2021 IEEE/ACM 21st International Symposium on Cluster, Cloud and Internet Computing (CCGrid) | 978-1-7281-9586-5/21/$31.00 ©2021 IEEE | DOI: 10.1109/CCGrid51090.2021.00065

AI-oriented Workload Allocation for Cloud-Edge Computing

Tianshu Hao*†§, Jianfeng Zhan*†, Kai Hwang‡§, Wanling Gao*†, Xu Wen*†
*State Key Laboratory of Computer Architecture, Institute of Computing Technology, Chinese Academy of Sciences
{haotianshu, zhanjianfeng, gaowanling, wenxu}@ict.ac.cn
‡ The Chinese University of Hong Kong, Shenzhen, China, hwangkai@cuhk.edu.cn
† University of Chinese Academy of Sciences
§ Shenzhen Institute of Artificial Intelligence and Robotics for Society

***Abstract*—Different placement or collaboration policies in handling datasets and workloads across cloud, edge, and user-end may substantially affect a cloud-edge computing environment's overall performance. However, the common practice is to optimize the performance only on the edge layer, while ignoring the rest of the system. This paper calls attention to optimize AI-oriented workloads' performance across all components in cloud-edge architectures holistically. Our goal is to optimize AI-workload allocation in cloud clusters, edge servers, and end devices, achieving the minimum response time in latency-sensitive applications.**

This paper presents new workload allocation methods for AI workloads in cloud-edge computing systems. We have proposed two efficient allocation algorithms to reduce the end-to-end response time of single-workload and multi-jobs scenarios, respectively. We apply six edge AI workloads from a comprehensive edge computing benchmark – Edge AIBench for experiments. Besides, we conduct experiments in a real edge computing environment. Our experiment results demonstrate the high efficiency and effectiveness of our algorithms in real-life applications and datasets. Our multi-job allocation algorithm's end-to-end response time outperforms the other four baseline strategies by 33% to 63%.

***Index Terms*—edge-cloud computing, edge AI, workload allocation, end-to-end response time**

I. Introduction

With the rapid development of user-end Internet of Things (IoT) [11], edge computing emphasizes real-time computing near the end devices [23]. A common hierarchical framework of distributed cloud and edge computing consists of three layers: cloud cluster, edge server, and end devices. For a better user experience and performance acceleration, artificial intelligence (AI) technology is widely used in edge computing to support edge intelligence [26]. Most of these edge AI applications are latency-sensitive, e.g., autonomous vehicles with strict requirements for response time, especially the end-to-end response time. Here, end-to-end response time covers the total time consumption from receiving an end-user request to feeding back the result to users, including processing time on different layers. Even for the edge applications without stringent time requirements, low response time is also essential to attract and retain users [14], e.g., smart home. Thus, how to reduce the response time of these edge AI applications has become a critical problem.

There have been several efforts about edge AI made by academia and industry now. Many corporations put efforts on edge AI accelerator chips to speed up the processing time on the edge device, such as Google's edge TPU [4], Huawei's Atlas 200DK and Intel's neural compute stick [6]. Moreover, Caffe and Tensorflow Lite implement the light-weighted model to reduce the complexity of the AI models which can be applied to edge servers. Additionally, there is also a set of benchmarking works to evaluate the inference of AI workloads on edge devices. For example, Luo et al. [15] and Ignatov et al. [12] have evaluated the inference performance on the edge devices.

It is a common practice to deploy AI workloads on edge servers. Still, it's not the optimal solution because different placement or collaboration policies in handling datasets and workloads across cloud, edge, and user-end may substantially affect the overall system's behaviors. An AI workload-allocation model is needed for latency-sensitive edge computing applications now. Moreover, performing a trade-off between the computational resources and network latency is still a problem in the edge computing hierarchical framework.

We single out six different AI workloads as the target applications from Edge AIBench [9]: three patient monitor AI tasks in the ICU scenario, one face recognition task in the smart home scenario, one lane detection task in the autonomous vehicle scenario, and one action detection task in the surveillance camera scenario. Based on the hierarchically-structured framework, this paper proposes a workload allocation strategy for edge AI workloads. We perform a breakdown of each workload's end-to-end response time: processing and transmission time across three layers.

The workload allocation aims at reducing the end-to-end response time for edge AI applications. We propose two efficient allocation algorithms to minimize the end-to-end response time for single and multi-job workloads, respectively. Finally, we use typical edge AI workloads and real-life datasets from Edge AIBench [9] to validate the efficiency and effectiveness. The experiment results show our allocation algorithms achieve the minimum response time comparing with the other four baseline strategies.

Jianfeng Zhan is the corresponding author.

978-1-7281-9586-5/21/$31.00 ©2021 IEEE
DOI 10.1109/CCGrid51090.2021.00065

The rest of this paper is organized as follows. Section II introduces the problem environment of the hierarchically-structured framework and edge AI workload. Section III and IV present the workload allocation problem and an optimal algorithm for a single job. In sections V and VI, we present an efficient allocation and scheduling algorithm for multiple jobs. Section VII introduces the experiment setup, and section VIII shows the experiment results analysis. Section IX introduces recent and related work. Finally, we conclude in section X.

II. The Problem Environment

A. Hierarchically-structured Cloud-Edge Computing

Edge computing is a distributed computing paradigm with the three-layer framework: cloud cluster, edge computing server, and user-side end devices. Figure 1 shows the hierarchically-structured cloud-edge computing framework of edge computing.

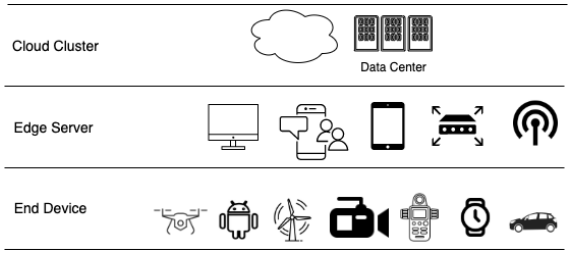


Fig. 1. Hierarchical Framework of Edge Computing.

