IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received May 28, 2020, accepted June 18, 2020, date of publication June 23, 2020, date of current version July 2, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3004509

Computation Offloading and Task Scheduling for DNN-Based Applications in Cloud-Edge Computing

ZHEYI CHEN[1], JUNQIN HU[2], XING CHEN[2], JIA HU[1], XIANGHAN ZHENG[2], AND GEYONG MIN[1]

[1]College of Engineering, Mathematics and Physical Sciences, University of Exeter, Exeter EX4 4QF, U.K.
[2]College of Mathematics and Computer Science, Fuzhou University, Fuzhou 350108, China

Corresponding authors: Xing Chen (chenxing@fzu.edu.cn) and Jia Hu (j.hu@exeter.ac.uk)

This work was supported by the National Key R&D Program of China under Grant No. 2018YFB1004800, the Talent Program of Fujian Province for Distinguished Young Scholars in Higher Education, the Guiding Project of Fujian Province under Grant No. 2018H0017, and the China Scholarship Council.

ABSTRACT Due to the high demands of deep neural network (DNN) based applications on computational capability, it is hard for them to be directly run on mobile devices with limited resources. Computation offloading technology offers a feasible solution by offloading some computation-intensive tasks of neural network layers to edges or remote clouds that are equipped with sufficient resources. However, the offloading process might lead to excessive delays and thus seriously affect the user experience. To address this important problem, we first regard the average response time of multi-task parallel scheduling as our optimization goal. Next, the problem of computation offloading and task scheduling for DNN-based applications in cloud-edge computing is formulated with a scheme evaluation algorithm. Finally, the greedy and genetic algorithms based methods are proposed to solve the problem. The extensive experiments are conducted to demonstrate the effectiveness of the proposed methods for scheduling tasks of DNN-based applications in different cloud-edge environments. The results show that the proposed methods can obtain the near-optimal scheduling performance, and generate less average response time than traditional scheduling schemes. Moreover, the genetic algorithm leads to less average response time than the greedy algorithm, but the genetic algorithm needs more running time.

INDEX TERMS Cloud-edge computing, DNN-based applications, computation offloading, task scheduling, greedy algorithm, genetic algorithm.

I. INTRODUCTION

With the rapid development of deep learning (DL) [1], deep neural network (DNN) based applications, such as personalized recommendation systems [2], face recognition systems [3], and license plate recognition systems [4], have become an integral part of people's daily life. The high intelligence of DNN-based applications relies on large-scale and complex DNNs, and thus they commonly require sufficient resources and lead to high energy consumption [5]. However, mobile systems are usually equipped with limited resources [6], including battery life, network bandwidth, storage capacity, and processor performance. Thus, complex DNN-based applications cannot be directly run on mobile devices. One feasible solution is to offload all or part of computational tasks to remote clouds with sufficient resources [7]. More specifically, DNNs are first divided by the granularity of neural network layers [8]. Next, some computationally-complex neural network layers are offloaded to remote clouds for execution, while other tasks with simpler neural network layers are processed locally. Finally, the results are returned and integrated on mobile devices.

However, offloading tasks to remote clouds is significantly limited by the distance between users and remote clouds. Such long-distance leads to huge delays that might cause applications lagging with frequent user interactions, and it seriously affects the user experience [9]. Moreover, the

The associate editor coordinating the review of this manuscript and approving it for publication was Shagufta Henna.

leakage of user privacy might happen when offloading tasks to remote clouds. With the rise of edge computing, mobile edges have become the main platforms for implementing computation offloading [10]. Compared with remote clouds, mobile edges are closer to the user data and provide services nearby. Therefore, they can offer a faster network service response and meet the basic requirements of users for privacy protection [11]. However, it is difficult to realize computation offloading for DNN-based applications, due to the geographical distribution of mobile edges and the mobility of mobile devices [12]. To address this problem, in the early work of this paper, we designed an adaptive offloading framework for DNN-based applications in mobile edge environments. Correspondingly, we proposed a design pattern and reconstruction method to support the computation offloading of DNN-based applications.

In the process of computation offloading, task scheduling has become a new challenge [13]–[16], where various types of delays occur. For example, the data transmission delay happens when the data is transmitted between different computing nodes. After tasks are offloaded to target nodes, they might need to wait in queues due to the limited concurrency capability of nodes, and it results in the waiting delay. If the total delays of offloading are excessive, the average response time of tasks will be significantly increased, and thus it will seriously affect the user experience. Besides, different DNN-based tasks require various amounts of data transmission, while the network connections and data transmission rates between nodes are also diverse. Therefore, different scheduling schemes might lead to different delays, and it has become a tough issue to find an optimal scheduling scheme with the lowest average response time. The traditional computation offloading schemes are to offload all tasks to mobile edges or remote clouds for execution. However, they result in huge data transmission time. Therefore, it is necessary to design an effective scheduling scheme, especially when multiple tasks are executed concurrently.

To solve these problems, we propose an effective method for offloading and scheduling DNN-based applications in cloud-edge environments. The optimization goal is to reduce the average response time of multi-task parallel scheduling. Moreover, the proposed method is able to make scheduling decisions with high-efficiency in response to the mobility of mobile devices. The main contributions of this paper are summarized as follows.

- The problem of computation offloading and task scheduling for DNN-based applications in cloud-edge computing is formulated. Meanwhile, a scheme evaluation algorithm is designed to evaluate the solutions.
- A greedy algorithm based method is first proposed to address the problem, and it can achieve a near-optimal scheduling scheme in a short time. Next, a genetic algorithm based method is developed with better scheduling performance, but it requires more running time than the greedy algorithm.
- The extensive experiments are conducted to validate the effectiveness of the proposed methods under different scenarios of cloud-edge environments. The results show that the proposed methods achieve less average response time than traditional scheduling schemes.

The rest of this paper is organized as follows. In Section II, the related work is analyzed. Section III formulates the problem of offloading and scheduling for DNN-based applications in cloud-edge computing, and a scheme evaluation algorithm is introduced in Section IV. Section V and Section VI discuss the greedy and genetic algorithms based methods for the scheduling problem, respectively. In Section VII, the proposed methods are evaluated. Finally, we conclude this paper and look for future work in Section VIII.

II. RELATED WORK

To relieve the limitation of mobile devices on computational capability, local tasks can be partially offloaded to remote clouds by using cloud computing technology. As a new type of business computing model, cloud computing is regarded an extension of distributed processing, parallel processing, and grid computing [17], [18]. At the early stages, most of the researches for computation offloading rely on cloud environments. Suradkar and Bharati [19] pointed out that offloading computing-intensive tasks to cloud platforms can improve battery life and the performance of mobile devices. For example, a computation offloading system (Phone2Cloud) was designed in [20], and it can offload application tasks from smart-phones to the remote cloud.

However, a lot of delays are generated when offloading application tasks to remote clouds. Thus, this offloading scheme is unsuitable for real-time applications. To address this issue, mobile edge computing (MEC) has emerged as a promising way to optimize the performance of computation offloading [21], [22]. With the rapid development of MEC, the research focus of computation offloading has gradually developed from clouds to edges. Moreover, Jeong *et al.* [7] indicated that machine learning (ML) based applications (especially using DNNs) consume a large number of computational resources. Therefore, mobile devices with limited computational capability cannot well support DNN-based applications. One feasible solution is to offload partial computing tasks of DNNs from mobile devices to nearby edge servers.

