Li *et al. J Wireless Com Network* (2023) 2023:80
https://doi.org/10.1186/s13638-023-02284-x

EURASIP Journal on Wireless Communications and Networking

RESEARCH Open Access

Partitioning multi-layer edge network for neural network collaborative computing

Qiang Li[1], Ming-Tuo Zhou[2*], Tian-Feng Ren[2], Cheng-Bin Jiang[2] and Yong Chen[2]

*Correspondence:
mingtuo.zhou@mail.sim.ac.cn

[1] State Grid Information & Telecommunication Co., Ltd, Beijing, China
[2] Jushri Technologies, Inc., Shanghai, China

Abstract

There is a trend to deploy neural network on edge devices in recent years. While the mainstream of research often concerns with single edge device processing and edge-cloud two-layer neural network collaborative computing, in this paper, we propose partitioning multi-layer edge network for neural network collaborative computing. With the proposed method, sub-models of neural network are deployed on multi-layer edge devices along the communication path from end users to cloud. Firstly, we propose an optimal path selection method to form a neural network collaborative computing path with lowest communication overhead. Secondly, we establish a time-delay optimization mathematical model to evaluate the effects of different partitioning solutions. To find the optimal partition solution, an ordered elitist genetic algorithm (OEGA) is proposed. The experimental results show that, compared with traditional cloud computing, single-device edge computing and edge-cloud collaborative computing, the proposed multi-layer edge network collaborative computing has a smaller runtime delay with limited bandwidth resources, and because of the pipeline computing characteristics, the proposed method has a better response speed when processing large number of requests. Meanwhile, the OEGA algorithm has better performance than conventional methods, and the optimized partitioning method outperforms other methods like random and evenly partition.

Keywords: Neural network, Edge computing, Collaborative computing, Genetic algorithm

1 Introduction

With rapid development of artificial intelligence and Internet-of-things (IoT) applications, more and more neural networks are deployed on clouds. However, when IoT end devices generate great amount of data, and if we send all the data to cloud for processing, it introduces big pressure to current communication networks. To alleviate the problem, edge computing emerged recently. Edge computing may utilize devices of the network edge to process IoT data, so reducing needs to send all data to cloud [1–4]. Meanwhile, it may reduce overall time delay since edge devices locate closer to end user devices where raw data generate. With edge computing, data generated at end devices (such as smart phones, wearables, smart cars, etc.) can be processed by edge devices like base stations, routers, switches, unmanned aerial vehicle (UAV) and so on [5–10].

Due to above mentioned advantages, recently, many researchers focus on deploying neural network models on edge devices. However, deploying neural networks directly on a single-edge device also introduces problems [5, 12]. Usually, an edge device has limited computing resource, therefore a neural network model needs to be compressed to fit the edge device resource. However, this leads to reduced computing accuracy. And for some neural network model, it is difficult to compress, or if being compressed the effect becomes not good enough. One way to alleviate this issue is to partition the neural networks into two sub-models and deploy on an edge device and a cloud for collaborative computing. By setting appropriate partition point, both the computing load of a single edge device and the communication overhead to cloud can be reduced. Therefore, this method may balance the computing complexity and the computing resource of the edge and the cloud. However, this method has not explored available edge devices in networks for further performance improvement.

With collaborative computing in mind, all possible edge devices may be utilized for neural network sub-models processing. As we know, there are a lot of edge devices from end user to a cloud. The edge computing resource does not only vertically distribute along the path from end devices to cloud, but also across various IoT networks. The edge computing resource from different IoT networks can be organized for neural network computing. In this way, the edge devices form an edge network, of which the device computing resources can be flexibly utilized to achieve optimized performance.

However, in current literatures, there are few research about optimized methods of deploying neural networks on edge networks. In this paper, we propose a method partitioning multi-layer edge network for neural network collaborative computing. With the proposed method, a communication path from end device to cloud with optimized communication overhead is determined, and then the neural network is divided into several sub-models, and each sub-model is deployed on an edge device along the communication path. The neural network is so partitioned, that the overall delay, which comprising of the computing delays at all neural network sub-layers, and the communication delays of each layer's output data to the next layer, is minimized.

Our contribution in this paper is threefold as listed below.

1. Firstly, based on ant colony algorithm, we can select a communication path from end user to cloud with least communication overhead, for any end device request of data processing with neural network.
2. Secondly, based on a proposed running time prediction model of neural network, we form a mathematical model of overall time delay for the neural network sub-models collaborative computing. We then propose Ordered Elitist Genetic Algorithm (OEGA) to solve the problem, and achieve optimized solution, by which we determine the optimal partitioning solution to get shortest overall delay.
3. Thirdly, we design experiment to validate the proposed method. The experimental results show that when the communication bandwidth is limited, the portioned multi-layer edge neural network computing has better performance than single-edge computing mode and cloud computing mode. The proposed OEGA has better con-

vergence speed and better convergence value than Simple Genetic Algorithm (SGA) and Elitist Genetic Algorithm (EGA). In addition, the proposed computing method has advantages in batch processing. With increase of the number of requests per unit time, multi-layer edge network computing can achieve better computational efficiency.

The rest of paper is organized as follows. Section 2 discusses the related works. Section 3 analyses the structure and characteristics of multi-layer edge networks, discusses the method of partitioning the neural network and establishes a mathematical model for time-delay optimization, and presents an ordered elitist genetic algorithm to solve the optimization problem. Section 4 shows and analyses experimental results and presents discussions. Section 5 summarizes the work of this paper and prospects the future work.