The cloud cluster is a centralized server cluster physically far away from the end devices. Traditionally in cloud computing, most of the computing tasks are executed in the data center. However, the transmission latency from the cloud server to user-side devices may be long, sometimes even longer than the processing time. Therefore, in the edge computing framework, the primary responsibility of the cloud server is executing offline tasks and store vast amounts of data.

Edge computing layer servers locate closer to users than the cloud cluster. Moreover, edge servers have more computational resources than user-side end devices. Therefore, complicated online tasks are usually executed on the edge computing layer.

There is a great diversity of user-side end devices, such as wearable devices, sensors, unmanned aerial vehicles, and smart vehicles, and etc. The end devices usually gather data and conduct some simple preprocess tasks because of the restriction of the computational resources.

Generally, in the three-layer framework, the higher the layer, the more computational resources of the device, the faster the processing speed, but the longer the data transmission time. Thus, how to trade off the computational ability and data location becomes a problem.

B. Edge AI Workloads Benchmark

Edge AIBench [9] is a comprehensive end-to-end scenario-based benchmark towards AI edge computing. By surveying the edge computing scenarios extensively, Edge AIBench finds out that most of these workloads use AI technology. Therefore, we focus on edge AI workloads in this paper.

Figure 2 shows the main workflow of AI workloads, including offline and online processes.

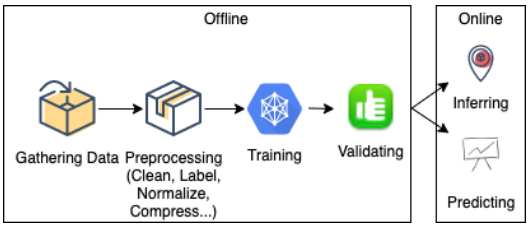


Fig. 2. AI Online and Offline Workflow.

Considering the complexity of AI workloads, there are high requirements for computing machines. Because cloud clusters have better computational ability, offline processes in Figure 2 are always executed on cloud clusters in edge computing frameworks. Then, the pre-trained model is sent to the online processing device after completing training on cloud clusters.

Due to the latency-sensitive characteristic being the major concern in this paper, we focus on the inference process of AI workloads. Actually, the most common edge AI workload allocation method is putting the offline training on the cloud server and putting the online inference or other online processes on the edge computing layer [20]. However, where to deploy the online process greatly influences the response time, considering the data transmission time. Additionally, many edge AI workloads are latency-sensitive applications. Therefore, response time reduction is important in edge computing workload allocation.

From Edge AIBench [1], we extract six workloads as the workload allocation problem background: three of them are from ICU patient monitor scenario, and three other workloads are from three other scenarios (autonomous vehicle, surveillance camera, and smart home).

ICU Patient monitor is a typical latency-sensitive edge AI scenario. Because ICU is the treatment place for critical patients, the response latency of any task is very important for doctors' further action. As Figure 3 shows, there are several patients in serious condition in an ICU room. Each patient has several end devices to measure vital signs such as temperature, respiration, pulse, and heartbeat, and etc. There is one end computing device for one patient and one edge server for one ICU room. Moreover, there is a cloud server for heavy computation tasks (such as training) and data storage. Additionally, the other three scenarios in Edge AIBench have some latency-sensitive applications.

Three typical medical AI workloads we choose from ICU patient monitor are: short-of-breath alerts, patient phenotype classification, and life-death prediction to conduct the evaluation experiments. Based on the purpose of these workloads, we set different priorities of them for multi-job experiments. In addition, we choose three more complicated delay-sensitive edge AI workloads: face recognition from smart home, lane

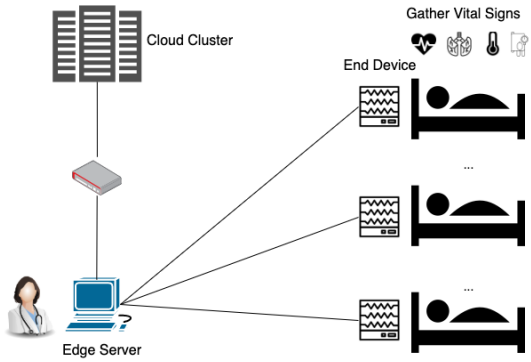


Fig. 3. ICU Patient Monitor Edge Computing Scenario in Hospital.

detection from autonomous vehicle, and action detection from surveillance camera [1].

III. Latency Analysis for Single Workload

A. Basic Assumptions

As Figure 1 shows, the hierarchically-structured framework consists of three levels: CC represents cloud clusters, ES represents edge computing servers, and ED represents end devices. The three major factors that determine the latency of the inference process are the computational capability of each level, the complexity of the model, the size of the inference data, and the network latency. In this paper, we make the following assumptions:

(a). The data are gathered from end devices. Thus, data transmission does not need to be considered if the inference process is on end devices.

(b). The transmission time from the cloud cluster to the end device T_{CC-ED} equals to the sum of the transmission time from the cloud cluster to the edge server T_{CC-ES} plus the transmission time from the edge server to the device T_{ES-ED}.

(c). Considering AI workloads are compute-intensive, the computational resources of each layer are expressed by the floating point operations per second (FLOPS) [8], [28] of devices.

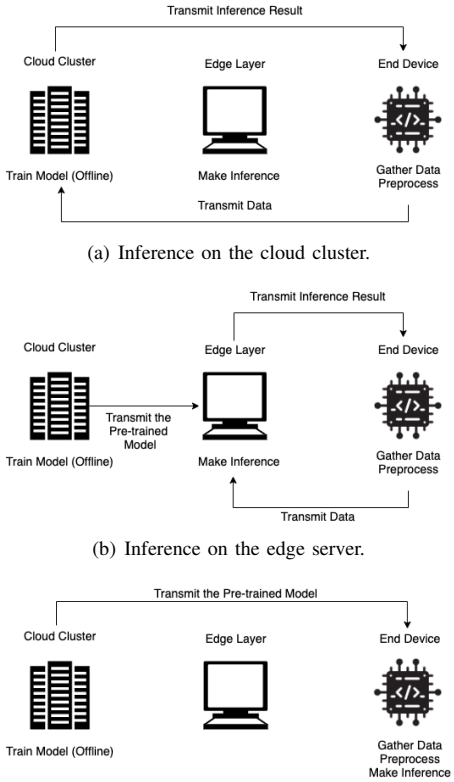
(d). To simplify the problem, there are only one cloud server and one edge server considered in this model.