To better perform the computation offloading for DNNs, Kang *et al.* [8] designed a lightweight scheduler (Neurosurgeon) that can automatically divide the computation of DNNs at the granularity of neural network layers between clouds and mobile edges. To avoid the high latency and connection errors caused by offloading all DNNs to external devices, Saguil and Azim [23] proposed that some DNNs should be executed locally while the others can be split and offloaded to different devices. Moreover, many researchers have contributed to the problems of computation offloading and task scheduling. For example, Jia *et al.* [24] proposed an online heuristic algorithm for task offloading that

can minimize the completion time of applications on mobile devices. Based on the Lyapunov optimization, a dynamic computation offloading algorithm was developed in [25] to jointly determine offloading decisions and CPU-cycle frequencies for mobile execution and transmit power. Liu *et al.* [26] formulated the problem of delay minimization with power constraint, and then proposed a one-dimensional search algorithm to explore the optimal scheme of task scheduling. Guo *et al.* [27] provided an energy-efficient dynamic offloading and scheduling strategy, in order to reduce energy consumption and shorten the complement time of applications. Besides, the task scheduling was first modeled as an optimization problem in [28], and then a two-stage task scheduling cost optimization (TTSCO) algorithm was proposed to reduce the cost of edge computing systems by offloading the latency-sensitive tasks of IoT devices to the edge cloud. Its goal is to minimize the computational cost and meet the delay requirements of tasks. Moreover, a novel many-objective optimization algorithm based on hybrid angles (MaOEA-HA) was proposed in [29] to enhance the performance of task scheduling in cloud computing. Rahmani Hosseinabadi *et al.* [30] studied the selection of crossover and mutation operators in the genetic algorithm for addressing the open-shop scheduling problem (OSSP). To optimize the task scheduling problem, Keshanchi *et al.* [31] designed an improved genetic algorithm by integrating the evolutionary genetic algorithm with heuristics. Similarly, Ahmad *et al.* [32] improved the genetic algorithm by involving a heuristic in genetic operators and developed a hybrid genetic algorithm for scheduling workflow applications in heterogeneous computing systems.

Different from the above work, the research objective of this paper is DNN-based applications. There are some researches about the computation offloading problem for DNN-based applications. For example, Qi *et al.* [33] designed an adaptive scheduling algorithm for choosing the processing environments (e.g., clouds or mobile devices) for DNN-based system models, according to the network condition between the remote cloud and mobile devices. However, only the current network condition is considered. If the network condition is good, the models can be offloaded to the remote cloud. Otherwise, the models should be processed locally. Therefore, this method cannot be applied to the complex multi-task scheduling problem proposed in this paper.

III. PROBLEM FORMULATION

A. CLOUD-EDGE ENVIRONMENT

In a cloud-edge environment, there are commonly three types of computational resources, including mobile devices, edge nodes, and a remote cloud. In general, these resources are with different levels of performance, which are mainly reflected in their computational capability and concurrency.

Assume that there are k computational resources (denoted by $S = \{s_1, s_2, \ldots, s_k\}$) in a cloud-edge environment. Among these resources, there are a mobile devices (denoted by $M = \{m_1, m_2, \ldots, m_a\}$), b edge nodes (denoted by $E = \{e_1, e_2, \ldots, e_b\}$), and a remote cloud (denoted by c). For the clarity of presentation, these computational resources are regarded as k nodes, and each node is denoted as s_i ($i \in [1, k]$). More specifically, each mobile device is denoted as m_i ($i \in [1, a]$), and it corresponds to the node set $\{s_1, s_2, \ldots, s_a\}$. For example, the mobile device m_1 corresponds to the node s_1, the mobile device m_2 corresponds to the node s_2, and so on. Each edge node is denoted as e_i ($i \in [1, b]$), and it corresponds to the node set $\{s_{a+1}, s_{a+2}, \ldots, s_{a+b}\}$. For example, the edge node e_1 corresponds to the node s_{a+1}, the edge node e_2 corresponds to the node s_{a+2}, and so on. Moreover, the remote cloud c corresponds to the node s_k.

Besides, p_i ($i \in [1, k]$) is used to indicate the number of concurrent lanes of the node s_i, and it represents the maximum number of tasks that can be concurrently processed on the node s_i. For example, if the number of concurrent lanes of a node is 3, up to 3 tasks can be simultaneously processed on the node, where each concurrent lane can be used to process a task.

Next, the connections between different nodes are represented by the two-dimensional matrices V and R as

$$V = \begin{bmatrix} v_{1,1} & \cdots & v_{1,k} \\ \vdots & \ddots & \vdots \\ v_{k,1} & \cdots & v_{k,k} \end{bmatrix}, \quad R = \begin{bmatrix} r_{1,1} & \cdots & r_{1,k} \\ \vdots & \ddots & \vdots \\ r_{k,1} & \cdots & r_{k,k} \end{bmatrix}, \tag{1}$$

where $v_{i,j}$ is the data transmission rate between the nodes s_i and s_j, and $r_{i,j}$ is the response time between the nodes s_i and s_j.

Moreover, the values of $v_{i,j}$ and $r_{i,j}$ are defined as

$$v_{i,j} = \begin{cases} C^v_{i,j}, & s_i \text{ is connected to } s_j \ \& \ i \neq j \\ \infty, & s_i \text{ is connected to } s_j \ \& \ i = j \\ 0, & s_i \text{ is not connected to } s_j, \end{cases} \tag{2}$$

$$r_{i,j} = \begin{cases} C^r_{i,j}, & s_i \text{ is connected to } s_j \ \& \ i \neq j \\ 0, & s_i \text{ is connected to } s_j \ \& \ i = j \\ \infty, & s_i \text{ is not connected to } s_j, \end{cases} \tag{3}$$

where $C^v_{i,j}$ and $C^r_{i,j}$ are the constant values under different connection conditions.

B. DESCRIPTION OF DNN-BASED TASKS

In a DNN-based application with m neural network layers, each layer is regarded as a subtask. Thus, each DNN-based task consists of m different subtasks. Assume that the scheduling process is with n tasks, and the task set is denoted as $T = \{T_1, T_2, \ldots, T_n\}$. Meanwhile, each task can be denoted as $T_i = \{t_{i,1}, t_{i,2}, \ldots, t_{i,m}\}$, where $t_{i,j}$ represents the j-th subtask of the i-th task. Moreover, each task is generated on a mobile device and arrives with a rate of λ_i, where $i \in [1, a]$.

As each neural network layer in DNNs is processed orderly, m subtasks of a DNN-based task are also processed orderly. For example, the subtask $t_{i,j}$ will not be generated until the subtask $t_{i,j-1}$ is processed. Similarly, the subtask $t_{i,j+1}$ will only be generated after the subtask $t_{i,j}$ is processed.

Next, $time_{i,j}$ is used to represent the processing time for the subtask $t_{x,i}$ of the task T_x on the node s_j. Thus, the set of processing time for the subtasks on different nodes is defined as

$$Time = \begin{bmatrix} time_{1,1} & \cdots & time_{1,k} \\ \vdots & \ddots & \vdots \\ time_{m,1} & \cdots & time_{m,k} \end{bmatrix}. \quad (4)$$

When the same subtask is processed on different nodes, the nodes with stronger computational capability lead to the smaller value of $time_{i,j}$. Similarly, when different subtasks are processed on the same node, the subtasks with smaller resource requirements result in the smaller value of $time_{i,j}$.

Besides, $D = \{d_1, d_2, \ldots, d_m\}$ is used to represent the set of data transmission volume between different subtasks of a task, where d_j indicates the data transmission volume between the subtasks $t_{i,j}$ and $t_{i,j+1}$.

