Energy-Efficient Offloading for DNN-based Smart IoT Systems in Cloud-Edge Environments

Xing Chen, *Member, IEEE*, Jianshan Zhang, Bing Lin, Zheyi Chen, Katinka Wolter, *Senior Member, IEEE*, and Geyong Min, *Member, IEEE*

Abstract—Deep Neural Networks (DNNs) have become an essential and important supporting technology for smart Internet-of-Things (IoT) systems. Due to the high computational costs of large-scale DNNs, it might be infeasible to directly deploy them in energy-constrained IoT devices. Through offloading computation-intensive tasks to the cloud or edges, the computation offloading technology offers a feasible solution to execute DNNs. However, energy-efficient offloading for DNN based smart IoT systems with deadline constraints in the cloud-edge environments is still an open challenge. To address this challenge, we first design a new system energy consumption model, which takes into account the runtime, switching, and computing energy consumption of all participating servers (from both the cloud and edge) and IoT devices. Next, a novel energy-efficient offloading strategy based on a Self-adaptive Particle Swarm Optimization algorithm using the Genetic Algorithm operators (SPSO-GA) is proposed. This new strategy can efficiently make offloading decisions for DNN layers with layer partition operations, which can lessen the encoding dimension and improve the execution time of SPSO-GA. Simulation results demonstrate that the proposed strategy can significantly reduce energy consumption compared to other classic methods.

Index Terms—Cloud-edge computing, IoT systems, energy-efficient offloading, deep neural networks, particle swarm optimization

1 Introduction

As the core technology for supporting modern Artificial Intelligence (AI) systems, Deep Neural Networks (DNNs) have made great achievements in smart systems. Due to the powerful analysis capabilities for large-scale data, DNNs are regarded as a promising approach for effectively mining valuable information in the complex environment of Internet-of-Things (IoT) [1]. It has been pointed out that DNNs with more complex and deeper network structures can commonly provide more accurate analysis results [2]. However, more computational resources are consumed when using deeper network structures, which seriously limits the application of large-scale DNNs on energy-constrained IoT devices. One feasible solution is to offload some DNN layers to the remote cloud with sufficient resources [3]. Specifically, DNNs can be divided according to the granularity of DNN layers [4]. For example, complex DNN layers can be offloaded to the cloud for execution, while simple ones are processed locally. However, it might increase the traffic load of core networks and cause high latency due to massive data transmission between IoT devices and the cloud.

Mobile Edge Computing (MEC) would be a promising technology for smart IoT services in response to the above problems. Through offloading computations from the remote cloud to the network edge in proximity to IoT devices, MEC can significantly reduce the data transmission during the preprocessing stage [5]. Moreover, MEC intensifies the processing capacities of mobile networks by deploying computational and storage resources at the network edge. Therefore, it would be an efficient way to alleviate the traffic load of core networks by partitioning DNNs and offloading DNN layers over the cloud, edge, and IoT devices [6].

However, it is still challenging to offload DNN layers while considering energy consumption in the cloud-edge environments. Due to the short battery lifetime of IoT devices, offloading decisions should consider both the deadline constraints of DNN-based smart IoT systems as well as the energy consumption on servers (from the cloud and edge) and IoT devices. When the cloud-edge environments change, the offloading decisions should be adapted. For example, vehicle identification (*i.e.*, an road traffic application) relies on computer vision whose core technology is DNNs. Traffic cameras periodically record the images of on-road vehicles, and these images are processed by the DNN-based applications deployed on the traffic cameras. But these energy-constrained cameras with limited processing capabilities may not complete these computation-intensive

- *Xing Chen and Jianshan Zhang are with the College of Mathematics and Computer Science, Fuzhou University, Fuzhou, 350118, China, and with Fujian Provincial Key Laboratory of Network Computing and Intelligent Information Processing, Fuzhou, 350118, China. E-mail: chenxing@fzu.edu.cn, zhangjs0512@163.com.*
- *Bing Lin is with College of Physics and Energy, Fujian Normal University, Fujian Provincial Key Laboratory of Quantum Manipulation and New Energy Materials, Fuzhou, 350117, China.Fujian Provincial Collaborative Innovation Center for Advanced High-Field Superconducting Materials and Engineering, Fuzhou, 350117, China. Fujian Provincial Collaborative Innovation Center for Optoelectronic Semiconductors and Efficient Devices, Xiamen, 361005, China. E-mail: WheelLX@163.com.*
- *Zheyi Chen and Geyong Min are with the Department of Computer Science, College of Engineering, Mathematics and Physical Sciences, University of Exeter, Exeter, EX4 4QF, United Kingdom. E-mail: {zc300, g.min}@exeter.ac.uk.*
- *Katinka Wolter is with the Institut für Informatik, Freie Universität Berlin 14195, Germany. E-mail: katinka.wolter@fu-berlin.de.*

(Corresponding authors: Bing Lin and Zheyi Chen)

applications within their deadlines. Therefore, when dealing with the offloading problem of DNN layers, the offloading decisions should be made subject to the consideration of both deadline constraints and energy consumption and adapt to the changes of cloud-edge environments.

In response to these challenges, two research questions are considered: (1) How to design a system model to evaluate the energy consumption of participating equipment including servers and IoT devices? (2) How to develop an energy-efficient offloading strategy to handle the NP-hard offloading problem for DNN-based Smart IoT systems in the cloud-edge environments? To address the above questions, we develop an energy-efficient offloading strategy based on a Self-adaptive Particle Swarm Optimization Algorithm using the Genetic Algorithm operators (SPSO-GA). The major contributions of this work are summarized below:

1) A new system energy consumption model is designed for DNN-based smart IoT systems, which takes into account the runtime, switching, and computing energy consumption of all participating servers (from both the cloud and edge) and IoT devices.

2) An energy-efficient offloading strategy based on SPSO-GA is proposed to reduce the system energy consumption. The proposed strategy can make efficient offloading decisions for DNN layers with layer partition operations, which lessens the encoding dimension and improves the execution time of SPSO-GA.

3) The extensive simulation experiments are conducted. The performance results demonstrate that the proposed strategy can make quick offloading decisions and achieve the superior performance than other classic methods while satisfying deadline constraints.

The remainder of this paper is organized as follows. We review the related work in Section 2. Section 3 introduces the proposed energy-efficient offloading model for DNN-based smart IoT systems in the cloud-edge environments. In Section 4, the proposed SPSO-GA strategy with layer partition is described in detail. Section 5 presents extensive simulation experiments and results to evaluate the performance of our proposed strategy. Section 6 discusses the issues of applicability of SPSO-GA. Finally, Section 7 concludes this paper.

2 Related Work

With the increasing complexity of DNN-based smart IoT systems, computation offloading has become an attractive solution for extending the battery lifetime of IoT devices and satisfying the latency requirements of mobile systems [7].

Many research efforts have been devoted to the offloading problems in cloud computing. For example, Altamimi *et al*. [8] pointed out that it would be feasible to extend the battery lifetime and capabilities of smartphones by offloading tasks directly to the cloud. They designed a mathematical model to estimate the energy consumption during the task offloading process. Elgazzar *et al*. [9] proposed a framework of cloud-assisted mobile service provisioning for reliable deliveries. This framework can support online offloading based on the current network and resource status of mobile systems, while meeting user-defined energy constraints. Fang *et al*. [10] focused on QoS-aware scheduling of heterogeneous servers for DNN inference workloads in cloud computing. They designed a deep reinforcement learning (DRL) based scheduler to maximize the Quality-of-Service (QoS) including the inference accuracy and response delay. Kumar and Lu [11] argued that IoT systems may not be time-saving or energy-efficient if their computation tasks are offloaded to the cloud directly. Since it would increase the traffic load of core network and cause high latency due to the massive data transmission between IoT devices and the cloud.

MEC is a promising technology to solve the above problem, where the data transmission time in mobile systems can be tremendously reduced by offloading computation from the remote cloud to the edge that is close to IoT devices [12]. Chen *et al*. [13] used the idea of software-defined network (SDN) to study the offloading problem in the ultra-dense networks, aiming to minimize the delay and save the battery life of users' equipments. Chen *et al*. [14] studied the multi-user offloading problem in a multi-channel wireless interference environment and proposed a distributed offloading algorithm based on the game theory, which realized the Nash equilibrium. Ali *et al*. [15] adopted a deep learning approach and comprehensive mathematical modeling to offload application components to cloudlets, which can achieve high model accuracy and reduce the energy consumption of users' devices.

The aforementioned work focused on the offloading problem without data dependency. However, the data dependency is of important for the offloading problem in MEC [22]. This is because the data dependency exists between each pair of DNN layers. Lo *et al*. [16] presented a dynamic DNN design technique to manage the workload transmission under the same accuracy requirement in edge computing. They utilized the dynamic network structure and authentic operation (AO) unit to enhance DNNs, which had a better performance in terms of reducing the amount of workload transmission while achieving the required accuracy. Jeong *et al*. [17] designed a snapshot-based offloading method for machine learning web systems in edge environments, which was suitable for real DNN-based web systems. Yang *et al.* [18] designed a framework for runtime system repartitioning in dynamic mobile cloud environments, which can solve the performance degradation problem caused by fluctuant device status and dynamic network. Hu *et al*. [23] designed a dynamic DNN surgery strategy to partition DNN inference between the cloud and edge at the granularity of the DNN layers. This strategy reduced the system latency and improved throughput by limiting data transmission, but it paid less attention to the offloading problem for DNN layers. Mohammed *et al*. [19] proposed an adaptive DNN partition scheme and a distributed algorithm based on the matching game method, where the DNN layers were offloaded to fog nodes. Neurosurgeon [4] claimed that excessive latency and energy consumption were generated when uploading massive data of DNNs to the cloud via the wireless network. To cope with this problem, a lightweight scheduler was designed to partition DNN-based applications automatically between end devices and the cloud at the granularity of DNN layers.