2 Related works

Many researchers have studied edge computing for neural network. Some of them focus on compressing neural network models. In literatures, researchers mainly study various types of neural network models and propose different compression methods such as pruning and early termination [12–16]. The goal is to achieve smaller models to ensure deployment on edge or terminal devices with limited computing resources. Some other researchers focus on dividing neural networks to realize collaborative computing [17–24]. Among them, [17–21] mainly discuss collaborative computing of two-layer edge-cloud collaborative computing. By selecting appropriate partition points, they reduce the data traffic between edge and cloud, and then to reduce the communication overhead. Paper [22–24] proposes collaborative computing between two end devices. However, this is only a variant of two-layer edge-cloud collaborative computing, which is still not applicable in face of complex multi-device scenarios. Some other researchers combine the above two ideas, that is, compress the model while dividing the neural network, [25–27] process the model through quantification and early termination, and then conduct collaborative calculation through dividing the model. The studies of [22–27] all discuss the two-portion neural network and collaborative computing scenarios, which are simpler than multipoint partitioning and require further design of decision-making for task partitioning in face of more devices collaborative computing. Some scholars investigate how the neural network can be partitioned and deployed among multiple devices and propose the concept of pipeline processing [28, 29]. However, their studies focus on the hardware architecture design, and pay more attention to the neural network training process. There is still a lack of research on neural network inference process and collaborative computing with multi-layer edge devices.

In this paper, we propose collaborative computing with optimized communication overhead and delay for neural networks, while the neural networks are deployed on multi-layer edge networks. Compared with methods in [12–16], we do not compress neural network models so that the model accuracy will not be affected. And unlike two-layer deployment in [17–27], we utilize a few edge-network layers to deploy the neural

networks and may use more computing devices for more flexible and powerful edge computing. Meanwhile, we focus more on edge computing path selection for reduced communication overhead but does not like [28, 29] for hardware architecture design.

3 Methods

3.1 Computing architecture under edge network

In this subsection, we will discuss the characteristics of edge networks, and how to deploy a partitioned neural network on an edge network for collaborative computing.

3.1.1 Edge network architecture

The edge network considered in this research is a network connecting devices to a cloud and generally consists of access devices, gateways, routers, switches, servers, etc. Access devices are nodes that connect end users like Wi-Fi access routers and 4G/5G base stations. Our current networks are full of gateways, routers, switches, and servers. With fast development of hardware, these devices may have good computing power, so they may not only be used to deliver data from network edge to cloud, but also can be used to process data. Therefore, vertically along all the way from network edge to cloud, and horizontally across various vertical IoT networks, these devices can be connected to form an edge network for communication and computing. Proper use of the computing power of these edge devices can greatly improve the computing efficiency and reduce the data throughput in the network.

Figure 1 illustrates an edge network that can access data from end devices and process the data across the network. Between any two edge devices there is at least one communication link [30]. A neural network can be partitioned to sub-models and be deployed on different edge devices with proper communication connection and computing capabilities. By carefully choosing the edge devices and designing the partition points of the neural network, we can manage to get the overall computing work done with shortest time.

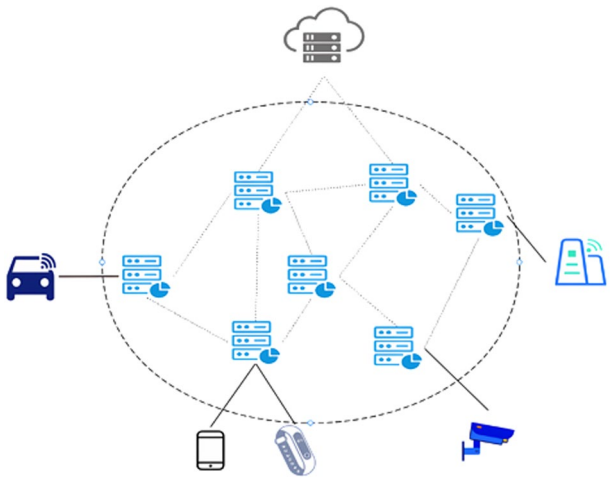

Fig. 1 An example of edge network. An edge network can access data from end devices and process the data across the network

3.1.2 Neural network computation under edge network

In above subsection, we described basic forms of edge networks considered in this study. Overall, a cloud is designed to coordinate the communication path selection, and the neural network partitioning, since a cloud may have full knowledge of the connectivity of edge devices and their computing capabilities. The cloud may statically or dynamically choose the data communication path of the end users, determine the neural network sub-model partitioning and deployment on the edge devices along the communication path, with the goal of minimizing the overall delay, which consists of computing delay and communication delay.

With the above idea, when an end user has data to process using the neural network, it firstly sends the raw data to the first computing edge device, where the first neural network sub-model is deployed. Note that the first computing edge device is not necessarily its access point edge device. After computing, the first computing edge device forwards the processing results, i.e., the output data of the first sub-model, to the next selected computing edge device. And then, this second edge device processes the data with the second sub-model of the neural network, and then forwards the processing results, i.e., the output data of the second sub-model, to the third selected computing edge device. This procedure is continued to the last selected computing edge device, where the final processing result is achieved. Generally, as the edge devices closer the cloud have more powerful computing capabilities, the computing edge devices chosen usually to distribute from the network end to the cloud. Of course, the computing path may be horizontally distributed, depending on the available communication bandwidth between each other and the computing power of the edge devices.

