Ultra-Low-Latency Distributed Deep Neural Network over Hierarchical Mobile Networks

Jen-I Chang[§], Jian-Jhih Kuo[†], Chi-Han Lin[§], Wen-Tsuen Chen[§], and Jang-Ping Sheu[§]
[§]Dept. of Computer Science, National Tsing Hua University, Hsinchu, Taiwan
[†]Dept. of Computer Science & Information Engineering, National Chung Cheng University, Chiayi, Taiwan
E-mail: cozy46th@gmail.com, lajacky@cs.ccu.edu.tw, finalspaceman@gmail.com, {wtchen,sheujp}@cs.nthu.edu.tw

***Abstract*—Recently, the notions of partitioning the Deep Neural Network (DNN) model over the multi-level computing units and making a fast inference with the early-inference technique have been proposed to shorten the inference time. Such computing units form a hierarchical mobile network to provide locality-aware computation, and the early-inference technique allows the prediction results to early exit the model with a probability. However, an inadequate model partition and misapply early inference may prolong response time. Previous studies focus on the classifier design for early inference, and thus, the optimal model partition with classifier deployment has not been explored. In this paper, we study DEMAND-OPE to consider response time and throughput. We first design the COLT for the simplified DEMAND-OPE without Optional Exit Points (DEMAND) to carefully balance the computing time and data transfer time. Then, an extension termed COLT-OPE is developed to achieve the lower response time. Simulation results show that our algorithms (COLT-OPE) outperform previous methods by 200%.**

***Index Terms*—Hierarchical mobile network, deep neural network, model partition, model deployment, early inference**

I. Introduction

Recently, the demand of Artificial Intelligence (AI) grows rapidly with the development of smart devices. Many applications, such as voice/image recognition and Augmented/Virtual Reality (AR/VR), benefit from AI models to improve system accuracy. In practice, AI model usually performs on either the User Equipment (UE) or the cloud. However, UEs usually rely on a simple model (e.g., Support Vector Machine) due to their limited capability, which incurs poor accuracy. By contrast, the plentiful resources of cloud facilitate the handling of more complex models like Deep Neural Network (DNN). Thus, due to powerful computing capacity, the cloud is still the main platform for marketing analysis or scientific research [1].

However, the paradigm will run into a serious problem in the near future. The first is the considerable connections from smart devices to the cloud. Cisco predicts more than 12 billion smart devices will access the network services in 2022 [2]. In addition, the services like video surveillance require real-time image recognition, which may overwhelm backhaul network and consume enormous computing capacity if videos are sent to the remote cloud for process. Another is the progression towards much *deeper* DNN structures. Model depth grows rapidly in recent years, such as 8-layer AlexNet, 19-layer VGGNet, and 152-layer ResNet in 2012, 2014, and 2015. A single computer may no longer meet the requirement and the response time may also be prolonged since every input has to experience more computations to make an inference.

To remedy the issues, we take advantage of the following technologies in this work: 1) Fog/Mobile Edge Computing (FC/MEC). The concept is to build multi-level computing units from the UEs to the cloud, which forms *hierarchical mobile networks*, to perform locality-aware computing to relieve backhaul load and reduce response time [3]–[5]. 2) Model partition. The idea is to partition a model into *stages*. Each stage contains a subset of consecutive layers and thus can be performed on different levels in parallel [6]. 3) Early inference. By attaching extra classifiers (i.e., *exit points*) after some layers, the result can be determined earlier with a probability to greatly shorten the expected response time [6], [7].

Nevertheless, a naïve model partition may incur long data transfer time between the levels and such data transfer will become the throughput bottleneck since the computing units have to wait for it. The rising *completion time* (i.e., the time to make an inference for an input) may also violate the response time requirement for delay-sensitive applications (e.g., 20 ms for AR/VR to avoid motion sickness [8]). Besides, the less-efficient exit points may cause the redundant computing and exacerbate the high response time. However, the *completion-time-aware deployment for DNN models* has not been explored to divide a model into stages and deploy them with a proper number and places of exit points over the hierarchical mobile network to achieve the low response time and high throughput.

In this paper, therefore, we make the first attempt to explore completion-time-aware deployment for DNN models. It raises the new research challenges. 1) *Proximity trade-off*. Computing the layers at the level near to the users can save more data transfer time. However, computing excessive layers at the levels near users may induce overdone computation time since these levels usually have weaker computing units. 2) *Suitable stage number*. The time of each stage must be constrained to guarantee the throughput because the bottleneck occurs at the slowest stage. Partitioning the model into the more stages enjoys a smaller stage time but may induce larger completion time since data transfer is needed between two consecutive stages. 3) *Optional exit points*. Intuitively, the addition of exit points is expected to reduce completion time. However, adding an exit point requires extra computing independent to the original model. If an exit point has a low probability to make an early inference, most data still have to deliver to later layers, which wastes the extra computation and makes completion time longer. Thus, whether and where to deploy exit points should be jointly decided, which leads to a challenge.

To address the above challenges, we formulate an optimization problem termed DEMAND-OPE to find the completion-time-aware deployment over the hierarchical mobile network.