C. FORMAL DEFINITION OF PROBLEM

During the process of task scheduling, $t_{i,j}.arrival$ is used to indicate the time when the subtask $t_{i,j}$ arrives at the node s_y, and $t_{i,j}.begin$ is used to represent the time when the subtask $t_{i,j}$ begins to be processed. Therefore, the waiting time of the subtask $t_{i,j}$ on the node s_y is defined as

$$w_{i,j}(y) = t_{i,j}.begin - t_{i,j}.arrival. \quad (5)$$

Next, the data transmission time of the subtask $t_{i,j}$ between the nodes s_x and s_y is defined as

$$g_{i,j}(x, y) = max\left(\frac{d_j}{v_{x,y}}, r_{x,y}\right). \quad (6)$$

Moreover, the response time of the task T_i is calculated from its generation to completion, and it is equal to the sum of the response time of all subtasks. More specifically, the response time $r_{i,j}$ of the subtask $t_{i,j}$ consists of three components, including the processing time, data transmission time, and waiting time, which can be denoted as

$$r_{i,j}(y) = time_{j,y} + g_{i,j}(x, y) + w_{i,j}(y). \quad (7)$$

Therefore, the response time of the task T_i can be calculated by

$$f_{resp}(T_i) = \sum_{j=1}^{m} r_{i,j}(y). \quad (8)$$

Correspondingly, the average response time of n tasks is defined as

$$f_{ave}(T) = \frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{m} r_{i,j}(y). \quad (9)$$

Based on the problem formulation, the objective of our work is to find an optimal scheduling scheme and use it to schedule the tasks of DNN-based applications, in order to minimize the average response time $f_{ave}(T)$. Therefore, the objective function is needed to measure and guide the potential scheduling schemes for achieving the optimal one with the lowest value of $f_{ave}(T)$. Meanwhile, the scheduling scheme can specify the processing node for each subtask and the processing order of subtasks on different nodes.

To solve this problem, one simple idea is to explore all possible scheduling schemes and find one with the lowest average response time. However, this strategy is with exponential complexity while it requires a large amount of running time. Therefore, it is necessary to design a more efficient method to solve this complicated problem of task scheduling.

IV. SCHEME EVALUATION

The scheme evaluation algorithm is used to evaluate scheduling schemes, where the average response time of a specific scheduling scheme is calculated. In general, better scheduling schemes lead to less average response time. In this paper, the proposed scheduling algorithms are optimized based on the results of the scheme evaluation algorithm, where the scheme evaluation algorithm simulates the scheduling process according to a specific scheme. During this process, the scheme evaluation algorithm first records the arrival and completion time of each subtask, and then it calculates the average response time of a scheduling scheme.

More specifically, *curTime* is first used to indicate the current time with an initial value of 0. Next, a scheduling scheme (denoted by *scheme*) is represented by a two-dimensional array with k rows. This array corresponds to k nodes, where each row orderly records the subtasks that will be processed on a node. Moreover, each subtask $t_{i,j}$ is with three attributes, including $t_{i,j}.arrival$, $t_{i,j}.end$, and $t_{i,j}.time$. These three attributes indicate the time when the subtask $t_{i,j}$ arrives at the node, the time when the subtask $t_{i,j}$ is completed, and the remaining processing time of the subtask $t_{i,j}$, respectively. They are initialized as

$$\begin{cases} t_{i,j}.arrival = \begin{cases} 0, & i = 1 \ \& \ j = 0 \\ None, & other\ cases \end{cases} \\ t_{i,j}.end = None \\ t_{i,j}.time = time_{i,j} \end{cases}.$$

The response time of a task is the time elapsed from its generation to completion. The time when the task T_i is generated is the time when its first subtask $t_{i,1}$ arrives (denoted by $t_{i,1}.arrival$), while the completion time of the task T_i is the time when its last subtask $t_{i,m}$ is completed (denoted by $t_{i,m}.end$). Therefore, the response time of the task T_i is defined as

$$f_{resp}(T_i) = t_{i,m}.end - t_{i,1}.arrival. \quad (10)$$

Therefore, the average response time can be calculated by

$$f_{ave}(T) = \frac{1}{n}\sum_{i=1}^{n}(t_{i,m}.end - t_{i,1}.arrival). \quad (11)$$

As shown in Algorithm 1, the key steps of calculating the average response time of a scheduling scheme are as follows.

Algorithm 1 The Scheme Evaluation

Input: A scheduling scheme (denoted by *scheme*).

Output: The arrival time of each subtask (denoted by $t_{i,j}.arrival$), the completion time of each subtask (denoted by $t_{i,j}.end$), and the average response time (denoted by $f_{ave}(T)$).

1: **Initialize** $t_{i,j}.time \leftarrow t_{i,j}$, $curTime \leftarrow 0$, $t_{i,j}.arrival \leftarrow (0 or None)$, and $t_{i,j}.end \leftarrow None$.
2: $slice \leftarrow \infty$.
3: # Fill lanes.
4: **for** s_x **in** S **do**
5: **while** $s_x.empty > 0$ **do**
6: **if** $t_{i,j}.arrival \leq curTime$ **then**
7: Add $t_{i,j}$ into $pool_x$.
8: Remove $t_{i,j}$ from *scheme*.
9: $s_x.empty \leftarrow (s_x.empty - 1)$.
10: **end if**
11: **end while**
12: **end for**
13: # Find the smallest time slice.
14: **for** s_x **in** S **do**
15: **for** $t_{i,j}$ **in** $pool_x$ **do**
16: **if** $t_{i,j}.time \leq slice$ **then**
17: $slice \leftarrow t_{i,j}.time$.
18: **end if**
19: **end for**
20: **end for**
21: $curTime \leftarrow (curTime + slice)$.
22: # Calculate the remaining processing time of subtasks.
23: **for** s_x **in** S **do**
24: **for** $t_{i,j}$ **in** $pool_x$ **do**
25: $t_{i,j}.time \leftarrow (t_{i,j}.time - slice)$.
26: # Generate the new subtask.
27: **if** $t_{i,j}.time \leq 0$ **then**
28: $t_{i,j}.end \leftarrow curTime$.
29: Remove $t_{i,j}$ from *scheme*.
30: $s_x.empty \leftarrow (s_x.empty + 1)$.
31: # Assume that the new subtask is processed on the node s_y.
32: **if** $j \neq m$ **then**
33: $t_{i,j+1}.arrival \leftarrow (curTime + g_{i,j+1}(x, y))$.
34: **end if**
35: **end if**
36: **end for**
37: **end for**
38: **for** $t_{i,j}$ **in** *scheme* **do**
39: **if** $(t_{i,j}.arrival = None) \parallel (t_{i,j}.end = None)$ **then**
40: **goto** line 4.
41: **end if**
42: **end for**
43: # Calculate the average response time of *scheme*.
44: $f_{ave}(T) \leftarrow \frac{1}{n}\sum_{i=1}^{n}(t_{i,m}.end - t_{i,1}.arrival)$.

Step 1: Fill lanes. According to *scheme*, the subtasks on each node are orderly placed into lanes until there is no idle lane on the node. More specifically, $s_x.empty$ is used to indicate the number of idle lanes on the node s_x, and the subtasks that have been placed into lanes will be removed from *scheme*. The subtask $t_{i,j}$ in a lane needs to meet the condition (i.e., $t_{i,j}.arrival \leq curTime$), which means that this subtask must have arrived at the node s_x. Moreover, $pool_x$ is used to represent the subtasks that have been placed into the lanes of the node s_x.