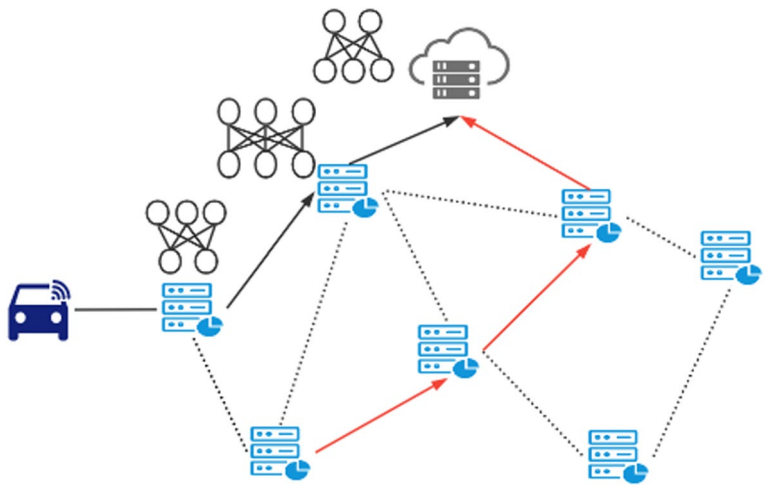

Fig. 2 Example of selecting a communication path & carrying out neural network collaborative computing on the devices of a selected path. It shows an example of the collaborative computing method of neural network in an edge network. The end user has two paths to the cloud. However, as the first path (on left in black) is shorter and has larger available bandwidth when comparing with the second one (on right in red), it may achieve shorter overall communication delay. And we assume that the overall computing delay with the properly designed three neural sub-models deployed on the first path is smaller than the five neural sub-models deployed on the second path, then the first path is selected to process the data, and it illustrates the deployment of the neural network sub-models along the first path. Meanwhile, we can observe that this computing method has the characteristics of pipeline computing. When a batch request is made by the end device, the edge devices on the path can perform parallel computing as pipelines

Figure 2 shows an example of collaborative computing method of neural network in an edge network. The end user has two paths to the cloud. However, as the first path (on left in black) is shorter and has larger available bandwidth when comparing with the second one (on right in red), it may achieve shorter overall communication delay. And we assume that the overall computing delay with the properly designed three neural sub-models deployed on the first path is smaller than the five neural sub-models deployed on the second path, then the first path is selected to process the data, and Fig. 2 illustrates the deployment of the neural network sub-models along the first path. Meanwhile, we can observe that this computing method has the characteristics of pipeline computing, with which a number of computing devices are connected in series and the output of one device is the input of the next one so that high-efficiency parallel computing can be executed like 'stream pipelines' [35]. And then in our scheme, when a batch request is made by an end device, edge devices on the path can perform parallel computing as pipelines.

In above-described computing method, with the path selection scheme proposed in this paper, finally a few edge devices along a communication path will be selected to execute the computing tasks. Usually, rewards are necessary to stimulate edge devices to perform task processing and data transmissions, particularly when the edge devices belong to different owners. To do this, the edge devices can form a service resource pool and 'sell' their service through a blockchain infrastructure [36]. With the author's previous work in [36], the transactions of edge device computing and transmission service can be automatically executed by smart contracts of a blockchain.

Security is one important consideration in above-described computing mode. When data of an end device are transmitted and processed along the selected path, each edge device of the path is possible to threaten the security of the data. Fortunately, the middle edge devices along the path may only access middle-status data as they are in middle part of a distributed neural network, this reduces the possibility of key information leaky. To improve the security of the data, no matter where it locates, technology of Trusted Execution Environment (TEE) can be applied to protect privacy at processor level [37]. Intel has implemented Software Guard Extension (SGX) technology in their advanced processors, by which data with high security requirement may be accessed and processed only in a special isolated part of the processors. ARM implemented a technology called TrustZone, by which the data in processing are isolated and protected in a special zone of an ARM processor with strict protection. AMD and some other companies have similar technology.

Note that in this study, to simply the model, we assumed the computing resource and communication resource are fully available for a target computing task. This could be achieved by time division multiplexing. However, there are possibly multiple tasks to be processed at an end user. In this case, the resource scheduler should apply proper algorithm to assign available resource to different tasks based on different requirements, e.g., to evenly assign available resources to different tasks. In this case, the resource parameters (bandwidth, CPU, etc.) should be set as the value of the actually available resource.

3.1.3 Edge network path selection

With this computing method, we note that there are two main factors that affect the overall processing delay: (a) the path selection of edge-to-cloud; (b) partition method of neural network.

For the first factor, the lower the communication cost of a path we select, the more efficient the computing process will be. For the second factor, the method of partitioning the neural network determines both the data output between edge devices and the computing cost of the devices. With this consideration, we design the procedure of optimizing an edge network with two steps. The first step is to select a computing path consisting of numbers of edge devices from an end user to a cloud, with which the communication cost of the computing path could be lowest. And the second step is to partition a neural network model into sub-networks and deploy them on the edge devices along the computing path. The neural network model is so partitioned that the overall computing delay and communication delay along the path could be minimized, and then the main task of the second step is to balance the time delays at different edge devices, as this computing mode has characteristics of pipeline computing, any edge device that has larger time delay than others may somewhat block the data pipeline processing and increase the overall time delay. Although the second step involves consideration of communication delay as in the first step, however, as in edge computing, the communication delay is generally higher than the computing delay, so the second step may have little impact on the first step.

In this subsection, we focus on the first factor and design a path selection algorithm with least communication overhead. We will discuss setting up a mathematical model of the second factor in the next subsection.

The core of this solution is to find an appropriate communication path between user-accessed edge devices and the cloud. In this paper, we design an ant colony algorithm to optimize the path by considering the communication cost between devices.

The topology of an edge network can be regarded as a weighted undirected graph, in which the weights are determined by the bandwidth between the edge devices. An edge device may be an access point (AP), a router, a server, a base station, and so on, that usually has a fix location, so an edge network has usually a static topology. The bandwidth between such edge network devices is generally fixed and for a given communication link between two edge devices the serving data rate could be assigned in stage of link establishment, so in this study, we assume during an edge computing service period, the corresponding edge communication link uses a fixed bandwidth. To conveniently describe the connection relationship between devices, we introduce the concept of adjacency matrix, which is defined as follows:

$$E[i][j] = \begin{cases} overhead[i][j] & if\ i\ and\ j\ is\ connected \\ inf & otherwise \end{cases} \tag{1}$$

To simply the model, besides time delay, communication overhead such as routing protocol expense is not considered in this study. The cost of communication between two devices is defined as the reciprocal of bandwidth:

$$\text{overhead}[i][j] = \frac{1}{B[i][j]} \tag{2}$$

where $B[i][j]$ is the bandwidth between edge devices i and j. That is, the larger the bandwidth between two devices, the smaller the communication overhead. Next, we construct parameters of ant colony algorithm.

