Mathematical Biosciences and Engineering

http://www.aimspress.com/journal/mbe

MBE, 20(2): 2793–2814.
DOI: 10.3934/mbe.2023131
Received: 11 October 2022
Revised: 08 November 2022
Accepted: 16 November 2022
Published: 01 December 2022

Research article

An adaptive offloading framework for license plate detection in collaborative edge and cloud computing

Hong Zhang[1], Penghai Wang[1,*], Shouhua Zhang[2] and Zihan Wu[1]

[1] School of Cyber Security and Computer, Hebei University, Baoding, Hebei, China
[2] Information Technology and Electrical Engineering, University of Oulu, Oulu, Finland

* **Correspondence:** Email: wph@stumail.hbu.edu.cn.

Abstract: With the explosive growth of edge computing, huge amounts of data are being generated in billions of edge devices. It is really difficult to balance detection efficiency and detection accuracy at the same time for object detection on multiple edge devices. However, there are few studies to investigate and improve the collaboration between cloud computing and edge computing considering realistic challenges, such as limited computation capacities, network congestion and long latency. To tackle these challenges, we propose a new multi-model license plate detection hybrid methodology with the tradeoff between efficiency and accuracy to process the tasks of license plate detection at the edge nodes and the cloud server. We also design a new probability-based offloading initialization algorithm that not only obtains reasonable initial solutions but also facilitates the accuracy of license plate detection. In addition, we introduce an adaptive offloading framework by gravitational genetic searching algorithm (GGSA), which can comprehensively consider influential factors such as license plate detection time, queuing time, energy consumption, image quality, and accuracy. GGSA is helpful for Quality-of-Service (QoS) enhancement. Extensive experiments show that our proposed GGSA offloading framework exhibits good performance in collaborative edge and cloud computing of license plate detection compared with other methods. It demonstrate that when compared with traditional all tasks are executed on the cloud server (AC), the offloading effect of GGSA can be improved by 50.31%. Besides, the offloading framework has strong portability when making real-time offloading decisions.

Keywords: task offloading; edge computing; internet of things; resource allocation

1. Introduction

With the rapid development of deep learning [1], a large number of complex applications often rely on high-end hardware, which is difficult for resource-constrained edge devices to handle. Although the emergence of 5G technique can immigrate tasks from edge devices to central servers more quickly

to mitigate this issue, problems such as network instability, transferring delay, and load imbalance have become new challenges [2]. Compared with traditional centralized cloud computing, exploring a collaborative edge and cloud computing by taking both efficiency and accuracy into account can effectively reduce load imbalance and system latency.

For tens of thousands of surveillance cameras in smart cities, it is really tough to achieve real-time license plate detection and recognition [3]. Currently, many license plate detection systems rely on server-grade hardware with unlimited resources to solve it [4]. However, cost reduction and energy conservation are of great importance in energy-constrained edge environments.

The development of edge computing has made task offloading gradually become popular [5]. In general, there are two computation offloading models: binary offloading [6–10] and partial offloading [11–15]. In the binary offloading model, the task can only be computed by an edge device or a cloud server as a whole. Partial offloading splits a task into two parts and partially executes one of them on an edge device or an cloud device. Selecting binary or partial offloading exactly rests with the requirements of tasks. For powerful central servers, complex deep learning models are more suitable to achieve high accuracy. Unfortunately, there still exists a challenge to balance the load and effectiveness between edge computing and cloud computing. Moreover, due to the high cost of GPUs, it is impossible to equip GPUs for all edge devices. Therefore, a single-model license plate detection scheme is not efficient enough in the environment of edge computing.

For the purpose of effectively mitigating the conflict between energy consumption and execution efficiency of different devices, we propose a brand-new multi-model license plate detection binary offloading framework to cope with the problem. The multi-model license plate detection framework takes full advantage of the fast detection speed of traditional machine learning model and the high detection accuracy of deep learning model. It is conducive to deploy traditional machine learning models on edge nodes with less resource consumption and low latency. Benefit from the deep learning model deployed in the cloud server, the accuracy of license plate detection is guaranteed.

A new criterion of QoS is identified, including task execution time, network delay, task queuing time, energy consumption, etc. In order to minimize the transmission time including task uploading time and results downloading time, it is necessary to optimize the initial offload decision. Therefore, the offloading scheme proposed in this paper introduces a novel task initial assignment scheme in combination with the size, brightness and other characteristics of the image.

In addition, the offloading decision generated by our proposed GGSA can improve the overall service quality. It can also reduce the time and energy consumption of license plate detection. In a word, this paper proposes an adaptive offloading framework for license plate detection in collaborative edge and cloud computing, which can locate the position of license plates in real-time for tens of thousands of surveillance cameras in smart cities with minimal cost. The main contributions of this paper can be summarized as follows:

- A new multi-model license plate detection scheme is applied to meet the requirements of real-time and accuracy at the same time. It can effectively mitigate issues such as network instability, transferring delay, and load imbalance.
- We propose a new probability-based offloading initialization assignment scheme, which can achieve near-optimal offloading decisions with a smaller number of iterations.
- We design a novel GGSA to generate the most appropriate offloading decisions with a minimum weighted sum of time and energy consumption.

The rest of this paper is organized as follows: Section II provides a review of related work. We present a multi-model license plate detection model in Section III. A new initial offloading assignment scheme and an offloading scheme called GGSA are proposed in Section IV. Section V shows parameter settings, data sets, and experiments in details. Conclusions and future work are summarized in Section VI.

2. Related works

In recent years, computation offloading can improve computational capabilities and provide better latency [16]. However, computation offloading problems are often considered as constrained optimization problems, which are NP-hard [17]. Scholars have investigated a variety of architectures and offloading policies to solve it. Saeik et al. [18] provided a detailed survey of how the Edge and Cloud can be combined together to facilitate the task offloading problem.

On the one hand, deep learning methods have been proposed to deal with the offloading problem [19]. In [20], Huang et al. [20] proposed a distributed deep learning-based offloading (DDLO) algorithm for MEC networks. They adopted a shared replay memory to store newly generated offloading decisions. And they also investigated a Deep Reinforcement learning-based Online Offloading (DROO) framework in [21]. This framework can learn the binary offloading decisions from the experience in the wireless powered mobile edge computing networks. An autonomous computation offloading framework in [22] is proposed to address some challenges related to time-intensive and resource-intensive applications. And Mao et al. [23] researched a deep-learning-based offloading policy optimization strategy. Their policy considered the long short-term memory model to address the dynamics of energy harvesting performance. Wu et al. [17] proposed a distributed deep learning-driven task offloading (DDTO) algorithm. The hybrid offloading model can effectively generate near-optimal offloading decisions in edge and cloud computing environments.

On the other hand, heuristic optimization algorithms showed great advantages in many studies. In [24], Kuang et al. [24] formalized an offloading decision problem in such environment with multiple users, multiple offloading points, and structured tasks as a cost-minimization problem. And they designed an improved genetic algorithm (GA) to solve the minimization problem under constraints. Liao et al. [25] formulated a distributed offloading strategy based on a binary-coded GA. Peng et al. [26] proposed three constrained multi-objective evolutionary algorithms (CMOEAs) for solving IoT-enabled computation offloading problems in collaborative edge and cloud computing networks. Xu et al. [27] researched non-dominated sorting genetic algorithm III (NSGA-III) to address the multi-objective optimization problem in IoT. Bi et al. [28] formulated a genetic simulated annealing-based particle (GSP) swarm optimization hybrid algorithm to produce a close-to-optimal solution. Alfakih et al. [29] presented a dynamic task scheduling and load-balancing technique based on an integrated accelerated particle swarm optimization (APSO) algorithm with dynamic programming as a multi-objective.

