Online Scheduling Unbiased Distributed Learning over Wireless Edge Networks

Ziyi Han*, Ruiting Zhou*, Jinlong Pang*, Haisheng Tan‡, Yue Cao*
*School of Cyber Science and Engineering, Wuhan University, China
‡School of Computer Science and Technology, University of Science and Technology of China, China
Email: {ziyihan, ruitingzhou, jinlongpang, yue.cao}@whu.edu.cn, hstan@ustc.edu.cn

***Abstract*—To realize high quality smart IoT services, such as intelligent video surveillance in Auto Driving and Smart City, tremendous amount of distributed machine learning jobs train unbiased models in wireless edge networks, adopting the parameter server (PS) architecture. Due to the large datasets collected geo-distributedly, the training of unbiased distributed learning (UDL) brings high response latency and bandwidth consumption. In this paper, we propose an online scheduling algorithm, *Okita*, to minimize both the latency cost and bandwidth cost in UDL. *Okita* schedules UDL jobs at each time slot to jointly decide the execution time window, the amount of training data, the number and the location of concurrent workers and PSs in each site. To evaluate the practical performance of *Okita*, we implement a testbed based on Kubernetes. Extensive experiments and simulations show that *Okita* can reduce up to 60% of total cost, compared with the state-of-the-art schedulers in cloud systems.**

***Index Terms*—Distributed Learning Job, Wireless Edge Network, Online Scheduling**

I. Introduction

The breakthrough in 5G and edge computing has promoted the development of smart IoT applications, *e.g.,* intelligent video surveillance, smart city and autonomous driving. Traditional machine learning (ML) job usually arrives with its own data and trains centrally in the remote cloud. While the training of such IoT services prefers operating at the edge of network to support real-time inferences and responses, aiming to reduce response latency and protect privacy [1], [2], [3]. Furthermore, to guarantee the quality of service, it is crucial to train an *unbiased* [4] model with a sufficient amount of data selected from each geo-distributed site uniformly *to represent the data source* in the wireless edge network [5], [6]. We define such jobs as *unbiased distributed learning* (UDL) jobs. For example, to support smart traffic management system, cameras installed in each intersection collect real-time road traffic data, and then upload to an edge cloudlet for processing [7], [8]. The training of UDL usually adopts *data parallelism* with the parameter server (PS) architecture at the edge [5], [9]. In each training iteration, the workers (implemented as containers or virtual machines) compute gradients and send to PSs. The PSs then average gradients to update model parameters and push back to all workers.

However, the training of UDL at the edge is quite nontrivial. *First*, there exists a trade-off between response latency and bandwidth consumption. To reduce the response latency, one tends to centralize all data to one place. The centralized approach can speed up training by eliminating the communication time between workers and PSs, but consumes large amounts of cross-site wireless bandwidth which is costly and scarce at the edge [2], [3]. *Second*, considering heterogeneous sensitivity of jobs to response latency, the scheduling order of jobs will affect the total response latency. Therefore, given heterogeneous resources and data volumes in different edge sites, a challenging problem exists: *how to decide the training workload in each site, and place the workers and PSs, such that the bandwidth cost (for data transmission and parameter exchange) and the latency cost (total training time) are minimized.*

The scheduling algorithms commonly adopted in today's cloud systems [10], [11] follow simple strategies, *e.g.,* First In First Out (FIFO) or Dominant Resource Fairness Scheduling (DRF), and require job owners to provide the exact number of workers/PSs and the training time. Researches on the edge network mainly study task offloading problems [12], [13], [14] or develop new training algorithms [15], [16], [17]. Different from existing literature, our work is the *first formal study of scheduling design for UDL jobs in wireless edge networks*. We capture the distinct features of UDL jobs (training data is geo-distributed over sites and frequent communication happens between workers and PSs) while modeling the problem and propose an online scheduling algorithm, *Okita*, to minimize both response latency and bandwidth consumption. Our main contribution can be summarized as follows.

- We formulate the total cost (the sum of latency cost and bandwidth cost) minimization problem into a mixed integer nonlinear program (MINLP) which is proven to be NP-hard even in the offline setting. The challenge further escalates when UDL jobs arrive online.
- To handle the above problem, we propose an online algorithm *Okita*. *Okita* first preprocesses unfinished jobs to compute their tentative optimal schedule independently and then determines the deployment of workers and PSs based on a well-designed priority. When there are not

Corresponding author: Ruiting Zhou and Haisheng Tan.
This work is supported in part by the NSFC Grants (62072344, 61772489 and U20A20177), Hubei Science Foundation (2020CFB195) and Compact Exponential Algorithm Project of Huawei (YBN2020035131).

enough resources, *Okita* selects the placement scheme with the lower cost in migrating deployment and delaying training for the low-priority job. Next, *Okita* matches the training data with the number of assigned workers in each site, and transfers data if necessary.

- We evaluate practical effectiveness of *Okita* through testbed experiments and simulation studies, and obtain some promising results: (i) *Okita* achieves a near-optimal solution with a small competitive ratio (< 1.8); (ii) *Okita* reduces up to $20\% \sim 65\%$ of the total cost, compared to three benchmarks; (iii) The running time of *Okita* remains at a low level (< 50s); (iv) *Okita* can complete the training of UDL jobs with desired model accuracy;

In the rest of the paper, we review the related work in Sec. II. We formulate the system model in Sec. III. Sec. IV presents the online scheduling algorithm for UDL. We evaluate the proposed algorithm in Sec. V. Sec. VI concludes the paper.

II. Related Work

Distributed Learning in Edge/Cloud. Lyu *et al.* [5] develop a new online approach to partition streaming data evenly under time-varying network conditions to speed up the training. Hu *et al.* [15] propose a parameter synchronization scheme for heterogeneous edge systems to eliminate significant waiting time while ensuring model convergence. Amiri *et al.* [16] focus on power- and bandwidth-limited multiple access channel properties of collaborative ML at the edge. Wang *et al.* [17] propose a control algorithm to dynamically adapt the frequency of global aggregation to achieve a balance between communication and computation. The above work mainly studies the performance of training model at the edge, rather than scheduling jobs to reduce the total training cost.

Scheduling and Placement in Edge. Tun *et al.* [12] study the problem of optimizing task offloading and resource allocation to minimize total energy consumption while meeting the latency requirements of devices. Considering energy-constrained mobile users, Saleem *et al.* [13] propose a scheduling scheme to minimize the total task execution latency. Alameddine *et al.* [14] consider the joint problem of task offloading and scheduling for latency-sensitive tasks to maximize the number of admitted tasks in a fixed time span. You *et al.* [18] study the online resource provisioning for jobs with distributed workload across edge cloudlet networks to achieve minimizing the total cost. The above work allocates resources for tasks by using virtual machine (VM) or cloud container instead of workers and PSs, which is not suitable for ML applications.

