# Cost-Driven Off-Loading for DNN-Based Applications Over Cloud, Edge, and End Devices

Bing Lin, Yinhao Huang, Jianshan Zhang, Junqin Hu, Xing Chen, and Jun Li, *Senior Member, IEEE*

***Abstract*—Currently, deep neural networks (DNNs) have achieved a great success in various applications. Traditional deployment for DNNs in the cloud may incur a prohibitively serious delay in transferring input data from the end devices to the cloud. To address this problem, the hybrid computing environments, consisting of the cloud, edge, and end devices, are adopted to offload DNN layers by combining the larger layers (more amount of data) in the cloud and the smaller layers (less amount of data) at the edge and end devices. A key issue in hybrid computing environments is how to minimize the system cost while accomplishing the offloaded layers with their deadline constraints. In this article, a self-adaptive discrete particle swarm optimization (PSO) algorithm using the genetic algorithm (GA) operators is proposed to reduce the system cost caused by data transmission and layer execution. This approach considers the characteristics of DNNs partitioning and layers off-loading over the cloud, edge, and end devices. The mutation operator and crossover operator of GA are adopted to avert the premature convergence of PSO, which distinctly reduces the system cost through enhanced population diversity of PSO. The proposed off-loading strategy is compared with benchmark solutions, and the results show that our strategy can effectively reduce the system cost of off-loading for DNN-based applications over the cloud, edge and end devices relative to the benchmarks.**

***Index Terms*—Cloud computing, cost-driven off-loading, deep neural networks (DNNs), edge computing, workflow scheduling.**

Manuscript received July 31, 2019; revised November 12, 2019; accepted December 12, 2019. Date of publication December 24, 2019; date of current version April 13, 2020. This work was supported in part by the National Key R&D Program of China under Grant 2018YFB1004800, in part by the Natural Science Foundation of China under Grant 61972165, Grant 61872184, Grant 61727802, and Grant 41801324, in part by the Natural Science Foundation of Fujian Province under Grant 2019J01286 and Grant 2019J01244, in part by the Young and Middle-aged Teacher Education Foundation of Fujian Province under Grant JT180098, and in part by the Talent Program of Fujian Province for Distinguished Young Scholars in Higher Education. Paper no. TII-19-3472. *(Corresponding authors: Xing Chen; Jun Li.)*

B. Lin is with the College of Physics and Energy, Fujian Normal University, Fujian Provincial Key Laboratory of Quantum Manipulation and New Energy Materials, Fuzhou 350117, China, and with the Fujian Provincial Collaborative Innovation Center for Optoelectronic Semiconductors and Efficient Devices, Xiamen 361005, China, and also with the Engineering Research Center of Big Data Application in Private Health Medicine, Fujian Province University, Putian, Fujian 351100, China and also with the Fujian Provincial Collaborative Innovation Center for Advanced High-Field Superconducting Materials and Engineering, Fujian 350117, China (e-mail: WheelLX@163.com).

Y. Huang, J. Zhang, J. Hu, and X. Chen are with the College of Mathematics and Computer Science, Fuzhou University, Fuzhou 350118, China, and also with the Fujian Provincial Key Laboratory of Network Computing and Intelligent Information Processing, Fuzhou 350118, China (e-mail: fzuhyh@foxmail.com; zhangjs0512@163.com; jackinhu@qq.com; chenxing@fzu.edu.cn).

J. Li is with the School of Electronic and Optical Engineering, Nanjing University of Science and Technology, Nanjing 210094, China, and also with the School of Computer Science and Robotics, National Research Tomsk Polytechnic University, 634050 Tomsk, Russia (e-mail: jun.li@njust.edu.cn).

Color versions of one or more of the figures in this article are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TII.2019.2961237

## I. Introduction

CONTEMPORARILY, deep neural networks (DNNs) have achieved a great success in various applications, such as natural language processing, speech recognition, and computer vision [1]. Meanwhile, the number of Internet of Things (IoT) devices has increased dramatically. These end devices, equipped with sensors (e.g., microphones, cameras, and gyroscopes) for obtaining a large amount of environment data, are usually attractive to machine learning (ML) applications [2].

However, these IoT devices with limited energy and computing resources cannot afford computation-intensive tasks (e.g., DNNs). Performing classification directly on the IoT devices by simple ML model leads to low system accuracy [3]. As such, DNNs are conventionally deployed in the cloud with powerful computation capability. This results in a prohibitively serious delay when off-loading input data from sensors to DNNs in the cloud, due to the long distance between the cloud and IoT devices.

Mobile edge computing (MEC) is proposed as a promising computing model for solving the problem by deploying servers at the network edge close to the end devices [4], [5]. One solution to reducing the system delay of off-loading for DNN-based applications is to partition DNNs [6] in hybrid computing environments, consisting of the cloud, edge, and end devices, and combine the larger layers (more amount of data) in the cloud and the smaller layers (less amount of data) at the edge and end devices. In this way, the traffic load of core network and the transmission delay will be alleviated significantly, and the overall system accuracy will be improved [3].

Off-loading for DNN-based applications in MEC has been broadly studied [6]–[10]. These studies mostly focus on off-loading DNN layers to the edge instead of to the cloud. However, much less attention is paid to off-loading DNN layers

in hybrid computing environments [3]. It is a challenging task to partition DNNs and schedule different layers to their suitable servers for distinct applications while satisfying each application's deadline constraint. Moreover, when the input/intermediate data is scheduled to different cloud/edge servers, the cost of computation and transmission are also different. Therefore, how to minimize the system cost while accomplishing the offloaded layers within their deadlines in hybrid computing environments is still an open issue.

The time-driven data placement for a scientific workflow combining edge computing and cloud computing [11], as well as the cost-driven scheduling for deadline-based workflow across multiple clouds [12], have been addressed respectively. Off-loading DNN layers in hybrid computing environments and workflow scheduling across multiple clouds are both NP-hard problems with many similarities, such as the structure between the DNNs and the workflow [13]. In this article, we propose a self-adaptive particle swarm optimization (PSO) algorithm using the genetic algorithm (GA) operators (PSO-GA) to reduce the system cost caused by data transmission and layer execution, with the deadline constraints of all DNN-based applications. This approach considers the characteristics of DNNs partitioning and layers off-loading over the cloud, edge, and end devices.

The main contributions of this article are summarized as follows.

1) A cost-driven off-loading strategy based on PSO-GA is proposed to optimize the system cost during DNN layers off-loading over the cloud, edge, and end devices.
2) A preprocessing operation is proposed to compress the number of layers in a DNN, which leads to reducing the dimensions of a PSO particle.
3) An adjustment mechanism for the inertia weight of PSO-GA is designed to adaptively adjust the search ability according to the quality of current particle.

The remainder of this article is organized as follows. Section II reviews the related work. Section III discusses the process of cost-driven off-loading for DNN-based applications in hybrid computing environments. Section IV presents the proposed PSO-GA algorithm in detail. Section V compares our algorithm with other state-of-the-art algorithms. Finally, Section VI concludes this article.

## II. Related Work

DNNs achieve a great success in various fields, such as natural language processing and computer vision. Mobile devices cannot directly execute intelligent applications due to their limited computation resources. Many work offloaded DNN-based applications in cloud computing. Fang *et al.* [14] designed a simple heuristic scheduler for cloud-hosted DNN inference workloads, which satisfied the requirement on processing throughput. In addition, they proposed a deep reinforcement learning (RL) scheduler, which was trained to maximize the quality of service (QoS) including inference accuracy and response delay. This work greatly reduced the scheduling overhead when performing inference workloads on demand. Qi *et al.* [15] designed a DNN-based object detection system combining cloud and mobile devices. They proposed a model scheduling algorithm to adaptively select the execution platform (cloud or mobile devices) based on the conditions of network and mobile devices. This designed system had a better performance in running speed and detection accuracy. The above work mainly focused on off-loading DNN layers from end device to the cloud directly. However, the long-distance network communication between the cloud and end devices may cause a serious delay in execution, which failed to satisfy the requirements of real-time response in industry scenarios.