Most of the aforementioned work tried to reduce the sys-

TABLE 1
The comparative analysis of different work ("+": involved; "-": not involved)

Reference	Infrastructure			Fluctuation		Application type		Constraint		Object				
	IoT Device	Edge	Cloud	Static	Dynamic	Jobs	Workflow	Energy	Deadline	Time	Energy	Cost	Workload balance	Throughput
Our work	+	+	+	+	-	-	+	-	+	-	+	-	-	-
Hu *et al.* ' [3]	-	+	+	-	+	-	+	-	-	+	-	-	-	+
Kang *et al.* ' [4]	+	-	+	+	-	-	+	-	-	+	+	-	-	+
Lin *et al.* ' [6]	+	+	+	+	-	-	+	-	+	-	-	+	-	-
Altamimi *et al.* ' [8]	+	-	+	+	-	+	-	-	-	-	+	-	-	-
Elgazzar *et al.* ' [9]	+	-	+	+	-	+	-	+	-	+	-	-	-	-
Fang *et al.* ' [10]	+	-	+	+	-	-	+	+	-	+	+	-	-	-
Chen *et al.* ' [13]	+	+	-	+	-	+	-	-	-	+	+	-	-	-
Chen *et al.* ' [14]	+	+	-	+	-	+	-	+	-	-	-	-	+	+
Ali *et al.* ' [15]	+	+	-	+	-	+	-	+	-	-	-	+	-	-
Lo *et al.* ' [16]	+	+	-	+	-	-	+	-	-	+	+	-	-	-
Jeong *et al.* ' [17]	+	+	-	+	-	-	+	-	-	+	-	-	-	-
Yang *et al.* ' [18]	+	+	+	-	+	-	+	-	-	+	+	-	-	-
Mohamed *et al.* ' [19]	-	+	-	-	+	-	+	-	-	+	-	-	+	-
Wu *et al.* ' [20]	-	+	+	-	+	-	+	-	-	+	+	-	-	-
Teerapittayanon *et al.* ' [21]	+	+	+	+	-	-	+	-	-	-	+	-	-	-

tem latency in cloud/edge environments [16], [18], [23], but they did not consider reducing the system energy consumption of offloading DNN layers with deadline constraints. An application partitioning algorithm was presented in [20] for pursuing a trade-off between energy consumption and data transmission in dynamic mobile environments. Teerapittayanon *et al*. [21] proposed distributed DNNs over the cloud, edge, and IoT devices. They considered the data transmission cost but not the layer execution consumption, while deploying distributed DNNs in the cloud-edge environments. In the previous work [6], a cost-driven offloading scheme was designed for DNN-based smart IoT systems with deadline constraints over the cloud, edge, and IoT devices, where a discrete PSO algorithm was developed to reduce the system cost of executing DNN layers and transferring data. Different from this work, we further consider the energy consumption of each participating server and IoT device, and introduce the layer partition operations into the offloading decision-making process for DNN layers.

The comparative analysis of the previous work has been illustrated in Table 1. In general, most of these work focused on the offloading problem in MEC. However, it is still an open issue to optimize the system energy consumption when offloading DNN layers with deadline constraints in the cloud-edge environments.

3 Problem Definition and Analysis

In the cloud-edge environments, DNNs can be deployed in the cloud, edge, and IoT devices. The energy-constrained IoT devices periodically receive terminal information. Some DNN layers are performed on IoT devices while the others are performed in the edge or cloud. Offloading decisions should be made based on the current environment situation, including the server status, network status, and existing tasks, to reduce the system energy consumption by offloading DNN layers while satisfying deadline constraints.

TABLE 2
List of abbreviations

Abbreviation	Description
AI	Artificial Intelligence
DNN	Deep Neural Network
IoT	Internet-of-Thing
MEC	Mobile Edge Computing
PSO	Particle Swarm Optimization
GA	Genetic Algorithm
DRL	Deep Reinforcement Learning
QoS	Quality-of-Service
AO	Authentic Operation

Particularly, when the environment remains unchanged, the offloading decisions will keep the same; and when the environment changes, the offloading decisions need to be adapted for better performance. In this work, we assume that the cloud-edge environments remain constant during a period of time. The major abbreviations and symbols used in this paper are defined in Table 2 and 3, respectively.

3.1 Problem Definition

The cloud-edge environments, denoted by $\boldsymbol{C}$=$\{\boldsymbol{C}_{\mathrm{c}}, \boldsymbol{C}_{\mathrm{e}}, \boldsymbol{C}_{\mathrm{d}}\}$, consist of the cloud, edge, and IoT devices. Each platform is equipped with some computing nodes (*i.e.* servers or virtual machines) with different computational capacities. For the clarity of presentation, we use 'server' to represent the computing nodes in different platforms. Thus, there are n servers in the cloud-edge environments, denoted by $\boldsymbol{C} = \{s_1, s_2, ..., s_n\}$, and a server s_i is defined as

$$s_i =< p_i, t_i, c_i, \boldsymbol{R}_i, \gamma_i >, \tag{1}$$

where p_i is the computational capacity of s_i that is expressed by the CPUs and assumed to be known and static. As this

study focuses on offloading DNN layers, we assume that the storage capacity of each server can meet the corresponding needs. $t_i \in \{0, 1, 2\}$ represents the type of s_i, which belongs to IoT devices, edge, and cloud, respectively. c_i represents the maximum number of concurrent tasks on s_i, whose value is equal to the number of cores in s_i [13]. When the server s_i is fixed, the value of c_i will be constant. $\boldsymbol{R}_i = \{< \zeta_i^1, \tau_i^1 >, < \zeta_i^2, \tau_i^2 >, ..., < \zeta_i^m, \tau_i^m >\}$ is the execution status set of s_i, which expresses the switching frequency of s_i. Each tuple $< \zeta_i^j, \tau_i^j >$ represents the j-th running interval of s_i with the start time ζ_i^j and end time τ_i^j. γ_i represents the energy consumption of s_i, which is described as

$$\gamma_i = \partial_i + \ell_i + \eta_i, \tag{2}$$

$$\partial_i = \sum_{j=1}^{|\boldsymbol{R}_i|} e_i^r \cdot (\tau_i^j - \zeta_i^j), \tag{3}$$

$$\ell_i = \sum_{j=1}^{|\boldsymbol{R}_i|} e_i^s, \tag{4}$$

$$\eta_i = \sum_{j=1}^{|\boldsymbol{R}_i|} \sum_{k=1} e_i^c \cdot w_i^k \cdot (\tau_i^j - \zeta_i^j), \tag{5}$$

where ∂_i represents the energy consumption of s_i running an empty loop, and e_i^r is the runtime energy consumption of s_i per time unit (*e.g.* second) when it is idle. e_i^r is assumed to be constant [15]. ℓ_i represents the switching energy consumption of s_i, and e_i^s is the switching energy consumption of s_i from power on to power off each time. η_i represents the computing energy consumption of s_i, which is related to its workload. e_i^c is the computing energy consumption of s_i per second. w_i^k is the current workload on s_i, which may change during the offloading process.

The bandwidth between two servers is defined as

$$\boldsymbol{B} = \begin{bmatrix} b_{1,1} & b_{1,2} & \cdots & b_{1,|\boldsymbol{C}|} \\ b_{2,1} & b_{2,2} & \cdots & b_{2,|\boldsymbol{C}|} \\ \vdots & \vdots & \cdots & \vdots \\ b_{|\boldsymbol{C}|,1} & b_{|\boldsymbol{C}|,2} & \cdots & b_{|\boldsymbol{C}|,|\boldsymbol{C}|} \end{bmatrix}, \tag{6}$$

where $b_{i,j} (\forall i, j = 1, 2, ..., |\boldsymbol{C}|; i \neq j)$ is the bandwidth between s_i and s_j. We assume that there is no ad hoc network, and it is directly unreachable between any two IoT devices [24]. In addition, WiFi has a certain connection range, and thus IoT devices can only connect to the edge servers within its radiation range. Besides, we assume that the bandwidth is not fluctuant.

Each IoT device obtains terminal data periodically, and there are many types of DNNs, denoted by $\boldsymbol{N} = \{\boldsymbol{N}_1, \boldsymbol{N}_2, ..., \boldsymbol{N}_q\}$. A DNN, denoted by $\boldsymbol{N}_i = (\boldsymbol{L}_i, \boldsymbol{H}_i, \boldsymbol{D}_i)$, is regarded as a workflow, where $\boldsymbol{L}_i = \{l_i^1, l_i^2, ..., l_i^q\}$ is the set of layers of $\boldsymbol{N}_i$, $\boldsymbol{H}_i = \{h_i^{1,2}, h_i^{1,3}, ..., h_i^{j,k}\}$ is the set of data dependencies between layers, and $\boldsymbol{D}_i = \{d_i^1, d_i^2, ..., d_i^r\}$ is the set of data transmissions between layers in $\boldsymbol{N}_i$. Moreover, the deadline of $\boldsymbol{N}_i$ is denoted as $D(\boldsymbol{N}_i)$.

More specifically, a layer l_i^j in $\boldsymbol{N}_i$ is defined as

TABLE 3
Symbol definitions

Symbol	Definition
$\boldsymbol{C} = \{\boldsymbol{C}_\mathrm{c}, \boldsymbol{C}_\mathrm{e}, \boldsymbol{C}_\mathrm{d}\}$	Cloud-edge environments consisting of the cloud $\boldsymbol{C}_\mathrm{c}$, edge $\boldsymbol{C}_\mathrm{e}$, and IoT devices $\boldsymbol{C}_\mathrm{d}$
s_i	Server i
p_i	Computing capacity of s_i
t_i	Type of s_i
c_i	Maximum number of concurrent tasks on server s_i
$\boldsymbol{R}_i$	Execution status set of s_i
ζ_i^j	Start time of the j^{th} running interval of s_i
τ_i^j	End time of the j^{th} running interval of s_i
γ_i^j	Energy consumption of s_i
∂_i	Runtime energy consumption of s_i
e_i^r	Runtime energy consumption per second of s_i
ℓ_i	Switching energy consumption of s_i
e_i^s	Switching energy consumption per second of s_i each time
η_i	Computing energy consumption of s_i
e_i^c	Computing energy consumption per second of s_i per second
w_i^k	Current workload on s_i
$b_{i,j}$	Bandwidth between s_i and s_j
$\boldsymbol{N}_i = (\boldsymbol{L}_i, \boldsymbol{H}_i, \boldsymbol{D}_i)$	DNN i with layer set $\boldsymbol{L}_i$,data dependency set $\boldsymbol{H}_i$ and data transmission set $\boldsymbol{D}_i$
l_i^j	Layer j in $\boldsymbol{N}_i$
$h_i^{j,k}$	Data dependency between l_i^j and l_i^k
a_i^j	Calculation amount of l_i^j
i_i^j	Input data of l_i^j
o_i^j	Output data of l_i^j
d_i^j	A data transmission j in $\boldsymbol{N}_i$
Ω_i^j	Amount of d_i^j
$t_t(d_i^j, s_k, s_r)$	Time for transferring d_i^j from s_k to s_r
t_i^c	Completion time of $\boldsymbol{N}_i$
e_t	Total energy consumption for executing all DNN layers

$$l_i^j =< i_i^j, o_i^j, a_i^j >, \tag{7}$$

$$t_e(l_i^j, s_k) = \frac{a_i^j}{p_k}, \tag{8}$$

where i_i^j and o_i^j are the input and output data of l_i^j, respectively. a_i^j is the calculation amount of l_i^j. Therefore, the execution time of l_i^j on s_k is defined as $t_e(l_i^j, s_k)$).

