CoEdge: Exploiting the Edge-Cloud Collaboration for Faster Deep Learning

LIANGYAN HU1, GUODONG SUN1,3 (Member, IEEE), AND YANLONG REN2

1School of Information Science and Technology, Beijing Forestry University, Beijing 100083, China
2Network Information Management and Service Center, Beijing University of Civil Engineering and Architecture, Beijing 100044, China
3Engineering Research Center for Forestry-oriented Intelligent Information Processing, National Forestry and Grassland Administration, Beijing 10083, China

Corresponding author: Guodong Sun (sungd@bjfu.edu.cn)

This work was supported in part by NSF of China under Grant 61300180, and in part by the Fundamental Research Funds for the Central Universities of China under Grant 2017ZY20.

ABSTRACT Recently a great number of ubiquitous Internet-of-Things (IoT) devices have been connecting to the Internet. With the massive amount of IoT data, the cloud-based intelligent applications have sprang up to support accurate monitoring and decision-making. In practice, however, the intrinsic transport bottleneck of the Internet severely handicaps the real-time performance of the cloud-based intelligence depending on IoT data. In the past few years, researchers have paid attention to the computing paradigm of edge-cloud collaboration; they offload the computing tasks from the cloud to the edge environment, in order to avoid transmitting much data through the Internet to the cloud. To present, it is still an open issue to effectively allocate the deep learning task (i.e., deep neural network computation) over the edge-cloud system to shorten the response time of application. In this paper, we propose the latency-minimum allocation (LMA) problem, aimed at allocating the deep neural network (DNN) layers over the edge-cloud environment while the total latency of processing this DNN can be minimized. First, we formalize the LMA problem in general form, prove its NP-hardness, and present an insightful characteristic of feasible DNN layer allocations. Second, we design an approximate algorithm, called CoEdge, which can handle the LMA problem in polynomial time. By exploiting the communication and computation resources of the edge, CoEdge greedily selects the beneficial edge nodes and allocates the DNN layers to the selected nodes by a recursion-based policy. Finally, we conduct extensive simulation experiments with realistic setups, and the experimental results show the efficacy of CoEdge in reducing the deep learning latency compared to two state-of-the-art schemes.

INDEX TERMS Edge computing, deep learning, latency, allocation of DNN layers.

I. INTRODUCTION

In the past few years, the popularity of emerging Internet-of-Things (IoT) applications has been generating a huge amount of real-time data. Based on the IoT data, the IoT end-users can make accurate monitoring and effective decisions with their intelligent utility deployed in cloud. For such cloud-based intelligent IoT applications, however, the data generated by the ubiquitous IoT devices has to be delivered, through the Internet, to the cloud for further processing. In industry, cloud computing has played an indispensable role in executing large-scale and intensive-computing tasks, such as deep learning, and thus, the intelligence of the IoT applications usually resides in the cloud [1], [2]. Under such a cloud-centric paradigm, the data delivered from IoT devices to the cloud will be inevitably impacted by the inherent transmission bottleneck of the Internet [3]–[6]. Accordingly, the response time or delay of application will be significantly extended, especially in the intelligent IoT applications where massive real-time IoT data struggles to traverse the Internet, swarming into the cloud. As a matter of fact, the non-negligible transmission delay of the Internet has become the essential obstacle to expediting the deep learning-based IoT applications [7]–[9].

One promising way of addressing the above issue is edge computing. It is a new computing paradigm complementary to cloud computing, in which the IoT data processing or at least part of it is moved from the cloud to the edge of the Internet, so that the computing task can be executed at the proximity of data sources, rather than being done at the remote, hard-to-reach cloud [7], [10], [11]. Many recent studies show the ability of edge computing in reduc-
ing the system delay. In [12], for instance, the researchers offload the computation from the cloud to the wearable cognitive assistance. They reduce the response time of application by 80 ms to 200 ms; moreover, their work squeezes 30%-40% less energy consumption from the cloud-based strategy. Edge computing has been envisioned to be a unifying platform that can engender a new breed of emerging services and support a variety of new intensive-computing real-time applications.

Recently researchers have attempted to process the deep learning task in edge, mainly in order to reduce the system response delay. In general, the execution of deep learning task is a layer-by-layer process of the deep neural network (DNN) model, which typically consists of a set of consecutive perceptron layers [13]. The raw IoT data is fed into the first DNN layer, and the inference or classification can be finally yielded out of the last DNN layer. In the ongoing process of DNN, the intermediate results are transferred through the DNN layers and can be quickly scaled down in terms of size. For edge-based intelligent IoT systems, the whole DNN model or a part of its layers can be deployed on the edge node, which is far closer to the IoT devices than the cloud server is. Sometimes, the IoT devices are resource-rich and then can by themselves serve as a kind of edge devices. The edge node only needs to load up to the cloud some size-reduced intermediate data or even the final result, instead of the raw IoT data of large size. The edge participation can therefore shorten the response time of DNN-based IoT system by reducing the Internet traffic. In order to further reduce the system response time, however, it is still an open issue to design an effective paradigm that can take full advantage of the edge-cloud resources.

In this paper we design and implement a new allocation scheme, called CoEdge, which attempts to allocate the DNN layers over the edge and the cloud such that the deep learning delay can be further reduced in comparison with the schemes that only involve a single edge node. With a greedy criterion, CoEdge iteratively finds out the best edge node and forms a set of edge nodes, over which the DNN is then allocated with a recursion-based policy. Essentially, CoEdge unlocks the potential of edge in deep learning, by exploiting the high-speed connection and the computing capacity embraced by the edge environment.

The remainder of this paper is organized as follows. Section II briefly introduces major works related to ours. Section III and Section IV give the models and the detailed design of CoEdge. Section V evaluates our design and compares it with two state-of-art schemes via extensive experiments. Finally, Section VI concludes this paper.

II. RELATED WORK

In this section, we first introduce the task offloading in edge computing and then, the approaches to allocating DNN learning or inference tasks to edge-cloud system.