MEC is a novel computing model to solve the response delay problem for DNN-based applications by off-loading DNN layers from resource-constrained mobile devices to the edge. It is a promising way to partition DNNs in hybrid computing environments, and combine the larger layers in the cloud and the smaller layers at the edge and end devices [3], [16]. Jeong [6] proposed a lightweight off-loading system run on web-supported devices to offload DNN computations to edge servers. He designed a DNN partitioning algorithm to efficiently utilize the edge resources and reduce the system response time. However, this work ignored the difference in the computing capacity of each edge server. Wu *et al.* [17] proposed a novel off-loading partitioning algorithm to tackle the problem of dynamic application partitioning in mobile environments. The algorithm could arrive at the best tradeoffs between minimizing transmission delay/costs and saving energy/time. Teerapittayanon *et al.* [3] proposed a distributed computing hierarchy, consisting of the cloud, the edge, and end devices, for deploying DNN-based applications. This hierarchy exploited the geographical diversity of sensors to reduce communication cost and improve the object recognition accuracy of DNNs. However, they ignored the layer execution cost which was an important part of the system cost. Kang *et al.* [16] investigated DNN partitioning strategies that effectively offloaded DNN layers on the mobile edge and the cloud to achieve low energy consumption and latency. In addition, they proposed a lightweight scheduler called Neurosurgeon to automatically partition DNNs between the cloud and mobile edge. This work gave us inspiration, but it did not discuss minimizing the system cost with the deadline constraints.

The works discussed above mostly focused on reducing the system delay with the help of MEC [6], [9], [16]. There is little work aiming to minimize the system cost with deadline constraints. The data transmission cost was considered in [3] while deploying distributed DNNs in hybrid computing environments. However, it ignored the layer execution cost.

A DNN and a scientific workflow have many similarities, such as the overall structure, and the data dependencies between each pair of computing nodes. Cui *et al.* [18] proposed a data placement strategy based on GA for a scientific workflow to reduce the amount of data movement in cloud environment. They modified the mutation and crossover operator of GA to get a good performance from a global perspective. Guo *et al.* [11] have previously proposed a scheduling strategy for a deadline-constrained scientific workflow across multiple clouds. This strategy aimed to minimize the execution cost of a scientific workflow within its deadline, which introduced the discrete PSO technique. This work has specific guiding significance for our work. However, it did not consider the impact of MEC.

In summary, previous studies have widely investigated the off-loading strategies for DNN-based applications. However,

TABLE I
SYMBOLS DEFINITION

| Symbol | Definition |
|---|---|
| $\boldsymbol{C} = \{\boldsymbol{C}_{\rm cld}, \boldsymbol{C}_{\rm edg}, \boldsymbol{C}_{\rm dev}\}$ | The hybrid computing environments consisting of the cloud $\boldsymbol{C}_{\rm cld}$, edge $\boldsymbol{C}_{\rm edg}$, and end devices $\boldsymbol{C}_{\rm dev}$. |
| $s_i$ | A server $i$ in the hybrid computing environments. |
| $p_i$ | The computing power of $s_i$. |
| $c_i^{com}$ | The computation cost per second of $s_i$. |
| $t_i$ | The category that $s_i$ belongs to. |
| $b_{i,j}$ | The bandwidth between $s_i$ and $s_j$. |
| $\ell_{i,j}$ | The value of $b_{i,j}$. |
| $c_{i,j}^{tran}$ | The data transmission cost per MB from $s_i$ to $s_j$. |
| $\boldsymbol{G}_i = (\boldsymbol{L}_i, \boldsymbol{E}_i, \boldsymbol{D}_i)$ | A DNN $i$ with layer set $\boldsymbol{L}_i$, data dependency set $\boldsymbol{E}_i$ and data set $\boldsymbol{D}_i$. |
| $D(\boldsymbol{G}_i)$ | The corresponding deadline of $\boldsymbol{G}_i$. |
| $l_i^j$ | A layer $j$ in $\boldsymbol{G}_i$. |
| $e_i^{j,k}$ | A data dependency between $l_i^j$ and $l_i^k$. |
| $a_i^j$ | The calculated amount of $l_i^j$. |
| $i_i^j$ | The input datasets of $l_i^j$. |
| $o_i^j$ | The output datasets of $l_i^j$. |
| $T_{exe}(l_i^j, s_k)$ | The execution time of offloaded layer $l_i^j$ on a ready $s_k$. |
| $d_i^j$ | A dataset $j$ in $\boldsymbol{G}_i$. |
| $\partial_i^j$ | The size of $d_i^j$. |
| $\Omega_i^j$ | The original server that stores $d_i^j$. |
| $f_i^j$ | The final server that uses $d_i^j$. |
| $T_{\rm trans}(d_i^j, s_k, s_r)$ | The transmission time for transferring $d_i^j$ from $s_k$ to $s_r$. |
| $T_i^{\rm compl}$ | The completion time of $\boldsymbol{G}_i$. |
| $C_{\rm total}$ | The total system cost for executing all DNN layers. |

it is still an open issue to optimize the system cost caused by data transmission and layer execution, while off-loading DNN layers within the corresponding deadlines in hybrid computing environments.

## III. PROBLEM DEFINITION AND ANALYSIS

The purpose of this article is to minimize the system cost caused by data transmission and layer execution during DNN layers off-loading while satisfying each DNN-based application's deadline constraint. Table I defines the symbols used in this article.

### A. Problem Definition

The problem definition includes the hybrid computing environments, some DNN-based applications, and an off-loading strategy.

The hybrid computing environments $\boldsymbol{C} = \{\boldsymbol{C}_{\rm cld}, \boldsymbol{C}_{\rm edg}, \boldsymbol{C}_{\rm dev}\}$ consist of the cloud, edge, and end devices. The cloud $\boldsymbol{C}_{\rm cld} = \{s_1, s_2, \ldots, s_n\}$ consists of $n$ servers[1] in a region, and we only consider the off-loading process in one region. The edge $\boldsymbol{C}_{\rm edg} = \{s_1, s_2, \ldots, s_m\}$ consists of $m$ servers in $m$ different regions, i.e., each region has only one server. There are $r$ IoT devices $\boldsymbol{C}_{\rm dev} = \{s_1, s_2, \ldots, s_r\}$, and each device only has a microserver. This study pursues an off-loading scheme, therefore we focus on the computing power of each server and ignore their storage capacity. A server $s_i$ is expressed as

$$s_i = < p_i, c_i^{\rm com}, t_i > \tag{1}$$

where $p_i$ represents the computing power of server $s_i$, which is usually measured by its CPUs [19]. $c_i^{\rm com}$ represents the computation cost of $s_i$ per second,[2] which has a strong positive proportional relationship with $p_i$. $t_i = \{0, 1, 2\}$ represents the category that the server $s_i$ belongs to. When $t_i = 0$, $s_i$ belongs to the cloud, and it has strong computing power. When $t_i = 1$, $s_i$ belongs to the edge, and it has general computing power. When $t_i = 2$, $s_i$ belongs to the end devices, and it has poor computing power. The computing power of each server is assumed to be known and not fluctuant.

Formula (2) represents the bandwidth across different servers

$$\boldsymbol{B} = \begin{bmatrix} b_{1,1} & b_{1,2} & \cdots & b_{1,|\boldsymbol{C}|} \\ b_{2,1} & b_{2,2} & \cdots & b_{2,|\boldsymbol{C}|} \\ \vdots & \vdots & \cdots & \vdots \\ b_{|\boldsymbol{C}|,1} & b_{|\boldsymbol{C}|,2} & \cdots & b_{|\boldsymbol{C}|,|\boldsymbol{C}|} \end{bmatrix} \tag{2}$$

$$b_{i,j} = < \ell_{i,j}, c_{i,j}^{\rm tran}, t_i, t_j > \tag{3}$$

where $b_{i,j}$ is the bandwidth between server $s_i$ and server $s_j$. $\ell_{i,j}$ represents the value of bandwidth $b_{i,j}$, where $\forall i, j = 1, 2, \ldots, |\boldsymbol{C}|$ and $i \neq j$. We do not consider ad hoc networks [20], therefore, there is no direct connection between two end devices (i.e., if $t_i$ and $t_j$ are both equal to 2, $\ell_{i,j}$ is 0). $c_{i,j}^{\rm tran}$ represents the data transmission cost per MB from server $s_i$ to server $s_j$. In addition, end devices connect to the edge via WIFI, so that the servers belong to IoT devices only communicate with the servers belong to the edge within a certain range of WIFI radiation. The bandwidth is assumed to be known and not fluctuant.

There are many DNNs $\boldsymbol{G} = \{\boldsymbol{G}_1, \boldsymbol{G}_2, \ldots, \boldsymbol{G}_q\}$ from different end devices. A DNN is modeled as a directed acyclic graph $\boldsymbol{G}_i = (\boldsymbol{L}_i, \boldsymbol{E}_i, \boldsymbol{D}_i)$ [21], where $\boldsymbol{L}_i = \{l_i^1, l_i^2, \ldots, l_i^s\}$ represents a finite set of nodes containing $s$ layers in $\boldsymbol{G}_i$, $\boldsymbol{E}_i = \{e_i^{1,2}, e_i^{1,3}, \ldots, e_i^{j,k}\}$ represents the data dependencies between each pair of layers, and $\boldsymbol{D}_i = \{d_i^1, d_i^2, \ldots, d_i^n\}$ represents all the datasets including input data, intermediate data, and output data in $\boldsymbol{G}_i$. Each DNN has a corresponding deadline $D(\boldsymbol{G}_i)$, and an off-loading strategy is called feasible solution if the DNN is completed within its deadline.