III. System Model

A. System Overview

Unbiased Distributed Learning System. As shown in Fig. 1, we consider an edge distributed learning system with a number of geo-distributed "sites", denoted as a set $\mathcal{R}$, where a "site" can be an edge server or an edge cloudlet. Each site collects and supplies training data for UDL jobs to ensure the unbiasedness of ML model [5]. Let D_i^r denote the training data size UDL job i selected in site r. Here, we assume that the minimum unit of data is data chunk, *i.e.*, the number of data chunks for job i in site r is D_i^r. Moreover, each site r can provide C_r^k units type-k resource, and connects with other sites through wireless network. K represents the number of resource types, including GPU, CPU, disk storage and memory. Over a large time span T, I UDL jobs arrive randomly online and request for training. Each job i needs to be trained E_i epochs. Let $\mathcal{X}$ denote the integer set $\{1, 2, \ldots, X\}$, *e.g.,* $\mathcal{I}$ represents the integer set $\{1, 2, \ldots, I\}$.

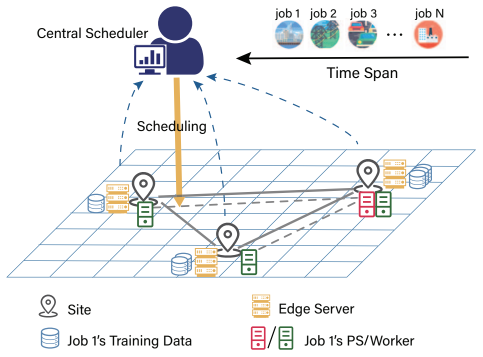

Fig. 1: The system of edge distributed learning.

Parameter Server Architecture. To reduce training cost, instead of centralized training [19], we choose to use *data parallel* training with the PS architecture [20]. We assume that only one PS allocated for each UDL job, which can represent multiple PS instances deployed on the same site in practice. Workers and PSs are all implemented as VMs or containers and are customized for each UDL job. For each UDL job $i \in \mathcal{I}$, a worker (PS) requires m_i^k (n_i^k) units of type-k resource, and let ρ_i denote the processing capacity of job i's worker, *i.e.,* a worker of job i can process ρ_i data chunks at one time slot. Due to the unreliability of wireless channels and the scarcity of wireless resources, the bandwidth resource is not pre-allocated.

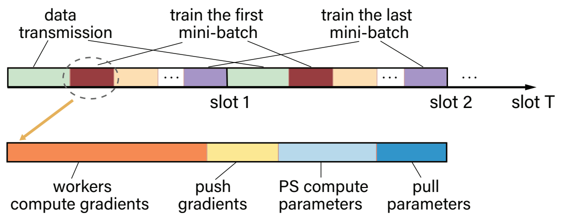

Fig. 2: Training workflow in one time slot.

Training Workflow. To repeat the training on the input dataset in each time slot, we assume that the length of one time slot is much longer than the duration of a training epoch, *e.g.,* one time slot can be half an hour or longer. Moreover, to ensure the convergence of ML model, synchronous training [21] which can achieve higher model accuracy than asynchronous training [22] is adopted. Fig. 2 shows the training workflow in one time slot. The beginning of each time slot is data transmission phase, during which data is transferred from the original site to other sites for training in this time slot. In the training process, each data chunk is divided into equal-sized mini-batches. Next is the training phase, where all allocated mini-batches are trained one by one. Workers compute model gradients based on an input mini-batch and upload to PSs. Each PS updates parameters and sends back to all workers. Workers continue to compute gradients with the next min-batch until all mini-batches have been trained for the required number of epochs. Important notations are listed in Table I.

Decision Variables. After UDL job i arrives at time slot a_i, the decisions made by the central scheduler at each time slot include: i) $y_i^r(t)$, the number of allocated workers for job i in site r at t; ii) $s_i^r(t)$, a binary variable which represents whether job i's PS is placed in site r at t or not; iii) $g_i^{rr'}(t)$, the amount of training data of job i transferred from site r to r' at t.

B. Problem Formulation

Overall Cost. As a key metric to measure the quality of service (QoS), the response latency of IoT applications is a main focus for edge learning, [1], [2], [3]. In this paper, we convert the response latency into a non-decreasing cost function $f_i(t_i - a_i)$ where t_i represents the job i's completion time. In the scenario of geo-distributed sites, the bandwidth cost is far larger than the computation cost of ML model training and the deployment cost of workers and PSs. Therefore, we only focus on the bandwidth cost which includes data transfer cost and parameter exchange cost and can be defined as follows.

$$B_i = \sum_{t\in\mathcal{T}}\sum_{r\in\mathcal{R}}\sum_{r'\in\mathcal{R}} q^{rr'}(\delta_i g_i^{rr'}(t) + \pi_i y_i^r(t) s_i^{r'}(t)), \qquad (1)$$

where $q^{rr'}$ denotes the unit cost of bandwidth (per MB) between site r and r' ($q^{rr'} = 0$ when $r' = r$), δ_i indicates the size (MB) of a data chunk for job i, and π_i represents the size (MB) of the parameters exchanged between a worker and a PS of job i per time slot. Specially, when workers and PS are deployed in the same site, we ignore their parameter exchange cost. Therefore, the total cost of training job i is $f_i(t_i - a_i) + B_i$.

Problem Formulation. The online cost minimization problem is formulated as follows.

$$\text{minimize} \quad \sum_{i\in\mathcal{I}}(f_i(t_i - a_i) + B_i) \qquad (2)$$

$$\text{subject to:} \quad \sum_{t\in\mathcal{T}}\sum_{r\in\mathcal{R}}\sum_{r'\in\mathcal{R}} g_i^{rr'}(t) = \sum_{r\in\mathcal{R}} D_i^r, \quad \forall i \in \mathcal{I}, \quad (2a)$$

$$\rho_i y_i^{r'}(t) \geq E_i \sum_{r\in\mathcal{R}} g_i^{rr'}(t), \quad \forall i \in \mathcal{I}, \forall r' \in \mathcal{R}, \forall t \in \mathcal{T}, \quad (2b)$$

$$\sum_{r\in\mathcal{R}} s_i^r(t) = 1, \quad \forall i \in \mathcal{I}, \forall t \in \mathcal{T}, \quad (2c)$$

$$\sum_{i\in\mathcal{I}}(m_i^k y_i^r(t) + n_i^k s_i^r(t)) \leq C_r^k, \quad \forall k \in \mathcal{K}, \forall r \in \mathcal{R}, \forall t \in \mathcal{T}, \quad (2d)$$

$$t_i = \arg\max_{t\in\mathcal{T}}(\sum_{r\in\mathcal{R}} y_i^r(t) > 0), \quad \forall i \in \mathcal{I}, \quad (2e)$$

$$y_i^r(t), g_i^{rr'}(t) \in \{0, 1, 2, \dots\}, \quad \forall i \in \mathcal{I}, \forall r, r' \in \mathcal{R}, \forall t \in \mathcal{T}, \quad (2f)$$

$$s_i^r(t) \in \{0, 1\}, \quad \forall i \in \mathcal{I}, \forall r \in \mathcal{R}, \forall t \in \mathcal{T}. \quad (2g)$$

Constraint (2a) guarantees that job i's amount of transferred data is equal to the training workload it requires. Specially, $g_i^{rr}(t)$ represents the amount of local training data which does not need to be transferred in site r at t. Constraint (2b) ensures that the workers of job i deployed in site r' at t can process E_i epochs of allocated data. Constraint (2c) means that only one PS is allocated for each job. The resource capacity of sites for worker and PS deployment is formulated by constraint (2d). Constraint (2e) defines the completion time of job i.