A. OFFLOADING COMPUTE TASK ONTO EDGE

In cloud computing, transferring data to the cloud server often needs a considerable amount of time, which will surely weaken the quality-of-service for the time-sensitive applications. To address this issue, a promising approach is to offload a part of computing tasks onto the resource-rich edge, alleviating the deficiencies of network congestion, latency and energy consumption [7]. Exiting works in edge-cloud offloading focus on how to determine the effective and efficient task offloading policy [14], [15], [16]–[18]. To improve the resource efficiency, reference [19] designs a resource-efficient edge computing framework, which enables intelligent IoT users to flexibly offload task across the edge device, the nearby assistant devices, and the adjacent edge cloud. In [20], the authors exploit the possibility that massive mobile devices collaboratively execute the on-edge task, aimed to optimize the energy efficiency of those devices. In [21], the authors consider a scenario where flying unmanned aerial vehicles serve as edge nodes; they propose a resource scheduling approach that can offload tasks in dynamic environments by leveraging based a learning algorithm. How to offload the computing task onto the edge has recently attracted more and more attention in industry and academia. We have observed that the sustainable edge computing can bring latency reduction and there is a great chance to deploy deep learning in edge.

B. ALLOCATING DNN TO EDGE-CLOUD

The high-accuracy deep learning usually depends on a lot of training data, and as a result, the demand for bandwidth rises up dramatically in the cloud-based intelligent applications. To accelerate the model training or inference of DNN, a popular means nowadays is to transfer to the edge environment a part of or even all the DNN layers, i.e., making them closer to the data sources.

Many hardware platforms including GPU or customized accelerators such as FPGA and ASIC have emerged. In [22], Djinn is designed, which is an open infrastructure for DNN with large-scale GPU servers to achieve high throughput and low network occupancy. There have been several approaches to accelerating machine learning [23]–[25]. FPGA-based accelerators have more flexibility than ASICs in accelerating large-scale CNN models. A deeply pipelined multi-FPGA architecture is designed in [26], which can achieve lower latency by using a dynamic programming method to map a DNN onto several pipelined FPGAs. For this approach based on a fixed pipeline, however, the FPGA devices are assumed to be homogeneous in computing capacity and each of them is required to undertake at least one DNN layer.

The demand for memory, computing and energy capacity has gradually grown up to be a critical bottleneck for the allocation of DNN to edge devices. How to deploy DNN into edge environment has been studied primarily over industry and academia. A software accelerator for DNN execution in edge network is presented in [27], in which the resource demands
are reduced by decomposing DNN layers into various unit-blocks that can be effectively processed by heterogeneous processors. In [28], a method called AAIoT is proposed to allocate DNN to a set of devices that form a multi-level IoT system. AAIoT balances the computation and transmission time to minimize the overall response time. Reference [29] schedules DNN layers in edge computing environment: it allocates as many deep learning tasks to the edge devices as possible, while satisfying a given constraint on response time. For a given deep learning task, however, the authors only employ a single edge device to share some DNN layers. Particularly, they do not consider the potential of collaboration across the edge devices, and thus, their allocation policy cannot well suit the deep learning task with a strict requirement for system latency.

Different from the above works, the CoEdge proposed in this study is designed for a general edge-cloud computing system. AAIoT balances the utilization of edge devices, where the edge devices could be heterogeneous in computing and communication resources. Additionally, in order to further shorten the deep learning latency, CoEdge can elaborately exploit the high-speed links and strong computing capacity contained in the edge network.

III. MODELS AND PROBLEM

In this section, we first introduce the system model in use and the corresponding notations. Second, we describe the proposed LMA problem, reformulate it as a constrained delay-minimization problem, and analyze its intractability; and particularly, we theoretically reveal an important feature of all optimal LMA solutions, which can exactly help reduce the searching space of our algorithm (to be given in Section IV) and thus improves the efficiency.

A. MODELS AND PRELIMINARIES

Typically, an edge-cloud consists of a cloud server and an edge network. We use $c$ to represent the cloud server, and $E = \{e_1, e_2, \ldots, e_u\}$, to represent the set of edge nodes. The edge nodes are interconnected via D2D links, 5G network, or other kinds of high-speed network. We assume that the transmission in edge is much faster than that between any edge node $e \in E$ and the cloud node $c$. If the deep learning task is completely processed in the edge, we still report the final result to the cloud node.

A deep neural network (DNN) can be represented with a hierarchical structure $\mathcal{L} = \{L_i|1 \leq i \leq m\}$, where each layer $L_i$ is a set of neurons. Because of the intrinsic directionality of processing DNN, $\mathcal{L}$ is a partially ordered set (or simply a sequence) of perception layers. In this paper, we define a partial-order relation (denoted by “$\prec$” and “$\preceq$”) on a set $U$—if $u_i \prec u_j$, $u_i$ proceeds $u_j$ in $U$, and if $u_i \preceq u_j$, we have $u_i \prec u_j$ or $i = j$. If there exists no $u_k \in U$ such that $u_i \prec u_k$ and $u_k \prec u_j$, we say that $u_i$ and $u_j$ are adjacent or consecutive in $U$. For two layers $L_i$ and $L_j$ of $\mathcal{L}$, “$L_i \prec L_j$” means that $L_i$ should be executed earlier in the deep learning task than $L_j$. Fig. 1 shows a typical DNN which involves five layers. The calculation of layer $L_i$ $(1 \leq i < m)$ yields the intermediate data, which will immediately be passed on to layer $L_{i+1}$ for further processing. We denote by $\theta_{i}^{in}$ the size of the data input to $L_i$, and by $\theta_{i}^{out}$ the size of the data output from $L_i$. Often, layer $L_1$ is called the input layer, which receives the input data to be classified; $L_m$ is called the output layer, which returns the final classification result; and other layers are called hidden layer because they are not connected with the external world.

