case. This is because the number of model layers in this case is larger than that of indoor intrusion detection application case. However, the overhead is acceptable since it occupied 5.6% of the total CPU time of the camera. This case illustrates that DNN surgery can be applied successfully on large-scale, multi-terminal IoT systems.

VIII. Related Work

Modification of DNN Models. To realize inference acceleration, one category of related work investigated how to modify DNN models for speedup [24]. For example, Microsoft and Google developed small-scale DNNs for speech recognition on mobile platforms by sacrificing the high prediction accuracy [25]. Han et al. [26] proposed generating alternative DNN models to trade off accuracy and performance/energy and choosing to execute either in the cloud or mobile. Gordon et al. [27] presented MorphNet, an approach to automate the design of neural network structures. Lane et al. [28] could scale down DNNs to run directly on a DSP only, offering energy efficiency and lower latency. Variani et al. [29] proposed deep models that are much smaller than normal and to be run on phones. Taylor et al. [30] allowed to use of a pool of DNNs, and the most effective one is selected to use at runtime.

There are optimization techniques, such as model compression and model quantization [31], that can reduce the NN model size and hence the model inference workload. Small models have also been developed: Google built small-scale DNNs for mobile platforms, MCDNN [32] develops alternative DNN models, etc. There are various optimization techniques to limit the number of frames supplied into model inference. Reducto [33] provides a lightweight filtering algorithm to filter out irrelevant frames. A region of targeted objects [34] was extracted based on

Authorized licensed use limited to: KAUST. Downloaded on February 19,2024 at 07:16:23 UTC from IEEE Xplore. Restrictions apply.

common-feature analysis. These techniques incur instruction-intensive computing workloads, which need to be considered in resource-constrained edge devices. Instead, it employs a full-scale deep model without sacrificing accuracy.

Computation Offloading. Research efforts focusing on offloading computation from the resource-constrained mobile to the powerful cloud will reduce inference time [35], [36]. Neurosurgeon [37] explores a computation offloading method for DNNs between the mobile device and the cloud server at layer granularity. However, Neurosurgeon is not applicable for the computation partition performed by DNN surgery for several reasons: 1) Neurosurgeon only handles chain-topology DNNs that are much easier to process. 2) Neurosurgeon can only handle one inference task without considering a sequence of tasks. The adaptation to network conditions was realized by DNN surgery. MAUI's [38] is an offloading framework that can determine where to execute functions (edge or cloud) of a program. However, it is not explicitly designed for DNN partitioning as the communication data volume between functions is small. Teerapittayanon et al. [39] proposed DDNN, a distributed deep neural network architecture that is distributed across computing hierarchies, consisting of the cloud, the edge, and end devices. DDNN aims at reducing the communication data size among devices for the given DNN. DNN surgery differs as it handles dynamic network conditions to reduce the inference latency (communication and computing latency) rather than communication overhead only.

From the perspective of offloading View, most works currently study task offloading local view assuming almost infinite resource at cloud [40], [41]. However, partitioning in the local view cannot be applied when the cloud has limited computation and communication resources. In this work, we study DNN inference partition in constrained computation in the cloud.

Hardware Acceleration. In the past years, we have seen a flourish in dedicated AI processors [42]. Commercial products emerge such as Google TPU, NVIDIA SCNN, etc. There are studies to increase the computation supplies with additional hardware, in particular GPUs, and develop algorithms to use GPUs to accelerate DNN inference, which is different from the scope of this paper. Hardware specialization is another method for inference acceleration. There are studies using FPGA for video analytics acceleration [43], such as recognition [44] and classification [45]. Then the programmable FPGAs can support partial reconfiguration [46], where a part of the FPGA can be reprogrammed while another part of the FPGA is being used [47]. Vanhoucke et al. [48] used fixed point arithmetic and SSSE3/SSE4 instructions on x86 machines to reduce the inference latency. DeepX [49] explored the opportunities to use mobile GPUs to enable real-time deep learning inferences. EC-DI investigates intelligent collaboration between the edge device and cloud for inference optimization and can be jointly applied with specialized hardware.

We would like to comment that there are studies from the perspective of cloud providers. The research question is how a cloud can support many DNN inference requests/queries with minimal cluster resource [15], [50]. For example, the system in [51] can summon 3,600 cores and start thousands of threads to perform video analytics. The system in [52] can analyze a large number of videos aggregated into a stream. DNN surgery looks from the perspective of an edge device. In DNN surgery, to reduce the delay experienced in the edge, the edge is responsible for managing the transmission delay; and partitioning a DNN inference task if necessary. Thus, both the problems we face and the solutions in DNN surgery we propose differ substantially from what the cloud provides.