In the past few decades, some weakly supervised learning and semi-supervised learning methods show advantages in the field of object detection,such as [30–32]. License plate detection has been a valuable topic in the field of object detection [33]. Many algorithms study license plate detection in terms of both traditional machine learning and deep learning. Yuan et al. [33] located license plates

by their shape and the aspect ratio of them. A novel technique to detect license plates regardless of their color, size, and content was proposed in [34]. A deep end-to-end model to localize and recognize Pakistani license plates with non-uniform and non-standardized sizes, fonts, and styles was introduce in [35].

Many research efforts like [36–40] have been devoted to energy-efficient offloading in mobile cloud computing. However, few studies have explored the offloading of multi-model object detection tasks in collaborative edge and cloud computing. For real-time detection, the images generated by terminals have to be processed locally or nearby to avoid the high latency and the performance bottleneck of servers in a centralized data center. However, for countless terminal cameras in a city, equipping GPUs to them is impractical and outweighs the gain. The motivation of this paper is to find the optimal approximation cost of energy and delay of such problems.

3. System model and problem formulation

In this work, we consider one central cloud server and multiple edge devices in collaborative edge and cloud computing. A large number of license plate detection tasks are captured by surveillance cameras and transmitted to nearby edge devices. Then, the edge devices transmit the specific information of each license plate detection task to the cloud server for offloading decisions, such as the size of the data, and the brightness of each picture.

Figure 1 shows the framework of multi-model license plate detection in collaborative edge and cloud computing. The cloud server collects the license plate detection picture information from the edge device and uploads it to the server. Then, the cloud server generates the appropriate offloading decision for edge devices and the cloud server through the offloading framework. Depending on the offloading decision, different images will be processed on different devices.

Due to the limited computing power, we employ a traditional machine learning model to detect license plates for edge devices. Firstly, we let the input image be transformed into an enhanced image by Gaussian filtering and median filtering. Then, we extract the color and edge features from filtered image and Sobel image. We form preliminary candidate images based on these features and morphological approaches. And a binary edge image is generated by thresholding each pixel adaptively in the fused image. Then, closed image is formed by reducing noise and removing small background structures. Finally, the candidate license plate is selected by combining the aspect ratio of the plate and the difference between the text and the background pixels. Therefore, the license plate candidates are selected by the machine learning model.

In addition, an improved Retina license plate detection deep learning model was deployed on cloud server. This model includes the feature pyramid network, the context head module, and the cascade multi-task loss. The feature pyramid network outputs three feature maps at different scales based on the input images. Then, the context head module obtains a feature map of a particular scale and calculates the multi-task loss with cascade regression to locate license plates. The Retina license plate detection model is an improved Chinese open source license plate detection model based on RetinaFace [41].

Edge devices are denoted by a set $N = \{1, 2, ..., n\}$. Each edge device has multiple independent license plate detection tasks processed locally or offloaded to the cloud server. License plate detection tasks are denoted by a set $M = \{1, 2, ..., m\}$. Each task has a corresponding offloading decision denoted by $O_{n,m} \in \{0, 1\}$. $O_{n,m} = 0$ denotes that task m on edge device n is executed locally, whereas $O_{n,m} = 1$

denotes that the task m on edge device n is offloaded to the cloud server for execution. Notations used in this paper are described in Table 1. The detailed operations of edge computing and cloud computing are illustrated as follows.

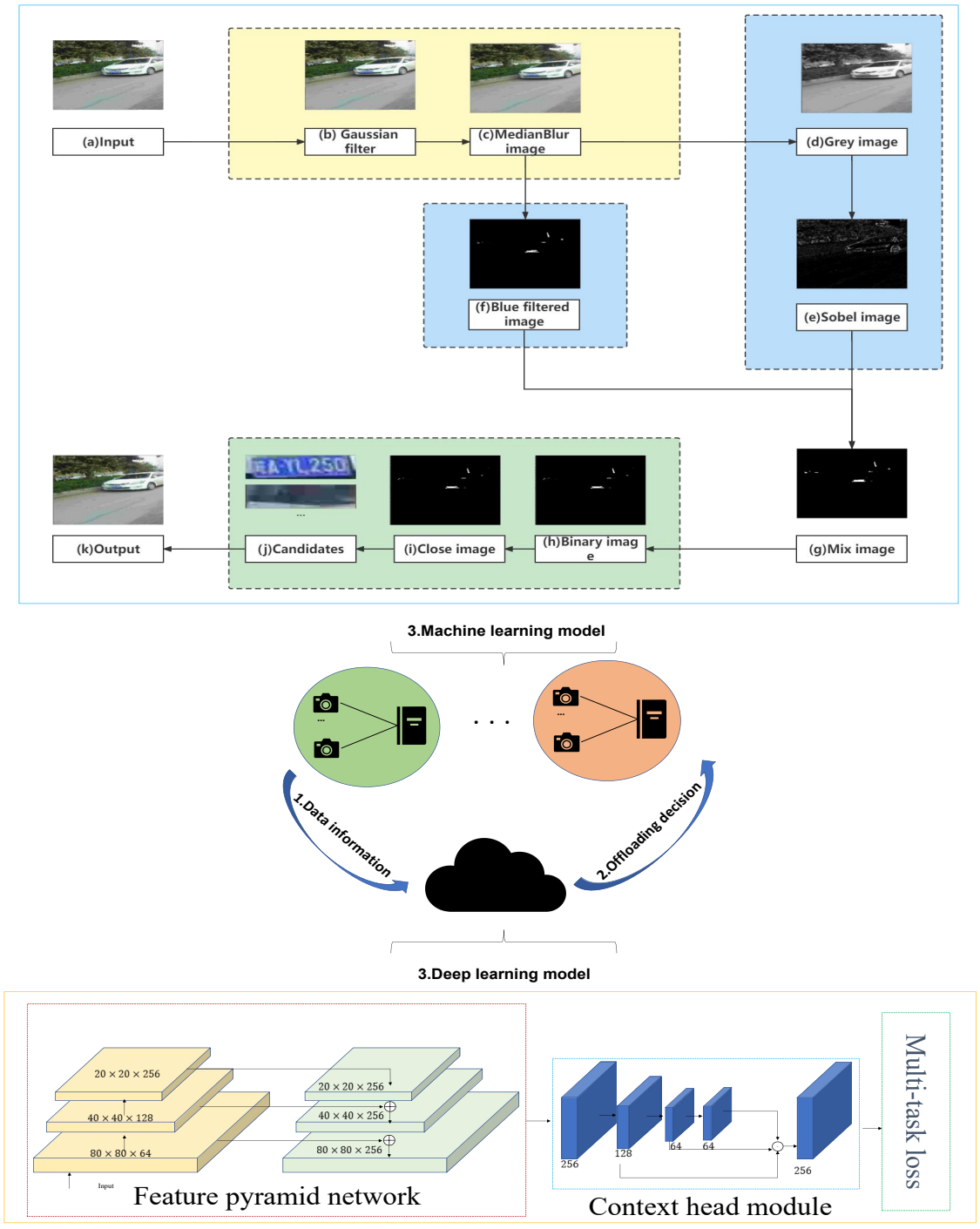

Figure 1. Framework of multi-model license plate detection in collaborative edge and cloud computing.