$e_i^{j,k} = (l_i^j, l_i^k)$ denotes a data dependency between layer $l_i^j$ and layer $l_i^k$, where layer $l_i^k$ is the direct successor of layer $l_i^j$, and layer $l_i^j$ is the direct precursor of layer $l_i^k$. In the process of off-loading DNN layers, a layer cannot start executing until all of its precursors have been completed.

For a layer $l_i^j = < a_i^j, i_i^j, o_i^j >$, $a_i^j$ is the calculated amount of $l_i^j$, $i_i^j$ is the input datasets of $l_i^j$, and $o_i^j$ is the output datasets of $l_i^j$. Serial processing model [22] is adopted in the execution process, which means that a server can only execute one layer at the same time and the whole layer is executed on the same server. Therefore, the execution time $T_{\rm exe}(l_i^j, s_k)$ of offloaded layer $l_i^j$ to a ready server $s_k$ is calculated as (4). The layers and datasets are many-to-many correspondence (i.e., a layer may require many input datasets from different servers, and a dataset may be used by many layers) shown as GoogleNet [1].

$$T_{\rm exe}(l_i^j, s_k) = \frac{a_i^j}{p_k}. \tag{4}$$

For a dataset $d_i^j = < \partial_i^j, \Omega_i^j, f_i^j >$, $\partial_i^j$ is the dataset size, $\Omega_i^j$ is the original server storing $d_i^j$, and $f_i^j$ is the final server using

[1] In order to have a unified expression for the computing resource in different platforms, we use "server" instead of "virtual machine" to express the instances in cloud.

[2] Although the computation cost in cloud is measured in an hour, the execution time of DNN is usually at the millisecond level. Therefore, we use "second" instead of "hour" to measure the computation cost of each server.

$d_i^j$. Therefore, the data transmission time $T_{\text{trans}}(d_i^j, s_k, s_r)$ for transferring dataset $d_i^j$ from server $s_k$ to server $s_r$ is calculated as

$$T_{\text{trans}}\left(d_i^j, s_k, s_r\right) = \frac{\partial_i^j}{\ell_{k,r}}. \tag{5}$$

The purpose of our off-loading strategy is to minimize the system cost caused by data transmission and layer execution, with the deadline constraints of all DNN-based applications. Any layer execution has to satisfy both conditions: (1) The layer has been offloaded to a specific server; (2) the datasets required by this layer have been transferred to the same server. The off-loading strategy for DNN-based applications is defined as $\boldsymbol{O} = (\boldsymbol{C}, \boldsymbol{L}_i, \boldsymbol{D}_i, \boldsymbol{M}, T_i^{\text{compl}}, C_{\text{total}})$, where $\boldsymbol{M} = \bigcup_{i=1,2,\ldots,|\boldsymbol{C}|}\{< l_i^j, s_s > \cup < d_i^j, s_k, s_r >\}$ represents the map set from DNN layers $\boldsymbol{L}_i$ and datasets $\boldsymbol{D}_i$ to hybrid computing environments $\boldsymbol{C}$. $< l_i^j, s_s >$ represents that the layer $l_i^j$ is offloaded to server $s_s$, and $< d_i^j, s_k, s_r >$ represents that dataset $d_i^j$ is transferred from server $s_k$ to server $s_r$. If all $< l_i^j, s_s >$ sets are determined, then all $< d_i^j, s_k, s_r >$ sets are determined. Therefore, the map set can be modified as $\boldsymbol{M} = \bigcup_{i=1,2,\ldots,|\boldsymbol{C}|}\{< l_i^j, s_s >\}$. $T_i^{\text{compl}}$ represents the completion time of DNN $\boldsymbol{G}_i$, and $C_{\text{total}}$ represents the total system cost for executing all DNN layers

$$T_i^{\text{compl}} = \max_{l_i^j \in \boldsymbol{L}_i}\{T_{\text{compl}}(l_i^j)\} \tag{6}$$

$$C_{\text{total}} = \sum_{i=1}^{|\boldsymbol{C}|} c_i^{com} \cdot (T_{\text{off}}(s_i) - T_{\text{on}}(s_i)) + \sum_{j=1,\Omega_l^m=j}^{|\boldsymbol{C}|} \sum_{k=j+1,f_l^m=k}^{|\boldsymbol{C}|} c_{j,k}^{\text{tran}} \cdot \partial_i^j \tag{7}$$

where $T_{\text{compl}}(l_i^j)$ is the completion time of layer $l_i^j$, $T_{\text{off}}(s_i)$ is the turn-OFF time of server $s_i$, and $T_{\text{on}}(s_i)$ is the turn-ON time of server $s_i$. Assuming that a server is turned on immediately with no delay when its first layer arrives on it. A server is turned off immediately with no delay when its last layer on this server is completed.

The problem of the cost-driven off-loading for DNN-based applications over the cloud, edge, and end devices can be formalized as (8). Its core purpose is to pursue a minimum total system cost while satisfying the deadline constraints for each DNN-based application.

$$\begin{aligned} &\text{Minimize} \quad C_{\text{total}} \\ &\text{subject to} \quad \forall i, T_i^{\text{compl}} \le D(\boldsymbol{G}_i). \end{aligned} \tag{8}$$

## B. Problem Analysis

Fig. 1(a) is a sample of off-loading for a DNN $\boldsymbol{G}_i$, which includes four layers $\{l_i^0, l_i^1, l_i^2, l_i^3\}$, and four datasets $\{d_i^1, d_i^2, d_i^3, d_i^4\}$ with size $\{1, 1, 0.5, 0.5\,\text{MB}\}$. The deadline of $\boldsymbol{G}_i$ is 3.7 s. Note that layer $l_i^0$ must be executed on the end device (i.e., $s_0$). There are six servers in hybrid computing environments. Table II shows the execution time of each layer on the six servers. Table III shows the computation cost per hour of six servers.

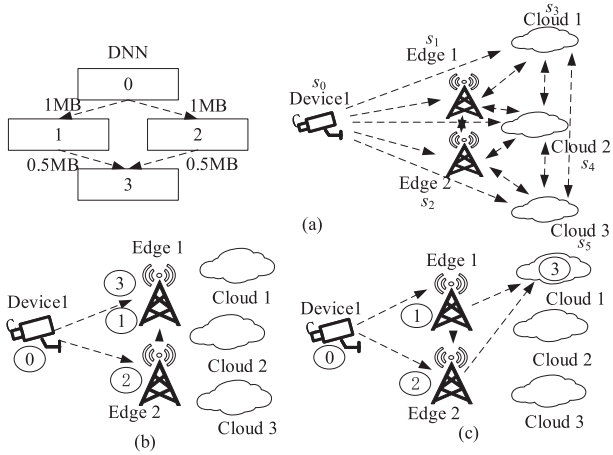


Fig. 1. Sample of cost-driven off-loading for a DNN.

TABLE II
EXECUTION TIME OF EACH LAYER ON DIFFERENT SERVERS

| | $s_0$ | $s_1$ | $s_2$ | $s_3$ | $s_4$ | $s_5$ |
|---|---|---|---|---|---|---|
| $l_i^0$ | 1.1s | - | - | - | - | - |
| $l_i^1$ | 1.92s | 0.98s | 0.62s | 0.31s | 0.19s | 0.09s |
| $l_i^2$ | 2.35s | 1.20s | 0.75s | 0.67s | 0.41s | 0.32s |
| $l_i^3$ | 2.12s | 1s | 0.8s | 0.56s | 0.45s | 0.21s |

TABLE III
COMPUTATION COST PER HOUR OF SIX SERVERS

| Servers | Cost/hour($) |
|---|---|
| $s_0$ | 0 |
| $s_1$ | 10 |
| $s_2$ | 15 |
| $s_3$ | 1 |
| $s_4$ | 2 |
| $s_5$ | 3 |

TABLE IV
BANDWIDTH BETWEEN TWO SERVERS IN DIFFERENT CATEGORIES AND THE CORRESPONDING COST PER MB

| $t_i$ | | $t_j$ | $\ell_{i,j}$ | $c_{i,j}^{tran}$ |
|---|---|---|---|---|
| 0 | $\leftrightarrow$ | 0 | 5 | 0.4 |
| 0 | $\leftrightarrow$ | 1 | 2 | 0.8 |
| 0 | $\leftrightarrow$ | 2 | 2 | 0.8 |
| 1 | $\leftrightarrow$ | 1 | 10 | 0.16 |
| 1 | $\leftrightarrow$ | 2 | 10 | 0.16 |

Table IV shows the bandwidth between two servers in different categories and the corresponding cost per MB.