IX. Conclusion

In this paper, we study DNN inference acceleration through collaborative edge-cloud computation. We propose a Dynamic Adaptive DNN surgery scheme that can partition DNN inference between the edge device and the cloud at the granularity of neural network layers, according to the dynamic network status. We present a comprehensive study of the partition problem under the lightly loaded condition and the heavily loaded condition. We also develop an optimal solution to the lightly loaded condition by converting it to the min-cut problem and design a 3-approximation ratio algorithm under the heavily loaded condition as the problem is NP-hard. We also study the partition problem under the cost-constraint problem with problem formulation, complexity analysis, and algorithm. We then implement a fully functioning system. Evaluations show that DNN surgery can effectively improve latency and throughput in order compared with executing the entire DNN on edge or on the cloud. Furthermore, depending on whether there are limited resources, it offers several use cases for DNN surgery. We anticipate that DNN surgery will function effectively on computers, smartphones, low-volume cameras, and large-scale, multi-terminal IoT devices.

References

[1] T. Zhao et al., "A survey of deep learning on mobile devices: Applications, optimizations, challenges, and research opportunities," *Proc. IEEE*, vol. 110, no. 3, pp. 334–354, Mar. 2022.
[2] D. Xu et al., "Edge intelligence: Empowering intelligence to the edge of network," *Proc. IEEE*, vol. 109, no. 11, pp. 1778–1837, Nov. 2021.
[3] C. Badue et al., "Self-driving cars: A survey," *Expert Syst. Appl.*, vol. 165, 2021, Art. no. 113816.
[4] J. Cao, K.-Y. Lam, L.-H. Lee, X. Liu, P. Hui, and X. Su, "Mobile augmented reality: User interfaces, frameworks, and intelligence," *ACM Comput. Surv.*, vol. 55, pp. 1–36, 2021.
[5] A. Grzywaczewski, "Training AI for self-driving vehicles: The challenge of scale," 2017. [Online]. Available: https://devblogs.nvidia.com/training-self-driving-vehicles-challenge-scale
[6] State of Mobile Networks: USA. Accessed: Jun., 2018. [Online]. Available: https://opensignal.com/reports/2017/08/usa/state-of-the-mobile-network
[7] J. Chen and X. Ran, "Deep learning with edge computing: A review," *Proc. IEEE*, vol. 107, no. 8, pp. 1655–1674, Aug. 2019.
[8] AWS DeepLens. Accessed: Jun., 2018. [Online]. Available: https://aws.amazon.com/deeplens
[9] J. Redmon and A. Farhadi, "YOLO9000: Better, faster, stronger," 2016, *arXiv:1612.08242*.
[10] L. Zeng, X. Chen, Z. Zhou, L. Yang, and J. Zhang, "CoEdge: Cooperative DNN inference with adaptive workload partitioning over heterogeneous edge devices," *IEEE/ACM Trans. Netw.*, vol. 29, no. 2, pp. 595–608, Apr. 2021.
[11] D. Raca, J. J. Quinlan, A. H. Zahran, and C. J. Sreenan, "Beyond throughput: A 4G LTE dataset with channel and context metrics," in *Proc. 9th ACM Multimedia Syst. Conf.*, Amsterdam, The Netherlands, 2018, pp. 460–465.
[12] M. Franceschinis, M. Mellia, M. Meo, and M. Munafo, "Measuring TCP over WiFi: A real case," in *Proc. 1st Workshop Wirel. Netw. Meas.*, Riva Del Garda, Italy, 2005.

Authorized licensed use limited to: KAUST. Downloaded on February 19,2024 at 07:16:23 UTC from IEEE Xplore. Restrictions apply.