(e). The response time of any workload depends on the workload model complexity, dataset size, the device processing the workload, and the network condition.

(f). The transmission time of the inference result T_{rt} is lower than the transmission time of the inference data T_{dt}. Therefore we don't consider T_{rt} in this problem.

Based on the above assumptions, the problem can be expressed as a cloud server, an edge server, and an end device constructed following the hierarchically structured 21 framework as Figure 1 shows. The problem is which layer to allocate the given online AI inference workload can get the minimum overall response time.

From the online tasks in Figure 2, we mainly consider the inference allocation problem in this paper for the actual production environment. Figure 4 shows the three different inference location methods, on cloud cluster, edge server, and end devices, respectively.



(a) Inference on the cloud cluster.

(b) Inference on the edge server.

(c) Inference on the end device.

Fig. 4. Trade-off among three processing layers of a hierarchically-structured cloud-edge environment.

B. Objective Function for Latency Reduction

In order to find the optimal solution for this problem, we express it as a numerical problem. Table I shows the notations in this problem.

TABLE I
Notations of Workload Allocation Problem.

Notation	Interpretation
D_i	the transmission time of data from end device to the execution layer i
I_i	the time of making the inference on the layer i
s	the size of the workload data
$comp$	the FLOPs of the AI model in workload i
CR_i	the computational resources (FLOPS) of the layer i
$\lambda 1$	the weight coefficient of the data transmission time
$\lambda 2$	the weight coefficient of the inference processing time

For different layers, the probability of whether to make inference on this layer is expressed by:

$$p_i \in \{0,1\}(i \in \{CC, ES, ED\})$$
$$where \quad \sum p_i = 1 \tag{1}$$

The data transmission time D is expressed by:

$$D = \lambda_1 s \sum P_i D_i \tag{2}$$

The computational resources of the device is expressed by $CR_i = FLOPS_i (i \in \{CC, ES, ED\})$.

The inference processing time I is expressed by:

$$I = \lambda_2 \sum P_i I_i, \quad where \quad I_i = \frac{s \times comp}{CR_i} \tag{3}$$

Where the inference processing time on layer i is expressed by I_i.

Therefore, the objective function can be expressed by:

$$minimize \quad T = D + I$$
$$subject \quad to \quad (1) - (3) \tag{4}$$

C. Performance Metrics for AI Workload

We measure the computational ability of devices from each layer by FLOPS [8]. The FLOPS is calculated by the number of processor cores × operating frequency × operations per cycle.

Moreover, we also measure the complexity of the AI model by FLOPs. FLOPs is computed using the number of parameters multiplying with the size of the feature map for convolution kernels. The computation formula is $FLOPs = 2HW(C_{in}K^2+1)C_{out}$. or fully connected layers, the FLOPS equals to the number of the parameters [18]. The computation formula is $FLOPs = (2I-1)O$. H indicates the height, W indicates the width and C_{in} indicates the number of channels of the input feature map. K indicates the kernel width and C_{out} indicates the number of output channels. I indicates the input dimension and O indicates the output dimensionality [18].

IV. Allocation Algorithm for Latency Reduction

This section gives an optimization strategy to get the minimum latency with a given workload and device environment. In order to solve the problem we defined, we simplify the problem by assuming that each layer has only one device.

For a given edge AI workload and device environment, we can calculate the computational resources of three layers and the network condition first. Then we can calculate the minimum response time by putting the inference in different layers. Algorithm 1 shows the procedure of our optimization strategy.

Firstly, for each workload, we analyze their computation model parameters and calculate the FLOPs $comp$ needed to execute.

Secondly, we calculate the unit network transmission latency D_{iu} when the inference is deployed on layer i. Therefore, we can calculate the given workload transmission time D_i by multiplying D_{iu} with the dataset size and the weight coefficient.

Thirdly, we calculate the computation ability CR_i of the device in each layer by FLOPS. Because most of the AI workloads are computation-intensity, we just consider FLOPS here.

The next step is to calculate the weight coefficient for processing time and transmission time. We conduct a simple experiment to compute the time of one respectively small dataset and obtain λ_1 and λ_2 by comparing them.

Next, we get the estimated response time for deploying the inference on each layer by summing up the processing time I_i and transmission time D_i.

Finally, we choose the minimum response time layer as the deployment layer for this workload.

Algorithm 1 Algorithm of Latency Reduction

Input: W, s, i, D_i
Output: the minimum response time T_{min}

Calculate the number of FLOPs of the AI workload model $comp$
Calculate the unit network latency
for i = CC, ES **do**
    D_{i_u} = latency time of unit dataset transmission
Calculate the computational of each layer's device
for i = CC, ES, ED **do**
    $CR_i = FLOPS \quad of \quad device \quad i$
Normalize the transmission time and inference time by calculating the weight coefficient λ_1, λ_2
Calculate the inference processing time.
for i = CC, ES, ED **do**
    $I_i = \frac{\lambda_2 \times s \times comp}{CR_i}$
Calculate the data transmission time.
for i = CC, ES **do**
    $D_i = \lambda_1 * s * D_{i_u}$
Calculate the minimum latency
Let minimum response time $T_{min} = +\infty$
for (**do** $i = CC, ES, ED$)
    $p_i = 1$
    $T_i = I_i + D_i$
    if $T_i < T_min$ **then**
        $T_{min} = T_i$
return T_{min}

V. Workload Allocation for Multiple Jobs

A. Multi-job Workload

Considering the real-life edge computing environment, there are several multi-job assignment problems. For example, in ICU patient monitoring scenario, each patient has an end device to pre-process collected data in one emergency room. These patients share one edge server in the emergency room and one cloud server remote from the ICU. These machines, including cloud server, edge server, and patients' end devices, can be considered as several unrelated parallel machines.

Each patient's end device may release an inference job randomly. Assuming these jobs are released in a time sequence, our goal is to minimize the overall response time of all jobs. Then, this problem can be considered as an unrelated parallel machine scheduling problem.