The heuristic factor is defined as:

$$\eta_{ij} = \frac{1}{\text{overhead}[i][j]} = B[i][j] \tag{3}$$

Then, for ant a, the probability that it will move from edge device i to edge device j at time t is:

$$p_{ij}^{m}(t) = \begin{cases} \frac{[\tau_{ij}(t)]^{\alpha}[\eta_{ij}(t)]^{\beta}}{\sum_{s\in J_m(i)}^{m}[\tau_{is}(t)]^{\alpha}[\eta_{is}(t)]^{\beta}}, & if \quad j \in J_k(i) \\ 0 \end{cases} \tag{4}$$

where $J_k(i)$ is the device set that ant a can choose to pass through in the next time. If j has not been visited by ant a before and $E[i][j]! = inf$, then j is in $J_k(i)$. $\tau_{ij}(t)$ represents the pheromone on the path from device i to j at time t. $\tau_{ij}(t)$ update according to the following rules:

$$\tau_{ij}(t) = (1-\rho)\tau_{ij}(t) + \Delta\tau_{ij} \tag{5}$$

$$\Delta\tau_{ij} = \sum\nolimits_{n=1}^{m} \Delta\tau_{ij}^{n} \tag{6}$$

$$\Delta\tau_{ij}^{n} = \begin{cases} \frac{Q}{L_n} & if\ ant\ n\ pass\ the\ path \\ 0 & otherwise \end{cases} \tag{7}$$

where ρ is pheromone volatilization factor, Q is the total pheromone that an ant carries, and L_n is the total communication overhead along the path. The basic principle of ant colony algorithm is as follows:

1. At the initial moment, it sets up m ants, which will release pheromones on paths.
2. If an ant encounters a device that has not yet passed, it randomly selects a device to move forward. At the same time, pheromones related to path length are released.
3. The pheromone concentration is inversely proportional to the overhead of the path. Later, when the ants encounter the device again, they choose the path with higher pheromone concentration.
4. The pheromone concentration on an optimal path increases.
5. Finally, the ant colony finds the optimal feeding path.

For the communication path selection problem in this paper, we finally design the communication path ant colony algorithm:

Algorithm 1: Communication path ant colony algorithm

Input: Topology of edge network

Output: Edge path $path$

Initialization:

1. *set inital pheromone* t_0
2. *set end number* T
3. while $t < T$
4. for each ant k
5. set the initial device
6. for i in $range(m)$
7. choose next device j with the probability given by $p_{ij}^m(t)$
8. compute the overhead of the path passed by ant k
9. for each edge
10. update the pheromone value by $\tau_{ij}(t)$
11. record the best path as $path$
12. $t+=1$
13. return $path$

Through the ant colony constantly updating pheromones on the path, we can finally choose the communication path with the least communication overhead. On the path we finally choose, we will pass through multiple edge devices to the cloud. In the next subsection, we will partition the neural network into multiple sub-models and deploy them on these devices. While data is transmitted along the edge devices, we also complete the computing task of the neural network.

In above, we assumed during an edge computing service period, the communication bandwidth between two connected edge devices is fixed. However, when a computing task has been completed and another end device computing task is ready to setup, we may assume that the available communication bandwidth can be changed, and then in this case, the new edge computing path can be selected based on the changed communication bandwidth.

It is possible that the colony algorithm has a slow convergence speed at the initial stage and easy to fall into local optimum. One possible solution is the authors another study, in which a forward-updating ant colony algorithm is proposed [38]. In this algorithm, a forward-updating criterion is set to select possible computing path with least time overhead even the computing path may be longer in term of tiers. When an ant is detecting a new edge device as a part of a new path and if the forward-updating criterion is met, the ant adds additional update pheromone to the path it traveled. By this operation, the possibility of the new device being selected by other ants increases, leading to faster convergence. Simulation results show that with 10 edge devices the algorithm converges in less than 160 iterations. Meanwhile, in implementation iterations, this algorithm can choose paths with least

time overhead in long running time even in short period the time overhead is relatively higher. In another words, it can jump out local optimums and approach global optimum.

3.2 Mathematical model for partitioning neural networks

In the previous subsection, we discussed how to select a communication path in an edge network and perform neural network collaborative computing on the selected path. In this subsection, we will design a mathematical model to evaluate the impact of different neural network partition methods on running time, and then to design a neural network partition method.

3.2.1 Neural network runtime prediction

To measure the running time of the neural network sub-models on different machines, we need to establish a time delay prediction model for the neural network models and the machines running the models. Generally, the running time of a neural network is determined by two factors: (1) the structure of the neural network; (2) performance of the device.

For a neural network, the more layers it has, the more complex the layer structure, and the longer it runs. To handle different structure of neural network conveniently, the existing research often does not directly establish the time-delay prediction model for the neural network but establishes the time-delay prediction model for the basic layer type of the neural network. When it is necessary to obtain the running time of a certain neural network, the overall running time is obtained by accumulating the running time of each basic layer in turn. For different base layers, their characteristics that affect runtime vary. Table 1 lists some common base layers and their features that affect latency.

For device performance, we mainly take the CPU primary frequency, allocatable memory, and CPU utilization as characteristics to evaluate its impact on the neural network. We denote it as $S = \{\text{frequency}, \text{memory}, \text{utilization}\}$. In this paper, we build a random forest prediction model based on the above characteristics and use the random forest model to evaluate the runtime of the neural network. Then, by giving (X^{type}, S), we can have the runtime of the basic layer:

$$T_{\text{comp}}^{\text{type}} = f(X^{\text{type}}, S) \tag{8}$$

where $f(x)$ represent the random forest prediction model. After experimentation, the average error of our prediction model is controlled within 0.1.