Fig. 1 shows an example where a pixelated image of cat is fed to a 5-layer CNN and after two stages (i.e., the feature extraction and the feature classification), a cat is recognized. Despite the number of layers, the DNN also specifies a neuron set for each layer and the connection pattern between two adjacent layers. Both factors collectively determine the computational cost at each layer and the volume of intermediate results to be transferred between two consecutive layers. Given the raw input data (often, an image), the latency of DNN-based learning or inference has two parts: the total computation time of all the layers, and the total time of transferring all the intermediate results between any two consecutive layers plus transferring the final result to the cloud server.

FIGURE 1. A typical example of layer-by-layer DNN structure.
B. PROBLEM DESCRIPTION AND ANALYSIS

In the edge-cloud system we consider, for a given deep learning task, some edge node of $E$ (denoted by $e_r$) receives the raw input data, and the cloud node $c$ receives the final result of learning or inference. So, both $e_r$ and $c$ will always participate in the deep learning task, although either of them does not have to undertake any DNN layers. In this paper, we participate in the deep learning task, although either of them does not have to undertake any DNN layers. In this paper, we present the LMA problem to be addressed and then prove its NP-hardness.

The allocation of $L$ over $E \cup \{c\}$ can be expressed with a tuple $A = (P(L), S_{rc}, \varphi)$, where function $\varphi : P(L) \to S_{rc}$ is the allocation policy. If $\varphi(L_j) = e_i$, node $e_i$ will perform all the layers in segment $L_j$; we denote by $d_i^j$ the time cost for $e_i$ to perform $L_j$. In practice, $d_i^j$ can be figured out in advance, according to the available computing resource of $e_i$ and the total computing load incurred by $L_j$. Noticeably, we can reasonably neglect the time consumed to transfer the intermediate data between any two adjacent layers of segment $L_j$, because such on-node data transfers are carried out by $e_i$ within its local space of memory access. Consider two adjacent segments $L_j$ and $L_{j+1}$ of partition $P(L)$, and assume that $\varphi(L_j) = e_i$ and $\varphi(L_{j+1}) = e_k$. If $e_i$ and $e_k$ are two different nodes of $S_{rc}$, the cross-edge data transfer is needed—node $e_i$ needs to transfer to node $e_k$ the intermediate data output from the last layer of $L_j$. We use $d_{i\rightarrow k}^j$ to represent the delay of such a cross-edge data transfer from node $e_i$ to node $e_k$.

We denote by $\delta(A)$ the latency that the allocation $A$ can achieve on processing a deep learning task. Thus, the LMA problem can be formally written as

$$\min \delta(A) = \sum_{L_j \in P(L)} \left( d_i^j + d_{i\rightarrow k}^j \right), \quad (1)$$

where $d_{i\rightarrow k}^j$ equals zero if $e_i$ is the cloud node under the current allocation policy $\varphi$. Given partition $P(L)$ and $S_{rc}$, Fig. 2 shows three feasible application policies. In the solution for LMA, some nodes of $S_{rc}$ might not process any segments but relay the intermediate data between nodes. In Fig. 2, for example, $e_3$ and $e_r$ are just relaying nodes under allocation policies $\varphi_2$ and $\varphi_3$, respectively.

**Theorem 1:** Generally, the LMA problem is NP-hard.

**Proof:** To show the NP-hardness of the LMA problem, we here consider a special instance of LMA, in which (1) the nodes that can process DNN $L$ are given in advance as $S_{rc} = \{e_1, e_2 \ldots e_k\} (k \geq 2)$, (2) the time cost $d_{p\rightarrow q}^j$ is extremely low and can be neglected for any pair of nodes $e_p, e_q \in S_{rc}$, and (3) each node $e_i$ will consume $\tau_i$ time in performing layer $L_j$ of $L$. For this special case, we can treat $S_{rc}$ to be a set of parallel machine and $L$, to be the set of jobs with precedence. This special problem instance is thus transformed into determining a partition of $L$ and a policy that can allocate each resulted segment to a machine of $S_{rc}$ while minimizing the total processing time. Clearly, this problem can be modeled as the Workload Partition Problem (WPP) with the precedence constraint of jobs. The WPP has proven NP-hard [30]–[32]. To solve the problem proposed in this paper, we need to determine not only the partition of $L$ and the allocation policy $\varphi$ but the subset $S_{rc}$ of $E \cup \{c\}$, and thus, the LMA problem is generally at least NP-hard.

**Theorem 2:** Suppose that there is an allocation of partition $P(L)$ over $S_{rc}$ ($|S_{rc}| \geq 2$) and the corresponding allocation policy is $\varphi$. Consider three consecutive nonempty segments $L_{i-1}$, $L_i$, and $L_{i+1}$ of $P(L)$ such that $\varphi(L_{i-1}) = \varphi(L_{i+1}) = e_p$ and $\varphi(L_i) = e_q$, where $e_p$ and $e_q$ are two distinct nodes of $S_{rc}$. We declare that the allocation policy $\varphi$ cannot lead to an optimal allocation of $P(L)$ over $S_{rc}$.

**Proof:** We prove this theorem by construction. The left part of Fig. 3 shows the allocation policy $\varphi$ on the given three consecutive segments. We next reshape $\varphi$ under two mutually exclusive cases:

- Case I: $e_p$ is at least as powerful as $e_q$ in terms of computing capacity, and
- Case II: $e_q$ is more powerful than $e_p$ in terms of computing capacity.