From the above section, we can calculate the response time for every single job deployed on different layers and get the best layer with the minimum latency. We set the following constraints in this problem:

C1. Each device can only execute one job at a time to simplify the problem.

C2. The preemption of jobs is not allowed in our system.

C3. The release time and response time are normalized as the non-zero integer units of time.

C4. The job data can be transmitted to the execution layer first and wait for execution.

C5. The workload is prioritized by its importance in real-life. The workload with the higher priority needs to be considered first.

Thus, the scheduling problem can be summarized as follows: there are n patients' jobs $(T_1, T_2, ..., T_n)$ waiting to be executed in one emergency room. These jobs have their own priority $(w_1, w_2, ..., w_n)$. The larger the w_i, the higher the priority of the job i. These jobs can be processed on cloud servers (M_c), edge servers (M_e), or individual patient end devices (M_d). Each job T_i has a known release time $(R_1, R_2, ..., R_n)$ in the time sequence. The response latency time of job T_i executed on machine M_j is L_{ij}. Table II shows the notation of this problem.

The main goal of this problem is to reduce the total response time of multiple jobs.

TABLE II
NOTATIONS OF ALLOCATION PROBLEM FOR MULTIPLE JOBS.

Notation	Interpretation
i	the index of the job, i = 1, 2, ... , n
j	the index of the machine, j = c (cloud), e (edge), d (device)
T	the unit of time, T = 1, 2, ... , $+\infty$
w_i	the priority weight of job i
P_{ij}	P_{ij} equals 1 if the job i is processed on the machine j, otherwise equals 0
I_{ijT}	I_{ijT} equals 1 if the job i is processed on the machine j at time T, otherwise equals 0
L_i	the response time of the job i
L_i^*	the response time of the job i considering the weight
R_i	the release time of job i
S_i	the start processing time of job i
E_i	the completion time of job i
L_{Sum}	the total response time of all jobs
E_{last}	the completion time of the last job

B. Objective Function to Minimize Response Time

To get the minimum total response time of all the jobs, the objective function can be expressed by:

$$minimize \quad L_{sum}$$
$$where \quad L_{sum} = \sum_{1}^{n} w_i(L_i - R_i) \tag{5}$$

The total response time is calculated by summing up the response time of all jobs. The response time L_i of job i equals to the end time E_i minus the release time R_i. For different priorities of these jobs, we multiply the response time with the priority weight W_i to get the new response time L_i^*. End devices are not subject to this constraint because we assume every job has its own end device.

We set p_{ij} and I_{ijT} as binary variables equal to 0 or 1. The constraint $\sum_{j=1}^{m} p_{ij} = 1$ needs to be considered to ensure the job i is assigned to exactly one machine. The constraint $\sum_{i=1}^{n} I_{ijT} \leq 1$ ensures that only one job is executed on machine j at a time.

From section III, we know the total execution time of job i consists of the transmission time D_i and processing time I_i. The exact processing time on the machine is I_i. The job can transmit data to the machine j while another job is running on the same machine.

VI. WORKLOAD ALLOCATION HEURISTIC ALGORITHM

We can change the lower bound from 0 to the sum of the minimum execution time of all the jobs. The lower bound can be expressed by:

$$L_{lb} = \sum_{i}^{n} \min w_i(I_i + D_i) \tag{6}$$

Therefore, the problem can be simplified as selecting the minimum response time of the last release job.

Because the scheduling problem for n jobs executed on m machines is very complicated, we develop a heuristic greedy algorithm to solve the problem in this section. Considering end devices are not shared among patients, jobs can be executed on them simultaneously.

We can obtain the initial feasible solutions by ensuring that the earliest released job has the shortest response time. Then, we optimize the solution by neighborhood search method [17]. Algorithm 2 shows the concrete steps, and it needs to follow the constraints we mentioned in the previous section.

Firstly, we use algorithm 1 to calculate the estimated execution time of each job deployed on each layer. Then, we normalize the response time to the integer time units. We use a heuristic greedy method to get the initial deployment strategy. We find the optimal deployment machine for each job to have the minimum completion time by time sequence. Then, we obtain the initial feasible solution.

Subsequently, we generate the neighborhood solutions from the current solution by swapping the current job i to another machine. After that, we calculate the deployment machine of other jobs using the above heuristic greedy method. If the total response time reduces, we swap the deployment machine of job i.

We set the max number of iterations $maxCount$ as the stopping condition for the algorithm.

Algorithm 2 Multi-job Allocation

Input: W_i, R_i
Output: the minimum response time of the whole jobs L_{sum}

Calculate the execution time L_ij of job i deployed on machine j. And get the response time matrix **l**.
Get $\mathbf{l}^*$ by multiplying with the job priority w_i.
$L^*_{ij} = w_i L_{ij}$
Normalize matrix $\mathbf{l}^*$.
Set $L_{sum} = \sum_i^n \min L_i$
Set $L^*_{sum} = \sum_i^n w_i \min L_i$
Initial the tabu array, 0 represents the job or machine can be switched and 1 represents can't be switched.
Set $tabu_m[cc, es, ed] = 0$
Optimize the solution by swapping the job to another machine reduce the total response time.
Set maxCount = C_{max}(large enough)
while maxCount **do**
    Set $tabu_j[1, ..., n] = 0$
    for i = 1, 2, ... , n **do**
        Set $tabu_m[cc, es, ed] = 0$
        from all the jobs which $tab_j[k] = 0$ choose the earliest completion job k
        Set $tab_j[k] = 1$
        Initial the max response time improvement.
        Set $V_{max} = 0$
        for j = cc, es, ed **do**
            if $tabu_m[j] = 0$ **then**
                Calculate the L^*_{sum} reduction V_{ij} when swap job i to machine j
                $tabu_m[j] = 1$
                if $V_{ij} \geq V_{max}$ **then**
                $V_{max} = V_{ij}$
                $swap = j$
        if $V_{max} > 0$ **then**
            Swap the job k to machine $swap$
    $MaxCount = Maxcount - 1$
return L_{sum}

VII. Experimental Setup

A. Experimental Environment

We construct the experimental environment to simulate the realistic cloud-edge computing scenario. Additionally, We assume that the cloud server has the highest computational ability, the edge server's performance is lower, and the end device's performance is the lowest. Meanwhile, the cloud server has a longer distance to the end device than to the edge server.