Table 1 Time-delay features of neural network layer

Layer type	Feature (X^{type})
Convolution and local layer	Input size, input/output channel, kernel size, stride
Fully connected layer	Input/output size
Pooling layer	Input size, output channel, kernel size, stride
Activation layer	Input size

By establishing a runtime prediction model, we can flexibly estimate the running time of the sub-models on different machines, thus making it easy to estimate the overall time delay of different partitioning solutions.

3.2.2 Mathematical model of collaborative computing under edge network

In this subsection, we will mathematically model the collaborative computing of neural networks based on the time-delay prediction model given in Sect. 3.2.1. As shown in Fig. 3, we can simplify the procedure to a linear transfer model.

Assuming that there are n edge devices in the selected path, we denote the status of these devices as $path = \{S_1, S_2, \ldots S_n\}$. We denote the neural network to be deployed as $L = \{l_1, l_2, .., l_t\}$ with t layer, the amount of data output for each layer is denoted as $D = \{d_1, d_2, \ldots, d_t\}$. We regard each device as a tier, as shown in Fig. 3.

We denote the partition solution of the neural network as $A = \{a_1, a_2, \ldots, a_n\}$, where $0 < a_i < n+1, a_i < a_{i+1}$; a_i represents layer i of the neural network assigned to the tier a_i. We use $C = \left\{c_1^1, \ldots c_i^j, \ldots c_t^n\right\}$ to denote the computing time of layer i of the neural network running on device j. Then, based on the prediction model, we have

$$c_i^j = f(X^{l_i}, S_j) \tag{9}$$

Since data is transmitted from the previous tier to the next tier, we can quickly obtain the computing time of the whole neural network on the path:

$$T_{\text{comp}} = \sum_{i=1}^{t} c_i^{a_i} = \sum_{i=1}^{t} f\left(X^{l_i}, S_{a_i}\right) \tag{10}$$

The communication delay depends mainly on the bandwidth between the tiers and the amount of data D to be transmitted. We still use $B[i][j]$ to represent the bandwidth between edge devices i and j. Then, the communication time between two tiers can be expressed as

$$T_{comm}(i-1, i) = \frac{D_l}{B[i-1][i]} \tag{11}$$

where D_l is the size of data transferred between tier $i-1$ and i. It depends on the last layer l of the sub-model to which tier $i-1$ is assigned, so it should satisfy $a_l = i-1, a_{l+1} = i$.

In this way, after the above analysis, we finally get a mathematical problem of time-delay optimization with constraints, i.e.,

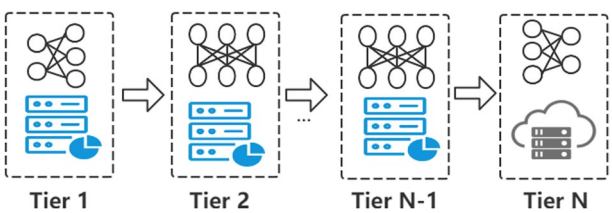

Fig. 3 Procedure of edge network collaborative computing. The procedure is simplified as a linear transfer model, and each device is regarded as a network tier

$$\begin{aligned} &\min\{T_{\text{comp}} + \sum_{i=1}^{n} T_{\text{comm}}(i, i+1)\} \\ &s.t. a_i < a_{i+1} \\ &0 < a_i < n+1 \\ &a_l = i-1 \\ &a_{l+1} = i \end{aligned} \tag{12}$$

In above analysis, we consider both computing time delay and communication time delay of a neural network. For computing delay, the key factors are the performance and status of the edge devices. For communication delay, the main factors are the amount of data output and the bandwidth between the layers. The above optimization problems describe the process that different partitioning solutions of neural networks can cause different changes in computing and communication time delays. Our goal is to obtain the optimal partition solution to minimize the overall delay of the neural network.

3.3 Ordered elitist genetic algorithm

The partition points of a neural network increase with the increase of tiers. Specifically, when the edge devices are divided into n tiers, it is necessary to determine $n-1$ partition points for the neural network model. Then, when a network with t layers need to be deployed, the complexity of the global search algorithm becomes $o\left(C_t^{n-1}\right)$, where C_t^{n-1} represents the number of combinations. As the device and neural network layers grow, the time complexity of the algorithm increases rapidly and becomes unacceptable.

To solve this problem, a heuristic algorithm is considered. Genetic algorithm is a heuristic optimization algorithm widely used in industry [31]. When solving some conventional combinatorial optimization problems, it can get the optimal results faster than some conventional optimization algorithms. When it is difficult to get the optimal solution directly, genetic algorithm provides us a good way to approach the optimal solution.

Based on the combination of objective optimization problem and simple genetic algorithm, an ordered elitist genetic algorithm (OEGA) is designed to solve the optimization problem that we proposed.

3.3.1 Constructing chromosomes

In this subsection, we will design the chromosome of the optimization problem.

First, suppose the neural network that we want to partition is $L = \{l_1, l_2 \ldots, l_t\}$, that is, the total number of layers is t, the edge device network is divided into n tiers, we want to select $n-1$ partition points eventually, then for the partition solution i, the chromosomal representation of the genetic algorithm can be denoted as:

$$x_i = \left\{x_i^1, x_i^2, \ldots x_i^{n-1}\right\} \tag{13}$$

where $0 \le x_i^j \le t$, x_i^j represents the $j-th$ partition point of the partition solution i, with a total of $n-1$ partition points.