It aims to minimize the expected completion time, whereas the *consecutive partition constraint* (i.e., each stage is responsible for a subset of consecutive layers) and the *stage time constraint* (i.e., the maximum stage time limit for throughput) are ensured. To explore the intrinsic properties of DEMAND-OPE, we first investigate DEMAND (i.e., DEMAND-OPE without Optional Exit Points). We design an algorithm named COLT to obtain the optimal solution for DEMAND by dynamic programming. COLT carefully examines every sub-problem to address *proximity trade-off* and *suitable stage number*. Afterward, we design COLT-OPE to further shorten the expected completion time by properly deploying *optional exit points*.

II. Related Work

A. Fog Computing and Edge Computing

FC and MEC have emerged to deal with the rising demands of the smart devices with novel applications such as VR/AR to provide proximity computing to reduce response time [3]–[5]. Xu *et al.* investigate dynamic offloading of popular services to the resource-limited edge server to remedy backhaul network overhead [9]. Wang *et al.* study the deployment of VR service proxy on the network edge to process user data and avoid massive data exchange between users [10]. Wang *et al.* address the trade-off between the local and global update across multiple edge nodes to reduce the loss function for training [11]. Ran *et al.* design an offloading mechanism for inference tasks based on the demand for accuracy and response time [12]. However, none of them considers the partition and deployment of AI models over the hierarchical mobile network to jointly minimize the completion time and limit the stage time.

B. Distributed Deep Neural Networks

Most research of decentralized DNN aims to accelerate the training. Dean *et al.* propose DistBelief to speed up the training a large neural network by using thousands of CPUs [13]. Iandola *et al.* extend the concept to exploit GPU clusters to accelerate training [14]. The decentralized training schemes evolve in two major ways, Module Parallelism and Data Parallelism [15]. The former employs the same training data set for all machines but each machine trains a part of the entire model, and the latter makes all machines with the complete model but assign different training data to each machine. Teerapittayanon *et al.* scale out AI models geographically [6]. However, none of them optimizes the partition and deployment of AI models to facilitate inference throughput, thereby motivating this paper.

III. The DEMAND-OPE Problem

We consider a mobile network with n-level computing units [3]. The computing unit in level 1 (i.e., the UE) usually (but not always) has the weakest computing and bandwidth capacity while the one in level n (i.e., the cloud) has the most plentiful capacity. To provide the service (e.g., real-time image recognition for video surveillance), the UE (e.g., camera) *continuously* sends data to the m-layer DNN model for processing [8]. Each layer in the model has different computing and bandwidth overhead, e.g., Convolutional Layers (CL) generates more data but demands less computing than Fully-connected Layers (FL). To guarantee the throughput, the model will be divided into multiple *stages*. Each stage will be assigned to a different level. The computing unit at each level will process the data of its assigned stage and then propagate the intermediate data to a higher level. All the stages will process *in parallel* to avoid idling. Each *stage time* is the time to process an input of the stage, which depends on the stage's layers and computing unit. Therefore, the slowest stage directly throttles the throughput. In other words, the longer the slowest stage, the worse the throughput. The maximum stage time is the *reciprocal* of throughput (e.g., 16-ms stage time for 60-fps image recognition [8]). Thus, we have to limit the maximum stage time to ensure high throughput. Also, every layer can add an exit point to early determine the results with an *exit probability* to shorten expected completion time [7]. We aim to deploy the layers and exit points to minimize the expected completion time for input data getting the final result.

Let binary variable x_{ij} denote whether layer j is deployed at level i. Constraint (1) ensures each layer j must be deployed.

$$\sum_{i\in[1,n]} x_{ij} = 1, \qquad \forall j \in [1,m] \tag{1}$$

Moreover, each layer (except 1^{st} layer) is deployed either at the same level of the previous layer or at a higher level due to the *consecutive partition constraint*. We have constraint (2).

$$\sum_{i'\in[1,i]} x_{i'j_1} - x_{i'j_2} \geq 0, \ \forall i \in [1,n], 1 \leq j_1 < j_2 \leq m \tag{2}$$

Let $c(i)$ and $b(i,i+1)$ denote the computing capacity of each level i and the bandwidth capacity between level i and $i+1$, and $f(j)$ and $s(j)$ denote the computing overhead and bandwidth overhead generated by layer j, where $j \in [0,m]$. Note that layer 0 indicates the input layer with data size $s(0)$ and computing overhead $f(0) = 0$, and $s(m) = 0$ since no layer exists after layer m. Hence, the computing unit of level i requires computing time $d_i^f = \frac{\sum_{j\in[1,m]} x_{ij} f(j)}{c(i)}$ to process the layers assigned to it. Besides, let $r(j) = \frac{s(j)}{s(j-1)}$ denotes *data size change ratio* through layer j since the input and output data size of layer j is $s(j-1)$ and $s(j)$, respectively. Thus, the size of data transferred from level i to level $i+1$ is $t_i = t_{i-1}\Pi_{j\in[1,m]}(1 + x_{ij} \cdot (r(j) - 1))$, where $t_0 = s(0)$. The data transfer time from level i to level $i+1$ is $d_i^s = \frac{t_i}{b(i,i+1)}$.