The cloud server is from CUHK (SZ) AIStation cloud server, which has 16 2.20-GHz Intel(R) Xeon(R) Gold 5220 CPU cores, 1 GeForce RTX 2080 Ti with 4352 CUDA cores and 128GB of DDR4 RAM. The edge server is from CUHK (SZ) edge computing platform server, which has 8 2.20-GHz Intel(R) Xeon(R) Silver 5220 CPU cores and 8GB of DDR4 RAM. The end device is a Raspberry Pi 4B with 4 1.5-GHz speed Quad-core Broadcom CPU cores and 4GB of DDR4 RAM.

Due to the fact that end device and edge server are close to the cloud server, we refer to [30], and set the network latency between the cloud server to the end device as 42ms and network bandwidth as 2.9MB/s. Moreover, we measure the network latency and bandwidth between the edge server and end device in our campus LAN environment as 0.239ms and 10MB/s, respectively.

Then we calculate the FLOPS of each device using the processor information. Table III shows the basic computational ability of devices of each layer.

TABLE III
Computational Ability of Device on Each Layer.

Layer	CPU FLOPS	GPU FLOPS
AIRS Cloud Server	1126.4 GFLOPS	110 TFLOPS
AIRS Edge Server	563.2 GFLOPS	✗
Raspberry	192 GFLOPS	✗

B. Edge AI Workloads

For the medical dataset, we choose MIMIC-III [13] as the real-world dataset. MIMIC-III includes vital signs and other medical information for over 60000 ICU stays of 40000 unique ICU patients. We pre-process the original data from the MIMIC-III website [2] and train them to obtain the offline model.

We choose three ICU AI applications from Edge AIBench [9] for the experiment: short-of-breath alerts, patient phenotype classification, and life-death prediction.

Short-of-breath alerts The purpose of this application is to predict whether the patient will suffer short-of-breath later using the LSTM model. Therefore the priority of this application is high. For the workload of short-of-breath alerts, we set the weight w to be 2. Using the number of parameters of the LSTM model in this application, we get 105089 for the number of FLOPs.

Life-death prediction The purpose of this application is to predict whether the patient will die in the hospital. The priority of this application is also high. Therefore, we also set the priority weight w of this application to be 2. Using the number of parameters of the LSTM model in this application, we get 7569 for the number of FLOPs.

Patient phenotype classification The purpose of this application is to conduct a 25 separate binary classification task using the LSTM model. Because it's not a very emergency task comparing with the above two tasks. We set lower priority weight w of this application as 1. Using the number of parameters of the LSTM model in this application, we get 347417 for the number of FLOPs.

In order to consider more complicated AI applications, we also choose three other workloads from Edge AIBench [9]: face recognition, lane detection, and action detection.

Face recognition uses the Labeled Faces in the Wild [10] as the dataset and FaceNet [22] as the network model with

1.7 GFLOPs. **Lane detection** uses CULane [19] as the dataset and LaneNet [27] as the network model, having 18.91 GFLOPs. **Action detection** uses UCF101 [24] as the dataset and ResNet18 as the network model, having 7.84 GFLOPs.

We implement these workloads using Edge AIBench reference implementation [1]. We train these models offline on our cloud server to get the pre-trained model for inference.

Moreover, we set 6 different inference data sizes for ICU AI applications to obtain 18 different workloads. The size of them are calculated in proportion of the number of record files, and the real sizes of inference data are [700, 1300, 2300, 5000, 10700, 21500, 479, 950, 1900, 3900, 7800, 15900, 836, 1700, 2900, 5300, 10800, 21600] KB.

VIII. Experiments and Performance Analysis

A. Single Medical Workload Allocation

Firstly, we use medical workloads to validate algorithm 1. For each workload, we calculate the estimated response time of deploying on different layers by using algorithm 1. Because the LSTM model used here is simple, we only use CPU cores in cloud server in this section. Then we choose the optimal deployment layer according to the results. Table IV shows our estimated computation results and the best deployment layer for each workload.

TABLE IV
Estimated Response Time Using Algorithm 1 at the Cloud, Edge, and Device Levels.

Workload No.	Chosen Deployment Layer	Estimated Response Time for Deploying on		
		Cloud Server	Edge Server	End Device
WL1-1	Edge Server	2091	1279	1394
WL1-2	Edge Server	4182	2558	2788
WL1-3	Edge Server	8364	5116	5576
WL1-4	Edge Server	16728	10232	11152
WL1-5	Edge Server	33456	20464	22304
WL1-6	Edge Server	66912	40928	44608
WL2-1	End Device	212	109	79
WL2-2	End Device	424	218	158
WL2-3	End Device	848	436	316
WL2-4	End Device	1696	872	632
WL2-5	End Device	3392	1744	1264
WL2-6	End Device	6784	3488	2528
WL3-1	Edge Server	3115	2931	3618
WL3-2	Edge Server	6230	5862	7236
WL3-3	Edge Server	12460	11724	14472
WL3-4	Edge Server	24920	23448	28944
WL3-5	Edge Server	49840	46896	57888
WL3-6	Edge Server	99680	93792	115776

For validating the effectiveness of the computational results, we deploy each workload on each layer to conduct the inference experiment on the real experimental environment, respectively. Then, we get the real response time for each workload deployed on different layers. Figure 5 shows the experimental results.

Figure 5a shows the results of the short-of-breath application. Deploying the workload on the edge server gets the lowest response time, and deploying on the cloud server gets the highest response time. Figure 5b shows the result of life-death prediction, the end device layer is the optimal deployment layer. Figure 5c shows that the edge layer is the optimal deployment layer for patient phenotype classification.

By comparing Table IV with Figure 5, we find most of the estimated deployment layer will get the lowest response time. For WL2-1 and WL2-2, it is better to deploy on edge servers but we predict to deploy on end devices. Nevertheless, the response time of deploying on the edge server and end devices are very close in these two workloads.

In general, the experiment results demonstrate the effectiveness of our optimal strategy for the single healthcare workload deployment.