For chromosome x_i, we note that it is order independent, specifically, $\left\{x_i^1, x_i^2, \ldots x_i^{n-1}\right\}$ and $\left\{x_i^2, x_i^1, \ldots x_i^{n-1}\right\}$ describe the same solution. Therefore, if a simple genetic algorithm is used to solve the problem, the duplicate solutions will be blindly searched in the

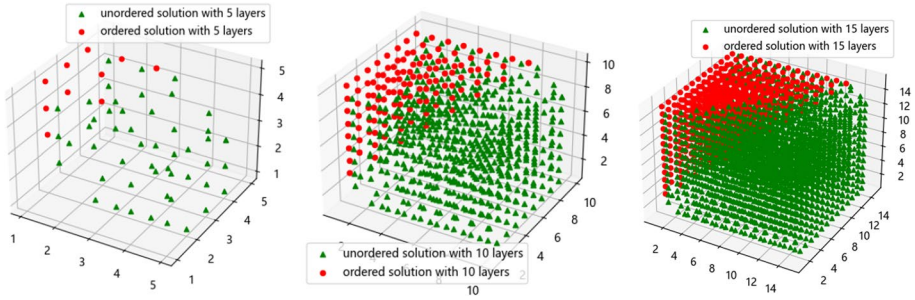

Fig. 4 Ordered solution space and disordered solution space. It shows the distribution of solution space when the number of layers of the neural network is 5, 10, and 15, respectively, with 3 partition points. The green points represent the solution space that simple genetic algorithm requires to search for and contain many duplicate solutions. The red point represents the solution space of the ordered partition solution

solution space, which results in an increase in the running time of the algorithm. Figure 4 shows the distribution of solution space when the number of layers of the neural network is 5, 10, and 15, respectively, with 3 partition points. The green points represent the solution space that simple genetic algorithm requires to search for and contain many duplicate solutions. The red point represents the solution space of the ordered partition solution; in fact, we only need to search for partitioned solutions in that space.

For this reason, we propose an ordered elitist genetic algorithm to speed up the convergence of the algorithm and to ensure the optimal solution as much as possible.

3.3.2 Ordered fitness function

In the genetic algorithm, for any solution x_i, its fitness reflects the probability that the solution will be retained. The higher the fitness, the higher the probability of being selected in the algorithm. For the neural network partition problem, there are two factors that affect the fitness: (1) time delay. The smaller the delay, the higher the fitness; (2) the degree of ordering, the higher the degree of ordering, the higher the fitness.

We can quickly define a time-delay fitness function as:

$$f_{\text{delay}}(x_i) = \sigma(T_{\text{comp}} + \sum_{j=1}^{t} T_{comm}(x_i^j, x_i^{j+1})) \tag{14}$$

where $\sigma(x) = \frac{2}{1+e^x}$, when $x > 0$, $\sigma(x)$ can be compressed to an interval (0,1), and as x increases,$\sigma(x)$ is reduced and therefore reflects the time-delay suitability very well.

Next, we need to define an ordered fitness function. Firstly, to evaluate the order degree of a partition solution, we introduce the concept of reverse order number. For the partition solution $x_i = \{x_i^1, x_i^2, \ldots x_i^{n-1}\}$, the reverse order number is defined as:

$$r(x_i) = \sum_{j=1}^{n} \sum_{k=j+1}^{n} I(x_i^j > x_i^k) \tag{15}$$

where $I(x > y) = \begin{cases} 1 & x \geq y \\ 0 & other \end{cases}$, is an indictor function.$r(x_i)$ reflects the disorder degree of a x_i. The higher the disorder degree, the greater the value of $r(x_i)$. When x_i is strictly increasing, $r(x_i) = 0$. With the help of reverse order number, we can construct ordered fitness function as follows:

$$f_{order}(x_i) = \sigma(r(x_i)) \tag{16}$$

In this way, two fitness functions are used to favorably select the range of solutions. In the solution space, the algorithm will choose a more ordered solution with less delay, thus speeding up the convergence rate. So, ultimately, our fitness function is defined as:

$$f_{\text{fitness}}(x_i) = \alpha f_{\text{delay}}(x_i) + (1-\alpha) f_{\text{order}}(x_i) \tag{17}$$

where α is the weight parameter, it depends on whether we want to prefer solutions with smaller delay or ordered solutions.

3.3.3 Overall flow of the algorithm

To ensure the solution is optimal as much as possible, we propose OEGA combined with elitist genetic algorithm (EGA). In the process of iteration, we will retain the elite individuals of the population.

For this purpose, we define terms as follows:

1. *Elite Individual*: Individuals with the largest fitness value in a population of a generation of genetic algorithms are elite individuals.

2. *Elite Individual Retention*: If the elite individuals in generation k have greater fitness than those in generation $k+1$, that is, $E(k) > E(k+1)$, then $E(k)$ is used to replace any individual in generation $k+1$.

By supplementing the above two definitions, combined with the ordered fitness function proposed in this paper, we finally summarize OEGA. The algorithm flow is as follows:

Algorithm 2: Ordered elitist genetic algorithm

Input: Neural network $L = \{l_1, l_2, \ldots, L_t\}$
Communication path $path$
Initialize population $pop(0) = \{x_1, x_2, \ldots x_{1000}\}$
Iteration Threshold T

Output: Partition solution x

1. $k = 0$
2. $E(k) = max\_fitness(pop(k), L, Path)$
3. **while** $k < T$:
4. $\quad pop(k+1) = cross(pop(k))$
5. $\quad mutation(pop(k+1))$
6. $\quad sort(f_{fitness}(pop(k+1), L, Path))$
7. $\quad select(pop(k+1))$
8. $\quad E(k+1) = max\_fitness(pop(k+1))$
9. $\quad$ **if** $E(k+1) > E(k)$:
10. $\quad\quad random\_replace(pop(k+1), E(k))$
11. $\quad k = k+1$
12. **return** $max\_fitness(pop(k), L, Path)$

By introducing an ordered fitness function, the algorithm greatly reduces the search range in the solution space and improves the convergence speed of the algorithm. By introducing elite individual preservation, the solution can be kept as close to the optimal solution as possible. Theoretically, it can be proved that the genetic algorithm for elite individual preservation can converge to the optimal solution when the number of iterations is close to infinity [32], but the convergence speed is slower. In our experiments, we can see that the algorithm that we proposed in this paper considers both the convergence speed and the convergence effect.