Step 2: Find the smallest time slice. First, the subtasks in each lane will be traversed, in order to find the subtask $t_{i,j}$ with the smallest value of $t_{i,j}.time$. Next, $t_{i,j}.time$ is regarded as a time slice (denoted by *slice*), and *curTime* is updated by adding *slice*.

Step 3: The remaining processing time of subtasks in lanes is subtracted by *slice*, which indicates that the subtasks have been executed for a *slice* of time. For the subtask $t_{i,j}$ on the node s_x, if $t_{i,j}.time \leq 0$, the subtask $t_{i,j}$ has been completed, and $t_{i,j}.end = curTime$ will be recorded. Next, the subtask $t_{i,j}$ will be removed from the lane, and $s_x.empty$ will be increased by 1. If the subtask $t_{i,j}$ is not of the last neural network layer (i.e., $j \neq m$), the following subtask $t_{i,j+1}$ will be generated. Thus, the processing node s_y for the subtask $t_{i,j+1}$ needs to be found, and the data transmission time can be calculated by

$$t_{i,j+1}.arrival = curTime + g_{i,j+1}(x, y). \tag{12}$$

Repeat the above steps until the arrival and completion time of all subtasks are determined. Finally, the average response time of *scheme* can be calculated by using Equation (11).

V. GREEDY ALGORITHM FOR SCHEDULING

The greedy algorithm always makes the current best choice as it is solving a problem. When it comes to the scheduling problem, the node with the lowest response time will always be chosen for processing newly-arriving subtasks. After all the subtasks are allocated to the processing nodes, a scheduling scheme can thus be generated.

According to Equation (7), the response time $r_{i,j}(y)$ of the subtask $t_{i,j}$ consists of the processing time, data transmission time, and waiting time. When the greedy algorithm makes decisions, it will always choose the node s_y that leads to the smallest $r_{i,j}(y)$.

On each node, the subtasks in waiting are sorted by their size (the required processing time). Based on the rule of the shortest job first (SJF), smaller jobs (with less processing time) will be processed with higher priority. For example, the processing order of the subtasks $t_{i,j}$ and $t'_{i,j}$ on the node s_y is determined by comparing $time_{j,y}$ and $time'_{j,y}$. Meanwhile, the beginning time of the subtask $t_{i,j}$ (i.e., $t_{i,j}.begin$) is also the completion time of the subtask that precedes it (i.e., $t_{i,j-1}.end$). Therefore, when calculating the waiting time $w_{i,j}(y)$ of the subtask $t_{i,j}$, both $t_{i,j}.begin$ and $t_{i,j}.arrival$ can be obtained by using Algorithm 1.

As shown in Algorithm 2, the key steps of the greedy algorithm for scheduling are as follows.

Step 1: Find all reachable nodes $s_x.avl$ of the node s_x. If there exists the network connection between two nodes

Algorithm 2 The Greedy Algorithm for Scheduling

Input: $\{S, M, E, c, p, V, R, T, Time, D, \lambda, x, k, m, n\}$.
Output: the processing node of the subtask $t_{i,j}$ (denoted by *minNode*).

```
# Find all reachable nodes of the node s_x.
for s_y in S do
    if v_{x,y} > 0 then
        Add s_y into s_x.avl.
    end if
end for
# Calculate the response time of each reachable node.
for s_y in s_x.avl do
    Call Algorithm 1 to calculate t_{i,j}.begin and t_{i,j}.arrival.
    w_{i,j}(y) ← (t_{i,j}.begin − t_{i,j}.arrival).
    g_{i,j}(x, y) ← max(d_j / v_{x,y}, r_{x,y}).
    r_{i,j}(y) ← (time_{j,y} + g_{i,j}(x, y) + w_{i,j}(y)).
end for
minNode ← s_x.
# Choose the node with the lowest response time.
for s_y in s_x.avl do
    if r_{i,j}(y) < r_{i,j}(minNode) then
        minNode ← s_y.
    end if
end for
```

while the data can be transmitted, these two nodes are mutually reachable.

Step 2: Calculate the waiting time $w_{i,j}(y)$, data transmission time $g_{i,j}(x, y)$, and response time $r_{i,j}(y)$ for each reachable node.

Step 3: Choose the node s_y with the lowest $r_{i,j}(y)$ to process the subtask $t_{i,j}$.

VI. GENETIC ALGORITHM FOR SCHEDULING

The genetic algorithm is considered as a useful meta-heuristic algorithm that can offer high-quality solutions to a wide range of combinatorial optimization problems, including the task scheduling problem [30]–[32].

In this paper, n tasks are divided into $(n * m)$ subtasks according to neural network layers. Each subtask corresponds to a gene loci, and thus there are $(n * m)$ gene loci in total. The numbering of a gene loci starts from 0, and the i-th gene loci corresponds to the subtask $t_{\frac{i}{m}+1,i\%m+1}$. For example, the 0th gene loci corresponds to the subtask $t_{1,1}$, the 1st gene loci corresponds to the subtask $t_{1,2}$, and the m-th gene loci corresponds to the subtask $t_{2,1}$. Moreover, the gene at each gene loci represents the node s_x for processing the subtask $t_{i,j}$. For instance, if the subtask $t_{1,1}$ is processed on the node s_1, the gene of the 0th gene loci is 1. Thus, there are k genes that correspond to k nodes.

More specifically, each individual u_i in the genetic algorithm is regarded as a potential scheduling scheme, where the average response time (denoted by $u_i.time$) is calculated by using Algorithm 1. Moreover, the population size is denoted as *size*, and the average response time of each generation of the population (denoted by *aveTime*) can be calculated by

$$aveTime = \frac{1}{size} \sum_{i=1}^{size} u_i.time. \tag{13}$$

Next, the individuals whose average response time is less than that of the population (i.e., $u_i.time < aveTime$) will be retained to the next generation. However, the selection operations reduce the population size. To maintain the initial population size, crossover operators are used to expand the offspring population size. For example, the individuals u_1 and u_2 are first selected from the remaining individuals according to the roulette selection method [34]. Next, these two individuals are used as the parent generation to perform the single-point crossover [35], and thus their offspring individuals u'_1 and u'_2 are generated. Finally, the above process is repeated until the population size reaches the initial value of *size*.

Besides, mutation operations are used to change the genes of the offspring population for increasing diversity. Therefore, premature convergence can be avoided [36]. More specifically, a random number (in the range of $[0, 0.1]$) is generated with a mutation rate of μ. If the random number is less than μ, mutation operations will be performed. When performing the operations, the number of mutated genes (denoted by *num*) is first randomly generated. Next, *num* gene loci are randomly generated, where the genes will be changed randomly. Thus, the genetic mutations of organisms in nature are simulated.

As shown in Algorithm 3, the key steps of the genetic algorithm for scheduling are as follows.

Step 1: Calculate the average response time of each individual (denoted by $u_i.time$) by using Algorithm 1.

Step 2: Find the best individual (denoted by *best*) with the minimum value of $u_i.time$.

Step 3: Calculate the average response time of the population (denoted by *aveTime*).

Step 4: Perform selection, crossover, and mutation operations, respectively.

VII. EXPERIMENTS

In this section, five different scenarios of a cloud-edge environment are simulated to evaluate the proposed greedy and genetic algorithms based offloading and scheduling methods for DNN-based applications.