Table 1. Notations used in this paper

Notation	Definition
N	The set of edge devices
M	The set of tasks on a edge device
$d^s_{n,m}(B)$	The size of picture task m on edge device n.
$d^b_{n,m}$	The brightness of picture task m on edge device n.
$O_{n,m}$	The offloading decision of task m on edge device n.
$T_e(ms)$	The total time cost of task m on edge device n.
$T^{e,p}_{n,m}(ms)$	The execution time of task m on edge device n.
$T^{e,q}_{n,m}(ms)$	Queuing time on edge device n before processing the task m.
$f_m(hz)$	The CPU cycles per bit of data processed by machine learning models on edge devices.
$f_n(hz)$	The CPU cycles of edge devices.
$t^w_{n,i}(ms)$	The average waiting time of the edge device n before processing the task i .
$E_e(J)$	The total energy cost of task m on edge device n.
$E^{e,p}_{n,m}(J)$	The energy consumption of task m on edge device n.
$E^{e,q}_{n,m}(J)$	Queuing energy consumption before processing the task m on edge device n.
$P^p_n(W)$	The average power consumed by processing tasks of edge device n.
$P^q_n(W)$	The average queue CPU power consumed on edge device n.
$T_c(ms)$	The total time cost of cloud server.
$T^{c,p}_{n,m}(ms)$	The time of cloud server processes task m on edge device n.
$T^{e,q}_{n,m}(ms)$	Queuing time before the cloud server processing the task m on edge device n.
$f_d(hz)$	The CPU cycles per bit of data processed by deep learning models on cloud server.
$f_c(hz)$	The CPU cycles of cloud server.
$t^w_{n,o}(ms)$	The average waiting time of cloud server before processing the task o on edge device n.
$R_{n,m}(\%)$	The transmission rate of task m on edge device n.
$P_n(W)$	Signal power of edge device n.
$H_{n,c}$	The channel gains of edge device n to cloud server.
$\theta^2(dBm)$	The receiver noise power from edge device n to the cloud server.
$E_c(J)$	The total energy cost of the cloud server.
$E^{c,p}_{n,m}(J)$	The energy cost of the cloud server when executing task m .
$E^{c,q}_{n,m}(J)$	Queuing energy consumed by cloud server before processing task m on edge device n.
$E^{c,q}_{n,m}(J)$	Energy consumed during transmission for task m from edge device n to the cloud server.
$P^c_n(W)$	The average power used by CPU processing tasks of cloud server.
$P^q_c(W)$	The average queue CPU power used of the cloud server.
$P^t_{n,c}(W)$	The average power during transmission from edge device n to cloud server.

3.1. Edge computing

When the offloading decision is $O_{n,m} = 0$, which means task m is executed on edge device n locally. We model the time cost of this task in Eqs (3.1)–(3.3).

$$T_e = T^{e,p}_{n,m} + T^{e,q}_{n,m} \tag{3.1}$$

$$T^{e,p}_{n,m} = \frac{d^s_{n,m} \cdot d^b_{n,m} \cdot f_m}{f_n} \tag{3.2}$$

$$T^{e,q}_{n,m} = \sum_{i=1}^{m-1} t^w_{n,i} \tag{3.3}$$

where $T^{e,p}_{n,m}$ is the execution time of task m on edge device n locally. f_m is the CPU cycles per bit of data processed by the local model. We note f_n as CPU cycles of edge devices n. The execution time of a picture is not only related to the size of the picture data $d^s_{n,m}$ but also affected by the brightness distribution of the picture $d^b_{n,m}$. $T^{e,q}_{n,m}$ represents the time of the picture task m queuing on edge device n. To calculate its queue time, we evaluate the wait time $t^w_{n,i}$ of tasks ahead on edge devices n.

After that, we calculate the total energy cost for task on edge device as:

$$E_e = E^{e,p}_{n,m} + E^{e,q}_{n,m} \tag{3.4}$$

$$E^{e,p}_{n,m} = T^{e,p}_{n,m} \cdot P^p_n \tag{3.5}$$

$$E^{e,q}_{n,m} = T^{e,q}_{n,m} \cdot P^q_n \tag{3.6}$$

where, $E^{e,p}_{n,m}$ is the energy consuming of task m on edge device n locally. $E^{e,q}_{n,m}$ represents the energy cost of the task m queuing on edge device n. P^p_n and P^q_n express the power used by CPU processing tasks and the energy queued in unit time, respectively.

3.2. Cloud computing

The license plate detection task is assigned to the cloud server only when the offloading decision is $O_{n,m} = 1$. In this scenario, it is necessary to consider not only the cost of cloud server, but also the cost of network transmission. Therefore, we denote the total time cost for edge device n offload task m to the cloud server in Eq (3.7):

$$T_c = T^{c,p}_{n,m} + T^{c,q}_{n,m} + T^t_{n,m} \tag{3.7}$$

where, $T^{c,p}_{n,m}$ is the time cost of processing tasks m on the cloud server transmitted from the edge device n. When multiple tasks are offloaded to the cloud server, the queuing cost of the subsequent tasks is expressed by $T^{c,q}_{n,m}$. Meanwhile, the network delay of transmission is denoted by $T^t_{n,m}$.

$$T^{c,p}_{n,m} = \frac{d^s_{n,m} \cdot d^b_{n,m} \cdot f_d}{f_c} \tag{3.8}$$

$$T^{c,q}_{n,m} = \sum_{i=1}^{n} \sum_{i=1}^{o-1} t^w_{n,o} \tag{3.9}$$

$$R_{n,m} = \frac{W}{N} \cdot \log_2\left(1 + \frac{P_n \cdot H_{n,c}}{\theta^2}\right) \tag{3.10}$$

$$T^t_{n,m} = \frac{d^s_{n,m}}{R_{n,m}} \tag{3.11}$$

where, f_d is the CPU cycles per bit of data processed by the deep learning model. $d^s_{n,m}$ and $d^b_{n,m}$ are the data size and brightness distribution of picture task m, respectively. f_c denotes CPU cycles of cloud server. And $T^{c,q}_{n,m}$ represents the time of the m picture task during the queue on cloud server. $t^w_{n,o}$ is the waiting time for tasks offloaded to the cloud server. In addition, we denote W as channel bandwidth, P_n as signal power, $H_{n,c}$ as the channel gains, and θ^2 as the receiver noise power.

Additionally, we make an evaluation model for the total energy cost on cloud server. It consists of three parts, which are the energy used by the cloud server to process the task $E^{c,p}_{n,m}$, the energy consumed during task queuing $E^{c,q}_{n,m}$, and the energy consumed by the transmission task $E^t_{n,m}$. P^p_c represents the power used per unit time on cloud server. We denote P^q_c and $P^t_{n,c}$ as the power used per unit time of queue and transmission, respectively. The formulas are shown in Eqs (3.12)–(3.15).

$$E_c = E^{c,p}_{n,m} + E^{c,q}_{n,m} + E^t_{n,m} \tag{3.12}$$

$$E^{c,p}_{n,m} = d^s_{n,m} \cdot P^p_c \tag{3.13}$$

$$E^{c,q}_{n,m} = T^{c,q}_{n,m} \cdot P^q_c \tag{3.14}$$

$$E^t_{n,m} = T^t_{n,m} \cdot P^t_{n,c} \tag{3.15}$$

3.3. Problem formulation

The authors explore the criterion of QoS Q for the collaborative edge computing and cloud computing, which is comprehensive consideration of energy consumption and task efficiency [20]. An mixed-integer optimization problem is formulated to minimize Q in Eq (3.16).

$$OP : \min Q = \sum_{n=1}^{N}(\sum_{m=1}^{M}(\alpha E_e + \beta E_c) + \max\{T_e, T_c\}) \tag{3.16}$$

where, $\alpha = 1 - O_{n,m}$, $\beta = O_{n,m}$. In collaborative edge and cloud computing environment, the energy consumption of edge devices and cloud servers is cumulative. Since different devices can perform tasks simultaneously, the time cost is only considered the maximum one.

4. Offloading algorithm

Since OP in Eq (3.16) is a NP-hard problem, it is indeed necessary to solve it with an approximation algorithm efficiently in tolerable time. The proposed heuristic algorithm can respond quickly and find the approximate optimal solution in limited iterations.

4.1. Probability-based offloading initialization algorithm

A large amount of heuristics initial solutions are usually generated randomly within the permitted ranges [42]. However, it takes a great deal of time to search the approximate optimal solution. Therefore, it is really essential to generate suitable initial solutions to provide better convergence efficiency. Here, we propose a probability-based offloading initialization algorithm to conquer it.