In the data dependency $h_i^{j,k} = (l_i^j, l_i^k)$, l_i^j is the direct precursor of l_i^k, and l_i^k is the direct successor of l_i^j. A layer can be executed only after all its direct precursors have been completed.

In the data transmission $d_i^j =< \Omega_i^j, s_k, s_r >$, Ω_i^j represents the amount of data transmission, and s_k and s_r represent the original and final servers storing d_i^j, respectively. Therefore, the data transmission time of d_i^j from s_k to s_r is defined as

$$t_t\left(d_i^j, s_k, s_r\right) = \frac{\Omega_i^j}{b_{k,r}}. \tag{9}$$

Moreover, the proposed offloading strategy is defined as $\boldsymbol{S} = (\boldsymbol{C}, \boldsymbol{L}_i, \boldsymbol{M}, t_i^c, e_t)$, where $\boldsymbol{M} = \{(l_i^j, s_k) | l_i^j \in \boldsymbol{L}_i, s_k \in \boldsymbol{C}\} \cup \{\Phi_k\}$ represents the mapping set from DNN layers to servers, (l_i^j, s_k) represents that layer l_i^j is executed on s_k, and Φ_k is the execution order of the concurrent layers on s_k. When the mapping set $\boldsymbol{M}$ is determined, all the datasets

TABLE 4
Bandwidth between servers

t_i		t_j	$b_{i,j}$(MB·s^{-1})
0	↔	1	10
0	↔	2	0.5
1	↔	1	10
1	↔	2	0.5

from one server to another is determined accordingly, t_i^c represents the completion time of $\boldsymbol{N}_i$, and e_t represents the total energy consumption of all participating servers for executing DNN layers.

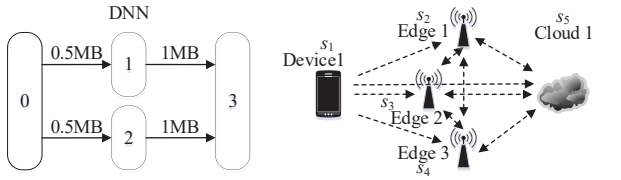
$$t_i^c = \max_{l_i^j \in \boldsymbol{L}_i} \{t^c(l_i^j)\}, \tag{10}$$

$$e_t = \sum_{i=1}^{|\boldsymbol{C}|} \mu \cdot (\partial_i + \ell_i + \eta_i), \tag{11}$$

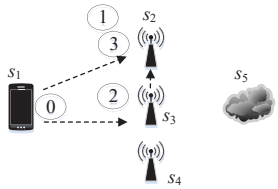
where $t^c(l_i^j)$ represents the completion time of l_i^j, and μ is the adjustment factor for energy consumption. If the running server belongs to IoT devices, $\mu = 1.2$. Otherwise, $\mu = 1.0$ [25]. This adjustment design tends to offload more layers to the cloud or edge within their deadlines. Therefore, the objective of the proposed energy-efficient offloading strategy for DNN-based smart IoT systems in the cloud-edge environments is formalized as

$$\begin{aligned} &\textbf{Minimize } e_t, \\ &\textbf{subject to } \forall i, t_i^c \leq D(\boldsymbol{N}_i). \end{aligned} \tag{12}$$

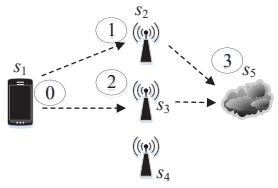
3.2 Problem Analysis



(a) A DNN and the cloud-edge environment.



(b) Offloading result by using Greedy.



(c) Offloading result by using the optimal strategy.

Fig. 1. An example of energy-efficient offloading for a DNN.

Fig. 1 illustrates an example of energy-efficient offloading for a DNN $\boldsymbol{N}_i$ in a cloud-edge environment consisting of six servers, denoted by $\{s_1, s_2, s_3, s_4, s_5, s_6\}$. $\boldsymbol{N}_i$ contains four layers $\{l_i^0, l_i^1, l_i^2, l_i^3\}$ and four data transmissions $\{d_i^1, d_i^2, d_i^3, d_i^4\}$ with different data sizes {0.5 MB, 0.5 MB, 1 MB, 1 MB}. The left subgraph in Fig. 1(a) shows the data transmissions between each pair of DNN layers. It should be noted that the input layer l_i^0 must be executed on the IoT device s_1. The deadline $D(\boldsymbol{N}_i)$ is 5.25 *ms*. Table 4 shows the bandwidth between servers with different types [6]. Table 5 shows the parameter settings about the energy consumption for different servers. Table 6 shows the calculation amount and execution time of different layers on feasible servers.

TABLE 5
Parameter settings about energy consumption for servers

Servers	e_i^r(μW·ms^{-1})	e_i^c(μW·ms^{-1}·byte^{-1})	e_i^s(μW)
s_1	0.1	0.1	0.3
s_2	0.2	0.2	0.95
s_3	0.18	0.18	0.45
s_4	0.22	0.22	0.65
s_5	0.3	0.3	0.1

TABLE 6
Calculation amount and execution time of different layers on feasible servers

	a_i^j	s_1	s_2	s_3	s_4	s_5
l_i^0	1 byte	1.41 ms	-	-	-	-
l_i^1	2 byte	1.87 ms	0.76 ms	0.82 ms	0.97 ms	0.24 ms
l_i^2	2.5 byte	2.78 ms	1.51 ms	1.42 ms	1.67 ms	0.82 ms
l_i^3	0.5 byte	3.32 ms	2.08 ms	1.23 ms	2.53 ms	0.32 ms

Fig. 1(b) depicts the offloading results by using the greedy algorithm [26] without considering the servers in the cloud. This algorithm tends to offload each layer to the corresponding most energy-efficient edge server or IoT device, and thus each layer is offloaded to the server with the lowest computing energy consumption within their deadlines. The completion time t_i^c of $\boldsymbol{N}_i$ is 5.06 *ms*, and the system energy consumption e_t is 5.067 μW. Fig. 1(c) depicts the optimal offloading results where the completion time t_i^c is 5.21 *ms* and system energy consumption e_t is 4.459 μW that is less than (about 12.01%) the results shown in Fig. 1(b).

4 Offloading Strategy based on SPSO-GA

The proposed offloading strategy, denoted by $\boldsymbol{S} = (\boldsymbol{C}, \boldsymbol{L}_i, \boldsymbol{M}, t_i^c, e_t)$, is to explore an optimal mapping $\boldsymbol{M}$ from the DNN layers $\boldsymbol{L}_i$ to different types of servers $\boldsymbol{C}$, which aims to reduce the system energy consumption from the perspectives of runtime, switching, and computing, while satisfying the deadline constraints of each DNN. Specifically, a DNN layer can be offloaded to different servers, and an available server can execute many layers from different DNNs. Therefore, it has been proved to be an NP-hard problem to explore the optimal mapping from the layers to servers [27]. The PSO algorithm is be a feasible solution for this problem [6], [28], but it may fall into the local optimum. As for the GA, its local search ability is poor, which results in the low search efficiency in the later stage of evolution [29]. Therefore, these two algorithms cannot well handle the complex offloading problem of DNN layers. In light of the PSO and GA, we propose a new SPSO-GA to explore an optimal offloading strategy for DNN layers in the cloud-edge environments.

4.1 Layer Partition for DNNs

The operations of layer partition are to partition consecutive DNN layers into different deployment units. For example,

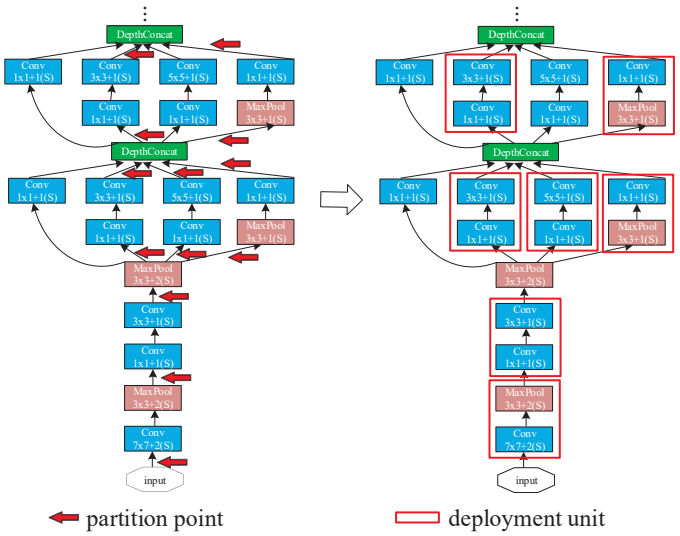


Fig. 2. Layer partition for GoogleNet.

as shown in Fig. 2, two DNN layers such as *Conv 3x3+1(S)* and *Conv 1x1+1(S)* are partitioned into a deployment unit, and the DNN layers in this deployment unit would be fitted to be deployed on the same server or IoT device. Algorithm 1 shows the process of layer partition for a DNN, where the input is the original DNN and the output is the DNN with deployment units. It should be noted that the subsequent offloading decisions would be made based on the output DNN. Specifically, different branches in a DNN are first divided into isolated modules (Line 2). For each module, the current layer is initialized as the start layer (Lines 6-8). From the current layer, every two adjacent layers will be checked orderly according to the fitness function defined in Eq. (16) (Lines 10-13). Once a partition point is found, the current layer will be updated to the next layer after this partition point (Lines 14-21). The above process will be repeated until the last two adjacent layers in a DNN are checked. Finally, the layers between every two adjacent partition points are combined into a deployment unit (Lines 23-24).

Under the cloud-edge environments with various computational resources and network connections, there might be different numbers of consecutive DNN layers that are fitted to be deployed together. Basically, with the higher performance ratio (λ) between servers and IoT devices and faster data transmission rate (v_p), the offloading tends to happen while the deployment units are constructed by fewer consecutive DNN layers. With the consideration of these factors, the layer partition can be used to find the consecutive DNN layers that are fitted to be deployed on the same server or IoT device and partition them into different deployment units according to the fitness function.

Property 1: Layer partition is able to improve the execution time of the proposed SPSO-GA by partitioning consecutive DNN layers into different deployment units.