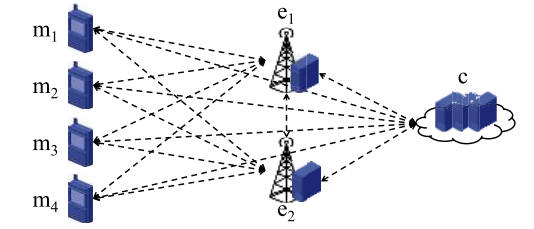
A. EXPERIMENTAL SETTINGS

We implement the cloud-edge simulation environments and the proposed scheduling methods for DNN-based applications based on Python 3.6, where NumPy is used to provide massive mathematical function libraries for array and matrix operations. As shown in Figure 1, we simulate the cloud-edge environment with different task arrivals. More specifically, Figure 1(a) depicts the node settings, including 4 mobile

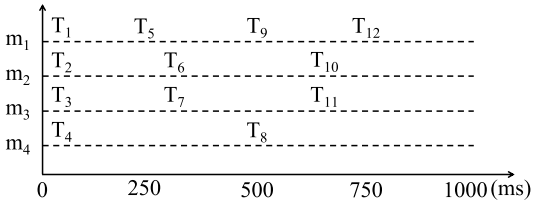
Algorithm 3 The Genetic Algorithm for Scheduling

Input: $\{S, M, E, c, p, V, R, T, Time, D, \lambda, x\}$.
Output: A scheduling scheme (denoted by *scheme*).

Initialize the first generation of the population.
Call Algorithm 1 to calculate the average response time of each individual (denoted by $u_i.time$).
$best \leftarrow u_0$.
# Find the best individual (denoted by *best*).
for $i \leftarrow 1$ **to** *size* **do**
    if $u_i.time < best.time$ **then**
        $best \leftarrow u_i$.
    end if
end for
# Calculate the average response time of the population.
$aveTime \leftarrow \frac{1}{size}\sum_{i=1}^{size} u_i.time$.
# Selection operations.
for $i \leftarrow 1$ **to** *size* **do**
    if $u_i.time < aveTime$ **then**
        Add u_i into *newPopulation*.
    end if
end for
# Crossover operations.
while len(*newPopulation*) $<$ *size* **do**
    Perform the single-point crossover.
end while
Perform mutation operations.



(a) Node settings.



(b) The timing diagram of task arrivals.

FIGURE 1. Simulation of cloud-edge environment with different task arrivals.

devices (i.e., m_1, m_2, m_3 and m_4), 2 edge nodes (i.e., e_1 and e_2), and a remote cloud c. The detailed performance metrics of the nodes are shown in Table 1. As for the essential parameters of the proposed methods, we set the population size as 1000, the maximum number of iterations as 500, and the mutation rate as 0.05.

TABLE 1. Performance metrics of different nodes.

Node	m_1	m_2	m_3	m_4	e_1	e_2	c
CPU cycle (GHz)	2.2	2.2	2.2	2.2	2.6	3.0	3.4
Memory (GB)	4	4	4	4	8	8	16
CPU core	2	2	2	2	4	4	8

In the experiments, a DNN-based application with 7 neural network layers is simulated, where the number of tasks is 12 (numbered from 1 to 12). Thus, there are 84 subtasks in total. Figure 1(b) shows the tasks generated on mobile devices and the arrival time of each task. More specifically, the tasks T_1, T_5, T_9 and T_{12} are generated on m_1, the tasks T_2, T_6 and T_{10} are generated on m_2, the tasks T_3, T_7 and T_{11} are generated on m_3, and the tasks T_4 and T_8 are generated on m_4, respectively. Meanwhile, the tasks on each mobile device arrive at a uniform speed within 1 second. For example, the 4 tasks on m_1 are with the task arrival rate of $\frac{1}{4}$ (one task arrives per 0.25 seconds).Moreover, the data transmission volume (Mb) between layers is $D = \{1.2, 0.3, 0.8, 0.2, 0.4, 0.1, 0.05\}$. Besides, the processing time (ms) of each neural network layer on nodes is

$$Time = \begin{bmatrix} 163 & 163 & 163 & 163 & 107 & 81 & 69 \\ 12 & 112 & 12 & 12 & 10 & 10 & 8 \\ 219 & 219 & 219 & 219 & 132 & 109 & 92 \\ 21 & 21 & 21 & 21 & 18 & 16 & 15 \\ 313 & 313 & 313 & 313 & 231 & 185 & 152 \\ 25 & 25 & 25 & 25 & 22 & 18 & 14 \\ 820 & 820 & 820 & 820 & 583 & 394 & 330 \end{bmatrix}.$$

Next, the following five different scenarios of cloud-edge environments are simulated with the above settings.

Scenario 1: The typical scenario. As shown in Table 2, the cloud-edge environment contains 2 edge nodes (i.e., e_1 and e_2) with the concurrent number of 2 (i.e., $p = 2$), 4 mobile devices (i.e., m_1, m_2, m_3 and m_4) with $p = 1$, and a remote cloud c with $p = 8$. c is connected to all mobile devices and edge nodes with the data transmission rate of 400 Kb/s and 600 Kb/s, respectively. e_1 is connected to m_2 and m_3, and e_2 is connected to m_1 and m_4, where the data transmission rate between an edge node and a mobile device is 2 Mb/s. But there is no connection between different mobile devices, neither for edge nodes.

Scenario 2: The scenario with limited mobile edges. As shown in Table 3, only one edge node (i.e., e_1) is simulated with $p = 2$, where e_1 is connected to all mobile devices and the remote cloud. Moreover, other settings are the same as **Scenario 1**.

Scenario 3: The scenario with sufficient mobile edges. As shown in Table 4, 2 edge nodes (i.e., e_1 and e_2) are simulated with $p = 3$. Moreover, other settings are the same as **Scenario 1**.

Scenario 4: The scenario with alternative mobile edges. As shown in Table 5, 2 edge nodes (i.e., e_1 and e_2) are simulated with $p = 2$ and $p = 3$, respectively. e_1 is

TABLE 2. The typical scenario.

Node	m_1 $(p=1)$	m_2 $(p=1)$	m_3 $(p=1)$	m_4 $(p=1)$	c $(p=8)$
e_1 $(p=2)$	×	r=30ms, v=2Mb/s	r=30ms, v=2Mb/s	×	r=100ms, v=600Kb/s
e_2 $(p=2)$	r=30ms, v=2Mb/s	×	×	r=30ms, v=2Mb/s	r=100ms, v=600Kb/s
c $(p=8)$	r=150ms, v=400Kb/s	r=150ms, v=400Kb/s	r=150ms, v=400Kb/s	r=150ms, v=400Kb/s	r=0ms, v=∞Kb/s

TABLE 3. The scenario with limited mobile edges.

Node	m_1 $(p=1)$	m_2 $(p=1)$	m_3 $(p=1)$	m_4 $(p=1)$	c $(p=8)$
e_1 $(p=2)$	r=30ms, v=2Mb/s	r=30ms, v=2Mb/s	r=30ms, v=2Mb/s	r=30ms, v=2Mb/s	r=100ms, v=600Kb/s
c $(p=8)$	r=150ms, v=400Kb/s	r=150ms, v=400Kb/s	r=150ms, v=400Kb/s	r=150ms, v=400Kb/s	r=0ms, v=∞Kb/s

TABLE 4. The scenario with sufficient mobile edges.

Node	m_1 $(p=1)$	m_2 $(p=1)$	m_3 $(p=1)$	m_4 $(p=1)$	c $(p=8)$
e_1 $(p=3)$	×	r=30ms, v=2Mb/s	r=30ms, v=2Mb/s	×	r=100ms, v=600Kb/s
e_2 $(p=3)$	r=30ms, v=2Mb/s	×	×	r=30ms, v=2Mb/s	r=100ms, v=600Kb/s
c $(p=8)$	r=150ms, v=400Kb/s	r=150ms, v=400Kb/s	r=150ms, v=400Kb/s	r=150ms, v=400Kb/s	r=0ms, v=∞Kb/s

TABLE 5. The scenario with alternative mobile edges.