Challenges. In the above optimization problem, there is a trade-off between latency cost and bandwidth cost. Moreover, problem (2) is non-convex and multi-objective, which involves three sets of integer variables and unconventional constraint (2e). Integer linear programming (ILP) (known to be NP-hard [23], [24]) can be reduced to it. Therefore, even in the offline setting, problem (2) is NP-hard. We further need to determine the deployment of workers and PSs, and data transmission under the online setting, without any future knowledge.

TABLE I: List of Notations

I	# of jobs	T	# of time slots
a_i	job i's arrival time	t_i	job i's completion time
C_r^k	capacity of type-k resource in site r		
D_i^r	# of data chunks for job i in site r		
$y_i^r(t)$	# of job i's workers in site r at t		
$s_i^r(t)$	whether job i's PS is deployed in site r at t		
ρ_i	processing capacity of job i's worker		
$q^{rr'}$	the unit cost of bandwidth (per MB) between site r and r'		
δ_i	the size (MB) of a data chunk for job i		
π_i	the size (MB) of the parameter exchanged between a worker and a PS of job i per time slot		
$g_i^{rr'}(t)$	the amount of training data of job i transferred from site r to site r' at t		

IV. Online Scheduling Algorithm Design

A. Algorithmic Idea

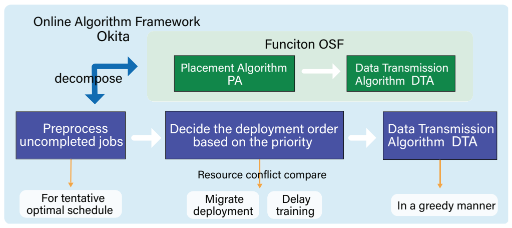
Online Algorithm Framework
Okita
decompose
Funciton OSF
Placement Algorithm PA
Data Transmission Algorithm DTA
Preprocess uncompleted jobs
Decide the deployment order based on the priority
Data Transmission Algorithm DTA
Resource conflict compare
For tentative optimal schedule
Migrate deployment
Delay training
In a greedy manner

Fig. 3: The structure of *Okita*.

In order to solve the cost minimization problem (2), we propose an effective heuristic algorithm *Okita* to schedule UDL jobs. We observe that it is hard to satisfy two objectives simultaneously. To address it, we assume that when preprocessing UDL jobs, the deployment of workers and PS remains unchanged during the training time window, to simplify the computation of job completion time. The structure of our algorithm is shown in Fig. 3. At each time slot t, *Okita* performs the following computation:

i. *Okita* first collects newly arrived jobs at t. Then it preprocesses uncompleted jobs, *i.e.,* enumerates the possible number of workers at each time slot to handle the non-conventional constraint (2e) and decompose the problem into a series of one-shot problems (solved by the function *OSF* in Alg. 1) for computing tentative optimal schedule without considering other jobs.

ii. For each one-shot problem of job i with the fixed number of workers at each time slot (call function *OSF* in Alg. 1), *Okita* further determines the number of workers in each site based on a well-designed metric (*i.e.,* $Q_i^r(t)$), and the placement of PSs (call algorithm *PA* in Alg. 2). Then the training workload in each site and the necessary data transmission with unchanged deployment are computed (call algorithm *DTA* in Alg. 3).
iii. *Okita* then decides the deployment order based on a well-designed priority (*i.e.,* $O_i(t)$). When resource conflict occurs (*i.e.,* there are not enough resources), *Okita* computes the cost of migrating deployment (call funciton *OSF* in Alg. 1) and delaying training for the low-priority job, and selects the smaller one.
iv. Next, *Okita* computes the amount of training data in all sites at t (*i.e.,* $d_i^r(t)$), based on the number of assigned workers on sites. If the local training data can not satisfy the required training workload, data transmission will happen. In this regard, *Okita* gathers data from other sites greedily based on the principle of the lowest bandwidth cost (call algorithm *DTA* in Alg. 3).
v. Finally, *Okita* checks whether the job is completed through the amount of remaining data (*i.e.,* $M_i^r(t)$).

B. Online Algorithm Okita

1) Algorithm Framework: ***Priority Metric.*** We define a priority metric $O_i(t)$ to capture job i's sensitivity to response latency. Let $M_i^r(t)$ denote the amount of remaining training data of job i in site r at t, which is formulated as follows.

$$M_i^r(t) = D_i^r - \sum_{t':t'<t} \sum_{r' \in \mathcal{R}} g_i^{rr'}(t'), \tag{3}$$

where the second term denotes the total amount of training data transferred from site r before t. Let $M_i(t) = \sum_{r \in \mathcal{R}} M_i^r(t)$, *i.e.,* the sum of $M_i^r(t)$ in all sites at t. $O_i(t)$ is expressed as:

$$O_i(t) = \alpha_1 \frac{\tau_i \cdot (t - a_i)}{\max_{i \in \mathcal{J}_a}(\tau_i \cdot (t - a_i))} - \alpha_2 \frac{M_i(t)}{\sum_{r \in \mathcal{R}} D_i^r}, \tag{4}$$

where the first term denotes the weight of the factors related to job i's latency cost function, τ_i is the parameter of the latency cost function and $t - a_i$ is the waiting time. The second term means the weight of the remaining training data of job i. In Eq. 4, α_1 and α_2 are proportional factors. The larger value of $O_i(t)$ means job i is more sensitive to latency, has longer waiting time, and can complete training faster. Therefore, when resource conflict happens, it has higher priority.

Algorithm Details. Our online algorithm framework *Okita* is presented in Alg. 1. *Okita* first adds the newly arrived job to uncompleted job set $\mathcal{J}_a$ in line 3. Lines 4-10 preprocess the jobs in $\mathcal{J}_a$, *i.e.,* enumerate the possible number of workers at each time slot by a *for* loop in lines 5-8. Lines 6-7 compute the running time and the schedule with minimum bandwidth cost (call *OSF*) under the fixed $N_i(t)$. By compare the cost, the tentative optimal schedule can be obtained (line 9). Lines 11-12 calculate the priority $O_i(t)$ and sort jobs in a decreasing order of $O_i(t)$. Then *Okita* determines the placement scheme of jobs. If there are not enough resources (line 14), compute the cost of migrating deployment (call *OSF*) (lines 15-16) and delaying training (where $\tilde{i}$ denotes the job that has deployed worker or PS in sites with insufficient resources) (line 17) for the low-priority job, and select the smaller one (lines 18-20). The amount of training data at each site $d_i^r(t)$ matches the number of assigned workers and necessary data transmission is computed by the subroutine *DTA* in Alg. 3 (lines 22-23). Line 24 saves the results and updates the available resources. Finally, if job is completed (*i.e.,* $M_i(t) = 0$), compute the total cost and move the job out of set $\mathcal{J}_a$ (lines 25-28).