Let binary variable y_{ij} denote whether the exit point of layer j is deployed at level i, and let $p(j)$ denote the *exit probability* of obtaining an early result with high confidence at the exit point of layer j. In other words, the *progressing probability* of the data from layer j to layer $j+1$ is $100\% - p(j)$ if the exit point of layer j is deployed. Note that the exit probability of exit point of layer m is 100% (i.e., $p(m) = 100\%$), and it must be deployed to ensure an output. That is,

$$\sum_{i\in[1,n]} y_{ij} \begin{cases} \leq 1, & \textbf{if } j \in [1, m-1] \\ = 1, & \textbf{else if } j = m \end{cases} \tag{3}$$

Following [6], an exit point is put on the last layer deployed at a level to achieve a low completion time and save backhaul communications. Therefore, we have constraints (4)−(6).

$$y_{ij} \leq x_{ij}, \qquad \forall i \in [1,n], \forall j \in [1,m] \tag{4}$$

$$\sum_{j\in[1,m]} y_{ij} \le 1, \qquad \forall i \in [1,n] \tag{5}$$

$$x_{ij_1} x_{ij_2}(y_{ij_2} - y_{ij_1}) \ge 0,\ \forall i \in [1,n], 1 \le j_1 < j_2 \le m \tag{6}$$

The exit point of layer j also causes the computing overhead denoted by $e(j)$. Then, the computing unit of level i needs extra computing time $d_i^e = \frac{\sum_{j\in[1,m]} y_{ij} e(j)}{c(i)}$. Recall that the stage time and the data transfer time between two levels must be bounded to guarantee the throughput. Therefore, assume that the maximum time for computing or data transfer at each level is Δ. The *stage time constraints* are

$$d_i^f + d_i^e \le \Delta, \qquad \forall i \in [1,n] \tag{7}$$

$$d_i^s \le \Delta, \qquad \forall i \in [1,n] \tag{8}$$

Next, the *progressing proportion* of data that will pass level i (i.e., deliver to level $i+1$) without early exiting is $\overline{\mathbb{P}}_i = \Pi_{i'\in[1,i]}\Pi_{j\in[1,m]}(1 - y_{i'j} \cdot p(j))$. Thus the *exit proportion* of data that will be early determined at each level i is

$$\mathbb{P}_i = \begin{cases} \sum_{j\in[1,m]} y_{ij} \cdot p(j) & \textbf{if } i = 1 \\ \overline{\mathbb{P}}_{i-1}(\sum_{j\in[1,m]} y_{ij} \cdot p(j)) & \textbf{otherwise} \end{cases} \tag{9}$$

We mathematically define this problem as follows.

Definition 1. Given an n-level network and an m-layer DNN, the Completion-Time-Aware **D**istributed D**e**ep Neural Network Deploy**m**ent Problem with Constr**a**i**n**e**d** Stage Time and **Op**tional **E**xit Points (DEMAND-OPE) finds the partition and deployment of layers as well as the suitable number and positions of exit points on the computing units at levels such that the above constraints (1)–(9) are satisfied and minimize the expected completion time,[i] i.e.,

$$\text{minimize} \sum_{i\in[1,n]} \mathbb{P}_i\Big(\sum_{i'\in[1,i]} (d_{i'}^f + d_{i'}^e) + \sum_{i'\in[1,i-1]} d_{i'}^s\Big) \tag{10}$$

Example 1. This example shows the effect of model partition and layer deployment with exit points on completion time. Tables I and II summarize the model and network information and Fig. 1 shows the results of different solutions, where $\Delta = 200$. Only-UE deploys all layers at level 1 and takes time $(50+60+30+250+150)/5 = 108$. Only-Cloud deploys all layers at level 3 and takes time $540/20 = 27$ and $100/2 + 100/5 = 70$ for computing and data transfer, and total time is $27+70 = 97$. Similarly, the optimum without early exit point spends time 78.5 and reduces 27% and 19% of completion time, where the subsets of layers $\{1,2\}$ and $\{3,4,5\}$ are deployed at levels 1 and 3. By contrast, the optimum with early exit points moves layer 3 to level 2 and puts early exit points of layers 2 and 3 at levels 1 and 2. The expected completion time consists of $95\% \times 180/5 = 34.2$ at level 1, $5\% \times 50\% \times (180/5 + 50/2 + 90/10) = 1.75$ at level 2, and $5\% \times 50\% \times 100\% \times (180/5 + 50/2 + 90/10 + 50/5 + 400/20) = 2.5$ at level 3. Therefore, it needs time $34.2 + 1.75 + 2.5 = 38.45$. The early exit points further reduce 64% and 60% of expected completion time.

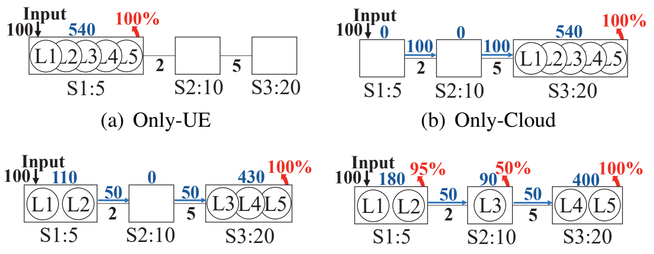
[i]Note that the time of transferring the inference result to the UE is ignored in our model since the output size is negligible compared with the input size.