In addiction, we explore the critical influence factor of the response time by the breakdown Figure 6. We choose the workload WL1-6, WL2-6, and WL3-6 to represent the three applications. Figure 6 shows the processing time and transmission time of the three workloads.

From Figure 6, we have the following observations.

The model of patient phenotype classification is more complicated than the other two applications. Therefore, the processing time on the end device is relatively higher. Therefore, the transmission time has a smaller influence on this condition.

For life-death prediction, the optimal deployment layer is the end device layer. Because the number of parameters of this model is small, the end device can satisfy the requirements of this model. In this situation, workloads aren't able to be offloaded on the edge or cloud server.

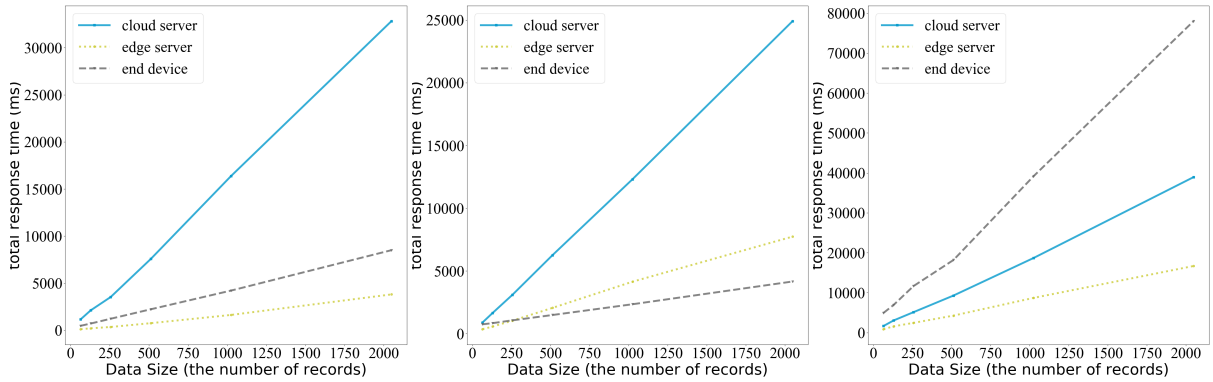
We can conclude that the more simplified the workload models are, the greater influence the transmission time has. Therefore, computing near the user may get the lowest response time. On contrary, for a heavy-weight workload, being processed on higher layer may lead to the lowest response time.

Therefore, for the AI workload deployment on edge computing framework, we need to estimate the computation ability of the devices on different layers and the network condition firstly. Then we can decide which layer to offload the workload by trading off between the processing time and transmission time.

B. Complicated AI Workload Allocation

Since the medical workloads we choose all use LSTM model, which is simpler than the most of the state-of-the-art deep learning models, we use three other state-of-the-art AI workloads to further demonstrate our algorithm. In this experiment, we use the GPU cores of cloud server because the network model is more complicated than that in the previous section. We use one image as the inference data for each workload. Table V shows the estimated computation results using algorithm 1 and the best deployment layer for each workload.

Figure 7 shows the experiment results which matches the estimated chosen layer in Table V. Therefore, this experiment demonstrates that algorithm 1 also satisfies the other state-of-the-art AI workloads.



(a) Short-of-breath alerts. (b) Life-death prediction. (c) Patient phenotype classification.

Fig. 5. The experiment response time results of each workload deployed on different layers.

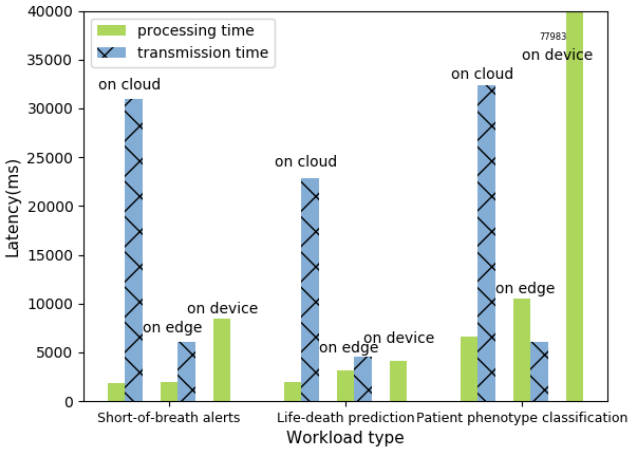


Fig. 6. Response time breakdown of each workload.

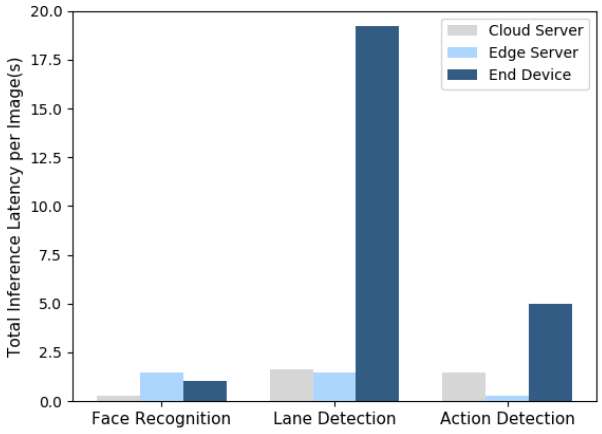


Fig. 7. Total inference time per image of each Workload deployed on different layers.

TABLE V
ESTIMATED RESPONSE TIME FOR SINGLE FRAME INFERENCE USING ALGORITHM 1.

Workload Name	Chosen Deployment Layer	Estimated Response Time for Deploying on		
		Cloud Server	Edge Server	End Device
Face Recognition	Cloud Server	513	5329	856
Lane Detection	Edge Server	2160	1596	7848
Action Detection	Edge Server	2120	1179	4083

C. Multi-job Scheduling Strategy

To validate the scheduling algorithm, we extract 10 jobs from the above experimental workload execution time results, and we normalize their response time. Meanwhile, we set a release time for each workload in order to simulate the real-world environment. Table VI shows the scheduling experiment setting.

We use algorithm 2 to calculate the efficient workload deployment strategy for these 10 jobs. Firstly we calculate the initial feasible deployment strategy, and choose the best deployment layer for each workload by time sequence. Then, we adjust the deployment strategy using the heuristic method iteratively.