The pseudo-code of our algorithm is presented in Algorithm 1 (Table 2). We denote the set of edge devices as N, the set of tasks on an edge device as M, the brightness of picture task m on edge device n as $d_{n,m}$. Firstly, the process of data probability initialization is shown from line 1 to 6. We normalize all the brightness of pictures on different edge devices in line 3. And according to the normalization of the brightness of picture, the picture's normal distribution $data_{probability}$ is calculated in line 4. The result of the normal distribution $data_{probability}$ is the basis for the offloading probability of a picture. Secondly, line 7 to 14 is the process of offloading decision making. Line 7 is to generate an array of initialization all-zero offloading decisions. And the number of tasks that can be offloaded by each edge device relies on the number of edge devices N and the number of tasks M in line 8. At last, we choose the task with the highest probability in $data_{probability}$ to offload in line 10 to 11. Line 12 is to free the assigned offloading task from $data_{probability}$.

Table 2. Probability-based offloading initialization algorithm pseudo-codes.

Algorithm 1: Probability-based offloading initialization algorithm

Input:
the set of edge devices N,
the set of tasks on a edge device$M$,
the brightness of edge device n task m picture $d_{n,m}$

Output:
$data_{offload}$

1: **for** n in range N **do**
2: **for** m in range M **do**
3: $d_{n,m} \leftarrow d_{n,m} \,/\, \sum_{i=1}^{M} d_{n,m}$
4: $data_{probability} \leftarrow \frac{1}{\sqrt{2\pi}} e^{\left(-\frac{(d_{n,m})^2}{2}\right)}$
5: **end for**
6: **end for**
7: $data_{offload} \leftarrow \sum_{i=1}^{N} \sum_{i=1}^{M} = 0$
8: $offload_{count} \leftarrow \lceil \frac{M}{N} \rceil$
9: **for** i in range $offload_{count}$ **do**
10: $offload^{i}_{index} \leftarrow data_{index} \max(data_{probability})$
11: $data_{offload} \leftarrow offload^{i}_{index} = 1$
12: $pop(\max(data_{probability}))$
13: **end for**
14: **return** $data_{offload}$

4.2. GGSA

To find the optimal approximate solution, the offloading decision generated by Algorithm 1 has to iterate several times for obtaining a converged solution. Traditional heuristic algorithms are poor efficient to handle this issue. Thus, we investigate a new GGSA in Algorithm 2 (Table 3), which shows better performance.

Table 3. GGSA pseudo-codes.

Algorithm 2: GGSA

Input:

The offloading decision array $offload_{states}$, a dictionary memory structure $memory = \{q, offload\}$, initialization v, G, K, $iter = 0$, maximum number of iterations $iter_{max}$

Output:

$final_{offload}, final_q$

while $iter < iter_{max}$ **do**
  $fitness \leftarrow Eva(offload_{states}, data_{probability}, data_{size})$
  if $\min(fitness_q) < memory_q$ **then**
    $memory_q \leftarrow \min(fitness_q)$
    $memory_{offload} \leftarrow data_{offload}[fitness_{index}]$
    $iter \leftarrow 0$
  else
    $Mass \leftarrow calculateMass(fitness)$
    $topK \leftarrow findTopK(fitness, K)$
    $G \leftarrow G * exp(-20 * iter/iter_{max})$
    $acceleration \leftarrow cal(fitnesses, Mass, G, topK)$
    for i in $range(len(fitness))$ **do**
      $index_v \leftarrow int((v + acceleration)\%M)$
      $index_{topK} \leftarrow randint(0, len(topK) - 1)$
      $offload_{newstates} \leftarrow offload_{states}[i][: index_v] + offload_{states}[index_{topK}][index_v :]$
    end for
    $iter \leftarrow iter + 1$
  end if
  $offload_{states} \leftarrow offload_{newstates}$
end while
$final_{offload} \leftarrow memory_{offload}$
$final_q \leftarrow memory_q$
return $final_{offload}, final_q$

For the purpose of fast convergence, we simultaneously employ Algorithm 1 to generate X sets of initial offloading decisions. Then, we store them into an array represented by $offload_{states}$. And a dictionary memory structure is used to store minimum offloading decisions with $memory_{offload}$ and corresponding quality of service with $memory_q$. The initialization velocity is denoted by v, and G is the gravitational parameters. K is the top k best values. Furthermore, $iter_{max}$ is the number of iterations

that $memory_q$ does not change anymore, which is the threshold to terminate the iteration. The final output is the optimal approximate solution, which includes final offloading decisions $final_{offload}$ and final QoS $final_q$.

GGSA consists of three parts. Line 1 to 7 is to update the minimum offloading decision as the first part. Lines 8 to 11 as the second part is the process of gravitational search. And the third part is lines 12 to 16, which is used for genetic search. In detail, $Eva()$ is to evaluate the Q of the $offload_{states}$ in line 2. If the Q of the offloading decision in $fitness$ is less than that in $memory_q$, we update $memory_q$ and the corresponding offloading decision $memory_{offload}$ in line 3 to 5. Then, we calculate $Mass, topK, G, acceleration$ in line 8 to 11. After that, we make the offloading decision in the corresponding $offload_{states}[i][: index_v]$ according to $index_v$. We concatenate it with the selected $offload_{states}[index_{topK}][index_v :]$ into a new offloading decision $offload_{newstates}$ in line 13 to 19. In the end, the final optimal results are obtained by $final_{offload}$ and $final_q$ from $memory_{offload}$ and $memory_q$, when $iter_{max}$ is satisfied in line 21 to 22.

4.3. Theoretical analysis of adaptive offloading framework

The implementation of the adaptive offloading framework relies on the combination of probability-based offloading initialization algorithm and GGSA. The quality of the initial offloading decision can have an impact on the performance of the heuristic search algorithm. A good initial offloading decision will be useful to speed up the heuristic search process. The probability-based offloading initialization algorithm makes a preliminary decision on the offloading position of the image based on the characteristics of the image, such as brightness, size, etc. Based on the above offloading decision, the GGSA can perform the search for the approximate optimal solution faster. Then, the final offloading decision of GGSA is sent back to each edge device and cloud server for license plate detection.

Thus, an adaptive offloading framework for license plate detection in collaborative edge and cloud is formed. The framework takes full advantage of the initial offloading decision and the heuristic search algorithm to approach the optimal approximate solution.

5. Performance evaluation

In this section, we conduct numerical simulations to evaluate the performance of proposed algorithm. We set the number of edge devices $N = [3, 5, 7, 9]$ and the number of tasks on an edge device $M = [100, 300, 500, 700, 900]$. We set the CPU rate of the edge device as 10×10^7 cycle/s. The cloud server CPU rate is 100 times as much as the edge device. In addition, We assign W as 20 Mhz [43]. The channel gains H_{nc} are generated using a distance-dependent pathloss model given as $L[dB] = 140.7 + 36.7log_{10}d[km]$ [44]. P_n is set to 20 dBm and the background noise θ^2 is $\theta^2 = -100$ dBm [43] . $iter_{max}$ is 10, as the threshold for stopping the iteration. And in order to avoid random chance of the experiment, each group of experiments is conducted ten times.

5.1. Datasets

The data size $d^s_{n,m}$ and brightness $d^b_{n,m}$ of each license plate detection image are obtained from the Chinese City Parking Dataset (CCPD) [45]. CCPD collects data from roadside parking in all the streets of one provincial capital in China where residuals own millions of cars.

5.2. Evaluation indicators

Q: Comprehensive consideration of energy consumption and task efficiency. Q value of different tasks on different edge devices.

Performance improvement: The performance improvement of GGSA compared with other algorithms of different tasks on different edge devices.

Average energy consumption: Average energy consumption of different tasks on different edge devices.

Number of iterations: Number of iterations on different edge devices with different heuristic algorithms.

Time of iterations: Time of iterations on different edge devices with different heuristic algorithms.