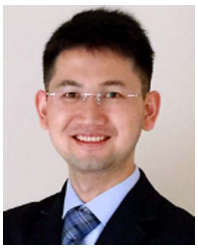
[13] P. Ballester and R. M. Araujo, "On the performance of GoogleNet and AlexNet applied to sketches," in *Proc. 30th AAAI Conf. Artif. Intell.*, 2016, pp. 1124–1128.
[14] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Las Vegas, Nevada, 2016, pp. 770–778.
[15] H. Zhang, G. Ananthanarayanan, P. Bodik, M. Philipose, P. Bahl, and M. J. Freedman, "Live video analytics at scale with approximation and delay-tolerance," in *Proc. USENIX Symp. Netw. Syst. Des. Implementation*, Boston, MA, 2017, pp. 377–392.
[16] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi, "Inception-V4, inception-ResNet and the impact of residual connections on learning," in *Proc. Conf. Assoc. Advance. Artif. Intell.*, San Francisco, CA, 2017, pp. 4278–4284.
[17] T. Stockhammer, "Dynamic adaptive streaming over HTTP: Standards and design principles," in *Proc. 2nd Annu. ACM Conf. Multimedia Syst.*, Santa Clara, CA, 2011, pp. 133–144.
[18] Y. Boykov and V. Kolmogorov, "An experimental comparison of min-cut/max- flow algorithms for energy minimization in vision," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 26, no. 9, pp. 1124–1137, Sep. 2004.
[19] E. L. Lawler and D. E. Wood, "Branch-and-bound methods: A survey," *Operations Res.*, vol. 14, no. 4, pp. 699–719, 1966.
[20] F. Yu et al., "BDD100K: A diverse driving dataset for heterogeneous multitask learning," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2020, pp. 2636–2645.
[21] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2015, pp. 91–99.
[22] Z. Qin, Z. Zhang, X. Chen, C. Wang, and Y. Peng, "Fd-Mobilenet: Improved Mobilenet with a fast downsampling strategy," in *Proc. 25th IEEE Int. Conf. Image Process.*, 2018, pp. 1363–1367.
[23] J. Choi, D. Chun, H. Kim, and H.-J. Lee, "Gaussian YOLOv3: An accurate and fast object detector using localization uncertainty for autonomous driving," in *Proc. IEEE/CVF Int. Conf. Comput. Vis.*, 2019, pp. 502–511.
[24] P. Guo, B. Hu, and W. Hu, "Mistify: Automating DNN model porting for on-device inference at the edge," in *Proc. 18th USENIX Symp. Netw. Syst. Des. Implementation*, 2021, pp. 705–719.
[25] X. Lei, A. W. Senior, A. Gruenstein, and J. Sorensen, "Accurate and compact large vocabulary speech recognition on mobile devices," in *Proc. INTERSPEECH Conf.*, 2013, pp. 662–665.
[26] S. Han, H. Shen, M. Philipose, S. Agarwal, A. Wolman, and A. Krishnamurthy, "MCDNN: An approximation-based execution framework for deep stream processing under resource constraints," in *Proc. ACM 14th Annu. Int. Conf. Mobile Syst. Appl. Serv.*, Singapore, 2016, pp. 123–136.
[27] A. Gordon et al., "MorphNet: Fast & simple resource-constrained structure learning of deep networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 1586–1595.
[28] N. D. Lane, P. Georgiev, and L. Qendro, "DeepEar: Robust smartphone audio sensing in unconstrained acoustic environments using deep learning," in *Proc. ACM Int. Joint Conf. Pervasive Ubiquitous Comput.*, Osaka, Japan, 2015, pp. 283–294.
[29] E. Variani, X. Lei, E. McDermott, I. L. Moreno, and J. Gonzalez-Dominguez, "Deep neural networks for small footprint text-dependent speaker verification," in *Proc. IEEE Int. Conf. Acoust. Speech Signal Process.*, Florence, Italy, 2014, pp. 4052–4056.
[30] B. Taylor, V. S. Marco, W. Wolff, Y. Elkhatib, and Z. Wang, "Adaptive selection of deep learning models on embedded systems," 2018, *arXiv: 1805.04252*.
[31] L. Deng, G. Li, S. Han, L. Shi, and Y. Xie, "Model compression and hardware acceleration for neural networks: A comprehensive survey," *Proc. IEEE*, vol. 108, no. 4, pp. 485–532, Apr. 2020.
[32] S. Han, H. Shen, M. Philipose, S. Agarwal, A. Wolman, and A. Krishnamurthy, "MCDNN: An approximation-based execution framework for deep stream processing under resource constraints," in *Proc. 14th Annu. Int. Conf. Mobile Syst. Appl. Serv.*, 2016, pp. 123–136.