If Case I holds, we can derive a new allocation policy $\varphi'$ from $\varphi$, by only re-allocating segment $L_i$ to $e_p$, as shown in the middle part of Fig. 3. We first evaluate by (2) the time cost...
of ϕ on processing these three consecutive segments.

\[
\delta(\phi) = d_p^{i-1} + d_p^{i-q} + d_q + d_{q-p}^i \\
+ d_p^{i+1} + \begin{cases} \\
\frac{d_p^{i+1}}{x} \quad \text{or} \\
\frac{d_p^{i+1}}{q} + d_{q-p}^i \\
\end{cases}
\]

(2)

Under allocation policy φ, nodes e_p and e_q process segments L_{i-1} and L_i, respectively, and then, e_p takes over segment L_{i+1}. φ results in two cross-edge intermediate data transfers, both of which consume times of d_p^{i-q} and d_q-p, respectively. In (2), the last item of the right-hand indicates that there are two alternatives for φ to transfer the intermediate data output from the last layer of L_{i+1}: one is transfer the data directly from e_p to some node e_x processing L_{i+2}, and the other, through e_q to some node e_y processing L_{i+2}. Similarly, the time cost of ϕ' on processing the three segments can be expressed with

\[
\delta(\phi') = d_p^{i-1} + d_p^{i} + d_q^{i+1} \\
+ d_p^{i+1} + \begin{cases} \\
\frac{d_p^{i+1}}{x} \quad \text{or} \\
\frac{d_p^{i+1}}{q} + d_{q-p}^i \\
\end{cases}
\]

(3)

In Case I, e_p is identical to or faster than e_q in terms of processing speed, i.e., d_p^i ≤ d_q^i. Comparing (3) and (2), we always have δ(ϕ') < δ(ϕ), regardless of how the allocation policy φ relays the intermediate data output from the last layer of L_{i+1}.

If Case II holds, i.e., e_q can process segment L_i faster than e_p, we can create a new allocation ϕ'' re-allocates segment L_i to e_q, the sole difference from ϕ. Also, we can easily prove δ(ϕ'') < δ(ϕ). In conclusion, we can always reshape ϕ given in this theorem into a latency-shorter allocation policy.

Comparing the three allocation policies shown in Fig. 3, we can see that both L_{i-1} and L_{i+1} are allocated by ϕ to node e_p but “cut in” by L_i, which is allocated to another node e_q. Theorem 2 implies that if an allocation policy allocates non-consecutive segments to some node, such a cut-in policy is not optimal. Not limited to the case of three consecutive segments, Theorem 2 can be easily extended such that it can apply to the case of any three nonempty segments L_i < L_j < L_k where L_i and L_k are both allocated to one node but L_j is allocated to another node. The heuristics offered by Theorem 2 make designers relievedly bypass those cut-in allocations, which will narrow down the solution space (or the feasible region) and then help speed up their algorithms.

IV. DESIGNS

Recall that for the deep learning considered in this paper, we let c and c_e represent the cloud node and the edge node that receives the input data, respectively. If we only allocate the L to e_c and c, as [29] does, we can easily figure out an optimal solution for our problem with polynomial time. Besides these two nodes, actually, any other nodes of E could be included in the optimal solution for the general case. To address the LMA problem, we design an approximate algorithm, called CoEdge. With a greedy policy, CoEdge attempts to iteratively insert a new edge node into the S_r which is initialized only with {e_r, c}, until the iteratively updated S_r cannot assure a shorter period of time to perform the deep learning task. Algorithm 1 shows how the proposed CoEdge works. Before diving into the algorithm description, we first introduce the information to be input to CoEdge and how to initialize these inputs.

Algorithm 1 CoEdge

**Input:** E ∪ {c}, L, W, and C

**Result:** the allocation of L over a subset S_r of E ∪ {c}

1. \( S_r \leftarrow \{e_r, c\} \)

2. Determine an optimal layer allocation \( \Lambda \) over \( S_r \) and then obtain the minimum total latency (i.e., \( \delta_{\text{min}} \))

3. while \( E \cup \{c\} - S_r \neq \emptyset \) do

4. foreach \( e \in E \cup \{c\} - S_r \) do

5. Determine a partition \( P(L) \) as well as an allocation policy \( \phi \) for \( S_r \oplus e \) to minimize the total latency

6. end

7. Select \( e^* \) from the nodes examined in the above for-loop such that the corresponding partition \( P^*(L) \) and allocation policy \( \phi^* \) on \( S_r \oplus e^* \) can achieve the minimum latency (denoted by \( \delta^* \))

8. if \( \delta^* < \delta_{\text{min}} \) then

9. \( \delta_{\text{min}} \leftarrow \delta^* \)

10. \( S_r \leftarrow S_r \oplus e^* \)

11. Update \( \Lambda \) with \( S_r, P^*(L) \), and \( \phi^* \)

12. else

13. \( \text{return} \ \Lambda \)

14. end

15. end

16. return \( \Lambda \)

A. INITIALIZATION OF THE CoEdge INPUTS

The input information needed by CoEdge consists of four data sets: \( E \cup \{c\}, L, W, \) and \( C \). The latter two sets profile the bandwidth resource and compute capacity of the edge-cloud environment. More specifically, \( W \) is a matrix of \( |E \cup \{c\}| \) rows and \( |E \cup \{c\}| \) columns; each element \( \omega_{ij} \) measures the best available bandwidth between two distinct nodes \( e_i \) and \( e_j \). Since we neglect the time consumed in on-node data transfers, we let \( \omega_{ij} = \infty \). Input \( C \) is also a matrix; its elements \( c_{ij} \) stores the computational time that each node \( e_i \) of \( E \cup \{c\} \) needs to pay if it is assigned to perform any possible segment \( L_j \) of \( L \). Next we introduce how to determinate matrices \( W \) and \( C \) before algorithm CoEdge can go ahead.

We assume that the edge network is connected and each edge node connects with the cloud node through the Internet. So, there exists at least one communication path between any two edge nodes or between any edge node and the cloud node. We employ the Floyd algorithm to calculate the best available
bandwidth between any two distinct nodes of $E \cup \{c\}$, and then, obtain the input matrix $W$. Considering symmetric communication bandwidth, the example given in Fig. 4 shows the determination of matrix $W$.

For a DNN $\mathcal{L}$ with $m$ perceptron layers, obviously, there are $\frac{m(m+1)}{2}$ different segments. We set up $C$ as a matrix of $|E \cup \{c\}|$ rows and $\frac{m(m+1)}{2}$ columns, and each element $c_{ij}$ stores the total computational time for node $e_i \in E \cup \{c\}$ to perform all the consecutive layers in the $j$-th segment of $\mathcal{L}$. Asymptotically, we need $O(n^2)$ time to determine matrix $C$. Since each segment can be possibly included in an optimal allocation, it is necessary for CoEdge to know how fast each node processes any possible segments.