Fig. 1(b) is the off-loading result according to the greedy strategy [23]. The completion time of $\boldsymbol{G}_i$ is 3.65 s, and the system cost is 0.0044013 with greedy strategy. Fig. 1(c) is the optimal off-loading result. The completion time of $\boldsymbol{G}_i$ is 3.41 s, and the system cost is 0.0036517. The system cost with the optimal strategy is 18.18% less than the former. The greedy off-loading strategy offloads each layer to the server with the lowest cost step by step within the corresponding deadline. However, Fig. 1(b) shows that each off-loading step with a greedy choice fails to achieve the best result from a local perspective. In this study, we need a suitable approach to effectively offload all DNN layers from a global perspective.

## IV. Off-loading Strategy Based on PSO-GA

For an off-loading strategy $\boldsymbol{O} = (\boldsymbol{C}, \boldsymbol{L}_i, \boldsymbol{D}_i, \boldsymbol{M}, T_i^{\text{compl}}, C_{\text{total}})$, its core purpose is to find a map from all $\boldsymbol{L}_i$ to $\boldsymbol{C}$ that has minimum system total cost $C_{\text{total}}$ while each DNN completion time $T_i^{\text{compl}}$ is not more than their corresponding deadline $D(\boldsymbol{G}_i)$. Finding the best map from all $\boldsymbol{L}_i$ to $\boldsymbol{C}$ is an NP-hard problem [24]. Therefore, an off-loading strategy based on PSO-GA algorithm to optimize the system total cost is proposed from a global perspective in hybrid computing environments. To improve the efficiency of our off-loading strategy, a preprocessing for a DNN is designed to compress the amount of layers and data transmission. This section mainly describes the preprocessing for a DNN in [25], and PSO-GA algorithm as follows.

### A. PSO-GA

The details of traditional PSO and the effect of relevant parameters are described in [25].

PSO has been widely used to solve continuous optimization problems. Off-loading DNN layers in hybrid computing environments is a discrete problem, and it needs a new coding approach. In addition, a suitable strategy for particle update should be introduced to avoid the premature convergence of traditional PSO. In this article, PSO-GA is proposed to solve the above shortages of traditional PSO. The off-loading strategy based on PSO-GA for DNNs is described as follows.

*1) Problem Encoding:* A good encoding strategy can enhance the search efficiency and improve the performance of PSO-based algorithm, which usually satisfies the following three principles [26].

*Definition 1. (Completeness):* Each candidate solution can be encoded as a particle in the problem space.

*Definition 2. (Nonredundancy):* Each candidate solution has only one corresponding encoded particle in the problem space.

*Definition 3. (Viability):* Each encoded particle represents a candidate solution in the problem space.

Designing an encoding strategy that simultaneously satisfies the three principles is difficult. Inspired by [27], we adopt a server-order nesting strategy to encode the layers off-loading problem. A particle represents a candidate solution of cost-driven off-loading for all DNNs in hybrid computing environments, and the $i$th particle in the $t$th iteration is described as (9)

$$X_i^t = (x_{i1}^t, x_{i2}^t, \ldots, x_{ip}^t) \tag{9}$$

$$x_{ik}^t = (s_j, \varphi_j)_{ik}^t \tag{10}$$

where $p$ is the total number of layers from all DNN-based applications. $x_{ik}^t (k = 1, 2, \ldots, p)$ indicates the final server $s_j$ for executing the $k$th layer with $\varphi_j$ order in the $t$th iteration in (10). It means that the $k$th layer is offloaded to server $s_j$ with a specified order $\varphi_j$, whose value ranges from 0 to $p-1$. The order of each layer in the same particle is different from each other, and the layer with smaller order is processed earlier when there are more than two concurrent layers on the same server. The concurrent layers mean that there is no direct or indirect data dependency between them. Therefore, the particle dimension is twice the total number of layers. Fig. 2 shows an encoded particle corresponding to the off-loading for a DNN in Fig. 1(c).

| layers | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| offloading server, order | 0, 3 | 1, 0 | 2, 1 | 3, 2 |

Fig. 2. Encoded particle corresponding to the off-loading for a DNN.

*Property 1:* The encoding strategy meets completeness and nonredundancy principles, but it may not meet the viability principle.

After DNN layers off-loading, each layer is offloaded to the corresponding server with a specified order. A layer can be offloaded to any server with any order under the data dependence constraints, and the corresponding dimension in a particle can be the corresponding value of server and order. Therefore, each off-loading strategy has the corresponding particle, which meets the completeness principle. An off-loading strategy for DNNs corresponds to a $2p$-dimensional particle. The value of the $k$th dimension in a particle is the server number processing the $k$th layer and the specified process order. Therefore, an off-loading strategy only corresponds to a specified particle, which meets the nonredundancy principle. However, some candidate solutions corresponding to the particles may not meet the deadline constraints. For example, if the final offloaded servers of layers in Fig. 1 is (0, 0, 2, 3), then layer $l_i^0$ and layer $l_i^1$ are offloaded to end device $s_0$. The completion time of this DNN is more than 4 s, which exceeds its deadline (i.e., 3.7 s). Therefore, the encoding strategy may not meet the viability principle.

*2) Fitness Function:* The fitness function is used to evaluate the performance of all particles. In general, a particle with smaller fitness represents a better candidate solution. This work pursues minimum system cost for off-loading DNN layers while satisfying their deadline constraints. Therefore, a particle with lesser system cost can be considered as a better solution. However, the encoding strategy designed in this article fails to meet the viability principle.

All particles can be divided into two categories: feasible particle and infeasible particle, whose definitions are described as follows.

*Definition 4. (Feasible particle):* A particle that corresponds to a DNN layer off-loading strategy meets all deadline constraints.

*Definition 5. (Infeasible particle):* A particle that corresponds to a DNN layer off-loading strategy fails to meet deadline constraints (i.e., at least one DNN's completion time exceeds its corresponding deadline).

The definition of fitness function that compares two candidate solutions has to be modified according to three different situations.

*Case 1:* Both particles are feasible. The particle with lesser system cost is selected, and the fitness function is defined as follows:

$$F(X_i) = C_{\text{total}(X_i)}. \tag{11}$$

*Case 2:* One particle is feasible, and the other one is infeasible. The feasible particle is selected, and the fitness function is defined as follows:

$$F(X_i) = \begin{cases} 0, & \text{if } \forall i, T_i^{\text{compl}}{}_{(X_i)} \le D(\mathbf{G}_i) \\ 1, & \text{else} \end{cases}. \tag{12}$$

*Case 3:* Both particles are infeasible. The particle with lesser total completion time is selected, and this particle is more likely to become a feasible particle after update operations. The fitness function is defined as follows:

$$F(X_i) = \max\{T_i^{\text{compl}}{}_{(X_i)}\}. \tag{13}$$

*3) Update Strategy:* PSO has three main parts: *Inertia*, *individual cognition*, and *social cognition*. The iterative update of each particle is affected by both its personal best position and the global best position in current generation [28]. Prematurely falling into a local optimum is a major defect of traditional PSO. To enhance the search ability of our algorithm and avoid premature convergence, the crossover operator and mutation operator of GA are introduced for particle update. The iterative update of the $i$th particle at the $t$th iteration is shown as (14), where $C_g()$ and $C_p()$ represent crossover operators, and $M_u()$ is mutation operator. In addition, the value of order $\varphi_j$ for each layer remains the same. It means that only the value of servers for each layer in a particle is updated.

$$\begin{aligned} X_i^t &= c_2 \oplus C_g \\ &\times (c_1 \oplus C_p(w \oplus M_u(X_i^{t-1}), p\text{Best}_i^{t-1}), g\text{Best}^{t-1}), \end{aligned} \tag{14}$$

For *individual cognition* component and *social cognition* component, the crossover operator of GA is introduced to refresh the corresponding component, which is described as (15) and (16), respectively

$$\begin{aligned} B_i^t &= c_1 \oplus C_p(A_i^t, p\text{Best}^{t-1}) \\ &= \begin{cases} C_p(A_i^t, p\text{Best}^{t-1}) & r_1 < c_1 \\ A_i^t & \text{else} \end{cases} \end{aligned} \tag{15}$$

$$\begin{aligned} C_i^t &= c_2 \oplus C_g(B_i^t, g\text{Best}^{t-1}) \\ &= \begin{cases} C_g(B_i^t, g\text{Best}^{t-1}) & r_2 < c_2 \\ B_i^t & \text{else} \end{cases} \end{aligned} \tag{16}$$

where $r_1$ and $r_2$ are both random factors, whose value is between 0 and 1. $C_p()$ (or $C_g()$) is crossover operator. It randomly chooses two locations in a particle to be updated, and then replaces server value of the segment between the two locations with the same interval in *pBest* (or *gBest*) particle. The crossover operator for *individual* (or *social*) *cognition* component is shown as Fig. 3(a). It randomly chooses $ind_1$ and $ind_2$ locations in an *old* particle, and then replaces the segment between $ind_1$ and $ind_2$ with the *pBest* (or *gBest*) particle in the same interval.