[33] Y. Li, A. Padmanabhan, P. Zhao, Y. Wang, G. H. Xu, and R. Netravali, "Reducto: On-camera filtering for resource-efficient real-time video analytics," in *Proc. Annu. Conf. ACM Special Int. Group Data Commun. Appl. Technol. Archit. Protoc. Comput. Commun.*, 2020, pp. 359–376.
[34] H. Kuang, L. Chen, F. Gu, J. Chen, L. Chan, and H. Yan, "Combining region-of-interest extraction and image enhancement for nighttime vehicle detection," *IEEE Intell. Syst.*, vol. 31, no. 3, pp. 57–65, May/Jun. 2016.
[35] J. Lu et al., "A multi-task oriented framework for mobile computation offloading," *IEEE Trans. Cloud Comput.*, vol. 10, no. 1, pp. 187–201, Jan./Mar. 2022.
[36] L. Lin, X. Liao, H. Jin, and P. Li, "Computation offloading toward edge computing," *Proc. IEEE*, vol. 107, no. 8, pp. 1584–1607, Aug. 2019.
[37] Y. Kang et al., "Neurosurgeon: Collaborative intelligence between the cloud and mobile edge," in *Proc. ACM 22nd Int. Conf. Archit. Support Program. Lang. Operating Syst.*, Xi'an, China, 2017, pp. 615–629.
[38] E. Cuervo et al., "MAUI: Making smartphones last longer with code offload," in *Proc. ACM 8th Int. Conf. Mobile Syst. Appl. Serv.*, San Francisco, CA, 2010, pp. 49–62.
[39] S. Teerapittayanon, B. McDanel, and H. Kung, "Distributed deep neural networks over the cloud, the edge and end devices," in *Proc. IEEE 37th Int. Conf. Distrib. Comput. Syst.*, Atlanta, GA, 2017, pp. 328–339.
[40] M. Liu, Y. Li, Y. Zhao, H. Yang, and J. Zhang, "Adaptive DNN model partition and deployment in edge computing-enabled metro optical interconnection network," in *Proc. Opt. Fiber Commun. Conf.*, Optical Society of America, 2020, pp. Th2A–28.
[41] Z. Zhao, K. Wang, N. Ling, and G. Xing, "EdgeML: An AutoML framework for real-time deep learning on the edge," in *Proc. Int. Conf. Internet-of-Things Des. Implementation*, 2021, pp. 133–144.
[42] A. Auten, M. Tomei, and R. Kumar, "Hardware acceleration of graph neural networks," in *Proc. IEEE/ACM 57th Des. Automat. Conf.*, 2020, pp. 1–6.
[43] M. Owaida, G. Alonso, L. Fogliarini, A. Hock-Koon, and P.-E. Melet, "Lowering the latency of data processing pipelines through FPGA based hardware acceleration," *Proc. VLDB Endowment*, vol. 13, no. 1, pp. 71–85, 2019.
[44] S. Han et al., "ESE: Efficient speech recognition engine with sparse LSTM on FPGA," in *Proc. ACM/SIGDA Int. Symp. Field-Programmable Gate Arrays*, Monterey, CA,USA, 2017, pp. 75–84.
[45] S. Jiang, Z. Ma, X. Zeng, and other, "SCYLLA: QoE-aware continuous mobile vision with FPGA-based dynamic deep neural network reconfiguration," in *Proc. IEEE Conf. Comput. Commun.*, Virtual Event, 2020, pp. 1369–1378.
[46] J. Wang, Y. Kang, W. Wu, G. Xing, and L. Tu, "DUPRFloor: Dynamic modeling and floorplanning for partially reconfigurable FPGAs," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 40, no. 8, pp. 1613–1625, Aug. 2021.
[47] X. Wang, Y. Niu, F. Liu, and Z. Xu, "When FPGA meets cloud: A first look at performance," *IEEE Trans. Cloud Comput.*, vol. 10, no. 2, pp. 1344–1357, Apr./Jun. 2022.
[48] V. Vanhoucke, A. Senior, and M. Z. Mao, "Improving the speed of neural networks on CPUs," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, Granada, Spain, 2011.
[49] L. N. Huynh, R. K. Balan, and Y. Lee, "DeepSense: A GPU-based deep convolutional neural network framework on commodity mobile devices," in *Proc. ACM Workshop Wearable Syst. Appl.*, 2016, pp. 25–30.
[50] J. Jiang, G. Ananthanarayanan, P. Bodik, S. Sen, and I. Stoica, "Chameleon: Scalable adaptation of video analytics," in *Proc. Conf. ACM Special Int. Group Data Commun.*, Budapest, Hungary, 2018, pp. 253–266.
[51] S. Fouladi et al., "Encoding, fast and slow: Low-latency video processing using thousands of tiny threads," in *Proc. 14th USENIX Symp. Netw. Syst. Des. Implementation)*, Boston, MA., 2017, pp. 363–376.
[52] B. Zhang, X. Jin, S. Ratnasamy, J. Wawrzynek, and E. A. Lee, "Awstream: Adaptive wide-area streaming analytics," in *Proc. Conf. ACM Special Int. Group Data Commun.*, Budapest, Hungary, 2018, pp. 236–252.