Figure 8 shows the scheduling method by our workload allocation algorithm 2. The horizontal axis represents the time sequence. It shows the execution starting time and completion time of each job on the deployment layer. The total response

TABLE VI
PROCESSING TIME, TRANSMISSION TIME AND RELEASE TIME OF JOBS DEPLOYED ON DIFFERENT LAYERS.

Job No.	Release	Priotity Weight	Deployed on Cloud Server		Deployed on Edge Server		Deployed on End Device
			Processing	Transmission	Processing	Transmission	Processing
J1	1	2	6	56	9	11	14
J2	1	2	3	32	3	6	12
J3	3	1	4	12	6	2	49
J4	5	1	7	23	11	5	69
J5	10	2	4	27	5	5	11
J6	20	2	5	70	5	14	22
J7	21	2	5	70	5	14	22
J8	21	1	4	12	6	2	49
J9	22	1	4	12	6	2	49
J10	25	1	7	23	11	5	69

time and the last completion time is 150 and 43, respectively. We need to deploy 4 workloads on end devices, 4 on the edge server, and 2 on the cloud server. We can see the deployment layer of each job is not optimal for the single workload.

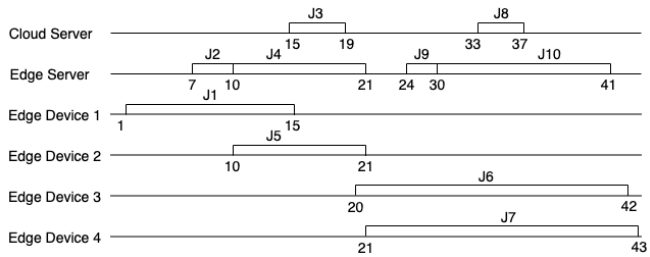


Fig. 8. Allocation Strategy Using Algorithm 2.

For comparison, we calculate the response time of the other 4 deployment strategies. We deploy all the workloads on the cloud server, the edge server, end devices, or their optimal layers.

Figure 9 shows the scheduling strategy through choosing the optimal deployment layer of each job. The optimal deployment layer is the cloud server and the edge server for job 1 and for other jobs, respectively. We can see from Figure 9 that a lot of jobs are dependent on the completion of previous jobs, which delays the completion. Therefore, our scheduling algorithm is significant in speeding up the job completion.

Fig. 9. Allocation Strategy Using the Optimal Layer for Each Job.

D. Measure Total Response Time of Multi Jobs

We calculate the sum of total response time and the last completion time of five strategies including our strategy and the other 4 baseline strategies. Table VII shows the results of these strategies.

Our strategy has the lowest total response time and the lowest last response time. For the total response time, our optimal deployment strategy outperforms 33%, 48%, 63%, and 59% over the other strategies, respectively. The results demonstrate that our strategy has the lowest total response time and last response time among all compared strategies.

TABLE VII
REPONSE TIME USING DIFFERENT ALGORITHMS.

Strategy	Total Response Time	Last Response Time
Our Allocation Strategy	150	43
Deployed on the Optimal Layer for Each Job	227	67
Deployed on Cloud Server	291	74
Deployed on Edge Server	416	100
Deployed on End Device	366	94

IX. RELATED WORK

This section reviews recent related work of edge computing workload deployment and resource allocation.

AIBench [25] is a comprehensive AI benchmark suite across cloud, edge, and user-end devices. AIBench distilling real-world application scenarios into AI Scenario, Training, Inference, Micro, and Synthetics Benchmarks. Each scenario benchmark models critical paths of a real-world application scenario as a permutation of the AI and non-AI modules [25]. Edge AIBench [9] is a scenario benchmark suite, modeling end-to-end performance across IoT, edge, and Datacenter. MLPerf [16] is another concurrent and complemental benchmark suite including seven training and five inference AI benchmarks.

Xu et al. [29] proposed a resource allocation model in the edge computing platform. Cao et al. [3] proposed a task allocation strategy to reduce resource consumption. However, these studies assume that deploying workloads on edge servers has the best performance, thus focus on resource allocation among only edge servers.

Chen et al. [5] proposed an optimal caching strategy for mobile services. However, they do not consider the multiple-job deployment problem. Feng et al. [7] proposed an optimal offload algorithm to maximize the data utility and minimize energy consumption. It only considers offloading computational tasks to MEC servers.

Richins et al. [21] characterize face recognition application in the edge computing framework and consider the end-to-end performance. Nevertheless, they do not evaluate other AI workloads that have different resource requirements.

X. Conclusions

Different placement or collaboration policies in handling datasets and workloads across cloud, edge, and user-end may substantially affect a cloud-edge computing environment's overall performance. This paper calls attention to optimize AI-oriented workloads' performance across all components in cloud-edge architectures holistically. We focus on the AI workloads allocation algorithm on cloud-edge computing hierarchically-structured framework, achieving the minimum response time in latency-sensitive applications.

We have proposed two efficient allocation algorithms to reduce the end-to-end response time of single-workload and multi-job scenarios, respectively. We apply six edge AI workloads from a comprehensive edge computing benchmark – Edge AIBench for experiments and conduct experiments in a real edge computing environment. We show our workload scheduling algorithms' effectiveness and efficiency by comparing them with the other four baseline strategies.

XI. Acknowledgements

This research is partially supported by the Shenzhen Institute of Artificial Intelligence and Robotics for Society (AIRS) and the Standardization Research Project of Chinese Academy of Sciences No.BZ201800001.

References

[1] Edge aibench specification. https://www.benchcouncil.org/file/EdgeAIBench_Specification.pdf. Accessed Dec 12, 2020.

[2] Mimic-iii database. https://mimic.physionet.org/gettingstarted/access/. Accessed Jan 6, 2020.

[3] Chunming Cao, Jin Wang, Jianping Wang, Kejie Lu, Jingya Zhou, Admela Jukan, and Wei Zhao. Optimal task allocation and coding design for secure coded edge computing. In *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*, pages 1083–1093. IEEE, 2019.

[4] Stephen Cass. Taking ai to the edge: Google's tpu now comes in a maker-friendly package. *IEEE Spectrum*, 56(5):16–17, 2019.