Node	m_1 $(p=1)$	m_2 $(p=1)$	m_3 $(p=1)$	m_4 $(p=1)$	c $(p=8)$
e_1 $(p=2)$	×	r=30ms, v=2Mb/s	r=30ms, v=2Mb/s	r=30ms, v=2Mb/s	r=100ms, v=600Kb/s
e_2 $(p=3)$	r=30ms, v=1.4Mb/s	r=30ms, v=1.4Mb/s	r=30ms, v=1.4Mb/s	×	r=100ms, v=600Kb/s
c $(p=8)$	r=150ms, v=400Kb/s	r=150ms, v=400Kb/s	r=150ms, v=400Kb/s	r=150ms, v=400Kb/s	r=0ms, v=∞Kb/s

TABLE 6. The scenario with connected mobile edges.

Node	m_1 $(p=1)$	m_2 $(p=1)$	m_3 $(p=1)$	m_4 $(p=1)$	e_1 $(p=2)$	c $(p=8)$
e_1 $(p=2)$	×	r=30ms, v=2Mb/s	r=30ms, v=2Mb/s	×	r=0ms, v=∞Kb/s	r=100ms, v=600Kb/s
e_2 $(p=3)$	r=30ms, v=2Mb/s	×	×	r=30ms, v=2Mb/s	r=30ms, v=2Mb/s	r=100ms, v=600Kb/s
c $(p=8)$	r=150ms, v=400Kb/s	r=150ms, v=400Kb/s	r=150ms,v=400Kb/s	r=150ms, v=400Kb/s	r=100ms, v=600Kb/s	r=0ms, v=∞Kb/s

connected to m_2, m_3 and m_4 with the data transmission rate of 2 Mb/s, while e_2 is connected to m_1, m_2 and m_3 with the data transmission rate of 1.4 Mb/s. Moreover, other settings are the same as **Scenario 1**.

Scenario 5: The scenario with connected mobile edges. As shown in Table 6, 2 edge nodes (i.e., e_1 and e_2) are simulated with $p = 2$ and $p = 3$, respectively. Different from other scenarios, e_1 is connected to e_2 with the data transmission rate of 2 Mb/s. Moreover, other settings are the same as **Scenario 1**.

B. EXPERIMENTAL RESULTS

Based on the above settings, we evaluate the performance of the proposed greedy and genetic algorithms based offloading and scheduling methods for DNN-based applications. As shown in Figure 2, the proposed methods are compared with traditional scheduling schemes, including the load balancing scheme (all tasks are evenly offloaded to the remote cloud and nearby edge nodes), edge scheme (all tasks are offloaded to nearby edge nodes), and cloud scheme (all tasks are offloaded to the remote cloud).

As shown in Table 7, we compare the average response time generated by using the proposed methods with the optimal results under different scenarios.

TABLE 7. Comparison of the average response time with the optimal results under different scenarios.

Scenario	1	2	3	4	5
Greedy algorithm (ms)	1603	1629	1560	1584	1595
Genetic algorithm (ms)	1472	1614	1417	1575	1530
Optimal scheme (ms)	1406	1587	1373	1524	1477

The above results show that the highest average response time is generated in **Scenario 2** with only one edge node. Thus, task congestion happens. In **Scenario 3**, edge nodes are with more concurrent numbers. Thus, task congestion is greatly relieved, and it leads to less average response time than the typical cloud-edge environment (**Scenario 1**). In **Scenario 4**, m_2 and m_3 are connected to the edge nodes with different performance. During the scheduling process, tasks tend to be processed on the edge node with better performance, where the task congestion happens. Thus, more

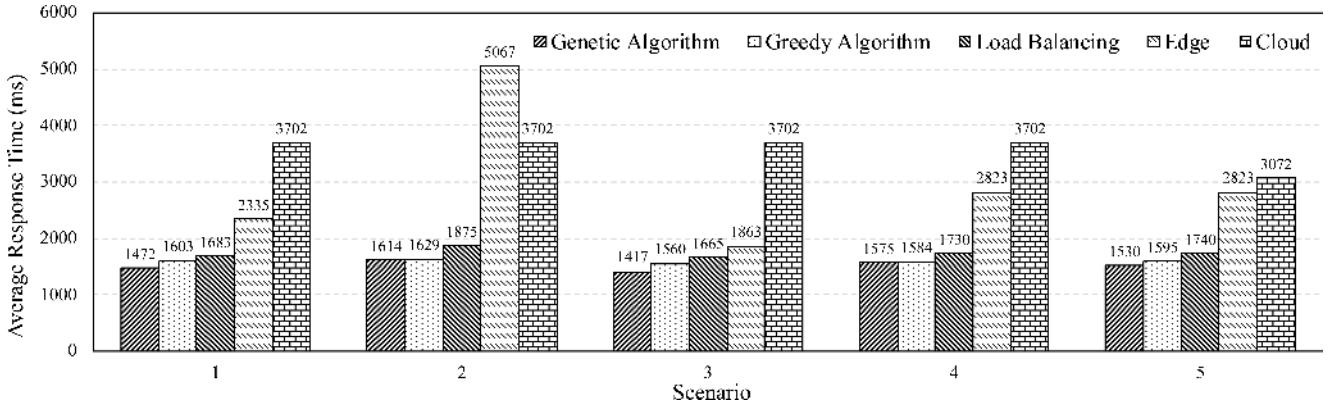
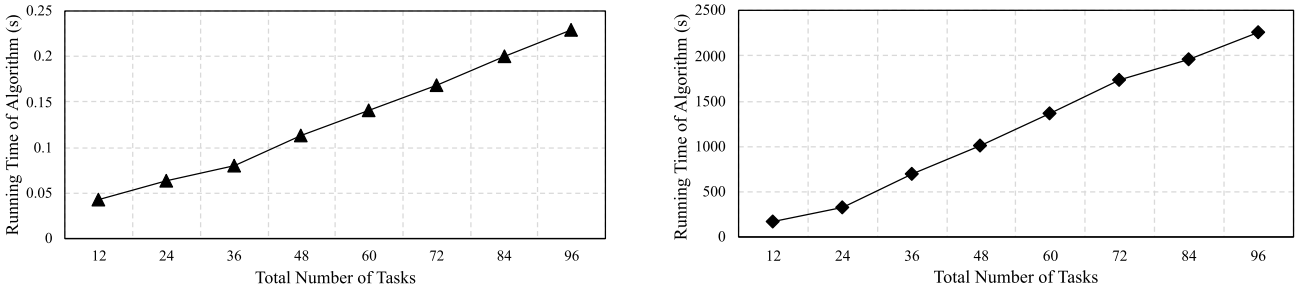


FIGURE 2. Comparison of the average response time among different scheduling schemes under various scenarios.



(a) Running time of the greedy algorithm.

(b) Running time of the genetic algorithm.

FIGURE 3. Influence of the total number of tasks on the running time of the greedy and genetic algorithms.

average response time is generated (only less than **Scenario 2**). In **Scenario 5**, edge nodes are connected, where the concurrent number of e_1 is less than e_2. Therefore, tasks tend to be processed on e_2. But tasks can be scheduled to e_1 when the task congestion occurs on e_2. Therefore, the congestion can be relieved, and the average response time generated in **Scenario 5** is less than **Scenario 4**. However, more average response time is still generated in **Scenario 5** than **Scenario 1** due to the data transmission delay between edge nodes.