In SPSO-GA, the encoding dimension is linearly positive to the number of deployment units in a DNN, and it determines the execution time of the algorithm. Compared to the total number of layers in a DNN, there are fewer deployment units after taking layer partition. Therefore, the layer partition can effectively reduce the encoding dimension and improve the execution time of SPSO-GA when considering an offloading problem.

ALGORITHM 1: Layer partition for a DNN

Input: The original DNN $\boldsymbol{N}_i$.
Output: The DNN $\boldsymbol{N}_j$ with deployment units.

1 **begin**
2 Divide different branches in $\boldsymbol{N}_i$ into isolated modules;
3 **foreach** *module* $\boldsymbol{M}_k^i \in \boldsymbol{N}_i$ **do**
4 //Initialize the set of partition points.
5 $\boldsymbol{P}_k = \emptyset$;
6 //Initialize the current layer and the index to the first and second layers of $\boldsymbol{M}_k^i$, respectively.
7 cur = 1;
8 ind = 2;
9 **while** $ind \leq |\boldsymbol{M}_k^i|$ **do**
10 $l_{left} = \{l_{cur}, ..., l_{ind-1}\}$;
11 $l_{right} = \{l_{ind}, ..., l_{|\boldsymbol{M}_k^i|}\}$;
12 //$U(x, y)$ decides whether x and y are suitable to be partitioned according to the fitness function defined in Eq. (16).
13 $p_{ind}^k = U(l_{left}, l_{right})$;
14 **if** $p_{ind}^k \neq \emptyset$ **then**
15 $\boldsymbol{P}_k = \boldsymbol{P}_k \cup p_{ind}^k$;
16 $cur = ind$;
17 $ind = cur + 1$;
18 **end**
19 **else**
20 ind += 1;
21 **end**
22 **end**
23 //Based on the original module and the set of partition points, $D(x, y)$ combines the layers between every two adjacent partition points into a deployment unit, and generates a new module.
24 $\boldsymbol{M}_k^j = D(\boldsymbol{M}_k^i, \boldsymbol{P}_k)$;
25 **end**
26 $\boldsymbol{N}_j = \bigcup_k \{\boldsymbol{M}_k^j\}$; //Generate a new DNN.
27 **return** $\boldsymbol{N}_j$;
28 **end**

4.2 Design of SPSO-GA

In the traditional PSO algorithm, the population commonly contains n_p particles, and a particle $Q_i^t = (X_i^t, V_i^t)$ has its own position, denoted by $X_i^t = (x_{i1}^t, x_{i2}^t, \ldots, x_{iy}^t)$, and velocity, denoted by $V_i^t = (v_{i1}^t, v_{i2}^t, \ldots, v_{iz}^t)$, in a continuous space at the t-th iteration. The position represents a candidate solution, and the velocity helps each particle to search for better positions. The global best particle in the population at the t-th iteration is defined as G^t, and the personal best particle in its own history at the t-th iteration is defined as P_i^t. Thus, the update process for the position and velocity of each particle can be described as

$$V_i^{t+1} = w \times V_i^t + c_1 r_1 (P_i^t - X_i^t) + c_2 r_2 (G^t - X_i^t), \quad (13)$$

$$X_i^{t+1} = X_i^t + V_i^{t+1}, \quad (14)$$

where w is the inertia weight that impacts the convergence of PSO. c_1 and c_2 are the acceleration coefficients that represent the cognitive ability of a particle for its personal and global best values, respectively. r_1 and r_2 are the random numbers on the interval [0,1) that are used to enhance the random search for an optimal result.

Commonly, the traditional PSO algorithm is used to solve continuous optimization problems. However, the energy-efficient offloading strategy for DNN layers is to explore an optimal mapping from deployment units to different servers, which is a discrete optimization problem. Therefore, the 'problem encoding' and 'population update' parts in SPSO-GA should be further adjusted. In addition, the expression of 'DNN layers' replaces that of 'deployment units' for ease of understanding in this Subsection.

4.2.1 Problem Encoding

Problem encoding affects the searchability of the PSO algorithm, which is expected to meet three basic criteria [30] as follows.

Definition 1 (Completeness). Each candidate solution must have at least one corresponding encoded particle.

Definition 2 (Non-redundancy). Each candidate solution has only one corresponding particle.

Definition 3 (Viability). Each candidate solution, corresponding to the encoded particles, is feasible.

The encoding of a constrained optimization problem usually fails to satisfy all three criteria described above. The position of a particle at the t-th iteration, denoted by $X_i^t = (x_{i1}^t, x_{i2}^t, \ldots, x_{iy}^t)$, represents a candidate solution for the optimization problem. Inspired by the work in [31], we use a server-order combination, to encode the energy-efficient offloading problem. Fig. 3 illustrates an example of the encoded particle for the offloading results in Fig. 1(c). Each dimension in a particle is defined as

layers	0	1	2	3
offloading location	<1,3>	<2,0>	<3,1>	<5,2>

Fig. 3. An example of the encoded particle for the offloading in Fig. 1(c).

$$x_{ij}^t =< s_k, \varphi_k >_{ij}^t, \quad (15)$$

where $x_{ij}^t (j = 1, 2, \ldots, y)$ is the j-th dimension of X_i^t, which represents the offloading decision of the j-th layer. The total number of DNN layers is y. s_k indicates the execution server for the j-th layer, and φ_k is the execution order of the j-th layer on s_k. $(s_k, \varphi_k)_{ij}^t$ indicates that the j-th layer is executed on s_k with the order φ_k in at the t-th iteration. φ_k is a unique integer on the interval [0, y). When more than two concurrent layers (without data dependency) are offloaded to the same server, the layers with smaller values of orders are executed first.

Property 2: Problem encoding meets the criteria of completeness and non-redundancy, but may fail to meet the viability criterion.

Each offloading solution represents the mappings from layers to servers with a specified order. After making offloading decisions, each DNN layer is allocated to an execution server with the corresponding execution order, which is depicted in the dimension of a particle. Therefore, our encoding method meets the completeness criterion. Moreover, a 2y-dimensional particle corresponds to an offloading solution. The j-th dimension of a particle represents the execution server s_k for the j-th layer with the order φ_k. Therefore, an offloading solution only maps to an encoding method, which meets the non-redundancy criterion. However, some particles may be infeasible, which means that the offloading solutions fail to satisfy deadline constraints. For example, a particle of the offloading results in Fig. 1 is (<1,3>, <2,0>, <3,1>, <4,2>), and thus the layers $(l_i^0, l_i^1, l_i^2, l_i^3)$ are offloaded to the servers (s_1, s_2, s_3, s_4), respectively. The completion time t_i^c of $\boldsymbol{N}_i$ is 5.51 *ms*, which exceeds the deadline (*i.e.* 5.25 *ms*). In this situation the encoding method may fail to meet the viability criterion. Therefore, there are two categories of particles in the problem space as follows.

Definition 4 (Feasible particle). A particle corresponding to a candidate solution meets the deadline constraints of all DNN-based smart IoT systems.

Definition 5 (Infeasible particle). A particle corresponding to a candidate solution fails to meet the deadline constraints of at least one DNN-based smart IoT system.

4.2.2 Fitness Function

Fitness function is used to evaluate the performance of particles and to select the global and personal best particles G^t and P_i^t, respectively. A particle with a smaller value of the fitness function indicates a better candidate offloading solution. According to our optimization objective, we regard the total system energy consumption as the fitness value of a particle. Since some solutions may exceed the deadlines of DNN-based smart IoT systems, we define the fitness function under three different cases as follows.

Case 1: Both particles are feasible. The one with less total energy consumption would be selected. Therefore, the fitness function is defined as

$$f(X_i) = e_t(X_i). \quad (16)$$

Case 2: One particle is feasible but another is infeasible. The feasible one would be selected. Therefore, the fitness function is defined as

$$f(X_i) = \begin{cases} 0, \text{if} \quad \forall i, t_i^c(X_i) \le D(N_i) \\ 1, \text{else} \end{cases}. \quad (17)$$

Case 3: Both particles are infeasible. The one with less completion time would be selected because it has a greater chance to become feasible after population update. Therefore, the fitness function is defined as

$$f(X_i) = \max\{t_i^c(X_i)\}. \quad (18)$$

4.2.3 Population Update

Eqs. (13) and (14) indicate the particle update for continuous optimization problems, which are affected by three factors: *inertia*, *individual cognition*, and *social cognition* [32]. When it comes to the DNN offloading, the particle update is defined as

$$X_i^{t+1} = F_3(F_2(F_1(X_i^t, w, r_1), P_i^{t-1}, c_1, r_2), G^t, c_2, r_3), \tag{19}$$

where $F_1()$ denotes the mutation operation. $F_2()$ and $F_3()$ denote the crossover operations. w is the inertia weight. c_1 and c_2 are acceleration coefficients. r_1, r_2, and r_3 are the random numbers on the interval [0,1).

For the *inertia* part, the velocity of particles is defined as

$$V_i^t = F_1(X_i^{t-1}, w, r_1) = \begin{cases} M_u(X_i^{t-1}) & r_1 < w \\ X_i^{t-1} & else \end{cases}, \tag{20}$$

where $M_u()$ denotes the mutation operator, and w denotes the mutation probability. Mutation operator randomly selects a location ind in a particle, and changes the corresponding value of the server-order tuple. The new value of server s_k is a random integer between 1 and $|\boldsymbol{C}|$, and the corresponding order value is also a random integer between 0 and $y-1$. Since the order value in a particle is unique, the other location with the same order is selected and its order value is changed to the original value of location ind. Fig. 4 depicts the mutation operator, which randomly selects a location ind_1 and changes the value of the server-order tuple from <5,2> to <4,3>. Next, the order value of the first location is changed to 2.

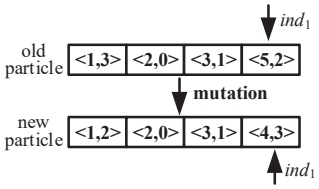


Fig. 4. Mutation operator.

Property 3: The mutation operator may change a particle from feasible to infeasible, and vice versa.

For example, the completion time of the old particle in Fig. 4 is 5.21 *ms*, and thus the old particle is feasible. After mutation operator, the completion time of the new particle is 5.51 *ms*, which exceeds its deadline (*i.e.* 5.25 *ms*). Therefore, the mutation operator may change the old particle from feasible to infeasible. If the positions of the old and new particles are reversed, the mutation operator will change the old particle from infeasible to feasible.