TABLE I
MODEL INFORMATION

j	0	1	2	3	4	5
$f(j)$	0	50	60	30	250	150
$e(j)$	0	80	70	60	50	0
$s(j)$	100	500	50	50	10	0
$p(j)$	0%	40%	95%	50%	70%	100%

TABLE II
NETWORK INFORMATION

i	$c(i)$	$b(i,i+1)$
1	5	2
2	10	5
3	20	-



(a) Only-UE (b) Only-Cloud (c) Optimum w/o early exit point (d) Optimum with early exit points

Fig. 1. Example of different solutions for DEMAND-OPE

IV. Algorithm Design

The naïve approach for DEMAND-OPE considers the two traditional solutions that deploy all the layers only in the smart device (i.e., Only-UE) and only in the cloud (i.e., Only-Cloud), respectively, and then chooses the better one. However, this approach does not divide the model and offload the overhead to the multi-level computing units. Thus, it misses opportunities to utilize the plentiful capacities. Moreover, no early exit point is well exploited to reduce expected completion time, which degrades real-time user experience.

In the following, to solve DEMAND, we first design an algorithm named COLT to carefully address *proximity trade-off* and *suitable stage number* to obtain the minimum completion time. Then, we extend COLT to exploit the *optional exit points* (i.e., COLT-OPE) to reduce the *expected* completion time.

A. *Completion-Time-Aware Layer Deployment Algorithm*

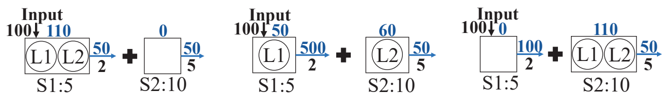
We first propose **Co**mpletion-Time-Aware **L**ayer Deployment Algorithm (COLT) to carefully address the first two challenges of DEMAND (i.e., DEMAND-OPE without Optional Exit Points). COLT exploits dynamic programming by first computing and recording the optimal solution of each smaller sub-problem in DEMAND, and then reusing these solutions to iteratively solve a larger sub-problem. For each sub-problem, COLT obtains the optimal solution by effectively combining 1) each one related previous solution that deploys a specific number of layers on fewer levels with 2) one extra higher level that supports the remaining layers to efficiently explore different combinations. That is, COLT carefully examines every possible combination to trade off the time for computing and data transfer (i.e., *proximity trade-off*). Meanwhile, to achieve the *suitable stage number*, COLT selects the combination that divides the model into the length-bounded stages.

More specifically, let $T(i,j)$ denote the minimum completion time for the consecutive layers $[1,j]$ deployed on the computing units in consecutive levels $[1,i]$, where $j \in [0,m]$ and $i \in [1,n]$. It can be envisaged that the solution of $T(i,j)$ must deploy a specific number of consecutive layers (i.e., layers 1 to some j') at levels $[1,i-1]$ and deploy the rest of layers (i.e., layers $[j'+1,j]$) at level i. To calculate the new

TABLE III
EXAMPLE OF COLT

$T(1,0)$	$T(1,1)$	$T(1,2)$	$T(1,3)$	$T(1,4)$	$T(1,5)$	-
50	no sol.	47	53	83	108	-

$T(2,0)$	$T(2,1)$	$T(2,2)$	$T(2,3)$	$T(2,4)$	$T(2,5)$	$T(3,5)$
70	155	57	60	77	90	78.5



(a) Candidate 1 (b) Candidate 2 (c) Candidate 3

Fig. 2. Candidate solution for $T(2,2)$ of COLT

completion time, let $D(i,[j_1,j_2])$ denote the total time for the computing unit at level i to process consecutive layers $[j_1,j_2]$ and transfer the data generated by layer j_2 to level $i+1$. Moreover, DEMAND forbids any stage violating the stage time bound Δ. Let $f(j_1,j_2)$ denote the total computing overhead of layers $[j_1,j_2]$, that is, $f(j_1,j_2)=\sum_{j\in[j_1,j_2]} f(j)$. Remind that $f(j_1,j_2)=0$ if $j_1>j_2$. Therefore, $D(i,[j_1,j_2])=$

$$\begin{cases} \text{no solution}, \textbf{if } \max\{\frac{f(j_1,j_2)}{c(i)}, \frac{s(j_2)}{b(i,i+1)}\}>\Delta \\ \frac{f(j_1,j_2)}{c(i)}+\frac{s(j_2)}{b(i,i+1)}, \textbf{otherwise} \end{cases} \tag{11}$$

COLT then derives $T(i,j)$ by examining each pair of $T(i-1,j')$ and $D(i,[j'+1,j])$ for $j'\in[0,j]$ since layer j' can be any layer among layers $[1,j]$, The recursive relation $T(i,j)=$

$$\begin{cases} D(1,[1,j]), \textbf{if } i=1 \\ \min_{j'\in[0,j]}\{T(i-1,j')+D(i,[j'+1,j])\}, \textbf{otherwise}. \end{cases} \tag{12}$$

Based on (12), COLT achieves the minimum completion time as follows. First, for the base cases $T(1,j)$ for $j\in[0,m]$, COLT derives the minimum completion time by calculating $D(1,[1,j])$. Next, COLT exploits the recurrence to compute the case $T(i,j)$ for $i\in[2,n-1]$ and $j\in[0,m]$. Finally, COLT computes and returns the solution $T(n,m)$.