B. DESCRIPTION OF CoEdge

After the above four input data sets come prepared, algorithm CoEdge initializes $S_{rc}$ with a partially-ordered set of $(e_r, c)$, in which $e_r$ is the edge node receiving the input data of DNN $\mathcal{L}$ and $c$ is the cloud server. At the very beginning, therefore, only $e_r$ and $c$ are prepared to share all the DNN layers. By brute-force enumeration, CoEdge can achieve an optimal solution $\Pi$ in linear time for allocating $\mathcal{L}$ to $(e_r, c)$, which yields the shortest latency of processing $\mathcal{L}$, denoted by $\delta_{min}$. CoEdge will next enter a greedy iterative procedure (line 3 of Algorithm 1), which tries to invite more edge nodes to collaboratively process DNN $\mathcal{L}$, aimed at updating $\delta_{min}$ with a shorter latency. In essential, the CoEdge algorithm continually grows the $S_{rc}$ set to pursue the acceleration of deep learning. In CoEdge, we regulate $S_{rc}$ such that is a partially-ordered set: for two distinct nodes $e_i, e_j \in S_{rc}$ that process segments $\mathcal{L}_i$ and $\mathcal{L}_j$, respectively, we have $e_i \prec e_j$ if $\mathcal{L}_i \prec \mathcal{L}_j$ holds. Such a partial-order regulation on $\mathcal{L}_{rc}$ makes the CoEdge algorithm bypass checking the cut-in allocation policies, which will not be optimal solutions, according to Theorem 2. Keeping $S_{rc}$ partially-ordered can thus help CoEdge narrow down the search space in its iterations. When a new node $e$ is examined in each iteration in hope of reducing the latency, CoEdge will always put it right ahead of the last node (i.e., the cloud server $c$) of $S_{rc}$. The addition of $e$ into $S_{rc}$ is expressed with “$S_{rc} \oplus e$” in this paper.

After entering the iteration in line 3 of Algorithm 1, CoEdge first checks all the nodes of $E$ that have not yet been inserted into $S_{rc}$, in order to decide which of these nodes can lead to a latency-minimum allocation (see the lines from 4 to 7). If the addition of $e^*$ into $S_{rc}$ and the corresponding partition $P^*(\mathcal{L})$ can achieve a minimum latency so far, we then update $\delta_{min}$ and the allocation while doing $S_{rc} \oplus e^*$. More specifically, CoEdge in line 5 determines the latency-minimum partition over $S_{rc}$ in a recursive way. We next resort to Fig. 5 to explain how to recursively obtain the best partition over the given $S_{rc}$.

As shown in Fig. 5, we suppose that there are $k$ nodes in sequence $S_{rc}$ (including the cloud server $c$) and an $m$-layer DNN $\mathcal{L}$ is to be allocated to $S_{rc}$ while minimizing the total latency. Recall that $e_r$ receives the raw input data and then should be included in sequence $S_{rc}$. To obtain a latency-minimum partition over $S_{rc}$, we need only to consider two cases: (1) allocating all the DNN layers $\mathcal{L}$ to $e_r$, and (2) allocating the layers from $L_1$ to $L_p$ to $e_r$ while the remaining layers are allocated to the subsequence $S_{rc}\setminus e_r$. After evaluating the latencies of these two cases, we can pick out the best allocation. When we are going to determine the allocation of layers $L_{p+1}$ to $L_m$ over the subsequence $S_{rc}\setminus e_r$, we can also calculate the corresponding shortest latency by recursion. We next denote by $\mathcal{L}_{i,j}$ the set of consecutive layers from $L_1$ to $L_j$, and by $\delta(\mathcal{L}_{i,j}, S)$ the latency of some partition of $\mathcal{L}_{i,j}$ over $S \subseteq S_{rc}$, where $1 \leq i, j \leq m$ and “$i \succ j$” makes $\mathcal{L}_{i,j}$ be an empty set. We use $\pi(S)$ to represent the first node of $S$ (i.e., the node who proceeds all the other nodes of $S$) and then, $\delta(\mathcal{L}_{i,j}, S) = \pi(S)$. For the allocation of $\mathcal{L}_{i,j}$ over the subsequence $S$, its shortest latency, denoted by $\delta^{*}(\mathcal{L}_{i,j}, S)$, can be recursively expressed as

$$\delta^{*}(\mathcal{L}_{i,j}, S) = \min_{1 \leq i \leq p \leq j} \left\{ \begin{array}{l} \delta(\mathcal{L}_{i,j}, \pi(S)), \\ \delta(\mathcal{L}_{i,p}, \pi(S)) + \tilde{d}_p \\ \delta^{*}(\mathcal{L}_{p+1}, S) \end{array} \right\}, \quad (4)$$

where $\tilde{d}_p$ is time consumed in transferring the intermediate data from layer $L_p$ to layer $L_{p+1}$. In (4), when $p = i - 1$, we have $\delta(\mathcal{L}_{i,p}, \pi(S)) = 0$ because none of layers is assigned to node $\pi(S)$. In this case, although $\pi(S)$ does not process any layers, it still needs to pay $\tilde{d}_p$ time to relay the intermediate data from layer $L_{i-1}$ to layer $L_i$. Given a DNN, the size of the intermediate data output from layer $L_p$ (i.e., $b_{out}$) is foreknown. If we allocate $L_p$ and $L_{p+1}$ to nodes $e_i$ and $e_j$, respectively, we can then evaluate the time cost $\tilde{d}_p$ by $b_{out} \log_2(\omega_{ij})$, where $\omega_{ij}$ is the best available bandwidth from $e_i$ to $e_j$ and stored in matrix $W$. For a given sequence $S_{rc}$
and an \( m \)-layer DNN, we can obtain a latency-minimum allocation by recursively solving \( \delta^*(L_{1-m}, S_{rc}) \) according to (4). At last, CoEdge returns a partition \( P(L) \), a nonempty \( S_{rc} \), and an allocation policy \( \psi \); and for any \( L_i \prec L_j \) of this partition, we always have \( \psi(L_i) < \psi(L_j) \).