The *individual cognition* and *social cognition* in the particle update are formulated as

$$C_i^t = F_2(V_i^t, P_i^{t-1}, c_1, r_2) = \begin{cases} C_p(V_i^t, P_i^{t-1}) & r_2 < c_1 \\ V_i^t & else \end{cases}, \tag{21}$$

$$X_i^{t+1} = F_3(C_i^t, G^t, c_2, r_3) = \begin{cases} C_p(C_i^t, G^t) & r_3 < c_2 \\ C_i^t & else \end{cases}, \tag{22}$$

where c_1 and c_2 denote the crossover probability of a particle with its personal best solution P_i^t and global optimal solution G^t, respectively. $C_p()$ denotes the crossover operator. For example, $C_p(A, B)$ randomly selects two locations in the particle A and then replaces the server-order segment between these two locations with the same interval in the particle B. The order of the generated particle needs to be adjusted. Fig. 5 depicts the crossover operator, which randomly selects the locations ind_1 and ind_2 in an old particle and replaces the server-order segment between ind_1 and ind_2 with the same interval in P_i^t (or G^t). Next, the order value of the 4th location in the old particle is changed to 2.

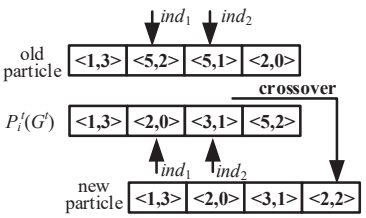


Fig. 5. Crossover operator.

Property 4: The crossover operator may change a particle from infeasible to feasible, and vice versa.

For example, the completion time of the old particle in Fig. 5 is 7.31 *ms*, which exceeds the deadline (*i.e.* 5.25 *ms*). Therefore, the old particle is infeasible. After crossover operator, the completion time of the new particle is 5.06 *ms*. Therefore, the crossover operator changes the old particle from infeasible to feasible. If the positions of the old and new particles are reversed, and the global optimal particle is (<1,3>, <4,0>, <2,1>, <5,2>), the crossover operator will change the old particle from feasible to infeasible.

4.2.4 Offloading corresponding a particle

Algorithm 2 shows the offloading corresponding to a particle. The inputs contain DNNs $\boldsymbol{N}$, all feasible servers $\boldsymbol{C}$, and an encoded particle X_i^t. According to the encoded particle X_i^t, the algorithm first assumes that each layer l_i^j is offloaded to the server $s_{x(j)}$ with the order $\varphi_{x(j)}$. For l_i^j, its start time $t_{start}(l_i^j)$ is equal to the ready time $t_{ready}(s_{x(j)})$ of $s_{x(j)}$ if it has no parents. Otherwise, the layer cannot start until the data transmission between $s_{x(j)}$ and its parents is completed (Lines 4-13). Next, the end time $t_{end}(l_i^j)$ of l_i^j is equal to the sum of its execution time $t_e(l_i^j, s_{x(j)})$ on $s_{x(j)}$ and start time (Line 14). If $t_{end}(l_i^j)$ exceeds the deadline, the algorithm stops immediately and return $Null$, which means that this particle is infeasible (Lines 15-17). The next ready time of $s_{x(j)}$ should be calculated until the data transmission between l_i^j and its children is completed (Lines 18-22). Moreover, the algorithm checks each free interval for all participating servers and shuts down the servers during the free intervals if their runtime energy consumption ∂_i is greater than their switching energy consumption ℓ_i (Lines 24-28). Finally, the system energy consumption e_t is calculated based on Eq. (11).

ALGORITHM 2: The offloading corresponding to a particle

Input: $\boldsymbol{N}, \boldsymbol{C}, X_i^t$.
Output: e_t or $Null$. // If X_i^t is infeasible, return $Null$.

begin
  //l_i^j is offloaded to $s_{x(j)}$ with $\varphi_{x(j)}$.
  foreach l_i^j *in* $\boldsymbol{N}$ **do**
    if l_i^j *has no parent* **then**
      $t_{start}(l_i^j) = t_{ready}(s_{x(j)})$;
    end
    else
      t_{trans}=0;
      foreach *parent* l_i^p *of* l_i^j **do**
        $t_{trans} = \max(t_{trans}, \frac{\Omega_i^p}{b_{x(p),x(j)}})$;
      end
      $t_{start}(l_i^j) = t_{ready}(s_{x(j)}) + t_{trans}$;
    end
    $t_{end}(l_i^j) = t_{start}(l_i^j) + t_e(l_i^j, s_{x(j)})$;
    if $t_{end}(l_i^j) > D(\boldsymbol{N}_i)$ **then**
      return $Null$; // Infeasible particle.
    end
    $t_{trans} = 0$;
    foreach *child* l_i^c *of* l_i^j **do**
      t_{trans} += $\frac{\Omega_i^j}{b_{x(j),x(c)}}$;
    end
    $t_{ready}(s_{x(j)})$ += $t_e(l_i^j, s_{x(j)}) + t_{trans}$;
  end
  foreach *free interval in* s_i **do**
    if $\partial_i > \ell_i$ **then**
      Shut down s_i in this interval;
    end
  end
  return e_t according to Eq. (11);
end

4.2.5 Parameter Settings

The inertia weight w may greatly influence the search ability and convergence of the PSO algorithm [33]. The larger value of w corresponds to a stronger global search ability, while the smaller one reflects a better local search ability. To fit in the nonlinear characteristics of energy-efficient offloading for DNN-based smart IoT systems, we design a discrete adjustment method for updating w as

$$w = w_u - (w_u - w_d) \times \exp(\frac{\Psi}{\Psi - 1.01}), \tag{23}$$

$$\Psi = div(G^{t-1}, X^{t-1}) = \frac{\sum_{i=1}^{y} z_i}{y}, \tag{24}$$

where w_u and w_d denote the maximum and minimum of w during initialization, respectively. Ψ represents the degree of difference between the global optimal solution G^{t-1} and the current candidate solution X^{t-1}. z_i denotes a statistical factor. $z_i = 1$ indicates that the server s_k of G^{t-1} and X^{t-1} in the same location is different otherwise is $z_i = 0$. Therefore, the search ability of the proposed algorithm can be adaptively adjusted based on the difference between the current and the global optimal particles. When the gap between G^{t-1} and X^{t-1} is large, the algorithm has a strong global search ability. On the contrary, the algorithm would enhance its local search ability to explore an optimal solution.

Moreover, the acceleration coefficients c_1 and c_2 are used for communications in the population whose settings are referred to the work in [34]. More specifically, c_1^s, and c_2^s are the start value of c_1 and c_2, while c_1^e and c_2^e are the end value of c_1 and c_2.

4.2.6 Algorithm Flowchart

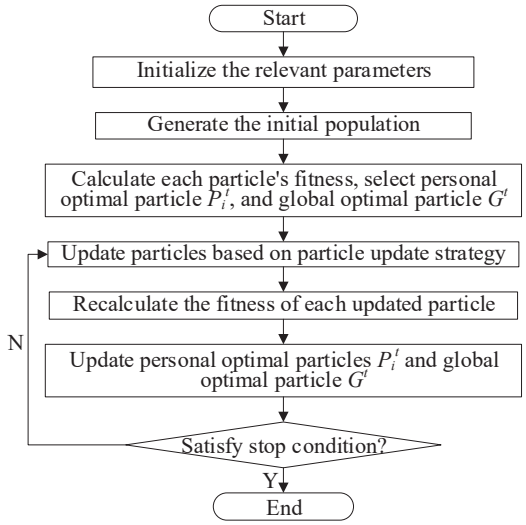


Fig. 6. Algorithm flowchart for SPSO-GA.

Fig. 6 depicts the flowchart of the proposed SPSO-GA, where the detailed process is described as follows.

Step 1: The relevant parameters of SPSO-GA are initialized, including the initial population size Γ, maximum number of iterations Θ, the maximum inertia weight w_u, the minimum inertia weight w_d, the start and end value of acceleration coefficients $c_1^s, c_2^s, c_1^e, c_2^e$. Next, the initial population is generated randomly.

Step 2: The fitness of each particle is calculated according to Eqs. (16) to (18). Each particle is selected as its own personal optimal solution, and the particle with the best fitness is selected as the global optimal solution in the current generation.

Step 3: Each particle is updated according to Eq. (19), and the fitness of each new particle is recalculated.

Step 4: The personal optimal particle for each particle is updated. The global optimal particle will be updated if there is a better solution than the original one.

Step 5: If the stop condition will be met, the algorithm is determined. Otherwise, the algorithm goes to ***Step 3***.

5 Performance Evaluation

To investigate and validate the effectiveness of the proposed SPSO-GA, extensive simulation experiments have been conducted in response to the research questions (**RQs**) as follows.

TABLE 7
The related configurations and energy consumption of 15 servers

Servers	p_i	c_i	t_i	e_i^r	e_i^s	e_i^c
$\{s_1, s_2, ..., s_{10}\}$	0.5	1	0	0.1	30	0.1
s_{11}	1	4	1	0.2	45	0.2
s_{12}	0.8	4	1	0.18	45	0.18
s_{13}	1.2	4	1	0.22	45	0.22
s_{14}	1.5	4	1	0.27	45	0.27
s_{15}	2	8	2	0.3	60	0.3

The unit of p_i, e_i^r, e_i^s, e_i^c is Hz, μW·ms^{-1}, μW, μW·ms^{-1}·byte^{-1}, respectively.

- **RQ1**: Compared with other classic methods, is SPSO-GA able to optimize the system energy consumption more effectively with different deadline constraints? (Section 5.3)
- **RQ2**: For the same optimization objective, is SPSO-GA superior to other classic methods with different workloads? (Section 5.4)
- **RQ3**: What impact do varying supplies of MEC resources have on the performance of SPSO-GA and other classic methods? (Section 5.5)
- **RQ4**: Whether the layer partition operations are able to effectively reduce the execution time of SPSO-GA? (Section 5.6)

5.1 Experimental Settings

There are four types of DNNs used in the experiments including AlexNet, VGG19, GoogleNet, and ResNet101 [35]. The detailed information about each type of DNN, including the structure, calculation amount of each layer, and data transmissions between layers, which are obtained from actual execution and simulation modeling [36], can be found in our GitHub project[1]. Besides, the deadline constraints for each DNN are needed to check whether an energy-efficient offloading strategy for DNN-based smart IoT systems is feasible or not. Therefore, five different deadline constraints for each DNN are set as

$$D_j(\boldsymbol{N}_i) = \varepsilon_j \cdot H(\boldsymbol{N}_i),\ \varepsilon_j = \{1.5, 2, 3, 5, 8\}, \quad (25)$$

where $H(\boldsymbol{N}_i)$ is the execution time of $\boldsymbol{N}_i$ by using the HEFT algorithm [37], and the values in ε_j are set according to [6].