*Property 2:* A particle can change from infeasible to feasible after crossover operator, and vice versa. More details are given in [25].

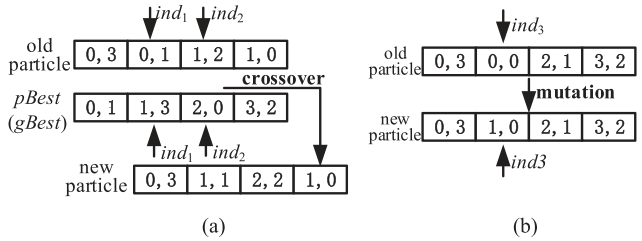


Fig. 3. Update operation. (a) Crossover operator for *individual* (*social*) *cognition* component. (b) Mutation operator for *inertia* component.

For *inertia* component, the mutation operator of GA is introduced to refresh the corresponding component, which is described as

$$A_i^t = w \oplus M_u(X_i^{t-1}) = \begin{cases} M_u(X_i^{t-1}) & r_3 < w \\ X_i^{t-1} & \text{else} \end{cases} \tag{17}$$

where $r_3$ is a random factor, whose value is between 0 and 1. $M_u()$ is mutation operator. It randomly chooses a location in a particle, and then alters the corresponding server value in the range of $|C|$. Fig. 3(b) illustrates the mutation operator for *inertia* component. It randomly chooses $ind_3$ location, and then alters the value of $ind_3$ from 0 to 1.

*Property 3:* A particle can change from infeasible to feasible after mutation operator, and vice versa. More details are given in [25].

*4) Map From a Particle to DNN Layers Off-Loading:* The pseudocode of mapping a particle to cost-driven off-loading for DNNs in hybrid computing environments is given in [25].

*5) Parameter Settings:* The inertia weight $w$ affects the convergence and search ability of PSO [29]. Formula (18) represents an adjustment mechanism for the inertia weight [30]

$$w = w_{\max} - iters_{\text{cur}} \times \frac{w_{\max} - w_{\min}}{\text{iters}_{\max}} \tag{18}$$

where $w_{\min}$ and $w_{\max}$ are the given minimum and maximum of $w$ in the initialization phase. $\text{iters}_{\text{cur}}$ and $\text{iters}_{\max}$ are the current number and the maximum number of iterations, respectively. Therefore, the algorithm focuses on the global search at the beginning of execution. With the increase of iterations, the value of $w$ reduces linearly and the algorithm gradually focuses on the local search.

The adjustment mechanism in (18) fails to meet the nonlinear characteristics of DNN layers off-loading. Therefore, we design an adjustment mechanism that can adaptively adjust the search ability according to the quality of current particle in (19)

$$w = w_{\max} - (w_{\max} - w_{\min}) \times \exp\left(\frac{d(X_i^{t-1})}{d(X_i^{t-1}) - 1.01}\right) \tag{19}$$

$$d(X^{t-1}) = \frac{\text{div}(g\text{Best}^{t-1}, X^{t-1})}{|\mathbf{C}|} \tag{20}$$

where $\text{div}(g\text{Best}^{t-1}, X^{t-1})$ indicates the number of coordinate difference between the global best particle $g\text{Best}^{t-1}$ and the current particle $X^{t-1}$. This mechanism can adaptively adjust its search ability according to the difference between the global

TABLE V
CONFIGURATIONS AND THE COST FOR ALL SERVERS

| Servers | Configurations | Cost/hour($) | $t_i$ |
|---|---|---|---|
| $\{s_1, s_2, ..., s_{10}\}$ | 2CPUs 4GB | 0 | 2 |
| $\{s_{10}, s_{11}, ..., s_{15}\}$ | 16CPUs 32GB | 2.43 | 1 |
| $s_{16}$ | 4CPUs 8GB | 0.225 | 0 |
| $s_{17}$ | 8CPUs 16GB | 0.45 | 0 |
| $s_{18}$ | 16CPUs 32GB | 0.9 | 0 |
| $s_{19}$ | 32CPUs 64GB | 1.8 | 0 |
| $s_{20}$ | 64CPUs 128GB | 3.6 | 0 |

best particles and current particle. When $\text{div}(g\text{Best}^{t-1}, X^{t-1})$ is small, it means that there is a small gap between $g\text{Best}^{t-1}$ and $X^{t-1}$, and the algorithm prefers to enhance the local search and accelerate the convergence to find an optimal solution.

The other two acceleration coefficients (i.e., $c_1$ and $c_2$) are given in [31]. Note that $c_1^{\text{start}}$ and $c_1^{\text{end}}$ are the start value and end value of $c_1$. $c_2^{\text{start}}$ and $c_2^{\text{end}}$ are the start value and end value of $c_2$.

*6) Algorithm Flowchart:* More details about algorithm flowchart are given in [25].

## V. EXPERIMENT RESULTS AND ANALYSIS

All the simulation experiments were conducted on a Win8 64-bit operating system with a G3250 3.20 GHz Intel Pentium processor and 8 GB RAM. The corresponding parameters of PSO-GA were set based on [30]: $S_{\text{pop}} = 100$, $\text{iters}_{\max} = 1000$, $w_{\max} = 0.9$, $w_{\min} = 0.4$, $c_1^{\text{start}} = 0.9$, $c_1^{\text{end}} = 0.2$, $c_2^{\text{start}} = 0.4$, and $c_2^{\text{end}} = 0.9$.

### A. Experimental Setup

We conducted our experiments using four types of DNNs: AlexNet, VGG19, GoogleNet, and ResNet101. The structure, datasets, and computing amount in each type of DNN are different. The basic layer execution time, data transmission amount between two layers, and the overall structure for our tested DNNs is recorded in a file which is available online.[3]

The hybrid computing environments have 20 servers $\{s_1, s_2, \ldots, s_{20}\}$, which are divided into three categories. The first 10 servers belong to the end devices, the last 5 servers belong to the cloud, and the other five servers belong to the edge. The processing capacity of a server in the same category is roughly proportional to its cost. We assume that the end servers have the lowest configurations and execute DNN layers without charging. Table V shows the configurations and the cost for all servers. In this article, each end server is connected to two nearby edge servers. Table IV shows the bandwidth between two servers in different categories and the corresponding data transmission cost.

Finally, each DNN needs a specific deadline to verify whether an off-loading strategy is feasible or not. We set five different deadlines for each DNN as

$$D_j(\boldsymbol{G}_i) = r_j \cdot H(\boldsymbol{G}_i),\ r_j = \{1.2, 1.5, 3, 5, 8\} \tag{21}$$

where $H(\boldsymbol{G}_i)$ is the execution time of a DNN $\boldsymbol{G}_i$ based on HEFT algorithm [32].

[3][Online]. Available: https://github.com/LinBin403/dataset-for-our-research

### B. Competitive Algorithms

To verify the effectiveness of PSO-GA, we modified the GA [18] and the Greedy [23] to adapt the cost-driven off-loading strategy for DNNs in hybrid computing environments.

GA adopts a binary encoding strategy, and its fitness function is according to [25]. The map from an encoded chromosome to a cost-driven off-loading solution should not only consider the computation cost for each layer, but also the transmission cost for each dataset.

Greedy offloads each layer to the cheapest server within the corresponding deadline. If a layer offloaded to the cheapest server cannot meet the deadline constraint, then the layer has to be offloaded to the second cheapest server. It follows these operations and iterates over.

Finally, prePSO is selected as another comparison algorithm, which is the PSO-GA with preprocessing discussed in Section IV.

### C. Experimental Results and Analysis

GA, PSO-GA, and prePSO belong to the metaheuristic algorithms. In the experiments, they are terminated if they maintain the same value in 50 continuous iterations. The off-loading results may be different with the same configurations in each experiment. Therefore, the system cost is measured as the average of 50 repeated experiments. If there are infeasible particles (solutions) and feasible particles (solutions), the system cost corresponding to the infeasible solutions is ignored. It means that we only consider the average system cost of feasible solutions when there is at least one feasible solution in the 50 repeated experiments. If there is no feasible particle (solution) after 50 repeated experiments, its system cost is represented as a negative value. In the experiments, the off-loading results corresponding to the negative system cost for DNNs are all the infeasible solutions.