Example 2. Following Example 1, Table III summarizes the derivation of $T(3,5)$. Take the calculation of $T(2,2)$ for example. COLT obtains three candidate solutions as shown in Figs. 2(a), 2(b), and 2(c), by combining $T(1,2)$, $T(1,1)$, and $T(1,0)$ with $D(2,[3,2])$, $D(2,[2,2])$, and $D(2,[1,2])$, respectively, based on the recursive function (12). Then, $T(1,2)=D(1,[1,2])=\frac{f(1,2)}{5}+\frac{s(2)}{2}=47$ and $T(1,0)=D(1,[1,0])=50$ at level 1, whereas $T(1,1)=D(1,[1,1])$ has no solution since $\max\{\frac{50}{5},\frac{500}{2}\}>\Delta=200$. Thus, computing $D(2,[2,2])$ is not necessary any more. Next, $D(2,[3,2])$ and $D(2,[1,2])$ require time $\frac{s(2)}{b(2,3)}=10$ and 21 at level 2. Overall, they spend total time $47+10=57$ and $50+21=71$, and thus $T(2,2)=\min\{57,71\}=57=T(1,2)+D(2,[3,2])$. Similarly, $T(3,5)=T(2,2)+D(3,[3,5])=78.5$.

Time Complexity. The overall time complexity of COLT is $O(nm^2)$. Due to the page limit, the details are omitted.

B. *COLT with **Op**tional **E**xit Points*

COLT-OPE extends COLT to deploy a suitable number of exit points to reduce the expected completion time. Intuitively, the solution of sub-problem jointly minimizing the progressing proportion of data and the expected completion time is the best choice for reuse. However, it is difficult to have both advantages at the same time since an exit point can help reduce the progressing proportion of data but prolong the completion time of data that has to deliver to the next level. Therefore, we prefer the solution with a lower progressing proportion of a sub-problem to form the solution of a large sub-problem if the completion time for the rest of layers is longer. Otherwise, the one with a shorter expected completion time is desired.

To this end, COLT-OPE adds the two factors to the recursive function, 1) the number of deployed exit points and 2) the progressing proportion of data. Thus, it computes and records the best deployment with the minimum expected completion time to meet each different value of the progressing proportion of data. It then selects the best one among all combinations built based on the previous solutions to balance the effects of the progressing proportion of data and the expected completion time. However, the number of possible progressing proportions COLT-OPE has to consider may grow exponentially as the numbers of layers increases. Thus, we exploit a novel random sampling technique to reduce the number of progressing proportions and approximate the optimum in polynomial time.

Specifically, both $D(i,[j_1,j_2])$ and $T(i,j)$ are extended to $\mathbb{D}(i,[j_1,j_2],\kappa,\rho)$ and $\mathbb{T}(i,j,k,\mathcal{P})$ to have a 3-tuple output. Note that $\mathbb{D}(i,[j_1,j_2],\kappa,\rho)$ bounds the progressing probability of data from level i to $i+1$ *within* ρ. The binary parameter $\kappa\in\{0,1\}$ then indicates whether layer j_2's exit point is deployed at level i (i.e., $\kappa=1$ *iff* j_2's exit point is deployed). The 1st tuple $\mathbb{D}_1(i,[j_1,j_2],\kappa,\rho)$ is the computing time for the layers $[j_1,j_2]$ with/without j_2's exit point (i.e., $\kappa\in\{0,1\}$) at level i. The 2nd tuple $\mathbb{D}_2(i,[j_1,j_2],\kappa,\rho)$ is the total time to compute the layers with/without j_2's exit point (i.e., $\mathbb{D}_1(i,[j_1,j_2],\kappa,\rho)$) and transfer the data of j_2 from level i to $i+1$. The 3rd tuple $\mathbb{D}_3(i,[j_1,j_2],\kappa,\rho)$ is the progressing probability of layer j_2 based on whether the exit point is deployed (i.e., $100\%-p(j_2)\cdot\kappa$). Then, $\mathbb{T}(i,j,k,\mathcal{P})$ denotes the best choice for the layers $[1,j]$ and $k\in[0,i]$ exit points deployed at levels $[1,i]$ with the progressing proportion of data through level i of *at most* $\mathcal{P}$. The 1st tuple $\mathbb{T}_1(i,j,k,\mathcal{P})$ is the expected completion time for the proportion of data that can early exit before (including) level i. The 2nd tuple $\mathbb{T}_2(i,j,k,\mathcal{P})$ is the completion time of data through level i (i.e., cannot exit before level i). The 3rd tuple $\mathbb{T}_3(i,j,k,\mathcal{P})$ is the progressing proportion of data through level i. Note that $\mathbb{T}(i,j,k,\mathcal{P})$ has no solution if $i<k$.