Furthermore, the cloud-edge environments consist of 15 servers $\{s_1, s_2, ..., s_{15}\}$, which are divided into 3 types including the cloud, edge and IoT devices. More specifically, the first 10 servers, the last server, and the other 4 servers belong to IoT devices, the cloud, and the edge, respectively. Table 7 shows the detailed configurations and energy consumption of the servers [25]. The number of DNNs executed by an IoT device is regarded as the workload, is set to one in **RQ1**, **RQ3** and **RQ4**, and ranges from 1 to 5 in **RQ2**. Besides, we assume that each IoT device can only connect to two nearby edge servers, and the bandwidths between different servers are shown in Table 4. During a period of time, the cloud-edge environment and the workload remain static, based on which the offloading decision is made.

1. https://github.com/SPSO-GA/dataset

To find all possible deployment units during the layer partition process, the most ideal offloading environment in our experimental environments (*i.e.* from an IoT device to the edge server s_{14}) is selected with the consideration of server performance and data transmission rate to conduct the simulation offloading experiment of Algorithm 1. λ and v_p are set to 3 and 10 MB·s^{-1}, respectively.

In addition, the parameters of SPSO-GA were initialized by following [38], where Γ = 50, Θ = 300, w_u = 0.8, w_d = 0.2, c_1^s = 0.9, c_1^e = 0.2, c_2^s = 0.4, and c_2^e = 0.9. Simulation experiments were conducted on the Win10 64-bit operating system with an i7-4790 3.60 GHz Intel(R) Core(TM) processor and 32GB RAM.

5.2 Classic Methods

GA [29] and Greedy [26] are introduced to make comparisons and evaluate the performance of the proposed SPSO-GA for DNN-based smart IoT systems in the cloud-edge environments.

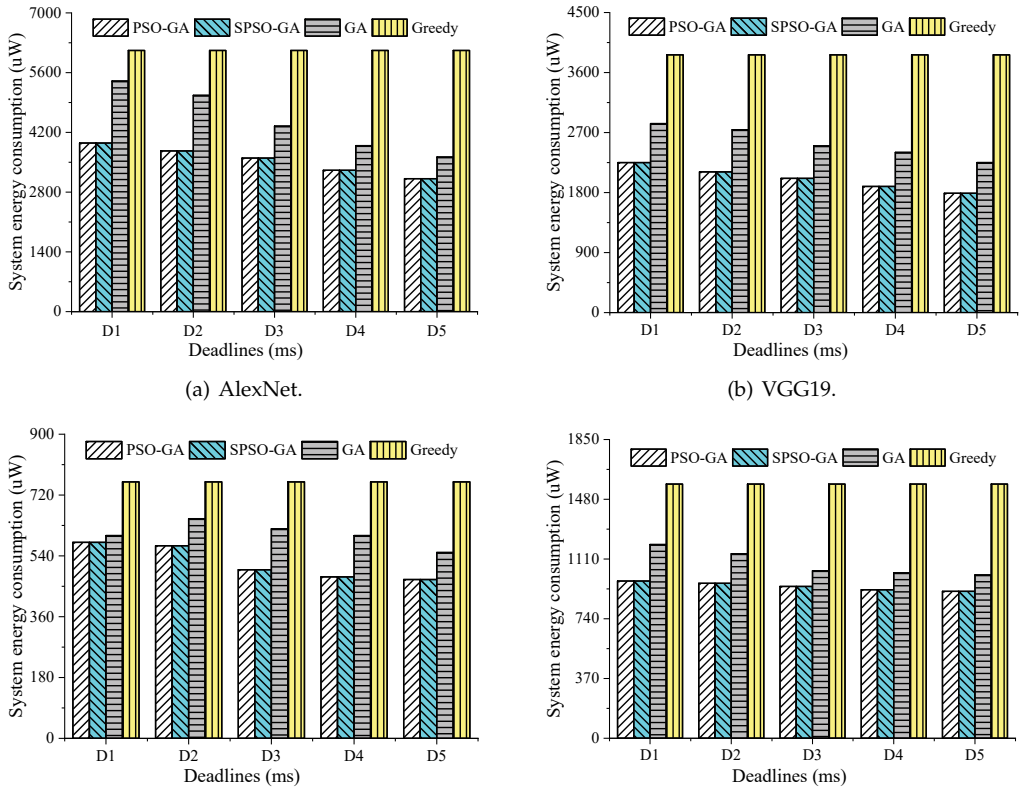
Greedy only considers edge servers and IoT devices but ignores the servers in the cloud. The algorithm offloads each layer to the servers with the lowest computing energy consumption within their deadlines. If a layer fails to satisfy the deadline constraint, it will be offloaded to the next cheapest server.

GA uses a binary problem encoding method, whose dimension is equal to the number of servers. Meanwhile, its fitness function is defined based on Eqs. (16)-(18). Thus, the offloading corresponding to an encoded chromosome also considers the runtime energy consumption, switching, and computing energy consumption of each server.

To verify the efficiency of layer partition, SPSO-GA without layer partition (*i.e.* PSO-GA) is also used as the comparison. GA, PSO-GA, and SPSO-GA belong to the evolutionary algorithms. In the experiments, these algorithms are considered convergent if they maintain the same global best particle in 50 continuous iterations. The offloading results may be different with the same configurations in an experiment. Therefore, the system energy consumption is measured by the average of 50 repeated experiments.

5.3 RQ1. SPSO-GA with different deadline constraints

Fig. 7 depicts the system energy consumption of different offloading strategies for one execution of DNN per IoT device with different deadline constraints. In general, the system energy consumption decreases as the deadline becomes looser by using SPSO-GA, PSO-GA, and GA. This is because the strategies based on meta-heuristic algorithms tend to allocate more layers to the energy-efficient servers when the deadline is not strict. Moreover, SPSO-GA and PSO-GA can achieve the same performance with different deadline constraints. The results also show that the system energy consumption for each type of DNNs is different. This is because they consist of different DNN layers with various data transmissions between layers. For example, the number of layers in AlexNet is more than the others, and thus more computing energy consumption is consumed by AlexNet. For GoogleNet, the computing energy consumption is less than that of other DNNs. This is because there are more layers that can be executed in parallel, and thus more layers



(a) AlexNet. (b) VGG19.

(c) GoogleNet. (d) ResNet101.

Fig. 7. System energy consumption of different offloading strategies for one execution of DNN per IoT device with different deadline constraints.

can be executed on servers with less energy consumption within the same deadline.

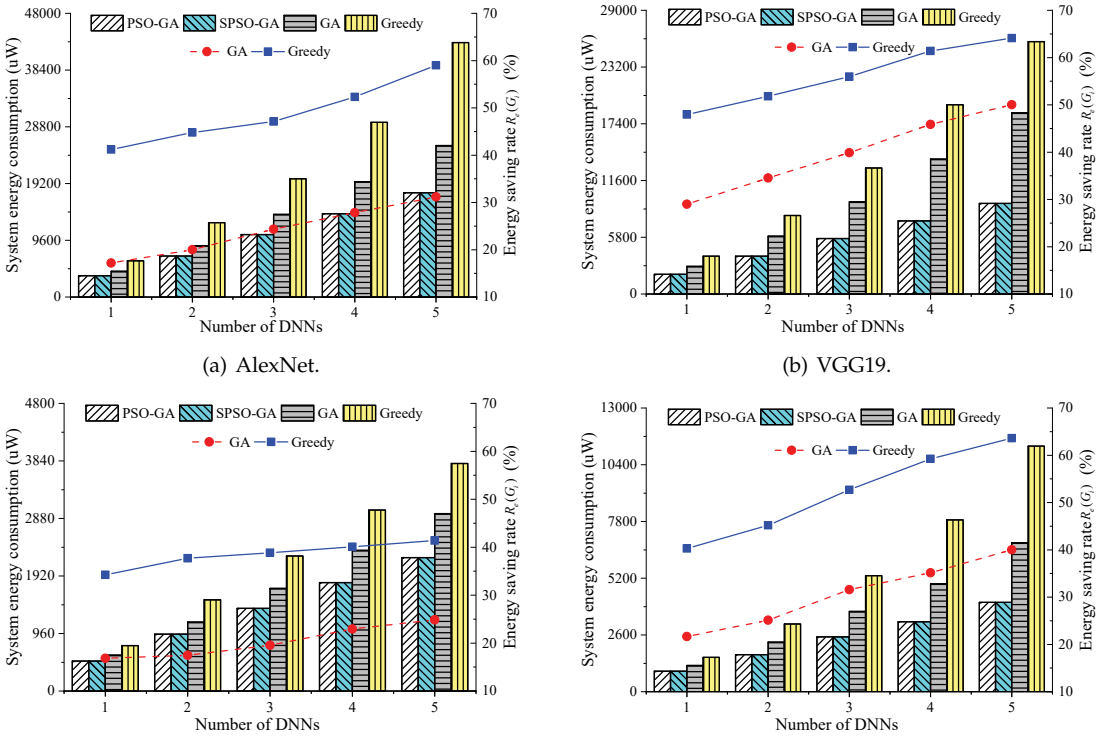
SPSO-GA and PSO-GA can adaptively adjust the search ability according to the current situation and evolve iteratively from a global perspective, which both perform best for all types of DNNs. Greedy only considers IoT devices and edge servers when offloading layers. It is a fixed strategy and only goes one way until it fails to meet deadline constraints. Therefore, the Greedy can obtain a feasible solution for all types of DNNs and consume the same system energy consumption with different deadline constraints. By contrast, the performance of GA is greatly influenced by deadline constraints. This is because its search scope is local at each iteration. As for the proposed SPSO-GA and PSO-GA, they can save around 12.91%~20.97% and 31.55%~47.98% compared with GA and Greedy, respectively.

Fig. 7(d) depicts the system energy consumption of different strategies for one execution of ResNet101 per IoT device, which is less than that in Figs. 7(a) and 7(b). This is because the calculation amount of layers in ResNet101 is less than that of AlexNet and VGG19 and thus leads to less computing energy consumption. In Fig. 7(c), the performance of SPSO-GA is a little better than GA with D_1. By using SPSO-GA, the layers can be offloaded to suitable servers from a global perspective even with strict deadline constraints.