4 Experimental results and discussion

In this section, we design experiments to verify our proposed algorithm and computing method and compare it with traditional methods. Our experiment used OSBrain to simulate edge devices and build edge networks. OSBrain is an open-source multi-agent simulation module based on python. It supports the functions of custom agents and provides communication modes between multiple agents. In this paper, we simulate edge devices through agents, and edge devices realize collaborative computing by calling the neural network model established by Pytorch.

4.1 Parameter selection of ant colony algorithm

The parameter selection of ant colony algorithm will greatly affect the result of our path selection. In theory, α represents an information heuristic. The larger the α, the greater the likelihood that an ant will choose the path it traveled before, and the less random the search path will be. β represents the desired heuristic. The larger the β, the easier it is for the ant colony to choose the local shortest path. This will speed up the convergence of the algorithm, but it is easy to fall into the local optimum.

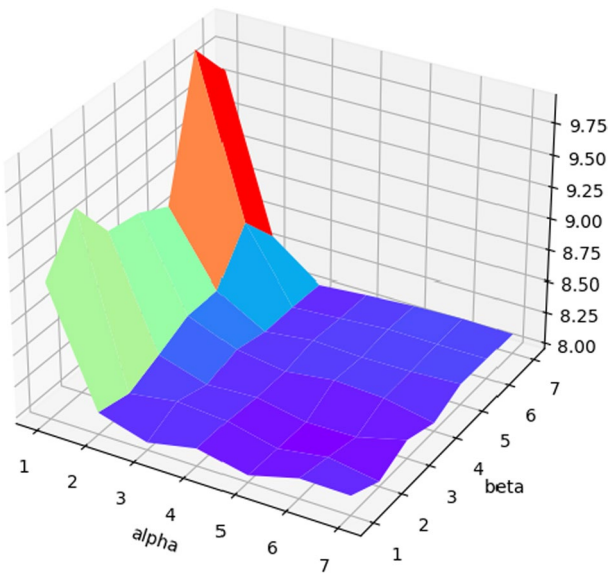

Fig. 5 Communication overhead changes with alpha and beta. It shows the computed convergence of the ant colony algorithm when the parameters alpha and beta are both set in range 1 to 7. When alpha is 5 and beta is 3, the ant colony algorithm converges best, so we set both alpha and beta as the two values in subsequent experiments

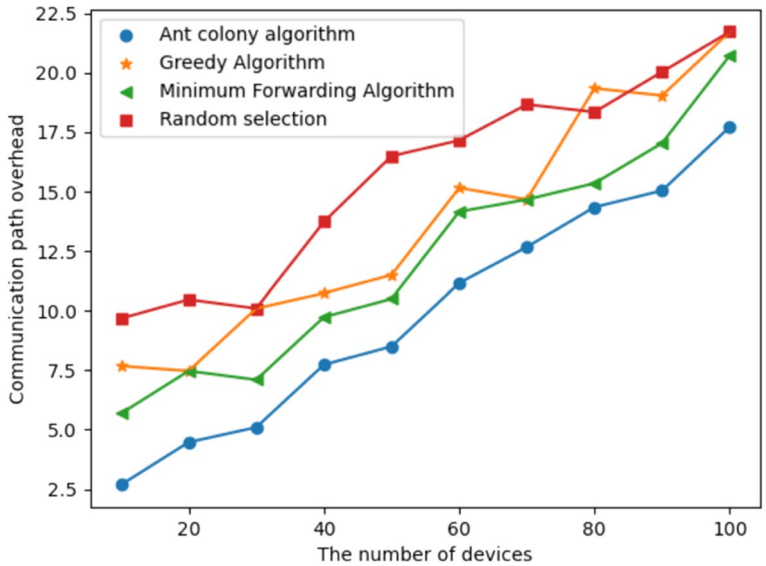

Fig. 6 Communication overhead for different paths. We set up an edge network with 10–100 edge devices to test the path selection ability of Ant Colony Algorithm by comparing with other three methods, i.e., Greedy Algorithm, Minimum Forwarding Algorithm, Random Selection. It shows that the cost of the paths increases with the number of the devices for all four methods. However, Ant Colony Algorithm can select a better path than other methods. Compared with the Greedy Algorithm and the Minimum Forwarding Algorithm, the Ant Colony Algorithm can jump out of the local optimal solution to better select the path with less communication overhead

We set the range of alpha and beta from 1 to 7 and computed the convergence of the ant colony algorithm within this range. The result is shown in Fig. 5. We can see that when the alpha value is 5 and the beta value is 3, the ant colony algorithm converges best, so in subsequent experiments, we set both alpha and beta as above values.

4.2 Comparison of different communication path

To test the path selection ability of the ant colony algorithm, we set up an edge network with 10-100 edge devices randomly and used four methods to select the communication path. The results are shown in Fig. 6.

As we can see, the cost of paths increases with the number of devices. However, ant colony algorithm can select a better path than other methods. Compared with the greedy algorithm and the minimum forwarding algorithm, the ant colony algorithm can jump out of the local optimal solution to better select the path with less communication overhead.