In each scenario, the average response time generated by using the proposed methods is close to the optimal result, and they outperform other scheduling schemes. More specifically, the edge and cloud schemes result in much data transmission delay and task queuing, and thus their average response time is much larger than other schemes. As edge nodes are close to users, the average response time generated by using the edge scheme is less than the cloud scheme. In **Scenario 2**, there is only one edge node with limited computational capability, and thus more average response time is generated by using the edge scheme. In **Scenario 3**, there are two edge nodes with better performance, and thus the average response time generated by using the edge scheme is much less than the cloud scheme. Compared with the load balancing scheme, the proposed methods achieve less average response time. This is because that the load balancing scheme does not consider the data transmission delay. If the data transmission time is more than the waiting time, the response time generated by using this scheme will be increased. By contrast, the proposed methods consider the processing time, data transmission time, and waiting time. Therefore, the response time of offloading and scheduling for DNN-based applications in cloud-edge environments can be effectively reduced.

TABLE 8. Comparison of the running time between the greedy and genetic algorithms under different scenarios.

Scenario	1	2	3	4	5
Genetic algorithm (s)	172	174	173	176	178
Greedy algorithm (ms)	42	41	40	41	45

As shown in Table 8, we compare the running time between the genetic and greedy algorithms under different scenarios. Although the average response time generated by using the genetic algorithm is less than the greedy algorithm (as shown in Table 7), the running time of the genetic algorithm is much more than the greedy algorithm. Moreover, when the total number of tasks is constant, there is almost no change in the running time of these two algorithms under different scenarios.

Furthermore, the running time of the greedy and genetic algorithms is evaluated by changing the total number of tasks in **Scenario 1**. As shown in Figure 3, the running time of the

algorithms has a linear relationship with the total number of tasks.

In general, these two algorithms have their pros and cons. The greedy algorithm needs less running time, while the genetic algorithm offers better scheduling results. Moreover, the cloud-edge environment might change with the mobility of mobile devices. In an unstable environment, the greedy algorithm will be regarded as a better choice than using the genetic algorithm, because the greedy algorithm can provide faster decision-making speed.

VIII. CONCLUSION AND FUTURE WORK

In this paper, we first formulate the problem of computation offloading and task scheduling for DNN-based applications in cloud-edge environments and design a scheme evaluation mechanism. Next, the greedy and genetic algorithms based methods are proposed to efficiently explore the suitable schemes. The extensive experiments are conducted to verify the effectiveness of the proposed methods in different scenarios of cloud-edge environments. The results show that the genetic algorithm leads to less average response time than other scheduling schemes but needs more running time than the greedy algorithm. Therefore, these two proposed algorithms are suitable for different scenarios with diverse objectives. For example, the genetic algorithm is more suitable for offline tasks, since these tasks are not sensitive to the training time while the genetic algorithm can promise better application performance. By contrast, the genetic algorithm would be a better choice when dealing with online tasks, because these tasks require fast decision-making ability, which is also the advantage of the greedy algorithm. In the future, we will continue the research by using the learning-based methods, such as reinforcement learning, for better balancing the performance and overheads of algorithms.

ACKNOWLEDGMENT

(Zheyi Chen and Junqin Hu contributed equally to this work.)

REFERENCES

[1] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016.

[2] Z. Huang, C. Yu, J. Ni, H. Liu, C. Zeng, and Y. Tang, "An efficient hybrid recommendation model with deep neural networks," *IEEE Access*, vol. 7, pp. 137900–137912, 2019.

[3] G. Goswami, N. Ratha, A. Agarwal, R. Singh, and M. Vatsa, "Unravelling robustness of deep learning based face recognition against adversarial attacks," in *Proc. 32nd AAAI Conf. Artif. Intell. (AAAI)*, 2018, pp. 6829–6836.

[4] W. Weihong and T. Jiaoyang, "Research on license plate recognition algorithms based on deep learning in complex environment," *IEEE Access*, vol. 8, pp. 91661–91675, 2020.

[5] X. Chen, L. Jiao, W. Li, and X. Fu, "Efficient multi-user computation offloading for mobile-edge cloud computing," *IEEE/ACM Trans. Netw.*, vol. 24, no. 5, pp. 2795–2808, Oct. 2016.

[6] F. Wang, J. Xu, X. Wang, and S. Cui, "Joint offloading and computing optimization in wireless powered mobile-edge computing systems," *IEEE Trans. Wireless Commun.*, vol. 17, no. 3, pp. 1784–1797, Mar. 2018.

[7] H.-J. Jeong, I. Jeong, H.-J. Lee, and S.-M. Moon, "Computation offloading for machine learning Web apps in the edge server environment," in *Proc. IEEE 38th Int. Conf. Distrib. Comput. Syst. (ICDCS)*, Jul. 2018, pp. 1492–1499.

[8] Y. Kang, J. Hauswald, C. Gao, A. Rovinski, T. Mudge, J. Mars, and L. Tang, "Neurosurgeon: Collaborative intelligence between the cloud and mobile edge," in *Proc. 22nd Int. Conf. Architectural Support Program. Lang. Oper. Syst. (ASPLOS)*, 2017, pp. 615–629.

[9] M. Jia, W. Liang, Z. Xu, M. Huang, and Y. Ma, "QoS-aware cloudlet load balancing in wireless metropolitan area networks," *IEEE Trans. Cloud Comput.*, vol. 8, no. 2, pp. 623–634, Apr. 2020.

[10] B. Li, Z. Fei, J. Shen, X. Jiang, and X. Zhong, "Dynamic offloading for energy harvesting mobile edge computing: Architecture, case studies, and future directions," *IEEE Access*, vol. 7, pp. 79877–79886, 2019.

[11] X. Li, S. Liu, F. Wu, S. Kumari, and J. J. P. C. Rodrigues, "Privacy preserving data aggregation scheme for mobile edge computing assisted IoT applications," *IEEE Internet Things J.*, vol. 6, no. 3, pp. 4755–4763, Jun. 2019.

[12] E. Ahmed, A. Ahmed, I. Yaqoob, J. Shuja, A. Gani, M. Imran, and M. Shoaib, "Bringing computation closer toward the user network: Is edge computing the solution?" *IEEE Commun. Mag.*, vol. 55, no. 11, pp. 138–144, Nov. 2017.

[13] F. Sun, F. Hou, N. Cheng, M. Wang, H. Zhou, L. Gui, and X. Shen, "Cooperative task scheduling for computation offloading in vehicular cloud," *IEEE Trans. Veh. Technol.*, vol. 67, no. 11, pp. 11049–11061, Nov. 2018.

[14] Z. Chen, J. Hu, and G. Min, "Learning-based resource allocation in cloud data center using advantage actor-critic," in *Proc. IEEE Int. Conf. Commun. (ICC)*, May 2019, pp. 1–6.

[15] Y.-H. Chiang, T. Zhang, and Y. Ji, "Joint cotask-aware offloading and scheduling in mobile edge computing systems," *IEEE Access*, vol. 7, pp. 105008–105018, 2019.

[16] X. Chen, F. Zhu, Z. Chen, G. Min, X. Zheng, and C. Rong, "Resource allocation for cloud-based software services using prediction-enabled feedback control with reinforcement learning," *IEEE Trans. Cloud Comput.*, early access, May 4, 2020, doi: 10.1109/TCC.2020.2992537.