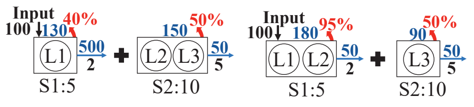
Computing $\mathbb{T}(i,j,k,\mathcal{P})$ has to reference each previous $\mathbb{T}(i-1,j',k',\mathcal{P}')$ with $\mathbb{D}(i,[j'+1,j],k-k',\frac{\mathcal{P}}{\mathcal{P}'})$, where $j'\in[0,j]$, $k'\in[k-1,k]$ and $\mathcal{P}'\in[0,1]$. Then it chooses the best combination of previous $\mathbb{T}(\cdot)$ with $\mathbb{D}(\cdot)$ that meets the progressing proportion bound Δ and has the smallest expected completion time for all data before (including) level i, i.e.,

$$\mathbb{T}_1(\cdot)+(\mathbb{T}_2(\cdot)+\mathbb{D}_1(\cdot))\mathbb{T}_3(\cdot)(100\%-\mathbb{D}_3(\cdot))+ \\ (\mathbb{T}_2(\cdot)+\mathbb{D}_2(\cdot))\times\mathbb{T}_3(\cdot)\mathbb{D}_3(\cdot) \tag{13}$$

Then, the 3 tuples of $\mathbb{T}(i,j,k,\mathcal{P})$ are set to $\mathbb{T}_1(\cdot)+(\mathbb{T}_2(\cdot)+\mathbb{D}_1(\cdot))\mathbb{T}_3(\cdot)(1-\mathbb{D}_3(\cdot))$, $\mathbb{T}_2(\cdot)+\mathbb{D}_2(\cdot)$, and $\mathbb{T}_3(\cdot)\mathbb{D}_3(\cdot)$. Due to the similarity, the details of recursive function is omitted.

TABLE IV
EXAMPLE OF COLT-OPE

...	$\mathbb{T}(1,2,1,0.09)$	$\mathbb{T}(2,3,2,0.09)$	...
...	$(34.2,61,0.05)$	$(35.95,80,0.025)$	...
$\mathbb{T}(3,5,1,0)$	$\mathbb{T}(3,5,2,0)$	$\mathbb{T}(3,5,3,0)$	T(3,5)
$(78.5,78.5,0.00)$	$(38.825,92.5,0.00)$	$(38.45,100,0.00)$	38.45



(a) Candidate 1 (b) Candidate 2

Fig. 3. Candidate solution for $\mathbb{T}(2,3,2,0.09)$ of COLT-OPE

Therefore, the optimal solution $T(i,j)$ of DEMAND-OPE can be obtained by calculating $\min_{k\in[1,n]}\{\mathbb{T}(n,m,k,0)\}$. However, the number of possible progressing proportions (i.e., argument $\mathcal{P}$) is at most 2^m since at most m exit points with different exit probability (i.e., $\{p(j)|\forall j \in [1,m]\}$) can be selected. It implies the number of states in dynamic programming may be exponential. Thus, we introduce a novel *random sampling technique* to select a few different probabilities to induce a polynomial number of states. That is, we randomly select at most $\lceil \log m \rceil$ exit probabilities of exit points from $\{p(1),...,p(m-1)\}$ and impose $\{0\%,100\%\}$ on the set of selected probabilities. All product combinations of the selected probabilities are then enumerated to obtain at most $2^{\lceil \log m \rceil}+1$ candidate progressing proportions. Hence, COLT-OPE only records at most $2^{\lceil \log m \rceil}+1$ candidate progressing proportions in dynamic programming. This subtle design greatly decreases the time complexity to polynomial, and acquires a near-optimal solution. Later, simulations will show that one time random sampling is sufficient to obtain a near-optimum solution.

Example 3. Following Example 2, the example demonstrates COLT-OPE. First, $\lceil \log 5 \rceil = 3$ exit probabilities 40%, 50%, and 70% from $\{40\%,95\%,50\%,70\%\}$ are sampled with the imposed probabilities 0% and 100%. Then, it lists $2^3+1=9$ progressing proportions, 0, $(1-40\%)(1-50\%)(1-70\%)=0.09$, 0.15, 0.18, 0.3 (duplicates of $(1-40\%)(1-50\%)$ and $1-70\%$), 0.5, 0.6, and 1. Table IV briefly shows the derivation of $T(3,5)$. Take $\mathbb{T}(2,3,2,0.09)$ for example. It has to examine the states of $\mathbb{T}(1,j,k,\mathcal{P})$ for any $j\in[0,3]$, $k\in\{1,2\}$, and $\mathcal{P}\in\{0,0.09,0.15,0.18,0.3,0.5,0.6,1\}$. The candidate solutions required to examine are shown in Fig. 3. Next, it selects $\mathbb{T}(1,2,1,0.09)$ to form $\mathbb{T}(2,3,2,0.09)$ since $34.2+(61+\frac{30+60}{10})\cdot 0.05\cdot(100\%-50\%)+(61+\frac{30+60}{10}+\frac{50}{5})\cdot 0.05\cdot 50\% = 35.95+80\times 0.025 = 37.95$ is the smallest. Then, $\mathbb{T}(2,3,2,0.09)$ is set to $(35.95,80,0.025)$. Finally, $T(3,5)$ chooses $\mathbb{T}(3,5,3,0)$ that reuses $\mathbb{T}(2,3,2,0.09)$.

Time Complexity. The time complexity of COLT-OPE is $O(nm^4)$. Due to the page limit, the details are omitted.