Fig. 4 shows the system cost of different strategies for one DNN per end device. In these experiments, there is only one DNN on each end device originally. It means that there are ten DNNs on ten end devices. In general, the system cost becomes less and less as the deadline is gradually loose for all off-loading strategies. With the looser deadline constraints, more layers can be offloaded to the cheaper servers under the same situation. The reasons of the performance difference on the four types of DNNs are that they have different overall structure, basic layer execution time, and data transmission amount between two layers. For VGG19, all layers in it are almost serially executed, and the data transmission amount is more than other three types of DNNs. Therefore, the same off-loading strategy has to spend more time or cost on off-loading each layer of VGG19 than other three types of DNNs. For GoogleNet, many layers can be processed in parallel, and it has a larger number of layers than AlexNet and VGG19. Although the data transmission amount of GoogleNet is similar to that of VGG19, the same off-loading strategy can spend less time or cost on off-loading each layer of GoogleNet than VGG19. This is mainly due to the parallel processing of layers in GoogleNet.

PSO-GA has the best performance due to that it evolves iteratively from a global perspective. Greedy is an extreme

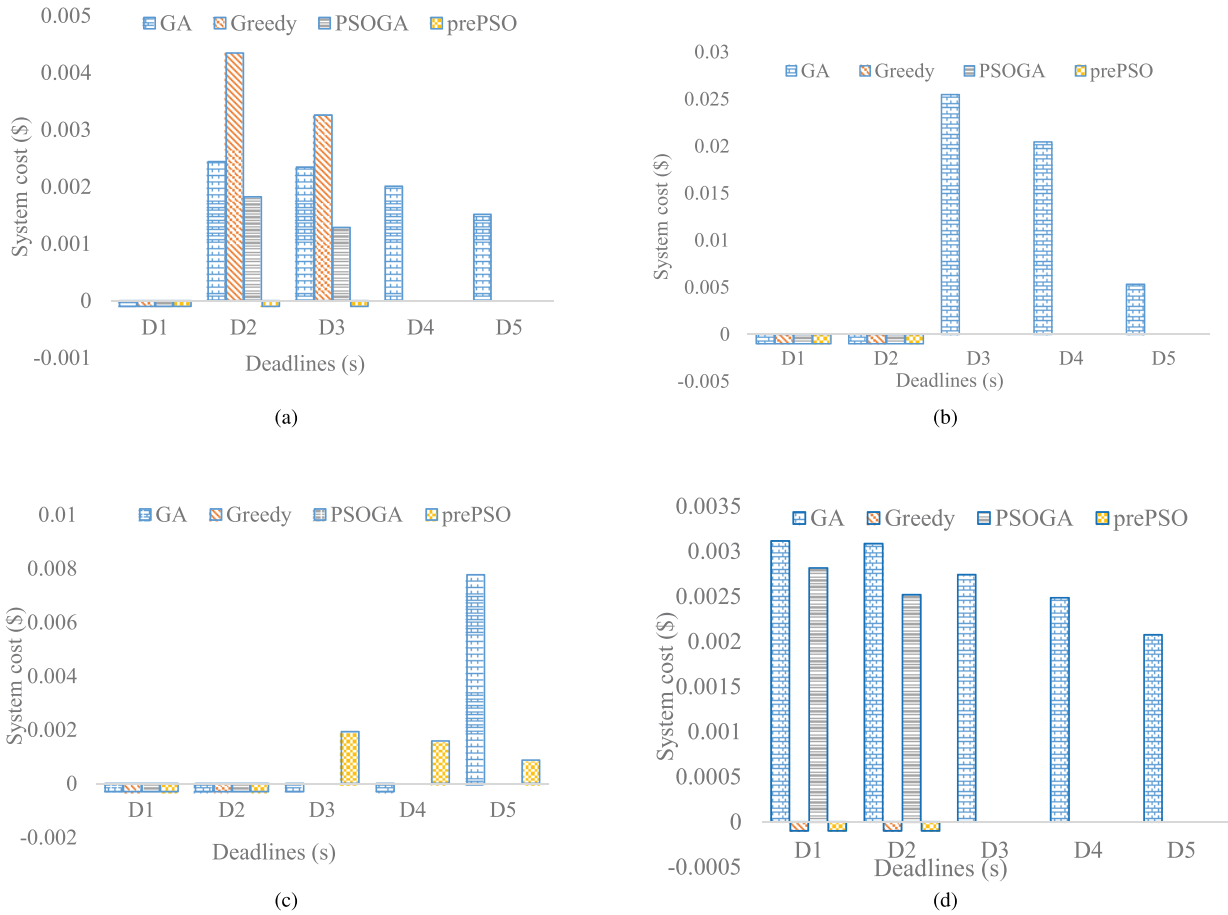


Fig. 4. System cost of different strategies for one DNN per end device. (a) AlexNet. (b) VGG19. (c) GoogleNet. (d) ResNet101.

strategy, which cannot find a feasible solution when the deadline is tight. GA's search scope is relatively limited at each iteration, and it does not adaptively adjust according to the performance of the current chromosome. For VGG19 and ResNet101, prePSO compresses all the layers into one layer in a DNN. Therefore, it offloads all layers of a DNN on its corresponding end devices, and its performance is similar to that of Greedy for VGG19 and ResNet101. For GoogleNet, there is a big gap between PSO-GA and prePSO after $D_3(\boldsymbol{G})$. The reason is that the preprocessing influences the final off-loading result discussed in [25]. The compressed layers have larger computation amount, which have to be offloaded on the servers with more computing power.

Fig. 4(a) shows the system cost of different strategies for one AlexNet per end device. The strategies in Fig. 4(a) have less system cost compared with that in Fig. 4(b) and (d). This is mainly due to that the number of layers, the average amount of each dataset, and the average execution time of each layer in AlexNet are much less than that in VGG19 and ResNet101. This leads to that the system cost among the three figures is not an order of magnitude. In Fig. 4(c), there is no strategy offering a feasible solution before $D_3(\boldsymbol{G})$. Although the deadline is approximately 1.5 times the completion time of HEFT algorithm for each DNN, each layer has to be executed according to the serial processing model on the same server.

Fig. 5 shows the system cost of different strategies for three DNNs per end device. It means that there are three DNNs on each end device originally. From Fig. 4, we find that almost all off-loading strategies have no feasible solution in $D_1(\boldsymbol{G})$ and $D_2(\boldsymbol{G})$. The number of DNN layers in Fig. 5 is three times that of the corresponding DNNs in Fig. 4. In order to decrease the generation of infeasible solutions, the deadlines in the experiments in Fig. 5 are twice that in Fig. 4.

In general, the system cost in Fig. 5 is almost four times that in Fig. 4. It is obvious that the system cost becomes less as the deadline is gradually loose for all off-loading strategies. Greedy has the worst performance due to that it is an extreme strategy from a local perspective. It prioritizes each layer on the cheapest server step by step within the corresponding deadlines. As the total number of DNN layers increase, the heavier layers usually fails to be completed within their deadlines by Greedy strategy. Through an overview of Fig. 5, we find that Greedy achieves feasible solutions within looser deadlines (i.e.,$D_5(\boldsymbol{G})$) for AlexNet and ResNet101. PSO-GA has the best performance. It averts the premature convergence of PSO and improves the diversity of population. This leads to that layers with larger amount of computation are offloaded to the cloud, and layers with larger amount of data transmission are offloaded to the edge and end devices within their deadlines. The overall trend of different strategies for system cost in Fig. 5 is similar to that in