Algorithm 1 Online Algorithm Framework *(Okita)*

```
Input: $T$, $D_i^r$, $m_i^k$, $n_i^k$, $q^{rr'}$, $E_i$, $\forall k \in \mathcal{K}, \forall r, r' \in \mathcal{R}, \forall i \in \mathcal{I}$;
Output: $\mathcal{L}$, $\{t_i\}_{\forall i \in \mathcal{I}}$, $min\_cost$;
 1: Initialize $\mathcal{L}_i = \varnothing, min\_cost = 0, bcost_i^* = 0, \mathcal{J}_a = \varnothing, y_i^r(t) = 0, s_i^r(t) = 0, d_i^r(t) = 0, Av_r^k(t) = C_r^k, \forall r \in \mathcal{R}, \forall i \in \mathcal{I}, \forall t \in \mathcal{T}$;
 2: for $t = 1$ to $T$ do
 3:     Collect jobs whose $a_i = t$ and add these jobs into set $\mathcal{J}_a$;
 4:     for all $i \in \mathcal{J}_a$ do
 5:         for $N_i(t) = 1$ to $\lceil E_i M_i(t)/\rho_i \rceil$ do  /*# of workers*/
 6:             $l_i = \lceil E_i M_i(t)/\rho_i/N_i(t) \rceil$;   /*execution time*/
 7:             $(bcost, \{\hat{y}_i^r(t)\}_{\forall r \in \mathcal{R}}, \{\hat{s}_i^r(t)\}_{\forall r \in \mathcal{R}}) = OSF(i, t, l_i, \{C_r^k\}_{\forall r \in \mathcal{R}, \forall k \in \mathcal{K}}, N_i(t))$;
 8:         end for
 9:         Compare $bcost + f_i(t + l_i - a_i)$ at different $N_i(t)$ and save the schedule of smallest one to $l_i^*$, $N_i^*(t)$, $y_i^r(t)$, $s_i^r(t)$;
10:     end for
11:     Calculate the quality $\{O_i(t)\}_{\forall i \in \mathcal{J}_a}$ by using Eq. 4;
12:     Sort jobs in $\mathcal{J}_a$ in decreasing order of $O_i(t)$ to $j_1$, $j_2$, $\cdots$;
13:     for $i = j_1, j_2, \cdots$ do
14:         if $Av_r^k(t) - (m_i^k y_i^r(t) + n_i^k s_i^r(t)) < 0$ then  /*conflict*/
15:             $(bcost1, \{y_i^r(t)\}_{\forall r \in \mathcal{R}}, \{s_i^r(t)\}_{\forall r \in \mathcal{R}}, ) = OSF(i, t, l_i^*, \{Av_r^k(t)\}_{\forall r \in \mathcal{R}, \forall k \in \mathcal{K}}, N_i^*(t))$;   /*migrate*/
16:             $cost1 = bcost_i^* + bcost1 + f_i(t + l_i^* - a_i)$;
17:             $cost2 = cost_i - f_i(t + l_i^* - a_i) + f_i(t + l_{\tilde{i}}^* + l_i^* - a_i)$;
18:             if $cost1 > cost2$ then
19:                 $y_i^r(t) = 0, s_i^r(t) = 0, \forall r \in \mathcal{R}$;  /*delay training*/
20:             end if
21:         end if
22:         Calculate $\{d_i^r(t)\}_{\forall r \in \mathcal{R}}$ based on $\{y_i^r(t)\}_{\forall r \in \mathcal{R}}$;
23:         $(bcost2, \{g_i^{rr'}(t)\}_{\forall r, r' \in \mathcal{R}}) = DTA(i, t, \{d_i^r(t)\}_{\forall r \in \mathcal{R}})$;
24:         $bcost_i^*$ plus the bandwidth cost of $t$, add schedule into $\mathcal{L}_i$, and update the available resources $Av_r^k(t)$;
25:         if $M_i(t) = 0$ then     /*job finished*/
26:             $min\_cost = bcost_i^* + f_i(t - a_i) + min\_cost$;
27:             $t_i = t$, $\mathcal{L} = \mathcal{L} \cup \mathcal{L}_i$, remove $i$ from set $\mathcal{J}_a$;
28:         end if
29:     end for
30: end for
31: Return $\mathcal{L}$, $\{t_i\}_{\forall i \in \mathcal{I}}$, $min\_cost$;

32: function $OSF(i, t, l_i^*, \{U_r^k\}_{\forall r \in \mathcal{R}, \forall k \in \mathcal{K}}, N_i(t))$
33:     Initialize $y_i^r(t) = 0, s_i^r(t) = 0, d_i^r = 0, \forall r \in \mathcal{R}$;
34:     $(bcost1, \{y_i^r(t)\}_{\forall r \in \mathcal{R}}, \{s_i^r(t)\}_{\forall r \in \mathcal{R}}) = PA(i, t, l_i, \{U_r^k\}_{\forall r \in \mathcal{R}, \forall k \in \mathcal{K}}, N_i(t))$;
35:     Calculate $\{d_i^r\}_{\forall r \in \mathcal{R}}$ based on $\{y_i^r(t)\}_{\forall r \in \mathcal{R}}$;
36:     $(bcost2, \{g_i^{rr'}(t)\}_{\forall r, r' \in \mathcal{R}}) = DTA(i, t, \{d_i^r\}_{\forall r \in \mathcal{R}})$;
37:     Return $cost = bcost1 + bcost2$, $\{y_i^r(t)\}_{\forall r \in \mathcal{R}}$, $\{s_i^r(t)\}_{\forall r \in \mathcal{R}}$;
38: end function
```

Function *OSF* computes the schedule with minimum bandwidth cost under the fixed number of workers at t (lines 32-38). Line 34 computes the deployment of workers and PS by the subroutine *PA* in Alg. 2. *OSF* then determines the amount of training data in all sites d_i^r based on the number of assigned workers and computes data transmission by *DTA* (lines 35-36). Line 37 calculates the bandwidth cost and returns the results.

2) Placement of Workers and PSs: For each one-shot problem with the fixed total number of workers, the specific placement of workers and PS needs to consider both the available resources and the bandwidth cost. The sub-problem of deploying workers and PS can be formulated as follows.

$$\text{minimize} \quad \sum_{r\in\mathcal{R}}\sum_{r'\in\mathcal{R}} q^{rr'}\pi_i y_i^r(t) s_i^{r'}(t) \tag{5}$$

$$\text{subject to:} \quad \sum_{r\in\mathcal{R}} y_i^r(t) \geq N_i(t), \tag{5a}$$

$$m_i^k y_i^r(t) + n_i^k s_i^r(t) \leq u_r^k(t), \quad \forall k\in\mathcal{K}, \forall r\in\mathcal{R}, \tag{5b}$$

$$\sum_{r\in\mathcal{R}} s_i^r(t) = 1, \tag{5c}$$

$$y_i^r(t) \in \{0,1,2,\dots\}, \quad \forall r\in\mathcal{R}, \tag{5d}$$

$$s_i^r(t) \in \{0,1\}, \quad \forall r\in\mathcal{R}. \tag{5e}$$

Constraint (5a) ensures that the number of workers deployed for job i at t is not less than the required. Constraints (5b) and (5c) are equivalent to constraints (2b) and (2c), respectively. Variable $u_r^k(t)$ in constraint (5b) denotes the available type-k resource at t, *i.e.*, $u_r^k(t) = C_r^k - \sum_{i':a_{i'}\leq a_i}(m_{i'}^k y_{i'}^r(t) + n_{i'}^k s_{i'}^r(t))$.