5.3. Algorithm comparison

We compare the performance of our proposed GGSA method with other offloading algorithms, specifically, the Simulated Annealing (SA) algorithm [46] and the GA [47]. An improved particle swarm optimization based algorithm (PSO) is also used for comparison [48]. And AE means that all tasks are processed on edge devices locally, whereas AC means all tasks are executed on the cloud server.

Figure 2 demonstrates the Q value of 100 tasks computed on different edge devices. And Table 4 is the performance improvement of GGSA compared with other algorithms for different edge devices. In general, the Q value of GGSA is better than other algorithms. With the increase of the edge device number, the Q value gap between GGSA and AC gradually increases. When the number of edge device is 3, the Q value of GGSA is 46.19% of AC. Furthermore, GGSA is 32.79% of AC as the number of device increases to 9. And the GGSA is 61.02% better on average than that of AC. The reason is that the transmission delay and server load skyrocketing increase when all tasks are offloaded to the cloud server. Besides, GGSA is 17.66% better than SA when edge device is 3. As the edge device number increases to 9, GGSA is 21.34% better than SA. The Q value of GGSA is on average 19.54% better than GA. In general, the Q values of the GA, SA, PSO algorithm are very close. The performance of GGSA improved by 20.75 and 21.08% over SA and PSO, respectively. The reason is that they cannot further improve the performance during the search process. In a word, the efficiency of GGSA is significantly improved than other offloading algorithms. The average energy consumption of 100 tasks on different edge devices is shown in Figure 3. The average energy consumption of GGSA is the lowest on different edge devices. Due to the limited transmission rate of edge devices, the average energy consumption of AC is the highest among these algorithms. In detail, the average energy consumption of GGSA is 17.21% better than GA and 61.56% better than AC. The average energy consumption of AE, SA, PSO are also higher than that of GGSA. GGSA improved by 37.65% over AE. Compared to SA and PSO, GGSA improved by 19.30 and 17.40%, respectively.

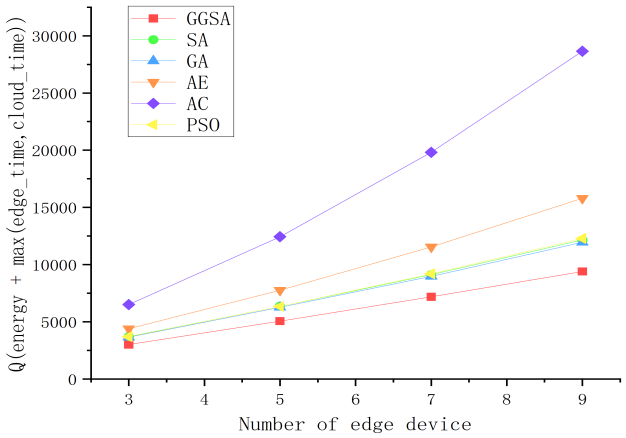

Figure 2. Q value of 100 tasks on different edge devices.

Table 4. The performance improvement of GGSA compared with other algorithms under different edge devices.

Algorithms	3	5	7	9	Average improvement rate
SA	18.89%	20.27%	21.14%	22.68%	20.75%
GA	17.66%	19.31%	19.83%	21.34%	19.54%
AE	31.58%	34.68%	37.76%	40.49%	36.13%
AC	53.81%	59.29%	63.75%	67.21%	61.02%
PSO	18.73%	20.15%	21.91%	23.55%	21.08%

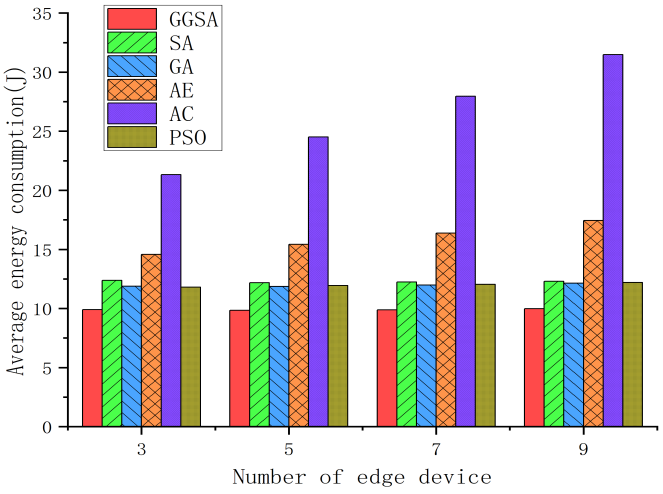

Figure 3. Average energy consumption of 100 tasks on different edge devices.

We compare the number of iterations on different devices in Figure 4. Since the offloading decisions of AE and AC are deterministic, they do not need to iterate to find the optimal offloading decision. The average number of iterations of GGSA is 12.75, which performs best among these algorithms. The number of iterations of SA is the most on different edge devices among these algorithms. The number of iterations of GA decreases gradually with the number of devices, while the number of iterations of PSO increases gradually. In general, the number of iterations of the other algorithms is higher than that of GGSA. In addition, Figure 5 is the time of total iterations on different edge devices. The average iteration time of GGSA is 25.37 ms. However, the average iteration time of SA is 3.36 times as much as GGSA. While the average iteration of PSO is 1.94 times longer than that of GGSA. The average iteration time of GA also takes 32.06 ms, which is 6.69 ms slower than GGSA. Fewer iterations and fast convergence indicate the superiority of GGSA offloading framework, which means GGSA algorithm is faster for searching the approximate optimal offloading solutions.

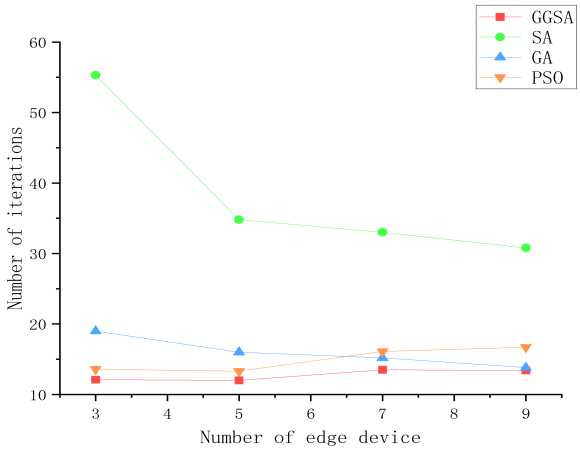

Figure 4. Number of iterations on different edge devices.

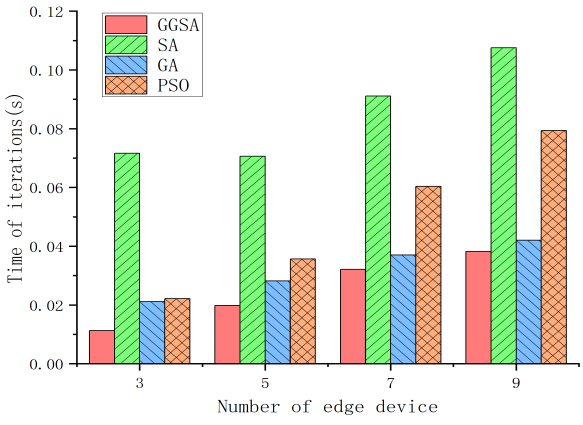

Figure 5. Time of iterations on different edge devices.

Figure 6 shows the Q value of different tasks on three edge devices. As we can see, GGSA always

has the lowest Q value in these different algorithms. The Q of AE is higher than AC as the tasks on edge devices. The reason is that the time for the edge device to process the task is better than cloud server. So the task queue time on edge device increases. When the number of tasks on the edge device is 900, GGSA is 61.41% better than AE. SA, GA, PSO have similar Q values for different tasks. The performance of GGSA is better than that of them. From Table 5, Q of GGSA compared with SA reduced by an average of 18.01%. And GGSA reduced by an average of 17.07% than GA. GGSA also outperformed PSO by 17.34. Additionally, the performance of GGSA is significantly better than that of AE and AC. GGSA is 48.76 and 50.31% better than AE and AC, respectively.