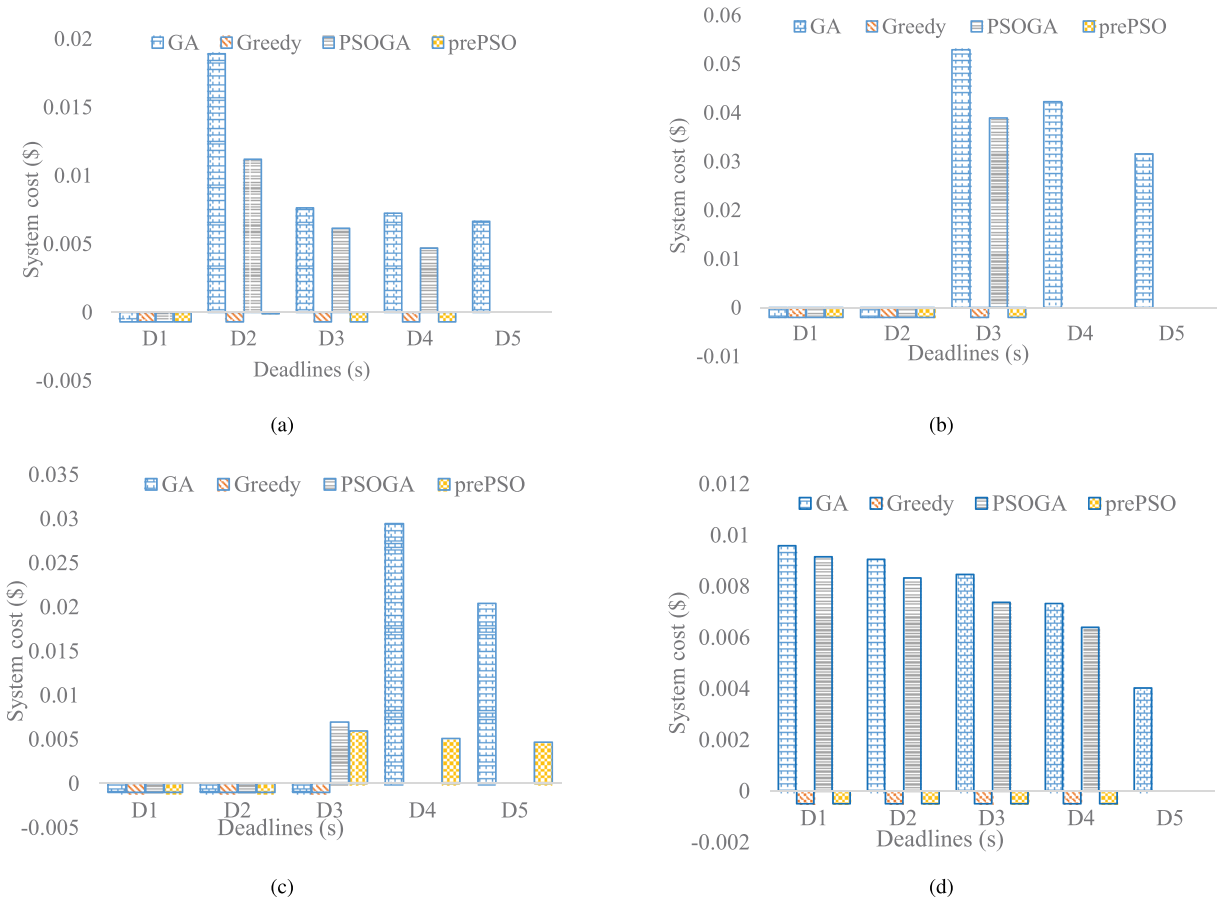


Fig. 5. System cost of different strategies for three DNN per end device. (a) AlexNet. (b) VGG19. (c) GoogleNet. (d) ResNet101.

Fig. 4. From the experiments, we find that data transmission cost accounts for the majority of the total system cost for GoogleNet and ResNet101. The reason for this result is that they have more layers, and the amount of layer execution is relatively smaller compared with AlexNet, whose number of layers is 11.

Fig. 5(a) shows the system cost of different strategies for three AlexNets per end device. From Figs. 4(a) and 5(a), you can find that Greedy strategy becomes extremely unsuitable as the total number of AlexNet layers increases. Although PSO-GA has the best performance, it cannot offload all layers to the end devices to achieve 0 system cost until in $D_5(\boldsymbol{G})$ of Fig. 5(a). In Fig. 5(b), all off-loading strategies except GA achieve 0 system cost after $D_3(\boldsymbol{G})$. It means that all layers in a VGG19 are offloaded to its original end device. In Fig. 5(d), PSO-GA and PSO both achieve the feasible solutions from $D_1(\boldsymbol{G})$ to $D_5(\boldsymbol{G})$. Greedy and prePSO cannot find any feasible solution until $D_5(\boldsymbol{G})$. prePSO compresses all the layers into one layer in a ResNet101, and offloads all layers in a ResNet101 to its original end device when the corresponding deadline is loose enough.

More details about experimental results are given in [25].

### D. Industrial Applications

Road traffic applications, such as vehicle identification, have to rely on computer vision, whose core is DNNs. Traffic cameras have limited process capacity, and the vehicle identification application usually has deadline constraint. With the hybrid computing environments, consisting of the cloud, edge, and end devices, the larger layers (more amount of data) with high business intelligence are offloaded to the cloud, while the smaller layers (less amount of data) are offloaded to the edge and cameras. These three platforms collaborate with each other and execute the layers of DNNs with low system cost and latency.

Traffic cameras periodically shoot traffic conditions, and the configurations of the hybrid computing environments usually keep the same for a long time. The off-loading strategy based on PSO-GA will make the same decision when the configurations keep the same after the first execution. Otherwise, it has to be re-executed to adapt to the new configurations of the hybrid computing environments.

## VI. Conclusion

A cost-driven off-loading strategy based on PSO-GA for DNN-based applications over the cloud, edge, and end devices was proposed in this article. The experimental results showed that the off-loading strategy effectively reduced the system cost within each DNN's corresponding deadline relative to the benchmarks. With the looser deadline constraints, more layers can be

offloaded to the cheaper servers under the same situation. Therefore, looser deadline constraints will lead to the system cost decrease. Moreover, all layers can be offloaded in their original end devices with no data transmission when their corresponding deadlines are loose enough.

In the future, we will consider the "load balancing" issue for all the participated servers, which may have a major impact on the system cost. In addition, each layer in the real environment has different price/performance ratios for different servers, and the bandwidth may fluctuate due to the changes in data traffic. Therefore, we will comprehensively optimize the system cost while considering the bandwidth fluctuation and the different price/performance ratios for different servers.

## References


[1] S. Han, H. Shen, M. Philipose, S. Agarwal, A. Wolman, and A. Krishnamurthy, "MCDNN: An approximation-based execution framework for deep stream processing under resource constraints," in *Proc. 14th Annu. Int. Conf. Mobile Syst., Appl. Services*, Singapore, 2016, pp. 123–136. [Online]. Available: http://dx.doi.org/10.1145/2906388.2906396

[2] H. Chen, C. Yang, and Y. Du, "Machine learning-assisted analysis of polarimetric scattering from cylindrical components of vegetation," *IEEE Trans. Geosci. Remote Sens.*, vol. 57, no. 1, pp. 155–165, Jan. 2019.

[3] S. Teerapittayanon, B. McDanel, and H. Kung, "Distributed deep neural networks over the cloud, the edge and end devices," in *Proc. 37th Int. Conf. Distrib. Comput. Syst.*, Atlanta, GA, USA, 2017, pp. 328–339. [Online]. Available: http://dx.doi.org/10.1109/ICDCS.2017.226

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

[5] H. Wu, "Multi-objective decision-making for mobile cloud offloading: A survey," *IEEE Access*, vol. 6, pp. 3962–3976, 2018.

[6] H. J. Jeong, "Lightweight offloading system for mobile edge computing," in *Proc. IEEE Int. Conf. Pervasive Comput. Commun. Workshops*, Kyoto, Japan, 2019, pp. 451–452.

[7] C. Hu, W. Bao, D. Wang, and F. Liu, "Dynamic adaptive DNN surgery for inference acceleration on the edge," in *Proc. IEEE INFOCOM*, Paris, France, 2019, pp. 1423–1431.

[8] 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. Int. Conf. Distrib. Comput. Syst.*, Vienna, Austria, 2018, pp. 1492–1499. [Online]. Available: http://dx.doi.org/10.1109/ICDCS.2018.00154

[9] C. Lo, Y. Y. Su, C. Y. Lee, and S. C. Chang, "A dynamic deep neural network design for efficient workload allocation in edge computing," in *Proc. 35th IEEE Int. Conf. Comput. Design*, Boston, MA, USA, 2017, pp. 273–280.

[10] Y. Mao, S. Yi, Q. Li, J. Feng, F. Xu, and S. Zhong, "Learning from differentially private neural activations with edge computing," in *Proc. 3rd ACM/IEEE Symp. Edge Comput.*, Bellevue, WA, USA, 2018, pp. 90–102.

[11] W. Guo, B. Lin, G. Chen, Y. Chen, and F. Liang, "Cost-driven scheduling for deadline-based workflow across multiple clouds," *IEEE Trans. Netw. Service Manage.*, vol. 15, no. 4, pp. 1571–1585, Dec. 2018.

[12] B. Lin *et al.*, "A time-driven data placement strategy for a scientific workflow combining edge computing and cloud computing," *IEEE Trans. Ind. Informat.*, vol. 15, no. 7, pp. 4254–4265, Jul. 2019.