In each iteration, for a given \( e \in E \cup \{c \} - S_{rc} \), CoEdge employs recursion to complete allocating all the layers of \( L \) across \( S_{rc} \oplus e \). According to (4), CoEdge needs to evaluate \( \delta(L_{i-j}, \pi(S_{rc})) \) for any \( 1 \leq i < j \leq m \) on the first node of sequence \( S_{rc} \). As analyzed above, there are \( m(m+1) \) different partitions for an \( m \)-layer DNN. In addition, we have created the set \( C \) before CoEdge enters greedy iterations, and \( c_{ij} \) stores the total computational time for node \( e_i \) to perform all the consecutive layers of partition \( L_j \). We thus know that in the iteration with \( S_{rc} \) and a given \( e \), we can obtain \( \delta(L_{i-j}, S_{rc} \oplus e) \) with the time complexity of \( O(m^2(|S_{rc}| + 1)) \). Furthermore, we know that in each greedy iteration, CoEdge needs \( O(m^2(|S_{rc}| + 1) \cdot |E - S_{rc}|) \) time to find out \( e^* \) and the corresponding \( \delta^* \). We then easily know that for an \( m \)-layer DNN and an edge-cloud of size \((n+1)\), the total complexity of CoEdge is upper bounded with \( O(m^2n^3) \).

V. EXPERIMENTS
In this section, we conduct simulation experiments with realistic setups to evaluate our designs and compare it with two baseline algorithms [26], [29], which are here termed by \textit{fixedEdge} and \textit{singleEdge}. We use the AlexNet [33] and VGGNet-19 [34] in our experiments to do image classification. AlexNet is an eight-layer DNN, including five convolution layers and three fully-connected layers; and the first, second, and fifth layers of AlexNet also involve the max pooling. VGGNet-19 is a 19-layer DNN, which is divided into five convolutional segments. Each convolutional segment of VGGNet-19 is followed by a max pooling layer that is used to reduce the size of the image data. Since our objective is to reduce the deep learning latency by turning to the collaborative edge, we evaluate our algorithm and the two baselines in terms of latency under a variety of experimental cases.

A. EXPERIMENTAL SETUP
In simulation, we set the parameters with realistic setups. The computing capacity of the cloud node is set to 3200 Gflops. We set the edge with four different setups in bandwidth resource and computing capacity; they are given as follows.

1) high-speed edge: the bandwidth of in-edge link ranges from 500 Mbps to 1000 Mbps,
2) low-speed edge: the bandwidth of in-edge link ranges from 10 Mbps to 200 Mbps.
3) high-capacity edge: the compute capacity of edge node ranges from 80 Gflops to 640 Gflops.
4) low-capacity edge: the compute capacity of edge node ranges from 4 Gflops to 32 Gflops.

All the above setups are advised by [29], [35], [36] on the basis of empirical measurements. We evaluate the proposed CoEdge and the baselines under four different cases. For different experimental cases, the computing capacity and the bandwidth are randomly chosen from the corresponding ranges. In all the experiments, we make the edge network connected, although not all the pairs of edge nodes are directly connected. Each edge node can communicate with the cloud through the Internet; for a given edge node, its bandwidth to the cloud is set to a random value between 1 Mbps and 10 Mbps. The input images for the AlexNet and the VGGNet-19 are \( 227 \times 227 \) pixels (about 1.1794 Mb) and \( 224 \times 224 \) pixels (about 1.1484 Mb) in size. The computing load and the reduced ratio of intermediate results are set with the default values of these two DNN models. Each experimental case is repeated 40 times, each with a randomly-chosen edge node as the data source (i.e., the receiver of the input images); and the average for that case is reported.

B. RESULTS AND ANALYSIS
Figures 6-9 show how these three schemes perform in different cases. In Fig. 6, we first examine the inference latency of our CoEdge when the edge network is formed with edge devices that are higher in processing capability than 80 Gflops and support high-speed communications of bandwidth ranging from 500 Mbps to 1000 Mbps. It can be seen that for the AlexNet and the VGGNet-19, the CoEdge can always achieve the fastest inference, regardless of what the edge network size is set to be. Additionally, compared with the other two baselines, especially with the fixedEdge, the CoEdge remains much stable under each DNN model—its inference time in need experiences a subtle fluctuation as the network size increases. It is worth noticing in Fig. 6 that although VGGNet-19 involves only more than twice layers of AlexNet, the deep inference for VGGNet-19 takes far longer time than that for AlexNet. For example, the inference latency of CoEdge in AlexNet is only 1.77 ms, on average, whereas the latency of CoEdge in VGGNet-19 is higher than 90 ms. The inference time needed by the singleEdge sharply grows up to the order of hundreds of milliseconds. Such an observation reflects that for a complex DNN with a huge amount of computation, it is necessary and feasible to “dissolve” this DNN into the edge-cloud to further improve the inference performance.

![Figure 6](image-url)
FIGURE 7. Variation of inference latency against edge size under the high-speed and low-capacity edge network.

Fig. 7 compares the three schemes in the cases where each edge node works with a constrained computing capacity that ranges from 4 Gflops to 32 Gflops. Comparing Fig. 7 with Fig. 6, we can find that the computing capacity impacts considerably the deep inference delay. For AlexNet and VGGNet-19, CoEdge outperforms the two baselines. The latency CoEdge on VGGNet-19 increases up to eight times the latency of CoEdge on AlexNet; a similar increase in latency also happens to the singleEdge scheme. For the fixed-Edge scheme, however, the constrained computing capacity considerably increases its latency when it works with VGGNet-19, deeper than AlexNet. The comparison shown in Fig. 7 leads to a two-fold indication. First, in the high-speed and low-capacity edge, the singleEdge scheme recruits only one edge node to share the DNN layer and then avoids the possible latency bottleneck caused by these low-capacity edge nodes. That is why the singleEdge performs better than the fixedEdge on scheduling the VGGNet-19 model, where the fixedEdge always pre-select a set of edge nodes without considering the actual edge network resources. Second, in comparison with the two baselines, CoEdge always dynamically cherry-picks the “best-fitting” edge nodes and lets them collaboratively perform the DNN task, thereby resulting in far lower latency.