The above sub-problem (5) is an ILP. To solve this problem, we enumerate all possible positions of PS and deploy workers based on an efficient greedy algorithm.

Algorithm 2 Placement Algorithm *(PA)*

```
Input: $i$, $t$, $l_i$,$\{U_r^k\}_{\forall r\in\mathcal{R},\forall k\in\mathcal{K}}$, $N_i$;
Output: $cost$, $\{\hat{y}_i^r(t)\}_{\forall r\in\mathcal{R}}$, $\{\hat{s}_i^r(t)\}_{\forall r\in\mathcal{R}}$;
 1: Initialize $cost = \infty$, $\hat{y}_i^r(t) = 0$, $\hat{s}_i^r(t) = 0$, $\forall r\in\mathcal{R}$;
 2: for site $\tilde{r}$ in $\mathcal{R}$ do /*deploy PS*/
 3:     $y_i^r(t) = 0$, $s_i^r(t) = 0$, $s_i^{\tilde{r}}(t) = 1$, $\forall r\in\mathcal{R}$;
 4:     for all site $r\in\mathcal{R}$ do
 5:         Calculate the quality $Q_i^r(t)$ by using Eq. (6);
 6:         $A_i^r(t) = \min_{k\in\mathcal{K}}\lfloor\frac{U_r^k - n_i^k\mathbb{I}(r=\tilde{r}))}{m_i^k}\rfloor$;
 7:     end for
 8:     Sort sites in $\mathcal{R}$ (include $\tilde{r}$) according to $Q_i^r(t)$ in decreasing
        order into $r_1$, $r_2$, $\cdots$, $r_R$;
 9:     for $j = 1$ to $R$ do /*deploy worker*/
10:         $y_i^{r_j}(t) = \min\{A_i^{r_j}(t), N_i - \sum_{j'=1}^{j-1} y_i^{r_{j'}}(t)\}$;
11:     end for
12:     if $l_i\sum_{r\in\mathcal{R}} q^{r\tilde{r}}\pi_i y_i^r(t) \leq cost$ and $\sum_{r\in\mathcal{R}} y_i^r(t) \geq N_i$ then
13:         $cost = l_i\sum_{r\in\mathcal{R}} q^{r\tilde{r}}\pi_i y_i^r(t)$, $\hat{y}_i^r(t) = y_i^r(t)$, $\hat{s}_i^r(t) =$
        $s_i^r(t)$, $\forall r\in\mathcal{R}$;
14:     end if
15: end for
16: Return $cost$, $\{\hat{y}_i^r(t)\}_{\forall r\in\mathcal{R}}$, $\{\hat{s}_i^r(t)\}_{\forall r\in\mathcal{R}}$;
```

Deployment Metric. We design a metric $Q_i^r(t)$ for the deployment of workers to minimize the parameter exchange cost which is formulated as follows.

$$Q_i^r(t) = \beta_1\frac{\sum_{k\in\mathcal{K}} U_r^k/C_r^k}{K} - \beta_2\frac{\sum_{r'\in\mathcal{R}} q^{rr'}}{R-1} + \beta_3\frac{M_i^r(t)}{D_i^r}, \tag{6}$$

where the first term means the weight of the average available resources among K types. U_r^k is the input variable of the algorithm and denotes the available type-k resource in site r. The second term indicates the weight of the average unit bandwidth cost from site r to other sites. The third term denotes the weight of the remaining training data in site r. In Eq. 6, β_1, β_2 and β_3 are proportional factors. The larger value of $Q_i^r(t)$ means site r has more available resources, smaller bandwidth cost, and more local untrained data. So we select sites to deploy workers in a greedy manner based on $Q_i^r(t)$.

Algorithm Details. Our placement algorithm *PA* is shown in Alg. 2. To handle the binary variable $s_i^r(t)$ in the constraint (5e), *PA* enumerates all positions of PS by a *for* loop in lines 2-15. Let $A_i^r(t)$ represent the maximum number of assignable workers for job i in site r at t. Lines 4-7 calculate $Q_i^r(t)$ and $A_i^r(t)$ for each site. *PA* then sorts all sites in decreasing order of $Q_i^r(t)$ (line 8) and places workers as much as possible while not violating the constraints (5a) and (5b) (lines 9-11). lines 12-14 compare the cost and update schedule when the required number of workers is met. Line 16 returns the results.

3) Online Data Transmission: After determining the number of training data at each site (*i.e.*, $d_i^r(t)$), we need to consider how to transfer data among sites with lowest bandwidth cost if necessary. The sub-problem of data transmission can be formulated as follows.

$$\text{minimize} \quad \sum_{r\in\mathcal{R}}\sum_{r'\in\mathcal{R}} q^{rr'} g_i^{rr'}(t) \tag{7}$$

subject to:

$$\sum_{r\in\mathcal{R}:r\neq r'} g_i^{rr'}(t) + M_i^{r'}(t-1) \geq d_i^{r'}(t), \quad \forall r'\in\mathcal{R}, \tag{7a}$$

$$g_i^{rr'}(t) \in \{0,1,2,\dots\}, \quad \forall r,r'\in\mathcal{R}, \tag{7b}$$

Constraint (7a) guarantees that the amount of data collected from other sites (including the local remaining data) is not less than the required workload at t. Obviously, the sub-problem (7) is NP-hard (Min Knapsack problem can be reduced to it) [23], [24]. To solve ILP (7), we propose a heuristic algorithm.

Algorithm 3 Data Transmission Algorithm *(DTA)*

```
Input: $i$, $t$, $\{d_i^r(t)\}_{r\in\mathcal{R}}$;
Output: $cost$, $\{g_i^{rr'}(t)\}_{\forall r,r'\in\mathcal{R}}$;
 1: Initialize $g_i^{rr'}(t) = 0$, $cost = 0$, $\forall r,r'\in\mathcal{R}$;
 2: for site $r$ in $\mathcal{R}$ do
 3:     if $H_i^r(t) < 0$ then
 4:         Sort sites in $\mathcal{R}$ according to $q^{rr'}$ in non-decreasing order
        into $r_1$, $r_2$, $\cdots$, $r_R$;
 5:         for $j = 1$ to $R$ do
 6:             if $H_i^{r_j}(t) > 0$ then
 7:                 $g_i^{r_j r}(t) = \min\{H_i^{r_j}(t) - \sum_{r\in\mathcal{R}} g_i^{r_j r}(t)$,
        $-H_i^r(t) - \sum_{j=1}^{j-1} g_i^{r_j r}(t)\}$;
 8:             end if
 9:         end for
10:     end if
11: end for
12: $cost = \sum_{r\in\mathcal{R}}\sum_{r'\in\mathcal{R}} q^{rr'}\delta_i g_i^{rr'}(t)$;
13: Return $cost$, $\{g_i^{rr'}(t)\}_{\forall r,r'\in\mathcal{R}}$;
```

Algorithm Details. As shown in Alg. 3, it is our data transmission algorithm *DTA*. Let $H_i^r(t)$ denote the remaining data after subtracting the corresponding allocated training workload in site r at t, *i.e.*, $H_i^r(t) = M_i^r(t) - d_i^r(t)$. In lines 3-12, the *for* loop is used to check whether the site has insufficient training data, if no (*i.e.*, $H_i^r(t) < 0$), data transmission needs to be carried out. In line 5, all sites are sorted in increasing order based on the unit bandwidth cost. *DTA* greedily collects data from the sites which still have affluent data meanwhile guaranteeing not to over-provisioning (lines 6-10). In fact, data transmission will not always occur throughout the training process. Due to the heterogeneity of initial data in different sites (D_i^r), data transmission usually happens at the end of training process.