[17] S. Singh and I. Chana, "A survey on resource scheduling in cloud computing: Issues and challenges," *J. Grid Comput.*, vol. 14, no. 2, pp. 217–264, Jun. 2016.

[18] Z. Chen, J. Hu, G. Min, A. Y. Zomaya, and T. El-Ghazawi, "Towards accurate prediction for high-dimensional and highly-variable cloud workloads with deep learning," *IEEE Trans. Parallel Distrib. Syst.*, vol. 31, no. 4, pp. 923–934, Apr. 2020.

[19] J. Ashok and R. D. Bharati, "Computation offloading: Overview, frameworks and challenges," *Int. J. Comput. Appl.*, vol. 134, no. 6, pp. 28–31, Jan. 2016.

[20] F. Xia, F. Ding, J. Li, X. Kong, L. T. Yang, and J. Ma, "Phone2Cloud: Exploiting computation offloading for energy saving on smartphones in mobile cloud computing," *Inf. Syst. Frontiers*, vol. 16, no. 1, pp. 95–111, Mar. 2014.

[21] P. Mach and Z. Becvar, "Mobile edge computing: A survey on architecture and computation offloading," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 3, pp. 1628–1656, 3rd Quart., 2017.

[22] N. Abbas, Y. Zhang, A. Taherkordi, and T. Skeie, "Mobile edge computing: A survey," *IEEE Internet Things J.*, vol. 5, no. 1, pp. 450–465, Feb. 2018.

[23] D. Saguil and A. Azim, "A layer-partitioning approach for faster execution of neural network-based embedded applications in edge networks," *IEEE Access*, vol. 8, pp. 59456–59469, 2020.

[24] M. Jia, J. Cao, and L. Yang, "Heuristic offloading of concurrent tasks for computation-intensive applications in mobile cloud computing," in *Proc. IEEE Conf. Comput. Commun. Workshops (INFOCOM WKSHPS)*, Apr. 2014, pp. 352–357.

[25] Y. Mao, J. Zhang, and K. B. Letaief, "Dynamic computation offloading for mobile-edge computing with energy harvesting devices," *IEEE J. Sel. Areas Commun.*, vol. 34, no. 12, pp. 3590–3605, Dec. 2016.

[26] J. Liu, Y. Mao, J. Zhang, and K. B. Letaief, "Delay-optimal computation task scheduling for mobile-edge computing systems," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Jul. 2016, pp. 1451–1455.

[27] S. Guo, J. Liu, Y. Yang, B. Xiao, and Z. Li, "Energy-efficient dynamic computation offloading and cooperative task scheduling in mobile cloud computing," *IEEE Trans. Mobile Comput.*, vol. 18, no. 2, pp. 319–333, Feb. 2019.

[28] Y. Zhang, X. Chen, Y. Chen, Z. Li, and J. Huang, "Cost efficient scheduling for delay-sensitive tasks in edge computing system," in *Proc. IEEE Int. Conf. Services Comput. (SCC)*, Jul. 2018, pp. 73–80.

[29] S. Geng, D. Wu, P. Wang, and X. Cai, "Many-objective cloud task scheduling," *IEEE Access*, vol. 8, pp. 79079–79088, 2020.

[30] A. A. Rahmani Hosseinabadi, J. Vahidi, B. Saemi, A. K. Sangaiah, and M. Elhoseny, "Extended genetic algorithm for solving open-shop scheduling problem," *Soft Comput.*, vol. 23, no. 13, pp. 5099–5116, Jul. 2019.
[31] B. Keshanchi, A. Souri, and N. J. Navimipour, "An improved genetic algorithm for task scheduling in the cloud environments using the priority queues: Formal verification, simulation, and statistical testing," *J. Syst. Softw.*, vol. 124, pp. 1–21, Feb. 2017.
[32] S. G. Ahmad, C. S. Liew, E. U. Munir, T. F. Ang, and S. U. Khan, "A hybrid genetic algorithm for optimization of scheduling workflow applications in heterogeneous computing systems," *J. Parallel Distrib. Comput.*, vol. 87, pp. 80–90, Jan. 2016.
[33] B. Qi, M. Wu, and L. Zhang, "A DNN-based object detection system on mobile cloud computing," in *Proc. 17th Int. Symp. Commun. Inf. Technol. (ISCIT)*, Sep. 2017, pp. 1–6.
[34] N. M. Razali and J. Geraghty, "Genetic algorithm performance with different selection strategies in solving TSP," in *Proc. 19th World Congr. Eng. (WCE)*, 2011, pp. 1–6.
[35] P. Kora and P. Yadlapalli, "Crossover operators in genetic algorithms: A review," *Int. J. Comput. Appl.*, vol. 162, no. 10, pp. 34–36, 2017.
[36] S. M. Lim, A. B. M. Sultan, M. N. Sulaiman, A. Mustapha, and K. Y. Leong, "Crossover and mutation operators of genetic algorithms," *Int. J. Mach. Learn. Comput.*, vol. 7, no. 1, pp. 9–12, 2017.

ZHEYI CHEN received the B.Sc. degree in computer science from Shanxi University, China, in 2014, and the M.Sc. degree in computer science from Tsinghua University, China, in 2017. He is currently pursuing the Ph.D. degree in computer science with the University of Exeter. His research interests include cloud computing, mobile edge computing, deep learning, and resource optimization.

JUNQIN HU received the B.S. degree in computer science and technology from Fuzhou University, Fujian, China, in 2019, where he is currently pursuing the M.S. degree in computer software and theory with the College of Mathematics and Computer Science. Since September 2019, he has been a part of the Fujian Key Laboratory of Network Computing and Intelligent Information Processing, Fuzhou University. His current research interests include computation offloading, mobile edge computing, and cloud computing.

XING CHEN received the B.S. and Ph.D. degrees in computer software and theory from Peking University, Beijing, China, in 2008 and 2013, respectively. He is currently an Associate Professor and the Deputy Director of the Fujian Provincial Key Laboratory of Network Computing and Intelligent Information Processing, Fuzhou University, and leads the Systems Research Group. He has authored or coauthored more than 30 journal and conference articles. His research interests include the software systems and engineering approaches for cloud and mobility. His current projects cover the topics from self-adaptive software, computation offloading, model-driven approach, and so on. Dr. Chen was a recipient of the first Provincial Scientific and Technological Progress Award, in 2018.

JIA HU received the B.Eng. and M.Eng. degrees in electronic engineering from the Huazhong University of Science and Technology, China, in 2006 and 2004, respectively, and the Ph.D. degree in computer science from the University of Bradford, U.K., in 2010. He is currently a Lecturer of computer science with the University of Exeter. His research interests include mobile and ubiquitous computing, resource optimization, applied machine learning, and network security.

XIANGHAN ZHENG received the M.Sc. degree in distributed system and the Ph.D. degree in information communication technology from the University of Agder, Norway, in 2007 and 2011, respectively. He is currently a Professor with the College of Mathematics and Computer Sciences, Fuzhou University, China. His current research interests include new generation network with a special focus on cloud computing services and applications and big data processing and security.

GEYONG MIN received the B.Sc. degree in computer science from the Huazhong University of Science and Technology, China, in 1995, and the Ph.D. degree in computing science from the University of Glasgow, U.K., in 2003. He is currently a Professor of high performance computing and networking with the Department of Computer Science, College of Engineering, Mathematics, and Physical Sciences, University of Exeter, U.K. His research interests include the future Internet, computer networks, wireless communications, multimedia systems, information security, high-performance computing, ubiquitous computing, modeling, and performance engineering.