Fig. 8 compares the three schemes in latency under low-speed and high-capacity edge. The overall performances of our CoEdge for both the AlexNet and the VGGNet-19 are much close to its counterparts shown in Fig. 6. On the contrary, the fixedEdge performs a little worse in low-speed edge than it does in the high-speed edge, which means the impact of the reduced edge bandwidth on the deep learning latency. Comparing the results in Fig. 8 and Fig. 9, we can find that when the edge resource is very constrained in terms of bandwidth and computing capacity, the proposed CoEdge scheme is more capable of reducing the deep learning latency, because it well leverages the edge-cloud collaboration.

FIGURE 8. Variation of inference latency against edge size under the low-speed and high-capacity edge network.

FIGURE 9. Variation of inference latency against edge size under the low-speed and low-capacity edge network.

In summary, the above experimental results show that CoEdge is more resource-aware than the two baselines. The singleEdge baseline uses only a single edge device to share the DNN layers with cloud, and the fixedEdge baseline processes a DNN on an already-configured set of pipelined edge devices. Both baselines allocate DNN layers without considering how to exploit as many available edge resources as possible, which is the essential reason they perform worse than CoEdge does.

VI. CONCLUSION

In this paper, we have designed and implemented CoEdge, which exploits the communication and computing resources to deploy the deep learning task on the edge-cloud computing environment, in order to minimize the deep learning latency. CoEdge involves effective approaches to determining beneficial nodes, and can in polynomial time allocate the deep learning layers to these selected nodes, laying a foundation of implementing collaborative processing of the deep learning task. Our extensive simulation experiments with realistic setups also demonstrate that CoEdge outperforms two state-of-the-art schemes in term of latency. In the future, we will take a further step towards achieving latency-aware allocation of DNN layers in the edge-cloud system with dynamic edge resources in terms of communication and computation.

REFERENCES

[1] H. El-Sayed, S. Sarkar, M. Prasad, D. Puthal, A. Gupta, M. Mohanty, and C.-T. Lin, “Edge of things: The big picture on the integration of edge, IoT and the cloud in a distributed computing environment,” IEEE Access, vol. 6, pp. 1706–1717, 2017.
[2] H. Song, J. Bai, Y. Yi, J. Wu, and L. Liu, “Artificial intelligence enabled Internet of Things: Network architecture and spectrum access,” IEEE Comput. Intell. Mag., vol. 15, no. 1, pp. 44–51, Feb. 2020.
[3] J. Ren, H. Guo, C. Xu, and Y. Zhang, “Serving at the edge: A scalable IoT architecture based on transparent computing,” IEEE Netw., vol. 31, no. 5, pp. 96–105, Aug. 2017.
[4] P. G. Lopez, J. Cao, Q. Zhang, Y. Li, and L. Xu, “Edge-centric computing: Vision and challenges,” ACM SIGCOMM Comput. Commun. Rev., vol. 45, no. 5, pp. 37–42, 2015.
[5] K. Wang, H. Yin, W. Quan, and G. Min, “Enabling collaborative edge computing for software defined vehicular networks,” IEEE Netw., vol. 32, no. 5, pp. 112–117, Sep. 2018.

[6] R. Liao, H. Wen, J. Wu, F. Pan, A. Xu, Y. Jiang, F. Xie, and M. Cao, “Deep-learning-based physical layer authentication for industrial wireless sensor networks,” Sensors, vol. 19, no. 11, p. 2440, 2019.

[7] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, “Edge computing: Vision and challenges,” IEEE Internet Things J., vol. 3, no. 5, pp. 637–646, Oct. 2016.

[8] T. X. Tran, A. Hajisami, P. Pandey, and D. Pompli, “Collaborative mobile edge computing in 5G networks: New paradigms, scenarios, and challenges,” IEEE Commun. Mag., vol. 55, no. 4, pp. 54–61, Apr. 2017.

[9] F. Xie, H. Wen, J. Wu, S. Chen, W. Hou, and Y. Jiang, “Convolution based feature extraction for edge computing access authentication,” IEEE Trans. Netw. Sci. Eng., early access, Dec. 3, 2019, doi: 10.1109/TNSE.2019.2957323.

[10] S. K. Sharma and X. Wang, “Live data analytics with collaborative edge and cloud processing in wireless IoT networks,” IEEE Access, vol. 5, pp. 4621–4635, 2017.

[11] X. Lyu, H. Tian, L. Jiang, A. Vinel, S. Maharjan, S. Gjessing, and Y. Zhang, “Selective offloading in mobile edge computing for the green Internet of Things,” IEEE Netw., vol. 32, no. 1, pp. 54–60, Jan. 2018.

[12] K. Ha, Z. Chen, W. Hu, W. Richter, P. Pillai, and M. Satyanarayanan, “Towards wearable cognitive assistance,” in Proc. 12th Annu. Int. Conf. Mobile Syst., Appl., Services (MobiSys), 2014, pp. 68–81.

[13] I. Goodfellow, Y. Bengio, A. Courville, and F. Bacn, Deep Learning, Cambridge, MA, USA: MIT Press, 2013.

[14] E. Cuervo, A. Balasubramanian, D. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl, “MAUL: Making smartphones last longer with code offload,” in Proc. 8th Int. Conf. Mobile Syst., Appl., Services, 2010, pp. 49–62.

[15] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, “The case for VM-based clouds in mobile computing,” IEEE Pervasive Comput., vol. 8, no. 4, pp. 14–23, Oct. 2009.

[16] M. Gordon, D. Jamshidi, S. Mahlke, Z. Mao, and X. Chen, “COMET: Code offload by migrating execution transparently,” in Proc. 10th USENIX Symp. Operating Sys. Design Implement. (OSDI), 2012, pp. 93–106.

[17] B.-G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti, “CloneCloud: Elastic execution between mobile device and cloud,” in Proc. 6th Conf. Comput. Syst., 2011, pp. 301–314.