5.4 RQ2. SPSO-GA with different workloads

Fig. 8 depicts the system energy consumption of different offloading strategies for different numbers of executing DNN per IoT device with a fixed deadline constraint (*i.e.* D_3). The bars depict the system energy consumption of different offloading strategies with various workloads, while the curves depict the energy saving rate $R_e(\mathrm{G}_i)$. $R_e(\mathrm{G}_i) = -(e_t(\mathrm{S_p}) - e_t(\mathrm{G}_i))/e_t(\mathrm{G}_i), \mathrm{G}_i = \{\mathrm{G_a}, \mathrm{G_r}\}$, where $\mathrm{S_p}$, $\mathrm{G_a}$, and $\mathrm{G_r}$ are the offloading strategies based on PSO-GA, GA, and Greedy, respectively. $e_t(\mathrm{A})$ is the system energy consumption by using the algorithm A. Generally, SPSO-GA and PSO-GA achieve better offloading performance than the other two strategies. The system energy consumption of different offloading strategies increases as the workloads become heavier. This is because more servers are needed to execute the increasing DNN layers, which results in more system energy consumption.

As shown in Fig. 8, the system energy consumption of different offloading strategies grows as the number of executing DNN per IoT device increases. This is because the offloading strategies need to handle more layers with more servers, which results in more system energy consumption. In general, the results of the system energy consumption in Fig. 8 are similar to that in Fig. 7. Moreover, DNN type significantly affects the system energy consumption of the offloading strategies. Specifically, as the number of executing DNN per IoT device increases, the energy saving rate by using GA and Greedy grows significantly for AlexNet, VGG19, and ResNet101 but not obviously for GoogleNet.



(a) AlexNet. (b) VGG19.

(c) GoogleNet. (d) ResNet101.

Fig. 8. System energy consumption of different offloading strategies for different numbers of executing DNN per IoT device with a fixed deadline constraint.

In addition, the parallel structure of GoogleNet causes the growth rate of the system energy consumption by using PSO-GA and SPSO-GA to be almost the same as that by using GA and Greedy. By contrast, the serial structures of AlexNet, VGG19, and ResNet101 make the performance advantages more obvious by using PSO-GA and SPSO-GA. As for the proposed SPSO-GA, they can save around 21.17%~36.52% and 40.92%~57.04% compared with GA and Greedy, respectively.

5.5 RQ3. Impact of varying the MEC resource supply

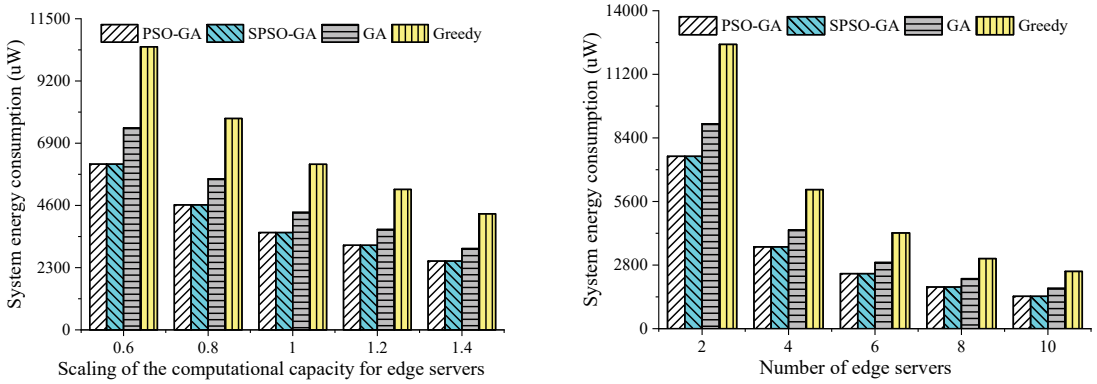
Fig. 9 depicts the system energy consumption of different offloading strategies for one execution of AlexNet per IoT device at D_3 with varying MEC resources. The impact of varying supplies of MEC resources on offloading results is analyzed from two aspects: the computational capacities of edge servers and the number of edge servers.

Fig. 9(a) shows the system energy consumption of different offloading strategies for one execution of AlexNet per IoT device at D_3 with varying computational capacities of edge servers. The scaling ratios of computational capacities for 4 edge servers are set to $\{0.6, 0.8, 1, 1.2, 1.4\}$. When the scaling ratio is set to 1, the offloading results are the same as that at D_3 in Fig. 7(a). In general, the system energy consumption decreases as the computational capacities of edge servers become stronger. This is because the offloading strategies tend to allocate more DNN layers to the edge rather than the cloud when edge servers are equipped with more computational resources, which leads to the reduction of energy consumption in cloud servers. Both SPSO-GA and PSO-GA perform best for all kinds of edge servers with different computational capacities. Specifically, they can reduce the system energy consumption by 15.43%~17.88% compared with the GA. Similarly, SPSO-GA and PSO-GA can reduce the system energy consumption by about 28% compared with the Greedy.

Fig. 9(b) shows the system energy consumption of different offloading strategies for one execution of AlexNet per IoT device at D_3 with varying numbers of edge servers, which are set to $\{2, 4, 6, 8, 10\}$. The other configurations are kept the same as in Subsection 5.1. The detailed information about the configurations of edge servers can be found in our GitHub project. In general, the system energy consumption decreases as the number of edge servers increases. When the number of edge servers is small, the offloading strategies tend to deploy more DNN layers in the cloud for satisfying deadline constraints. This results in more system energy consumption. Otherwise, the offloading strategies tend to deploy more DNN layers in the edge rather than the cloud, which leads to less system energy consumption. Specially, both SPSO-GA and PSO-GA perform best for different numbers of edge servers. The performance of the GA is better than that of the Greedy.

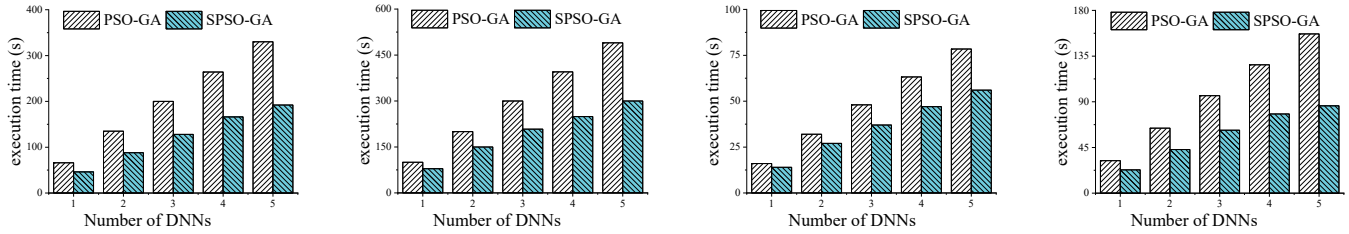
5.6 RQ4. Execution time of SPSO-GA and PSO-GA

The results of Sections 5.3 and 5.4 indicate that the layer partition operations have seldom negative impact on the proposed SPSO-GA for optimizing the system energy consumption. The layer partition is able to lessen the number of deployment units in a DNN and thus improve the execution time of the proposed algorithm. Fig. 10 depicts the execution time of SPSO-GA and PSO-GA with different workloads.



(a) Varying the computational capacity of edge servers. (b) Varying the number of edge servers.

Fig. 9. System energy consumption of different offloading strategies for one execution of AlexNet per IoT device at D_3 with varying MEC resource.



(a) AlexNet. (b) VGG19. (c) GoogleNet. (d) ResNet101.

Fig. 10. Execution time of SPSO-GA and PSO-GA with different workloads.

There are different numbers of executing DNN per IoT device during a period of time, whose value is between 1 to 5. In general, the execution time of SPSO-GA is much less than the PSO-GA on AlexNet, VGG19, and ResNet101, whose structures are serial. This is because DNNs with serial structures have more dimensionality reductions, which results in less deployment units after layer partition. The number of deployment units determines the problem encoding space and the execution time of the algorithms. Therefore, the execution time of SPSO-GA is much less than PSO-GA on AlexNet, VGG19, and ResNet101.

Fig. 10(a) depicts the execution time of SPSO-GA and PSO-GA with different workloads for AlexNet. Specifically, the execution time of SPSO-GA is around 36.01% less than PSO-GA on average. As shown in Figs. 10(b) to 10(d), the execution time of SPSO-GA is around 30.48%, 21.05%, and 35.96% less than PSO-GA on average for VGG19, GoogleNet, and ResNet101, respectively. After layer partition, GoogleNet still remains many deployment units, and thus the execution time of SPSO-GA cannot be well reduced.

6 Discussion

Some issues about applicability are discussed as follows.

Impact of transmission speed on the performance. Data transmission speed is positively-correlated with bandwidth. When the bandwidth between IoT devices and the cloud increases, IoT devices tend to offload more DNN layers to the cloud. Due to the increase of data transmission speed, the latency of data transmission between IoT devices and the cloud decreases, which would better satisfy the deadline constraints for the DNN-based Smart IoT systems. Similarly, the increase of bandwidth between IoT devices and the edge has positive impact on the offloading results for the DNN-based Smart IoT systems.

Time complexity. In each iteration of the proposed SPSO-GA algorithm, all particles are updated and their fitness is calculated. The number of calculations required to update the state of particles is determined by the initial population size Γ and the particle dimension D. The total number of deployment units in DNNs y determines the complexity of the fitness function based on Algorithm 2. Since $D = y$ in the proposed offloading strategy, the time complexity of SPSO-GA is $O(\Gamma \cdot y^2)$ per iteration. In addition, the convergence time is influenced by the total number of deployment units in DNNs y and the number of servers $|\boldsymbol{C}|$. The overall framework of the PSO-GA and GA is the same as SPSO-GA, and thus the time complexity of the PSO-GA and GA should be $O(\Gamma \cdot y^2)$ per iteration. Note that y in PSO-GA and GA is equal to the number of DNN layers before layer partition operations. Obviously, the time complexity of the Greedy is $O(y \cdot |\boldsymbol{C}|)$.

7 Conclusion

In this paper, we have proposed an energy-efficient offloading strategy based on SPSO-GA for DNN-based smart IoT systems in the cloud-edge environments, which aims to reduce the system energy consumption while satisfying the deadline constraints. The extensive simulation experiments show that the proposed strategy can achieve the superior performance than other classic methods. The system energy

consumption is reduced as the deadline becomes looser by using SPSO-GA, PSO-GA, and GA. With the looser deadline constraints, more layers tend to be allocated to the more energy-efficient servers in the same situation. However, the performance of GA is greatly influenced by the deadline constraints due to its local search scope at each iteration.