C. Theoretical Analysis

Theorem 1. *(Correctness): Our online algorithm Okita in Alg. 1 (including Alg. 2 and Alg. 3) computes a feasible solution for optimization problems (2), (5) and (7).*

All missing proofs are in our technical report [25].

Theorem 2. *(Time complexity): Okita runs in polynomial time, withtime complexity* $O(TI(\frac{D_i}{E_i}R^2\log(R+K)+\log I)$.

V. Evaluation

In this section, we evaluate the performance of *Okita* through testbed experiments (in Sec. V-A) and simulation studies (in Sec. V-B, Sec. V-C).

A. Testbed Experiments

Implementation. We implement a MXNet-based [26] distributed ML system with Kubernetes 1.19 [27]. The testbed is built with 9 servers for training and a physical machine as the cloud central scheduler, each server is equipped with 1 GeForce RTX 2060 GPU, 12 CPU cores, 16GB RAM, 500GB HDDs and a dual-port 1GbE NIC. Additionally, we adopt Hadoop Distributed File System (HDFS) to store training data and parameters [28], with data chunk size 2MB (*i.e.,* $\delta_i = 2$ MB). The model parameters of time slots are also checkpointed and stored in HDFS to deal with the dynamic changes of deployment during the training process [29].

TABLE II: Machine learning jobs used for experiments

Model	Dataset	# of Data chunks/ Mini-batches
ResNet-50 [30]	CIFAR10 [31]	27 / 58
ResNet-101	CIFAR10	27 / 58
GoogLeNet [32]	Caltech101 [33]	115 / 58
VGG-16 [34]	Caltech101	115 / 58
AlexNet [35]	Tiny-ImageNet [36]	150 / 58
Inception-BN [37]	Tiny-ImageNet	150 / 58

Setup. As given in Table. II, we run 6 types of deep learning jobs (*i.e.*, ResNet-50, ResNet-101, GoogLeNet, VGG-16, AlexNet and Inception-BN) on three respective datasets CIFAR10, Caltech101 and Tiny-ImageNet. Each worker or PS requires 0 to 1 GPU, 1 to 3 CPU cores and 2 to 6GB RAM, which implemented on a Docker container. For each job i, E_i is chosen randomly in $[20, 30]$, ρ_i is obtained by pre-training, and other settings are the same as simulator. Each experiment runs for 30 time slots, and the length of each time slot is set to 20 minutes. 10 jobs arrive randomly in the first 10 time slots.

Baselines. We compare *Okita* with the following three baselines, implemented on the same testbed or simulator:

- *FIFO* [38]: the default scheduler in Hadoop and Spark. Jobs run in order of arrival time a_i, with fixed number of workers. The number is fixed within $[1, 0.5A_i(a_i)]$, where $A_i(a_i)$ represents available number of workers at a_i.
- *DRF* [39]: a fairness-based scheduler adopted in YARN [40] and Mesos [41]. The fixed number of workers are computed to achieve multi-resource max-min fairness.
- *OASiS* [42]: for each newly arrived job, OASiS computes the optimal schedule based on the unit resource cost to achieve the goal of minimizing the deployment cost.

Furthermore, for FIFO and DRF, the deployment of workers and PSs are both selected based on the deployment metric $Q_i^r(t)$ at a_i. Additionally, since the number and the deployment of workers in FIFO and DRF are fixed throughout the training process, the training data distributed over sites will be dispatched evenly to sites in a greedy fashion once jobs arrive. As for OASiS, it first computes the workload of sites at each time slot, and then transfers data online based on a similar fashion in Alg. 3.

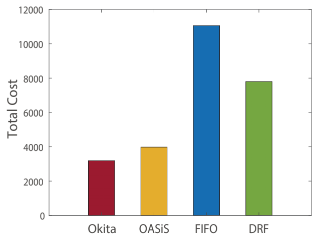

Fig. 4: Total cost.

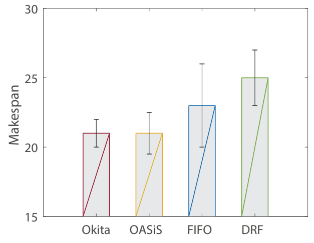

Fig. 5: Makespan.

Results. In our testbed experiments, the total cost of all algorithms is depicted in Fig. 4 which verifies that *Okita* outperforms three baselines. Fig. 5 further plots the makespan (*i.e.,* the distance in time that elapses from the start of the first job to the end of the last job) of all algorithms. It can be seen that the makespan of FIFO and DRF is larger than OASiS and *Okita*. This is because OASiS and *Okita* take into account the training time to avoid exponential explosion of latency cost function, so that further reducing the total cost.To show more details, Fig. 6 presents the training accuracy of the three datasets. From Fig. 6, we can observe that the training process in the experiment will not be affected by the dynamic deployment of workers and PS, and the convergence of the model can be achieved.

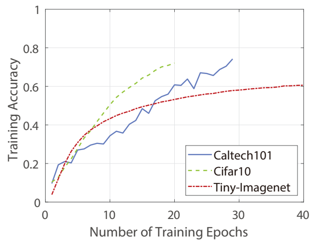

Fig. 6: Training accuracy.

B. Simulation Setup

To evaluate the performance at larger scale of inputs, we simulate an edge distributed learning system running for $T \in [20, 150]$ time slots ($R = 50$, $T = 150$ by default). 100 jobs (I) arrive randomly. Resources configuration of each site is set according to Amazon EC2 GPU instances P3, P2 and G3 [43]. Following the similar setting in [42], we set the resource demands of each worker as follows: 0 to 4 GPUs, 1 to 10 vCPUs, 2 to 32GB RAM and 5 to 10GB HDDs. Resource demands of each PS are: 1 to 10 vCPUs, 2 to 32GB RAM and 5 to 10GB HDDs. The unit bandwidth cost between site r and site r' ($q^{rr'}$) is evenly distributed at $[2, 8]$ per 100MB. For each job, D_i^r is $[10, 20]$, δ_i is set within $[20, 60]$ MB, (ρ_i/E_i) is picked within the range $[5, 15]$, and π_i is $[30, 575]$ MB [22]. Moreover, similar to [44], we set the latency cost function $f(t)$ to the following three types:

- *Sigmoid Function (default)*: $f(t) = \tau e^{0.25t}$, $\tau \in [20, 100]$.
- *Linear Function*: $f(t) = \tau t + b$, $\tau \in [20, 100]$, $b \in [50, 200]$.
- *Piecewise Function*: $f(x) = \begin{cases} \tau_1 & t < c \\ \tau_2 & t \geq c \end{cases}$, $\tau_1 \in [20, 100]$, $\tau_2 \in [300, 400]$, $c \in [3, 5]$.