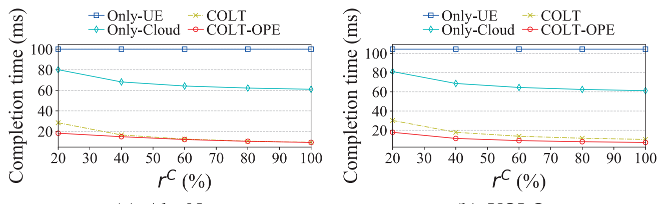
V. EVALUATION

We verify the performance of COLT and COLT-OPE in this section. In the simulation, the mobile network consists of 10 levels [3], where levels 1 and 10 are the UE and cloud. Since both the computing capacity and bandwidth of levels usually grow from UE to cloud [3], we set $c(1)=100$ Gflops, $c(i)=100+200i$ Gflops, $b(1,2)=\alpha$ Gbps, and $b(i-1,i)=(2300+150i)\alpha$ Mbps, where $i\in[2,10]$ and α is random in $[0.25,1]$.

Two popular image recognition models, i.e., AlexNet and YOLO, are adopted in our experiments. AlexNet is a small size model containing 5 CL, 3 FL and 3 Pooling Layers (PL), whereas YOLO is a relatively large model containing 24 CL, 2 FL and 4 PL. Note that a PL is considered as an independent layer in our simulation. Choosing models with different sizes shows that COLT-OPE could adapt to multi-sized DNN models. Note that the download time of the trained DNN model from the cloud to servers is ignored since a trained model can be shared by similar-purpose applications (e.g., DNN for image recognition applications) without frequent updates. To support exit points, we derive the exit probability of each layer by linear regression based on the probabilities given in [7]. We use high-quality images $s(0)\approx 3$ Mb and $\Delta=16$ ms for 60-fps throughput [8]. The performance metric is the expected completion time, and each result is averaged by 200 simulations.

A. Completion Time

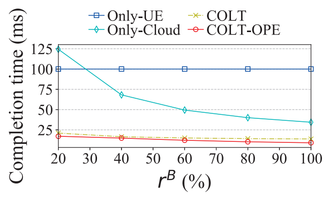
COLT and COLT-OPE are compared with two traditional methods: Only-UE and Only-Cloud (see Section IV). Since computing and bandwidth resources of servers may be occupied by other applications, we respectively define the available computing capacity ratio r^{C} and the available bandwidth capacity ratio r^{B} as the ratio of available resources to the maximum resources. For example, if r^{C} of level 2 equals 20%, the available computing capacity equals $c(2)\times 20\% = (100+200\times 2)\times 20\% = 100$ Gflops. Note that $r^{\mathrm{C}}=40\%$ and $r^{\mathrm{B}}=40\%$ in the default setting. In addition, since the UE usually wants to get the result as soon as possible, we assume that the UE always uses 100% computing resources.



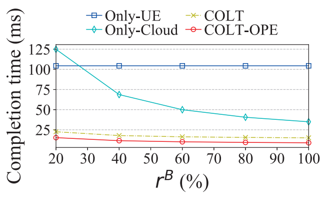
(a) AlexNet (b) YOLO

Fig. 4. Effect of available computing capacity on completion time.

Figs. 4 and 5 show the completion time over different r^{C}, r^{B}, and DNN models. Only-UE is a horizontal line due to its 100% computing usage. For other methods, the completion time descends as r^{C} or r^{B} increases. Compared with Only-UE and Only-Cloud, COLT-OPE spends time of 20%-47% since COLT-OPE aggressively utilizes all computing units in the network. It is worth noting that the completion time of COLT-OPE raises almost linearly, unlike that of Only-Cloud increasing exponentially, as r^{C} or r^{B} descends. This indicates that COLT-OPE is less influenced by r^{C} and r^{B} since it can assign tasks to servers near the UE if the computing time reduced by uploading to the remote servers is insufficient to compensate the time consumed by intermediate data transmissions. Besides, Only-Cloud is more sensitive to r^{B} since it has to send the raw image data to the cloud hop by hop. COLT-

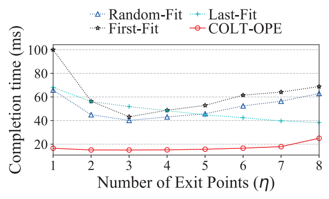


(a) AlexNet

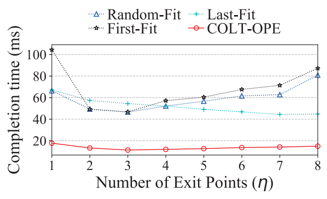


(b) YOLO

Fig. 5. Effect of available bandwidth capacity on completion time.

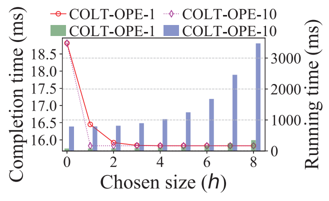


(a) AlexNet

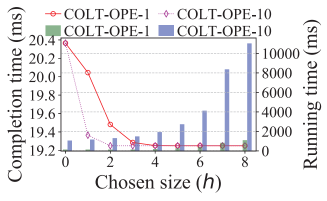


(b) YOLO

Fig. 6. Effect of exit point number on completion time.



(a) AlexNet



(b) YOLO

Fig. 7. Effect of chosen parameter size on completion time and running time.

OPE outperforms COLT since COLT-OPE deploys the exit points to reduce the data delivered to the next level.