[13] J. Liu *et al.*, "Online multi-workflow scheduling under uncertain task execution time in IaaS clouds," *IEEE Trans. Cloud Comput.*, vol. 1, pp. 1–10, 2019.

[14] Z. Fang, T. Yu, O. J. Mengshoel, and R. K. Gupta, "QoS-aware scheduling of heterogeneous servers for inference in deep neural networks," in *Proc. Int. Conf. Inf. Knowl. Manage.*, Singapore, 2017, pp. 2067–2070. [Online]. Available: http://dx.doi.org/10.1145/3132847.3133045

[15] B. Qi, M. Wu, and L. Zhang, "A DNN-based object detection system on mobile cloud computing," in *Proc. 17th Int. Symp. Commun. Inf. Technologies*, Cairns, Australia, 2017, pp. 1–6. [Online]. Available: http://dx.doi.org/10.1109/ISCIT.2017.8261188

[16] Y. Kang *et al.*, "Neurosurgeon: Collaborative intelligence between the cloud and mobile edge," in *Proc. Int. Conf. Architectural Support Program. Languages Operating Syst.*, Xi'an, China, 2017, pp. 615–629. [Online]. Available: http://dx.doi.org/10.1145/3037697.3037698

[17] H. Wu, W. J. Knottenbelt, and K. Wolter, "An efficient application partitioning algorithm in mobile environments," *IEEE Trans. Parallel Distrib. Syst.*, vol. 30, no. 7, pp. 1464–1480, Jul. 2019.

[18] L. Cui, J. Zhang, L. Yue, Y. Shi, H. Li, and D. Yuan, "A genetic algorithm based data replica placement strategy for scientific applications in clouds," *IEEE Trans. Services Comput.*, vol. 11, no. 4, pp. 727–739, Jul./Aug. 2018.

[19] Y. He *et al.*, "Deep-reinforcement-learning-based optimization for cache-enabled opportunistic interference alignment wireless networks," *IEEE Trans. Veh. Technol.*, vol. 66, no. 11, pp. 10433–10445, Nov. 2017.

[20] N. Zhao, X. Liu, F. R. Yu, M. Li, and V. C. M. Leung, "Communications, caching, and computing oriented small cell networks with interference alignment," *IEEE Commun. Mag.*, vol. 54, no. 9, pp. 29–35, Sep. 2016.

[21] H. Wu and K. Wolter, "Stochastic analysis of delayed mobile offloading in heterogeneous networks," *IEEE Trans. Mobile Comput.*, vol. 17, no. 2, pp. 461–474, Feb. 2018.

[22] M. Zorzi, "Serial processing in reading aloud: No challenge for a parallel model," *J. Exp. Psychol. Human Perception Perform.*, vol. 26, pp. 847–856, 2000. [Online]. Available: http://dx.doi.org/10.1037//0096-1523.26.2.847

[23] C. Jiang, Z. Chen, R. Su, and Y. C. Soh, "Group greedy method for sensor placement," *IEEE Trans. Signal Process.*, vol. 67, no. 9, pp. 2249–2262, May 2019.

[24] D. S. Hochba, "Approximation algorithms for NP-hard problems," *SIGACT News*, vol. 28, pp. 40–52, 1997. [Online]. Available: http://doi.acm.org/10.1145/261342.571216

[25] B. Lin, X. Chen, and J. Li, "Cost-driven offloading for DNN-based applications over cloud, edge and end devices," Tech. Rep., 2019. [Online]. Available: http://github.com/LinBin403/Paper_Online

[26] J. S. Su, W. Z. Guo, C. L. Yu, and G. L. Chen, "Fault-tolerance clustering algorithm with load-balance aware in wireless sensor network," *Jisuanji Xuebao/Chinese J. Comput.*, vol. 37, pp. 445–456, 2014. [Online]. Available: http://dx.doi.org/10.3724/SP.J.1016.2014.00445

[27] M. A. Rodriguez and R. Buyya, "Deadline based resource provisioning and scheduling algorithm for scientific workflows on clouds," *IEEE Trans. Cloud Comput.*, vol. 2, no. 2, pp. 222–235, Apr./Jun. 2014.

[28] H. Li, D. Yang, W. Su, J. Lu, and X. Yu, "An overall distribution particle swarm optimization MPPT algorithm for photovoltaic system under partial shading," *IEEE Trans. Ind. Electron.*, vol. 66, no. 1, pp. 265–275, Jan. 2019.

[29] D. O'Neill, A. Lensen, B. Xue, and M. Zhang, "Particle swarm optimisation for feature selection and weighting in high-dimensional clustering," in *Proc. IEEE Congr. Evol. Comput.*, Rio de Janeiro, Brazil, 2018, pp. 1–8.

[30] Y. Shi and R. Eberhart, "A modified particle swarm optimizer," in *Proc. IEEE Int. Conf. Evol. Comput.*, Anchorage, USA, 1998, pp. 69–73.

[31] M. Masdari, F. Salehi, M. Jalali, and M. Bidaki, "A survey of PSO-based scheduling algorithms in cloud computing," *J. Netw. Syst. Manage.*, vol. 25, pp. 122–158, 2017. [Online]. Available: http://dx.doi.org/10.1007/s10922-016-9385-9

[32] H. Topcuoglu, S. Hariri, and M.-Y. Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 13, no. 3, pp. 260–274, Mar. 2002.


**Bing Lin** received the B.S. and M.S degrees in computer science and the Ph.D. degree in communication and information system from Fuzhou University, Fuzhou, China, in 2010, 2013, and 2016, respectively.

He is currently an Assistant Professor with the College of Physics and Energy, Fujian Normal University, Fujian, China. He is also the Academic Secretary of CCF YOCSEF in Fuzhou. He has authored or coauthored over 20 journals and conference articles, such as IEEE TRANSACTIONS ON INDUSTRIAL INFORMATICS, IEEE TRANSACTIONS ON NETWORK AND SERVICE MANAGEMENT, and *Concurrency and Computation: Practice and Experience*. His research interests include parallel and distributed computing, computational intelligence, and data center resource management.

**Yinhao Huang** received the B.S. degree in software engineering in 2018 from Fuzhou University, Fujian, China, where he is currently working toward the M.S. degree in technology of computer application with the College of Mathematics and Computer Science.

Since September 2018, he has also been a student of the Fujian Key Laboratory of Network Computing and Intelligent Information Processing at Fuzhou University. His current research interests include deep neural network, edge computing, and cloud computing.

**Jianshan Zhang** received the M.S. degree in material engineering from the College of Physics and Energy, Fujian Normal University, Fujian, China, in 2018. He is currently working toward the Ph.D. degree in computer science and technology with the College of Mathematics and Computer Science, Fuzhou University, Fujian.

His current research interests include edge computing, computational intelligence, and cloud computing.

**Junqin Hu** received the B.S. degree in computer science and technology in 2019 from Fuzhou University, Fujian, China, where he is currently working toward the M.S. degree in computer software and theory with the College of Mathematics and Computer Science.

Since September 2019, he has also been a student of the Fujian Key Laboratory of Network Computing and Intelligent Information Processing at Fuzhou University. His current research interests include computation offloading, edge computing, and cloud computing.

**Xing Chen** received the B.S. and Ph.D. degrees in computer software and theory from Peking University, in 2008 and 2013, respectively.

Upon completion of the Ph.D. degree, he joined Fuzhou University where held the rank of Associate Professor since 2016. Now he is the Deputy Director of Fujian Provincial Key Laboratory of Network Computing and Intelligent Information Processing, Fuzhou University, and leads the Systems research group. His current projects cover the topics from self-adaptive software, computation off-loading, model driven approach, and so on. He has authored or coauthored over 30 journal and conference articles, and has won the first Provincial Scientific and Technological Progress Award in 2018. His research interests include the software systems and engineering approaches for cloud and mobility.

**Jun Li** (M'09–SM'16) received Ph.D. degree in electronic engineering from Shanghai Jiao Tong University, Shanghai, P. R. China, in 2009.

From January 2009 to June 2009, he worked with the Department of Research and Innovation, Alcatel Lucent Shanghai Bell. From June 2009 to April 2012, he was a Postdoctoral Fellow with the School of Electrical Engineering and Telecommunications, the University of New South Wales, Australia. From April 2012 to June 2015, he was a Research Fellow with the School of Electrical Engineering, the University of Sydney, Australia. Since June 2015, he has been a Professor with the School of Electronic and Optical Engineering, Nanjing University of Science and Technology, Nanjing, China. His research interests include network information theory, ultra-dense wireless networks, and mobile edge computing.