Jobs with the linear cost function are normally latency-sensitive, whose cost increases with response time. Jobs associated with hard deadlines can use the piecewise function to represent. The sigmoid function can be regarded as a mix of the above functions and is the most common type in practice.

C. Performance

Performance Ratio. We define the performance ratio of algorithm A for a minimization problem as: the objective value of the solution computed by A / the objective value delivered by the optimal algorithm. In our simulator, the performance ratio of our online algorithm *Okita* under the different numbers of sites (R) and jobs (I) in the system is displayed in Fig. 7. It can be seen that the performance ratio of *Okita* is stable at a small value (< 1.8). To further demonstrate the performance of *Okita*, we compare the performance ratio of all algorithms with different values of R and I in Fig. 8. We can observe that the performance ratio of *Okita* is the smallest, indicating that the solution computed by *Okita* is closer to the optimal.

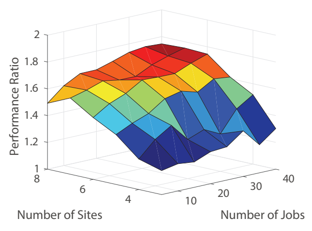

Fig. 7: Performance ratio of *Okita*.

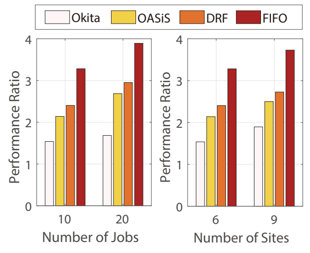

Fig. 8: Performance ratio under different numbers of sites and jobs.

Total Cost. The total cost can measure the overhead of the system. In Fig. 9 and Fig. 10, we compare the total cost of all algorithms as the number of sites and jobs increases. It can be observed that our online algorithm *Okita* performs better than other baselines, especially in a large scale of input. Combined with the testbed experiment results in Fig. 4, we are confident that *Okita* always outperforms regardless of the input scale.

Then we carry out simulations to evaluate the sensitivity of *Okita* based on the key proportional factors in the priority metric $O_i(t)$ (*i.e.,* α_1, and α_2) and the deployment metric $Q_i^r(t)$ (*i.e.,* β_1, β_2 and β_3). As described in Fig. 11 and Fig. 12, the total cost of *Okita* is roughly equal, and not affected greatly by proportional factors.

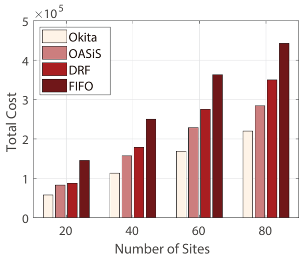

Fig. 9: Total cost under different numbers of sites.

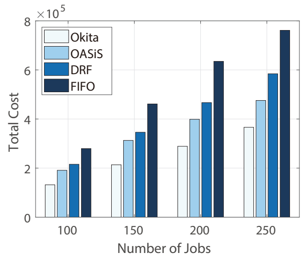

Fig. 10: Total cost under different numbers of jobs.

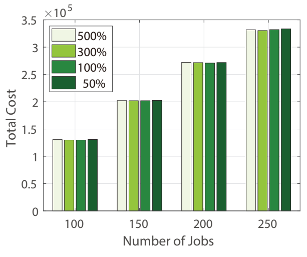

Fig. 11: Total cost under different $\alpha_1 : \alpha_2$.

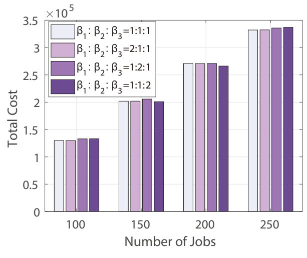

Fig. 12: Total cost under different β_1, β_2 and β_3.

Running time. Here, we use the *tic* and *toc* functions in MATLAB to measure the execution time of our online algorithm. We run *Okita* 20 times with different numbers of sites and jobs on laptop (Intel Core i5- 7300HQ/8GB RAM) and compute the average value to draw Fig. 13. We can observe that the running time of *Okita* increases with the number of sites/jobs, but still remains at a low level (< 50s).

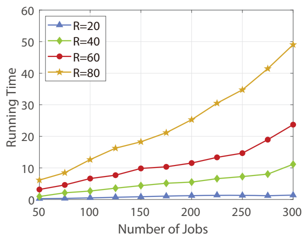

Fig. 13: Running time under different numbers of jobs.

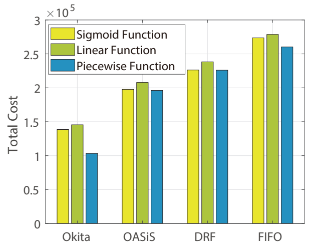

Fig. 14: Total cost under different latency cost functions.

Applicability. To study the impact of different latency cost functions, we also carry simulations on linear and piecewise cost functions. Fig. 14 shows the total cost of all algorithms under different latency cost functions. *Okita* always generates a lower cost. Compared with the sigmoid function, *Okita* outperforms under the linear function, *i.e.,* in the training of jobs which are more sensitive to latency. In the experiment of the piecewise function, change the first item of the priority metric to $\alpha_1 \cdot \min\left(\frac{t - a_i - c}{\max_{i \in \mathcal{J}_a}(t - a_i - c)}, 0\right)$, which means that the closer the response time $t - a_i$ is to the cost segment point c, the higher the priority. And if the cost segment point has passed, there is no need to consider its latency cost, to make all jobs tend to complete training with low latency cost, so the advantage of the piecewise function is the most obvious.

VI. Conclusion

In this paper, we proposed an online scheduling algorithm to speed up the training of UDL, meanwhile minimizing the bandwidth consumption among geo-distributed sites in wireless edge networks. In particular, no prior work has studied this problem to support new emerged AI applications with unbiased ML models in today's wireless edge system. To cope with the new challenges introduced by modeling this practical scenario, we designed an efficient online algorithm to schedule UDL jobs, aiming to minimize both the latency cost (which is related to the job completion time) and the bandwidth cost. Our simulations verified the effectiveness of our algorithm, and show that our algorithm can reduce the total training cost by up to 60% compared with state-of-the-art algorithms. For the direction of the further research, it is interesting to study the scheduling of inference at the wireless edge networks.

References

[1] H. Khelifi, S. Luo, B. Nour, A. Sellami, H. Moungla, S. H. Ahmed, and M. Guizani, "Bringing deep learning at the edge of information-centric internet of things," *IEEE Communications Letters*, vol. 23, no. 1, pp. 52–55, 2018.