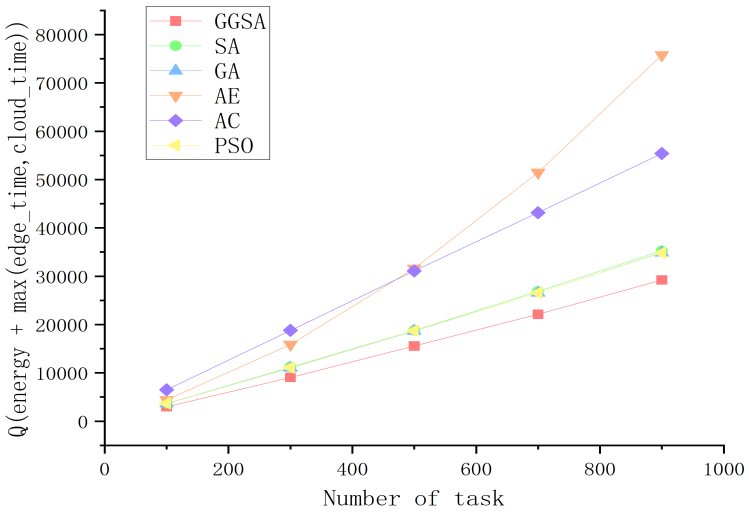

Figure 6. Q value of different tasks on three edge devices.

Table 5. The performance improvement of GGSA compared with other algorithms under different tasks.

Algorithms	100	300	500	700	900	Average improvement rate
SA	18.89%	19.22%	17.22%	17.62%	17.06%	18.01%
GA	17.66%	18.11%	16.71%	16.74%	16.14%	17.07%
AE	31.58%	42.98%	50.83%	57.01%	61.41%	48.76%
AC	53.81%	51.80%	49.99%	48.72%	47.23%	50.31%
PSO	18.73%	18.34%	16.86%	16.72%	16.05%	17.34%

The average energy consumption of different tasks on three edge devices is illustrated in Figure 7. Generally, the average energy consumption of GGSA is the lowest among these algorithms. Due to the CPU processing power of edge devices are limited, the average energy consumption of AE increases as the number of tasks increases. Moreover, the average energy consumption of GGSA is 15.14 and 57.26% better than GA and AE, respectively. And the average energy consumption of GGSA was also 16.88, 15.24 and 50.47% better than SA, PSO and AC, respectively.

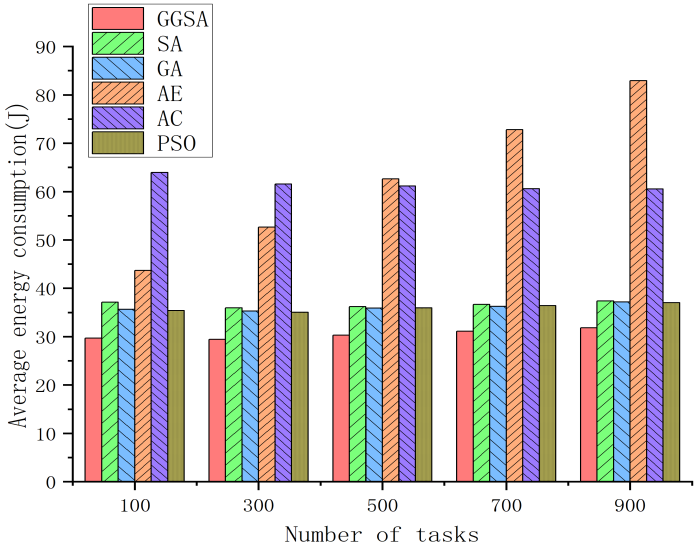

Figure 7. Average energy consumption of different tasks on three edge devices.

In Figure 8, it reveals the effect of different tasks on the number of iterations. It is obvious that the average number of iterations for GGSA is less than that for GA and SA. The average number of iteration of GGSA is 14.64. However, the average number of iterations of GA and SA are 17.48 and 31.36, respectively. And the average number of iterations of pso is 15.74. The time of iteration also shows the same trend in Figure 9. The average time of iterations of GGSA is 75.59 ms. While GGSA is 13 and 72% better than GA and SA, respectively. And GGSA is 36% better than PSO.

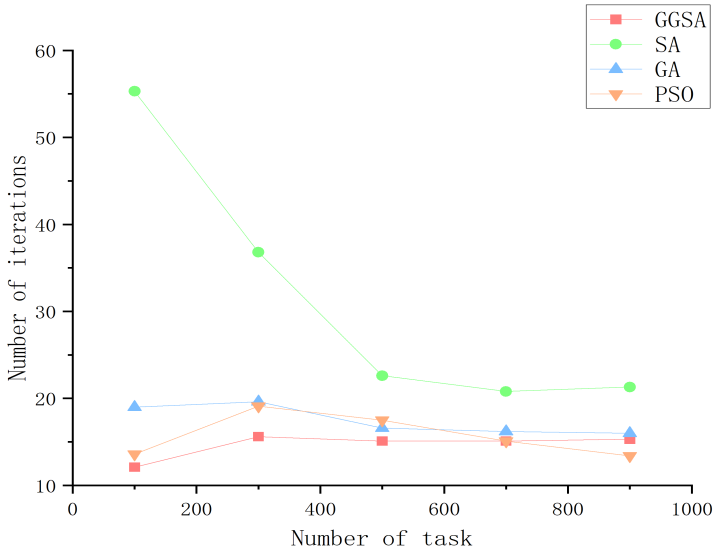

Figure 8. Number of iterations on three edge devices with different tasks.

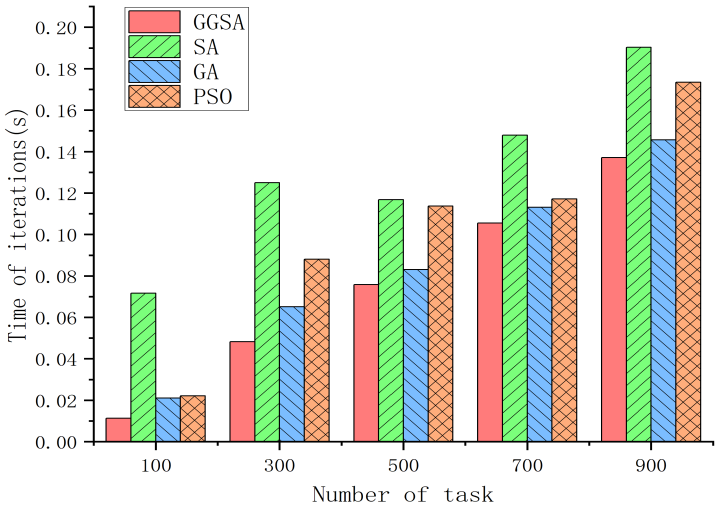

Figure 9. Time of iterations on three edge devices with different tasks.

The average accuracy of different algorithms is shown in Figure 10. AC has the highest license plate detection accuracy of 97.53%. The reason is that all license plate images are detected with a deep learning model on a powerful gpu. While the average detection rate of AE was only 76.69%. This is mainly due to the low detection efficiency of traditional machine learning models on edge devices. In addition, the average accuracy of license plate detection for SA, GA and PSO is 85.45, 86.61 and 85.84% respectively. However, the average license plate detection accuracy of GGSA is 96.02%.