B. The Efficiency of Exit Point Deployment

To further validate COLT-OPE's effectiveness, we examine each possible number of exit points and additionally design three naïve schemes as follows: 1) First-Fit. Given the number of exit points η, First-Fit first randomly divides the model into η stages and the final layer of each stage is equipped with an exit point. It then deploys the 1^{st} stage on level 1 (i.e., UE), the 2^{nd} stage on level 2, ..., and the η^{th} stage to level η. 2) Last-Fit. Similar to First-Fit, this scheme first generates η non-empty stages with exit points. However, it assigns the η^{th} stage to level 10 (i.e., cloud), the $\eta-1^{th}$ stage to level 9, and so on. 3) Random-Fit. Like First-Fit, this scheme first determines η stages. Then, it randomly selects η levels in the network and sequentially assigns the η stages to these levels.

Fig. 6 shows that COLT-OPE can find the optimal positions of exit points for any given number of exit points. COLT-OPE significantly outperforms other schemes since it carefully selects the most suitable positions of exits points. Moreover, the trends of First-Fit and Random-Fit is consistent with that of COLT-OPE since the positions of non-final exit points are almost deployed in the shallower levels (about level 2-7). However, Last-Fit tends to deploy stages near cloud, which makes the data transfer time be usually greater than computing time and thus appears degressive trend.

C. The Effectiveness of Random Sampling Technique

Recall that COLT-OPE uses *random sampling technique* to save running time. We claim that choosing only one random combination containing $\lceil \log m \rceil$ exit probabilities and searching the optimal solution based on these exit probabilities are sufficient. Here, we design two methods to examine the effectiveness as follows: 1) COLT-OPE-1. Given the number of exit probabilities h, it only chooses a random combination of size h and exhaustively searches the optimal solution from 2^h possible progressing proportions. 2) COLT-OPE-10. Given h, it chooses ten random combinations containing h exit probabilities and finds the optimal solution around 10×2^h possible progressing proportions. Fig. 7 demonstrates that, in both AlexNet and YOLO cases, COLT-OPE-1 and COLT-OPE-10 achieve the same completion time if $h = \lceil \log m \rceil$ (For AlexNet, $h = \lceil \log 11 \rceil = 4$; for YOLO, $h = \lceil \log 30 \rceil = 5$). Besides, one time *random sampling technique* significantly reduces running time of finding a near-optimum solution.

VI. Conclusions

This paper investigates the completion-time-aware deployment termed DEMAND-OPE. We first design COLT by dynamic programming to jointly address *proximity trade-off* and *suitable stage number* to acquire the optimal solution of DEMAND without early exit point. Then, we propose COLT-OPE to further shorten the expected completion time by carefully deploying the *optional exit points* for DEMAND-OPE. COLT-OPE balances the effects of the number of deployed exit points and the progressing proportion of data to effectively reduce completion time. Simulation results show that our algorithms outperform the traditional methods by at least 200%.

References

[1] J. Wang, B. Cao, P. Yu, L. Sun, W. Bao, and X. Zhu, "Deep learning towards mobile applications," in *IEEE ICDCS*, 2018.

[2] (2019) Cisco visual networking index: Global mobile data traffic forecast update. White paper.

[3] C. Marquez *et al.*, "How should I slice my network? a multi-service empirical evaluation of resource sharing efficiency," in *ACM MOBICOM*, 2018.

[4] P. Rost *et al.*, "Mobile network architecture evolution toward 5G," *IEEE Commun. Mag.*, vol. 54, pp. 84–91, 2016.

[5] T. G. Rodrigues, K. Suto, H. Nishiyama, and N. Kato, "Hybrid method for minimizing service delay in edge cloud computing through VM migration and transmission power control," *IEEE Transactions on Computers*, vol. 66, pp. 810–819, 2017.

[6] S. Teerapittayanon, B. McDanel, and H. T. Kung, "Distributed deep neural networks over the cloud, the edge and end devices," in *IEEE ICDCS*, 2017.

[7] S. Teerapittayanon, B. McDanel, and H. Kung, "Branchynet: Fast inference via early exiting from deep neural networks," in *ICPR*, 2016.

[8] G. Klein and D. Murray, "Parallel tracking and mapping for small AR workspaces," in *IEEE/ACM ISMAR*, 2007.

[9] J. Xu, L. Chen, and P. Zhou, "Joint service caching and task offloading for mobile edge computing in dense networks," in *IEEE INFOCOM*, 2018.

[10] L. Wang *et al.*, "Service entity placement for social virtual reality applications in edge computing," in *IEEE INFOCOM*, 2018.

[11] S. Wang *et al.*, "When edge meets learning: Adaptive control for resource constrained distributed machine learning," in *IEEE INFOCOM*, 2018.

[12] X. Ran, H. Chen, X. Zhu, Z. Liu, and J. Chen, "DeepDecision: A mobile deep learning framework for edge video analytics," in *IEEE INFOCOM*, 2018.

[13] J. Dean *et al.*, "Large scale distributed deep networks," in *NIPS*, 2012.

[14] F. N. Iandola, M. W. Moskewicz, K. Ashraf, and K. Keutzer, "FireCaffe: Near-linear acceleration of deep neural network training on compute clusters," in *IEEE CVPR*, 2016.

[15] J. Mao *et al.*, "AdaLearner: An adaptive distributed mobile learning system for neural networks," in *IEEE/ACM ICCAD*, 2017.