[18] S. Kosta, A. Aucinas, P. Hui, R. Mortier, and X. Zhang, “ThinkAir: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading,” in Proc. IEEE INFOCOM, Mar. 2012, pp. 945–953.

[19] X. Chen, Q. Shi, L. Yang, and J. Xu, “ThriftEdge: Resource-efficient edge computing for intelligent IoT applications,” IEEE Netw., vol. 32, no. 1, pp. 61–65, Jan. 2018.

[20] X. Chen, L. Pu, L. Gao, W. Wu, and D. Wu, “Exploiting massive D2D collaboration for energy-efficient mobile edge computing,” IEEE Wireless Commun., vol. 24, no. 4, pp. 64–71, Aug. 2017.

[21] X. Cheng, F. Lyu, W. Quan, C. Zhou, H. He, W. Shi, and X. Shen, “Space/Aerial-assisted computing offloading for IoT applications: A learning-based approach,” IEEE J. Sel. Areas Commun., vol. 37, no. 5, pp. 1117–1129, May 2019.

[22] J. Hauswald, Y. Kang, M. A. Laurenzano, C. Quan, and L. Tang, “Djinn and Tonic: DNN as a service and its implications for future warehouse scale computers,” in Proc. ACM/IEEE 42nd Annu. Int. Symp. Comput. Architecture (ISCA), Jun. 2015, pp. 23–40.

[23] T. Chen, Z. Du, N. San, W. Jia, C. Wu, Y. Chen, and G. Sun, “Diannao: A small-footprint high-throughput accelerator for ubiquitous machine-learning,” ACM SIGPLAN Notices, vol. 49, no. 4, pp. 269–284, 2014.

[24] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, and T. Darrell, “Caffe: Convolutional architecture for fast feature embedding,” in Proc. 22nd ACM Int. Conf. Multimedia, Nov. 2014, pp. 675–678.

[25] D. Liu, T. Chen, S. Liu, J. Zhou, S. Zhou, O. Teman, X. Feng, X. Zhou, and Y. and Chen, “PMLearning: A polyvalent machine learning accelerator,” ACM SIGARCH Comput. Archit. News, vol. 43, no. 1, pp. 369–381, 2015.

[26] C. Zhang, D. Wu, J. Sun, G. Sun, G. Luo, and J. Cong, “Energy-efficient CNN implementation on a deeply pipelined FPGA cluster,” in Proc. Int. Symp. Low Power Electron. Design (ISLPED), 2016, pp. 326–331.

[27] N. D. Lane, S. Bhattacharya, P. Georgiev, C. Forlivesi, L. Jiao, L. Qendro, and F. Kawsar, “DeepX: A software accelerator for low-power deep learning inference on mobile devices,” in Proc. 15th ACM/IEEE Int. Conf. Process. Sensor Netw. (IPSN), Apr. 2016.

[28] J. Zhou, Y. Wang, K. Ota, and M. Dong, “AAIoT: Accelerating artificial intelligence in IoT systems,” IEEE Wireless Commun. Lett., vol. 8, no. 3, pp. 825–828, Jun. 2019.

[29] H. Li, K. Ota, and M. Dong, “Learning IoT in edge: Deep learning for the Internet of Things with edge computing,” IEEE Netw., vol. 32, no. 1, pp. 96–101, Jan. 2018.

[30] A. Janiak, W. Janiak, and M. Lichtenstein, “Resource management in machine scheduling problems: A survey,” Decis. Making Manuf. Services, vol. 1, no. 2, pp. 59–89, Oct. 2007.

[31] D. Oron, D. Shabayt, and G. Steiner, “Approximation algorithms for the workload partition problem and applications to scheduling with variable processing times,” Eur. J. Oper. Res., vol. 256, no. 2, pp. 384–391, Jun. 2017.

[32] H. Wang and B. Alidaei, “Unrelated parallel machine selection and job scheduling with the objective of minimizing total workload and machine fixed costs,” IEEE Trans. Autom. Sci. Eng., vol. 15, no. 4, pp. 1955–1963, Oct. 2018.

[33] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” in Proc. Adv. Neural Inf. Process. Syst., 2012, pp. 1097–1105.

[34] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” 2014, arXiv:1409.1556. [Online]. Available: http://arxiv.org/abs/1409.1556.

[35] A. Coates, B. Huval, T. Wang, D. Wu, B. Catanzaro, and A. Ng, “Deep learning with COTS HPC systems,” in Proc. Int. Conf. Mach. Learn., 2013, pp. 1337–1345.

[36] Z. Zeng, K. Cao, and M. Zhang, “MobileDeepPill: A small-footprint mobile deep learning system for recognizing unconstrained pill images,” in Proc. 15th Annu. Int. Conf. Mobile Syst., Appl., Services, 2017, pp. 56–67.

LIANYONG REN received the B.E. degree in computer science and technology from the School of Information Science and Technology, Beijing Forestry University, Beijing, China, in 2018. She is currently pursuing the master’s degree in software engineering with Beijing Forestry University. Her current research interests include edge computing, deep learning, and mobile computing.

GUODONG SUN (Member, IEEE) was a Post-doctoral Researcher at Tsinghua University, China, before joining the Faculty of Beijing Forestry University. He was a Visiting Professor of computer science with North Carolina University at Charlotte, USA. He is currently an Associate Professor of computer science with the School of Information Science and Technology, Beijing Forestry University, Beijing, China. His research interests include mobile computing, wireless ad-hoc and sensor networks, combinatorial optimization, the Internet of Things, and machine learning. He is a member of the IEEE Computer Society.

YANLONG REN received the B.E. and M.E. degrees from the School of Information Science and Technology, Beijing Forestry University, Beijing, China, in 2013 and 2015, respectively. He is currently a Senior Engineer with the Network Information Management and Service Center, Beijing University of Civil Engineering and Architecture, Beijing. His major efforts are put to maintaining the large-scale network infrastructure and the cloud-based data center of his university.

* * *