Huanghuang Liang received the BS and MS degrees in automation engineering from the Anhui University of Technology in 2016 and the University of Electronic Science and Technology of China in 2019. He is currently working toward the PhD degree in computer science with Wuhan University. His research interests include cloud computing and Big Data systems.

Authorized licensed use limited to: KAUST. Downloaded on February 19,2024 at 07:16:23 UTC from IEEE Xplore. Restrictions apply.

Qianlong Sang received the BS degree in cyber science and engineering from Wuhan University in 2022. He is currently working toward the PhD degree in computer science with Wuhan University. His research interests include edge computing and Big Data systems.

Chuang Hu received the BS and MS degrees from Wuhan University in 2013 and 2016, and the PhD degree from the Hong Kong Polytechnic University in 2019. He is currently an associate researcher with the School of Computer Science at Wuhan University. His research interests include edge learning, federated learning/analytics, and distributed computing.



Dazhao Cheng (Senior Member, IEEE) received the BS and MS degrees in electrical engineering from the Hefei University of Technology in 2006 and the University of Science and Technology of China in 2009, and the PhD from the University of Colorado, Colorado Springs in 2016. He was an AP with the University of North Carolina at Charlotte in 2016-2020. He is currently a professor with the School of Computer Science, Wuhan University. His research interests include Big Data and cloud computing.

Xiaobo Zhou (Senior Member, IEEE) received the BS, MS, and the PhD degrees in computer science from Nanjing University, in 1994, 1997, and 2000, respectively. He was a professor with the Department of Computer Science, University of Colorado, Colorado Springs. He is currently a distinguished professor with the State Key Laboratory of Internet of Things for Smart City & the Department of Computer and Information Sciences, University of Macau. His research interests include distributed systems, cloud computing and datacenters, data parallel and distributed processing, and autonomic. He was the recipient of the NSF CAREER Award in 2009.

Dan Wang (Senior Member, IEEE) received the BS, MS, and the PhD degrees in computer science from Peking University, Case Western Reserve University, and Simon Fraser University, in 2000, 2004, and 2007, respectively. He is currently a professor with the Department of Computing at Hong Kong Polytechnic University. His research interests lie in networked systems, and recently in the inter-discipline domains of smart energy systems. He publishes in ACM SIGCOMM, ACM SIGMETRICS, IEEE INFOCOM and in inter-discipline conferences, such as ACM e-Energy, ACM Buildsys. He won the Best Paper Award of ACM e-Energy 2018 and the Best Paper Award of ACM Buildsys 2018. He is currently the steering committee chair of IEEE/ACM IWQoS and the steering committee chair of ACM e-Energy. He is an advisor of EMSD, the Hong Kong SAR government. He has extensive experiences in applying his research results to industry, including Huawei, IBM, Henderson, etc. He won the TechConnect Global Innovation Award in 2017.

Wei Bao (Senior Member, IEEE) received the BS degree in communications engineering from the Beijing University of Posts and Telecommunications in 2009, the MS degree in electrical and computer engineering from the University of British Columbia in 2011, and the PhD degree in electrical and computer engineering from the University of Toronton in 2016. He is currently a senior lecturer with the School of Computer Science, University of Sydney. His research interests include network science, with particular emphasis on Internet of things, mobile computing, and edge computing.

Yu Wang (Fellow, IEEE) received the BS and MS degrees in computer science from Tsinghua University in 1998 and 2000, and the PhD degree in computer science from Illinois Institute of Technology in 2004. He is currently a professor with the Department of Computer and Information Sciences, Temple University, Philadelphia, PA, USA. He has published more than 200 papers in peer-reviewed journals and conferences. His research interests include wireless networks, smart sensing, and mobile computing. He is a recipient of the Ralph E. Powe Junior Faculty Enhancement Awards from Oak Ridge Associated Universities in 2006, the Outstanding Faculty Research Award from the College of Computing and Informatics at the University of North Carolina at Charlotte in 2008, and the ACM distinguished member in 2020. He has served as the general Cchair, the program Cchair, and the program committee member for many international conferences, such as IEEE IPCCC, ACM MobiHoc, IEEE INFOCOM, IEEE GLOBECOM, and IEEE ICC, and served as an editorial board member for several international journals, including *IEEE Transactions on Parallel and Distributed Systems*.

Authorized licensed use limited to: KAUST. Downloaded on February 19,2024 at 07:16:23 UTC from IEEE Xplore. Restrictions apply.