[2] X. Wang, Y. Han, V. C. Leung, D. Niyato, X. Yan, and X. Chen, "Convergence of edge computing and deep learning: A comprehensive survey," *IEEE Communications Surveys & Tutorials*, vol. 22, no. 2, pp. 869–904, 2020.

[3] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE internet of things journal*, vol. 3, no. 5, pp. 637–646, 2016.

[4] A. Torralba and A. A. Efros, "Unbiased look at dataset bias," in *Proc. of IEEE CVPR*, 2011.

[5] X. Lyu, C. Ren, W. Ni, H. Tian, R. P. Liu, and E. Dutkiewicz, "Optimal online data partitioning for geo-distributed machine learning in edge of wireless networks," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 10, pp. 2393–2406, 2019.

[6] G. Zhu, D. Liu, Y. Du, C. You, J. Zhang, and K. Huang, "Toward an intelligent edge: Wireless communication meets machine learning," *IEEE communications magazine*, vol. 58, no. 1, pp. 19–25, 2020.

[7] J. Chen, K. Li, Q. Deng, K. Li, and S. Y. Philip, "Distributed deep learning model for intelligent video surveillance systems with edge computing," *IEEE Transactions on Industrial Informatics*, 2019.

[8] Y. Liu, M. Peng, G. Shou, Y. Chen, and S. Chen, "Toward edge intelligence: multiaccess edge computing for 5g and internet of things," *IEEE Internet of Things Journal*, vol. 7, no. 8, pp. 6722–6747, 2020.

[9] X. Zhang, M. Fang, J. Liu, and Z. Zhu, "Private and communication-efficient edge learning: a sparse differential gaussian-masking distributed sgd approach," in *Proc. of ACM MobiHoc*, 2020.

[10] *"Google Cloud TPU"*, 2017, https://cloud.google.com/tpu.

[11] *" Microsoft Azure Machine Learning"*, 2017, https://azure.microsoft.com/en-us/overview/machine-learning/.

[12] Y. K. Tun, Y. M. Park, N. H. Tran, W. Saad, S. R. Pandey, and C. S. Hong, "Energy-efficient resource management in uav-assisted mobile edge computing," *IEEE Communications Letters*, 2020.

[13] U. Saleem, Y. Liu, S. Jangsher, Y. Li, and T. Jiang, "Mobility-aware joint task scheduling and resource allocation for cooperative mobile edge computing," *IEEE Transactions on Wireless Communications*, 2020.

[14] H. A. Alameddine, S. Sharafeddine, S. Sebbah, S. Ayoubi, and C. Assi, "Dynamic task offloading and scheduling for low-latency iot services in multi-access edge computing," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 3, pp. 668–682, 2019.

[15] H. Hu, D. Wang, and C. Wu, "Distributed machine learning through heterogeneous edge systems," in *Proc. of AAAI*, 2020.

[16] M. M. Amiri and D. Gündüz, "Machine learning at the wireless edge: Distributed stochastic gradient descent over-the-air," *IEEE Transactions on Signal Processing*, vol. 68, pp. 2155–2169, 2020.

[17] S. Wang, T. Tuor, T. Salonidis, K. K. Leung, C. Makaya, T. He, and K. Chan, "When edge meets learning: Adaptive control for resource-constrained distributed machine learning," in *Proc. of IEEE INFOCOM*, 2018.

[18] W. You, L. Jiao, S. Bhattacharya, and Y. Zhang, "Dynamic distributed edge resource provisioning via online learning across timescales," in *Proc. of IEEE SECON*, 2020.

[19] I. Cano, M. Weimer, D. Mahajan, C. Curino, and G. M. Fumarola, "Towards geo-distributed machine learning," *arXiv preprint arXiv:1603.09035*, 2016.

[20] M. Li, D. G. Andersen, J. W. Park, A. J. Smola, A. Ahmed, V. Josifovski, J. Long, E. J. Shekita, and B.-Y. Su, "Scaling distributed machine learning with the parameter server," in *Proc. of USENIX OSDI*, 2014.

[21] F. Yan, O. Ruwase, Y. He, and T. Chilimbi, "Performance modeling and scalability optimization of distributed deep learning systems," in *Proc. of ACM SIGKDD*, 2015.

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

[23] R. M. Karp, "Reducibility among combinatorial problems," in *Complexity of computer computations*. Springer, 1972, pp. 85–103.

[24] A. Schrijver, *Theory of linear and integer programming*. John Wiley & Sons, 1998.

[25] *Technical report*, 2021, https://www.jianguoyun.com/p/DeVqhosQr764CRixkf8D.

[26] T. Chen, M. Li, Y. Li, M. Lin, N. Wang, M. Wang, T. Xiao, B. Xu, C. Zhang, and Z. Zhang, "Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems," *arXiv preprint arXiv:1512.01274*, 2015.

[27] *"Kubernetes"*, https://kubernetes.io/.

[28] *"HDFS"*, https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html.

[29] J. Gu, M. Chowdhury, K. G. Shin, Y. Zhu, M. Jeon, J. Qian, H. Liu, and C. Guo, "Tiresias: A {GPU} cluster manager for distributed deep learning," in *Proc. of USENIX NSDI*, 2019.

[30] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He, "Aggregated residual transformations for deep neural networks," in *Proc. of IEEE CVPR*, 2017.

[31] *The CIFAR-10 Dataset*, 2009, https://www.cs.toronto.edu/~kriz/cifar.html.

[32] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proc. of IEEE CVPR*, 2015.

[33] *Caltech101 Dataset*, 2006, http://www.vision.caltech.edu/Image_Datasets/Caltech101/.

[34] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[35] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Proc. of NIPS*, 2012.

[36] *Tiny imagenet visual recognition challenge*, 2019, http://tiny-imagenet.herokuapp.com/.

[37] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proc. of IEEE CVPR*, 2016.

[38] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, I. Stoica *et al.*, "Spark: Cluster computing with working sets." *Proc. of HotCloud*, 2010.

[39] A. Ghodsi, M. Zaharia, B. Hindman, A. Konwinski, S. Shenker, and I. Stoica, "Dominant resource fairness: Fair allocation of multiple resource types." in *Proc. of USENIX NSDI*, 2011.

[40] V. K. Vavilapalli, A. C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, S. Seth *et al.*, "Apache hadoop yarn: Yet another resource negotiator," in *Proc. of ACM SOCC*, 2013.

[41] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. H. Katz, S. Shenker, and I. Stoica, "Mesos: A platform for fine-grained resource sharing in the data center." in *Proc. of USENIX NSDI*, 2011.

[42] Y. Bao, Y. Peng, C. Wu, and Z. Li, "Online job scheduling in distributed machine learning clusters," in *Proc. of IEEE INFOCOM*, 2018.

[43] *Amazon EC2 Instances*, https://aws.amazon.com/ec2/instance-types/.

[44] C. Zhang, H. Tan, H. Huang, Z. Han, S. H. Jiang, N. Freris, and X. Li, "Online dispatching and scheduling of jobs with heterogeneous utilities in edge computing," in *Proc. of ACM MobiHoc*, 2020.