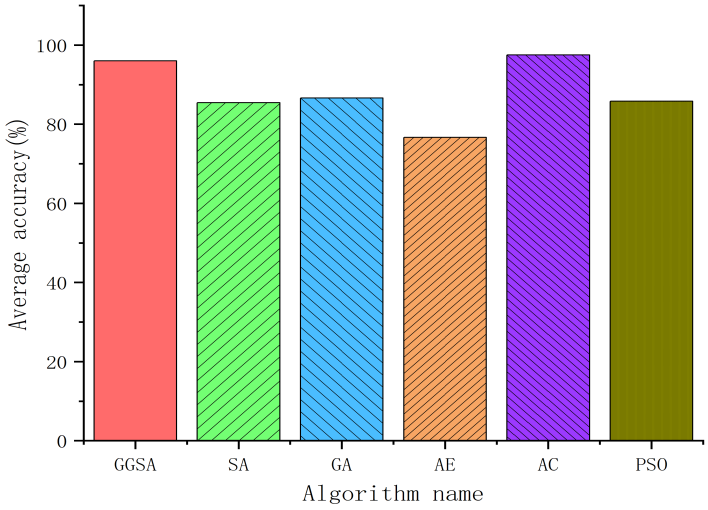

Figure 10. Average accuracy of different algorithms.

It is easily come to the conclusion that GGSA can get close to the optimal solution in the least

amount of time among these algorithms. GGSA offloading framework shows better performance in collaborative edge and cloud computing compared with other methods. Extensive experiments have shown that this adaptive framework for license plate detection in collaborative edge and cloud computing not only ensures the speed of detection, but also the accuracy of detection.

6. Conclusions

This paper proposes an adaptive offloading framework for license plate detection in collaborative edge and cloud computing. On the one hand, we propose a new multi-model license plate detection scheme based on network delay, task queue number, task execution time, and energy consumption, etc. On the other hand, we investigate a refreshing probability-based offloading initialization algorithm to solve the problem of imbalanced initial offloading decisions. We also introduce a new heuristic algorithm called GGSA to solve the contradiction between energy consumption and execution efficiency. Simulation results demonstrated the priority of the proposed method in terms of minimizing the weighted sum of energy consumption and time cost. This framework can be used in collaborative edge and cloud computing to meet realistic challenges, such as limited computation capacities, network congestion and long latency.

There are some rooms for improvement in our multi-model license plate detection scheme and adaptive offloading framework. For example, better multi-models can be constructed to replace the current ones. At present, the scope of application of this offloading framework is relatively narrow. In the future, we will apply this framework to other object detection tasks in edge computing, such as face detection. Furthermore, we will study more complex environments and more powerful offloading algorithms in collaborative edge and cloud computing.

Acknowledgments

This work is supported by Science and Technology Research Project of Hebei Higher Education Institutions (No. QN2020133), the Natural Science Foundation of Hebei Province of China (No. F2019201361).

Conflict of interest

The authors declare there is no conflict of interest.

References

1. I. Goodfellow, Y. Bengio, A. Courville, *Deep learning*, MIT press, Cambridge, 2016.
2. H. Xue, B. Huang, M. Qin, H. Zhou, H. Yang, Edge computing for internet of things: A survey, in *2020 International Conferences on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData) and IEEE Congress on Cybermatics (Cybermatics)*, 2020, 755–760. https://doi.org/10.1109/iThings-GreenCom-CPSCom-SmartData-Cybermatics50389.2020.00130

3. J. Shashirangana, H. Padmasiri, D. Meedeniya, C. Perera, Automated license plate recognition: a survey on methods and techniques, *IEEE Access*, **9** (2020), 11203–11225. https://doi.org/10.1109/ACCESS.2020.3047929
4. H. Padmasiri, J. Shashirangana, D. Meedeniya, O. Rana, C. Perera, Automated license plate recognition for resource-constrained environments, *Sensors*, **22** (2022), 1434. https://doi.org/10.3390/s22041434
5. M. Diamanti, P. Charatsaris, E. E. Tsiropoulou, S. Papavassiliou, Incentive mechanism and resource allocation for edge-fog networks driven by multi-dimensional contract and game theories, *IEEE Open J. Commun. Soc.*, **3** (2022), 435–452. https://doi.org/10.1109/OJCOMS.2022.3154536
6. X. Chen, H. Xu, G. Zhang, Y. Chen, R. Li, Unsupervised deep learning for binary offloading in mobile edge computation network, *Wirel. Pers. Commun.*, **124** (2022), 1841–1860. https://doi.org/10.1007/s11277-021-09433-9
7. R. Chen, X. Wang, Maximization of value of service for mobile collaborative computing through situation aware task offloading, *IEEE. Trans. Mob. Comput.*, **99** (2021), 1–1. https://doi.org/10.1109/TMC.2021.3086687
8. Y. Fu, X. Yang, P. Yang, A. K. Y. Wong, Z. Shi, H. Wang, et al., Energy-efficient offloading and resource allocation for mobile edge computing enabled mission-critical internet-of-things systems, *EURASIP J. Wirel. Commun. Networking*, **26** (2021), 1–16. https://doi.org/10.1186/s13638-021-01905-7
9. X. Li, L. Huang, H. Wang, S. Bi, Y. A. Zhang, An integrated optimization-learning framework for online combinatorial computation offloading in mec networks, *IEEE Wirel. Commun.*, **29** (2022), 170–177. https://doi.org/10.1109/MWC.201.2100155
10. C. Jiang, Y. Li, J. Su, Q. Chen, Research on new edge computing network architecture and task offloading strategy for internet of things, *Wirel. Networks*, (2021), 1–13. https://doi.org/10.1007/s11276-020-02516-8
11. Y. Mao, C. You, J. Zhang, K. Huang, K. B. Letaief, A survey on mobile edge computing: The communication perspective, *IEEE Commun. Surv. Tutor.*, **19** (2017), 2322–2358. https://doi.org/10.1109/COMST.2017.2745201
12. D. Xu, H. Zhu, Legitimate surveillance of suspicious computation offloading in mobile edge computing networks, *IEEE Trans. Commun.*, **70** (2022), 2648–2662. https://doi.org/10.1109/TCOMM.2022.3151767
13. V. D. Tuong, T. P. Truong, T. Nguyen, W. Noh, S. Cho, Partial computation offloading in noma-assisted mobile-edge computing systems using deep reinforcement learning, *IEEE Internet Things J.*, **8** (2021), 13196–13208. https://doi.org/10.1109/JIOT.2021.3064995
14. X. Deng, J. Yin, P. Guan, N. N. Xiong, L. Zhang, S. Mumtaz, Intelligent delay-aware partial computing task offloading for multi-user industrial internet of things through edge computing, *IEEE Internet Things J.*, (2021), 1–1. https://doi.org/10.1109/JIOT.2021.3123406
15. J. Baek, G. Kaddoum, Online partial offloading and task scheduling in sdn-fog networks with deep recurrent reinforcement learning, *IEEE Internet Things J.*, **9** (2022), 11578–11589. https://doi.org/10.1109/JIOT.2021.3130474