[5] Min Chen, Yongfeng Qian, Yixue Hao, Yong Li, and Jeungeun Song. Data-driven computing and caching in 5g networks: Architecture and delay analysis. *IEEE Wireless Communications*, 25(1):70–75, 2018.

[6] E Di Nardo, A Petrosino, and V Santopietro. Embedded deep learning for face detection and emotion recognition with intel© movidius (tm) neural compute stick. 2018.

[7] Jie Feng, Qingqi Pei, F Richard Yu, Xiaoli Chu, and Bodong Shang. Computation offloading and resource allocation for wireless powered mobile edge computing with latency constraint. *IEEE Wireless Communications Letters*, 2019.

[8] Mark Fernandes. Nodes, sockets, cores and flops, oh, my. https://www.dell.com/support/article/no/no/nobsdt1/SLN310893?lwp=rt. Accessed Jan 6, 2020.

[9] Tianshu Hao, Yunyou Huang, Xu Wen, Wanling Gao, Fan Zhang, Chen Zheng, Lei Wang, Hainan Ye, Kai Hwang, Zujie Ren, et al. Edge aibench: towards comprehensive end-to-end edge computing benchmarking. In *International Symposium on Benchmarking, Measuring and Optimization*, pages 23–30. Springer, 2018.

[10] Gary B Huang, Marwan Mattar, Tamara Berg, and Eric Learned-Miller. Labeled faces in the wild: A database forstudying face recognition in unconstrained environments. 2008.

[11] Kai Hwang and Min Chen. *Big-data analytics for cloud, IoT and cognitive computing*. John Wiley & Sons, 2017.

[12] Andrey Ignatov, Radu Timofte, William Chou, Ke Wang, Max Wu, Tim Hartley, and Luc Van Gool. Ai benchmark: Running deep neural networks on android smartphones. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 0–0, 2018.

[13] Alistair EW Johnson, Tom J Pollard, Lu Shen, H Lehman Li-wei, Mengling Feng, Mohammad Ghassemi, Benjamin Moody, Peter Szolovits, Leo Anthony Celi, and Roger G Mark. Mimic-iii, a freely accessible critical care database. *Scientific data*, 3:160035, 2016.

[14] Gang Lu, Jianfeng Zhan, Tianshu Hao, and Lei Wang. 10-millisecond computing. *arXiv preprint arXiv:1610.01267*, 2016.

[15] Chunjie Luo, Fan Zhang, Cheng Huang, Xingwang Xiong, Jianan Chen, Lei Wang, Wanling Gao, Hainan Ye, Tong Wu, Runsong Zhou, et al. Aiot bench: towards comprehensive benchmarking mobile and embedded device intelligence. In *International Symposium on Benchmarking, Measuring and Optimization*, pages 31–35. Springer, 2018.

[16] Peter Mattson, Vijay Janapa Reddi, Christine Cheng, Cody Coleman, Greg Diamos, David Kanter, Paulius Micikevicius, David Patterson, Guenther Schmuelling, Hanlin Tang, et al. Mlperf: An industry standard benchmark suite for machine learning performance. *IEEE Micro*, 40(2):8–16, 2020.

[17] Nenad Mladenović and Pierre Hansen. Variable neighborhood search. *Computers & operations research*, 24(11):1097–1100, 1997.

[18] Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. Pruning convolutional neural networks for resource efficient inference. *arXiv preprint arXiv:1611.06440*, 2016.

[19] Xingang Pan, Jianping Shi, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Spatial as deep: Spatial cnn for traffic scene understanding. *arXiv preprint arXiv:1712.06080*, 2017.

[20] Amir M Rahmani, Tuan Nguyen Gia, Behailu Negash, Arman Anzanpour, Iman Azimi, Mingzhe Jiang, and Pasi Liljeberg. Exploiting smart e-health gateways at the edge of healthcare internet-of-things: A fog computing approach. *Future Generation Computer Systems*, 78:641–658, 2018.

[21] Daniel Richins, Dharmisha Doshi, Matthew Blackmore, Aswathy Thulaseedharan Nair, Neha Pathapati, Ankit Patel, Brainard Daguman, Daniel Dobrijalowski, Ramesh Illikkal, Kevin Long, et al. Missing the forest for the trees: End-to-end ai application performance in edge data centers. In *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 515–528. IEEE, 2020.

[22] Florian Schroff, Dmitry Kalenichenko, and James Philbin. Facenet: A unified embedding for face recognition and clustering. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 815–823, 2015.

[23] Weisong Shi et al. The promise of edge computing. *Computer*, 49(5):78–81, 2016.

[24] Khurram Soomro, Amir Roshan Zamir, and Mubarak Shah. Ucf101: A dataset of 101 human actions classes from videos in the wild. *arXiv preprint arXiv:1212.0402*, 2012.

[25] Fei Tang et al. AIBench Training: Balanced Industry-Standard AI Training Benchmarking. In *2021 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE Computer Society, 2021.

[26] Xiaofei Wang, Yiwen Han, Chenyang Wang, Qiyang Zhao, Xu Chen, and Min Chen. In-edge ai: Intelligentizing mobile edge computing, caching and communication by federated learning. *IEEE Network*, 33(5):156–165, 2019.

[27] Ze Wang, Weiqiang Ren, and Qiang Qiu. Lanenet: Real-time lane detection networks for autonomous driving. *arXiv preprint arXiv:1807.01726*, 2018.

[28] Samuel Williams, Andrew Waterman, and David Patterson. Roofline: An insightful visual performance model for floating-point programs and multicore architectures. Technical report, Lawrence Berkeley National Lab.(LBNL), Berkeley, CA (United States), 2009.

[29] Jinlai Xu, Balaji Palanisamy, Heiko Ludwig, and Qingyang Wang. Zenith: Utility-aware resource allocation for edge computing. In *2017 IEEE International Conference on Edge Computing (EDGE)*, pages 47–54. IEEE, 2017.

[30] Amelie Chi Zhou, Yifan Gong, Bingsheng He, and Jidong Zhai. Efficient process mapping in geo-distributed cloud data centers. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, page 16. ACM, 2017.