In our future work, we will consider more complicated scenarios with the environmental fluctuation including network delay, bandwidth fluctuation, and server failure, and investigate the adaptiveness of the proposed offloading strategy. Moreover, we plan to apply deep reinforcement learning algorithm to offloading decision on DNN-based smart IoT systems.

ACKNOWLEDGMENTS

This work is partly supported by the Natural Science Foundation of China under Grant No. 62072108, the Natural Science Foundation of Fujian Province for Distinguished Young Scholar No. 2020J06014, the Natural Science Foundation of Fujian Province under Grant No. 2019J01286, and the Young and Middle-aged Teacher Education Foundation of Fujian Province under Grant No. JT180098.

REFERENCES

[1] H. Li, K. Ota, and M. Dong, "Learning IoT in edge: Deep learning for the internet of things with edge computing," *IEEE Network*, vol. 32, no. 1, pp. 96–101, 2018.

[2] Y. You, Z. Zhang, C. Hsieh, J. Demmel, and K. Keutzer, "Fast deep neural network training on distributed systems and cloud TPUs," *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 11, pp. 2449–2462, 2019.

[3] C. Hu, W. Bao, D. Wang, and F. Liu, "Dynamic adaptive DNN surgery for inference acceleration on the edge," in *IEEE International Conference on Computer Communications (INFOCOM)*, 2019.

[4] 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 *the 22nd International Conference on Architectural Support for Programming Languages and Operating Systems*, 2017.

[5] H. Badri, T. Bahreini, D. Grosu, and K. Yang, "Energy-aware application placement in mobile edge computing: A stochastic optimization approach," *IEEE Transactions on Parallel and Distributed Systems*, vol. 31, no. 4, pp. 909–922, 2020.

[6] B. Lin, Y. Huang, J. Zhang, J. Hu, X. Chen, and J. Li, "Cost-driven off-loading for DNN-based applications over cloud, edge, and end devices," *IEEE Transactions on Industrial Informatics*, vol. 16, no. 8, pp. 5456–5466, 2020.

[7] M. Chen, S. Guo, K. Liu, X. Liao, and B. Xiao, "Robust computation offloading and resource scheduling in cloudlet-based mobile cloud computing," *IEEE Transactions on Mobile Computing*, vol. 20, no. 5, pp. 2025–2040, 2021.

[8] M. Altamimi, A. Abdrabou, K. Naik, and A. Nayak, "Energy cost models of smartphones for task offloading to the cloud," *IEEE Transactions on Emerging Topics in Computing*, vol. 3, no. 3, pp. 384–398, 2015.

[9] K. Elgazzar, P. Martin, and H. S. Hassanein, "Cloud-assisted computation offloading to support mobile services," *IEEE Transactions on Cloud Computing*, vol. 4, no. 3, pp. 279–292, 2016.

[10] Z. Fang, T. Yu, O. J. Mengshoel, and R. K. Gupta, "Qos-aware scheduling of heterogeneous servers for inference in deep neural networks," in *International Conference on Information and Knowledge Management (ICIKM)*, 2017.

[11] K. Kumar and Y. Lu, "Cloud computing for mobile users: Can offloading computation save energy?" *Computer*, vol. 43, no. 4, pp. 51–56, 2010.

[12] H. J. Jeong, "Lightweight offloading system for mobile edge computing," in *IEEE International Conference on Pervasive Computing and Communications Workshops (ICPCCW)*, 2019.

[13] M. Chen and Y. Hao, "Task offloading for mobile edge computing in software defined ultra-dense network," *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 3, pp. 587–597, 2018.

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

[15] Z. Ali, L. Jiao, T. Baker, G. Abbas, Z. H. Abbas, and S. Khaf, "A deep learning approach for energy efficient computational offloading in mobile edge computing," *IEEE Access*, vol. 7, pp. 149 623–149 633, 2019.

[16] 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 *IEEE International Conference on Computer Design (ICCD)*, 2017.

[17] 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 *International Conference on Distributed Computing Systems (ICDCS)*, 2018.

[18] L. Yang, J. Cao, S. Tang, D. Han, and N. Suri, "Run time application repartitioning in dynamic mobile cloud environments," *IEEE Transactions on Cloud Computing*, vol. 4, no. 3, pp. 336–348, 2016.

[19] T. Mohammed, C. Joe-Wong, R. Babbar, and M. D. Francesco, "Distributed inference acceleration with adaptive DNN partitioning and offloading," in *IEEE International Conference on Computer Communications (INFOCOM)*, 2020.

[20] H. Wu, W. J. Knottenbelt, and K. Wolter, "An efficient application partitioning algorithm in mobile environments," *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 7, pp. 1464–1480, 2019.

[21] S. Teerapittayanon, B. McDanel, and H. Kung, "Distributed deep neural networks over the cloud, the edge and end devices," in *International Conference on Distributed Computing Systems (ICDCS)*, 2017.

[22] H. Lin, S. Zeadally, Z. Chen, H. Labiod, and L. Wang, "A survey on computation offloading modeling for edge computing," *Journal of Network and Computer Applications*, vol. 169, pp. 1–25, 2020.

[23] C. Hu, W. Bao, D. Wang, and F. Liu, "Dynamic adaptive DNN surgery for inference acceleration on the edge," in *IEEE International Conference on Computer Communications (INFOCOM)*, 2019.

[24] 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 Communications Magazine*, vol. 54, pp. 29–35, 2016.

[25] G. Xie, G. Zeng, X. Xiao, R. Li, and K. Li, "Energy-efficient scheduling algorithms for real-time parallel applications on heterogeneous distributed embedded systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 12, pp. 3426–3442, 2017.

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

[27] D. S. Hochba, "Approximation algorithms for NP-hard problems," *SIGACT News*, vol. 28, pp. 40–52, 1997.

[28] B. Lin, F. Zhu, J. Zhang, J. Chen, X. Chen, N. N. Xiong, and J. Lloret Mauri, "A time-driven data placement strategy for a scientific workflow combining edge computing and cloud computing," *IEEE Transactions on Industrial Informatics*, vol. 15, no. 7, pp. 4254–4265, 2019.

[29] 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 Transactions on Services Computing*, vol. 11, no. 4, pp. 727–739, 2018.

[30] 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 Journal of Computers*, vol. 37, pp. 445–456, 2014.

[31] M. A. Rodriguez and R. Buyya, "Deadline based resource provisioningand scheduling algorithm for scientific workflows on clouds," *IEEE Transactions on Cloud Computing*, vol. 2, no. 2, pp. 222–235, 2014.

[32] H. Li, D. Yang, W. Su, J. LÃij, and X. Yu, "An overall distribution particle swarm optimization MPPT algorithm for photovoltaic system under partial shading," *IEEE Transactions on Industrial Electronics*, vol. 66, no. 1, pp. 265–275, 2019.

[33] D. O'Neill, A. Lensen, B. Xue, and M. Zhang, "Particle swarm optimisation for feature selection and weighting in high-dimensional clustering," in *IEEE Congress on Evolutionary Computation (ICEC)*, 2018.

[34] M. Masdari, F. Salehi, M. Jalali, and M. Bidaki, "A survey of PSO-based scheduling algorithms in cloud computing," *Journal of Network and Systems Management*, vol. 25, pp. 122–158, 2017.
[35] T. R. Chavan and A. V. Nandedkar, "A hybrid deep neural network for online learning," in *the 9th International Conference on Advances in Pattern Recognition (ICAPR)*, 2017.
[36] X. Chen, M. Li, H. Zhong, Y. Ma, and C.-H. Hsu, "DNNOff: Offloading dnn-based intelligent iot applications in mobile edge computing," *IEEE Transactions on Industrial Informatics*, pp. 1–1, 2021.
[37] H. Topcuoglu, S. Hariri, and Min-You Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 13, no. 3, pp. 260–274, 2002.
[38] Y. Shi and R. Eberhart, "A modified particle swarm optimizer," in *IEEE International Conference on Evolutionary Computation Proceedings (ICECP)*, 1998.

Xing Chen received the B.S. degree and the Ph.D. degree from Peking University, in 2008 and 2013, respectively. Upon completion of the Ph.D. degree, he joined Fuzhou University and has held the rank of Professor since 2020. 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. Dr. Chen's research focuses on 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. He has published over fifty journal and conference articles, and was awarded two First Class Prizes for Provincial Scientific and Technological Progress, separately in 2018 and 2020.

Jianshan Zhang received his M.S. degree in Materials Engineering from Fujian Normal University, China, in 2018. He is currently working toward the PhD degree at the College of Mathematics and Computer Science, Fuzhou University. He has also been a part of the Fujian Key Laboratory of Network Computing and Intelligent Information Processing at Fuzhou University since September 2019. His current research interests include edge computing, computational intelligence and cloud computing.

Bing Lin received the B.S. and M.S degrees in Computer Science from Fuzhou University, Fuzhou, China, in 2010 and 2013, respectively, and the Ph.D. degree in Communication and Information System from Fuzhou University in 2016. He is currently an associate professor with the College of Physics and Energy at Fujian Normal University. Now he is the deputy director of the Department of Energy and Materials, and leads the Intelligent Computing research group. His research interest mainly includes parallel and distributed computing, computational intelligence, and data center resource management. He has published over twenty journals and conference articles, such as IEEE Transactions on Industrial Informatics, and IEEE Transactions on Network and Service Management.

Zheyi Chen is currently a Ph.D. candidate in computer science at the University of Exeter. He received his M.Sc. degree in Computer Science from Tsinghua University, China, in 2017, and B.Sc. degree in Computer Science from Shanxi University, China, in 2014. His research interests include cloud computing, mobile edge computing, deep learning, and resource optimization.

Katinka Wolter received her Ph.D degree from Technische Universität Berlin in 1999. She has been Assistant professor at Humboldt-University Berlin and lecturer at Newcastle University before joining Freie Universität Berlin as a professor for dependable systems in 2012. Her research interests are model-based evaluation and improvement of dependability, security and performance of distributed systems and networks. Special systems of interest to her are cryptocurrencies, data streaming systems and car traffic networks.

Geyong Min is a Professor of High Performance Computing and Networking in the Department of Computer Science within the College of Engineering, Mathematics and Physical Sciences at the University of Exeter, United Kingdom. He received the Ph.D. degree in Computing Science from the University of Glasgow, United Kingdom, in 2003, and the B.Sc. degree in Computer Science from Huazhong University of Science and Technology, China, in 1995. His research interests include Computer Networks, Wireless Communications, Parallel and Distributed Computing, Ubiquitous Computing, Multimedia Systems, Modelling and Performance Engineering.