16. A. Yousafzai, A. Gani, R. M. Noor, A. Naveed, R. W. Ahmad, V. Chang, Computational offloading mechanism for native and android runtime based mobile applications, *J. Syst. Softw.*, **121** (2016), 28–39. https://doi.org/10.1016/j.jss.2016.07.043
17. H. Wu, Z. Zhang, C. Guan, K. Wolter, M. Xu Collaborate edge and cloud computing with distributed deep learning for smart city internet of things, *IEEE Internet Things J.*, **7** (2020), 8099–8110. https://doi.org/10.1109/JIOT.2020.2996784
18. F. Saeik, M. Avgeris, D. Spatharakis, N. Santi, D. Dechouniotis, J. Violos, et al., Task offloading in edge and cloud computing: A survey on mathematical, artificial intelligence and control theory solutions, *IEEE Internet Things J.*, **195** (2021), 108177. https://doi.org/10.1016/j.comnet.2021.108177
19. F. Wang, M. Zhang, X. Wang, X. Ma, J. Liu, Deep learning for edge computing applications: A state-of-the-art survey, *IEEE Access*, **8** (2020), 58322–58336. https://doi.org/10.1109/ACCESS.2020.2982411
20. L. Huang, X. Feng, A. Feng, Y. Huang, L. P. Qian, Distributed deep learning-based offloading for mobile edge computing networks, *Mobile Networks Appl.*, **27** (2022), 1123–1130. https://doi.org/10.1007/s11036-018-1177-x
21. L. Huang, S. Bi, Y. A. Zhang, Deep reinforcement learning for online computation offloading in wireless powered mobile-edge computing networks, *IEEE. Trans. Mob. Comput.*, **19** (2019), 2581–2593. https://doi.org/10.1109/TMC.2019.2928811
22. A. Shakarami, A. Shahidinejad, M. Ghobaei-Arani, An autonomous computation offloading strategy in mobile edge computing: A deep learning-based hybrid approach, *J. Network Comput. Appl.*, **178** (2021), 102974. https://doi.org/10.1016/j.jnca.2021.102974
23. B. Mao, F. Tang, Y. Kawamoto, N. Kato, Optimizing computation offloading in satellite-uav-served 6G IoT: A deep learning approach, *IEEE Network*, **35** (2021), 102–108. https://doi.org/10.1109/MNET.011.2100097
24. L. Kuang, T. Gong, S. OuYang, H. Gao, S. Deng, Offloading decision methods for multiple users with structured tasks in edge computing for smart cities, *Future Gener. Comput. Syst.*, **105** (2020), 717–729. https://doi.org/10.1016/j.future.2019.12.039
25. Z. Liao, J. Peng, B. Xiong, J. Huang, Adaptive offloading in mobile-edge computing for ultradense cellular networks based on genetic algorithm, *J. Cloud Comput.*, **10** (2021), 1–16. https://doi.org/10.1186/s13677-021-00232-y
26. G. Peng, H. Wu, H. Wu, K. Wolter, Constrained multiobjective optimization for iot-enabled computation offloading in collaborative edge and cloud computing, *IEEE Internet Things J.*, **8** (2021), 13723–13736. https://doi.org/10.1109/JIOT.2021.3067732
27. X. Xu, Q. Liu, Y. Luo, K. Peng, X. Zhang, S. Meng, et al., A computation offloading method over big data for iot-enabled cloud-edge computing, *Future Gener. Comput. Syst.*, **95** (2019), 522–533. https://doi.org/10.1016/j.future.2018.12.055
28. J. Bi, H. Yuan, S. Duanmu, M. Zhou, A. Abusorrah, Energy-optimized partial computation offloading in mobile-edge computing with genetic simulated-annealing-based particle swarm optimization, *IEEE Internet Things J.*, **8** (2020), 3774–3785. https://doi.org/10.1109/JIOT.2020.3024223

29. T. Alfakih, M. M. Hassan, M. Al-Razgan, Multi-objective accelerated particle swarm optimization with dynamic programing technique for resource allocation in mobile edge computing, *IEEE Access*, **9** (2021), 167503–167520. https://doi.org/10.1109/ACCESS.2021.3134941

30. M. O. Lawal, Tomato detection based on modified YOLOv3 framework, *Sci. Rep.*, **11** (2021), 1–11. https://doi.org/10.1038/s41598-021-81216-5

31. A. M. Roy, J. Bhaduri, Real-time growth stage detection model for high degree of occultation using densenet-fused YOLOv4, *Comput. Electron. Agric.*, **193** (2022), 106694. https://doi.org/10.1016/j.compag.2022.106694

32. A. M. Roy, R. Bose, J. Bhaduri, A fast accurate fine-grain object detection model based on yolov4 deep neural network, *Neural Comput. Appl.*, **34** (2022), 3895–3921. https://doi.org/10.1007/s00521-021-06651-x

33. Y. Yuan, W. Zou, Y. Zhao, X. Wang, X. Hu, N. Komodakis, A robust and efficient approach to license plate detection, *IEEE Trans. Image Process.*, **26** (2016), 1102–1114. https://doi.org/10.1109/TIP.2016.2631901

34. M. R. Asif, Q. Chun, S. Hussain, M. S. Fareed, S. Khan, Multinational vehicle license plate detection in complex backgrounds, *J. Vis. Commun. Image Represent.*, **46** (2017), 176–186. https://doi.org/10.1016/j.jvcir.2017.03.020

35. U. Yousaf, A. Khan, H. Ali, F. G. Khan, Z. U. Rehman, S. Shah, et al., A deep learning based approach for localization and recognition of pakistani vehicle license plates, *Sensors*, **21** (2021), 76696. https://doi.org/10.3390/s21227696

36. Y. Yang, Y. Gong,Y. Wu, Intelligent reflecting surface aided mobile edge computing with binary offloading: Energy minimization for IoT devices, *IEEE Internet Things J.*, **9** (2022), 12973–12983. https://doi.org/10.1109/JIOT.2022.3173027

37. C. You, Y. Zeng, R. Zhang, K. Huang, Asynchronous mobile-edge computation offloading: Energy-efficient resource management, *IEEE Trans. Wirel. Commun.*, **17** (2018), 7590–7605. https://doi.org/10.1109/TWC.2018.2868710

38. G. Zhang, F. Shen, Z. Liu, Y. Yang, K. Wang, M. Zhou, Femto: Fair and energy-minimized task offloading for fog-enabled iot networks, *IEEE Internet Things J.*, **6** (2018), 4388–4400. https://doi.org/10.1109/JIOT.2018.2887229

39. Y. Chen, N. Zhang, Y. Zhang, X. Chen, W. Wu, X. Shen, Energy efficient dynamic offloading in mobile edge computing for internet of things, *IEEE Trans. Cloud Comput.*, **9** (2019), 1050–1060. https://doi.org/10.1109/TCC.2019.2898657

40. Y. Deng, Z. Chen, X. Yao, S. Hassan, A. M. Ibrahim, Parallel offloading in green and sustainable mobile edge computing for delay-constrained IoT system, *IEEE Trans. Veh. Technol.*, **68** (2019), 12202–12214. https://doi.org/10.1109/TVT.2019.2944926

41. J. Deng, J. Guo, E. Ververas, I. Kotsia, S. Zafeiriou, Retinaface: Single-shot multi-level face localisation in the wild, in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, (2020), 5203–5012. https://doi.org/10.1109/CVPR42600.2020.00525

42. Y. Wu, A survey on population-based meta-heuristic algorithms for motion planning of aircraft, *Swarm Evol. Comput.*, **62** (2021), 100844. https://doi.org/10.1016/j.swevo.2021.100844

43. T. X. Tran, D. Pompili, Joint task offloading and resource allocation for multi-server mobile-edge computing networks, *IEEE Trans. Veh. Technol.*, **68** (2018), 856–868. https://doi.org/10.1109/TVT.2018.2881191

44. X. Chu, D. Lopez-Perez, Y. Yang, F. Gunnarsson, *Heterogeneous cellular networks: Theory, simulation and deployment*, Cambridge University Press, Cambridge, 2013.

45. Z. Xu, W. Yang, A. Meng, N. Lu, H. Huang, C. Ying, et al. Towards end-to-end license plate detection and recognition: A large dataset and baseline, in *European Conference on Computer Vision*, (2018), 255–271. https://doi.org/10.1007/978-3-030-01261-8_16

46. N. Leite, F. Mel´ıcio, A. C. Rosa, A fast simulated annealing algorithm for the examination timetabling problem, *Expert Syst. Appl.*, **122** (2019), 137–151. https://doi.org/10.1016/j.eswa.2018.12.048

47. S. Katoch, S. S. Chauhan, V. Kumar, A review on genetic algorithm: Past, present, and future, *Multimed. Tools Appl.*, **80** (2021), 8091–8126. https://doi.org/10.1007/s11042-020-10139-6

48. E. H. Houssein, A. G. Gad, K. Hussain, P. N. Suganthan, Major advances in particle swarm optimization: theory, analysis, and application, *Swarm Evol. Comput.*, **63** (2021), 100868. https://doi.org/10.1016/j.swevo.2021.100868

AIMS Press