4.3 Comparison of different computing method

In this subsection, we compare single-edge computing, cloud computing, edge cloud collaborative computing, and edge network collaborative computing proposed in this paper. In the comparative experiment, we calculated the running time of LeNet5, AlexNet and VGG [33, 34] networks under the four computing modes, and set the bandwidth between devices to 0.1, 2, 5 MB. Among the three networks, LeNet5 has the smallest scale, it has eight layers including an input layer and an output layer and the size of its input layer is 32×32. AlexNet has a larger scale—it has 12 layers, and

its input layer size is 227 × 227 × 3. VGG has the largest scale—its input layer size is 224 × 224 × 3 but it has 22 layers (we used VGG-16 in this study). We observed the changes of their running time as listed in Tables 2, 3, 4:

In above tables, we bold the smallest running time of neural network. Firstly, we can see that the running time of single-edge computing does not change with the increase of bandwidth, because single-edge device computing is local and not affected by bandwidth. However, we can find that with the increase of the scale of neural network, the computing time of single-edge device increased rapidly, which makes the overall running time larger. The column of VGG in the table proves this.

Secondly, we can see that when the bandwidth between devices is 0.1 MB and 1 MB, multi-layer edge network collaborative computing can generally achieve shortest running time. This is because cloud computing is limited by bandwidth and will have large communication delay. By controlling the partition point, collaborative computing can reduce the amount of data transmitted as much as possible, so as to reduce the overhead of communication. From this point of view, multi-layer edge network collaborative computing and edge-cloud collaborative computing are better than only-cloud computing.

With further increase of bandwidth, we can see in Table 4 that the cloud computing has the shortest running time. In this case, the communication delay is no longer the bottleneck of the overall running time. Therefore, the powerful cloud computing can complete the task quickly.

Table 2 Neural network runtime at 0.1 MB

Running time (s)	LeNet5	AlexNet	VGG
Single-Edge computing	0.45	1.30	14.10
Cloud computing	1.71	1.77	4.05
Edge-cloud computing	0.38	1.10	2.45
Multi-layer Edge network computing	**0.25**	**0.71**	**2.31**

Table 3 Neural network runtime at 1 MB

Running time (s)	LeNet5	AlexNet	VGG
Single-Edge computing	0.45	1.30	14.10
Cloud computing	0.82	0.70	2.53
Edge-cloud computing	0.36	0.8	2.44
Multi-layer Edge network computing	**0.25**	**0.43**	**2.0**

Table 4 Neural network runtime at 5 MB

Running time(s)	LeNet5	AlexNet	VGG
Single-Edge computing	0.45	1.30	14.10
Cloud computing	**0.20**	**0.21**	**1.20**
Edge-cloud computing	0.26	0.4	2.00
Multi-layer Edge network computing	0.25	0.4	1.96

Therefore, the computing method proposed in this paper is a supplement to the traditional computing methods. Its purpose is to make full use of computing resources and reduce the computing overhead as much as possible in the case of limited bandwidth.

4.4 Algorithm performance comparison

In this subsection, we compare the convergence of OEGA proposed in this paper with traditional SGA and EGA. We use LeNet5, AlexNet, VGG network and set five partition points for computing. We performed 1000 iterations of the algorithm and counted the running time of each generation of optimal solution in the iterative process. The results are shown in Fig. 7.

It can be seen that OEGA converges faster than the other algorithms and has converged to the current optimal solution after about 200~400 iterations, while SGA and EGA need about 400~1000 iterations to converge. We can see that the convergence value of OEGA is better under the statistical final delay convergence value of the three neural networks, which shows that compared with SGA and EGA, OEGA has faster convergence speed and better convergence under the same number of iterations.

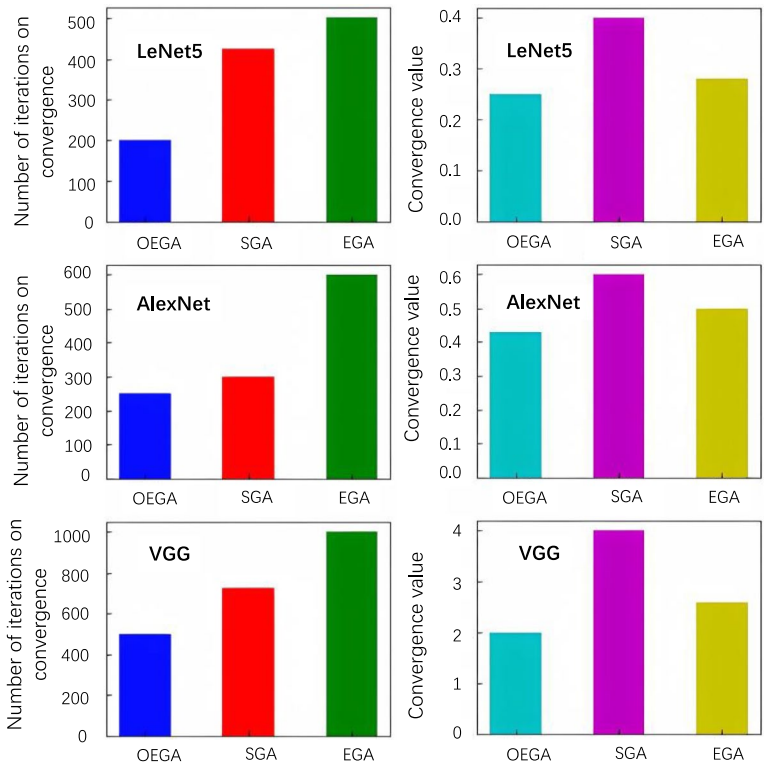

Fig. 7 Comparison of convergence speed between OEGA and other algorithms. OEGA converges faster than the other algorithms and has converged to the current optimal solution after about 200 ~ 400 iterations, while SGA and EGA need about 400 ~ 1000 iterations to converge, and the convergence value of OEGA is better under the statistical final delay convergence value of the three neural networks, showing that compared with SGA and EGA, OEGA has faster convergence speed and better convergence under the same number of iterations

4.5 Comparison of different partition method

In previous subsection, we showed that OEGA can reduce the search range of the solution space, so it is easier to find the optimal solution. Therefore, compared with other partition methods, it can obtain better partition results. To verify this, we compare the method of average partition and two-portion partition method with the method proposed in this paper. We randomly build a 10~1000-layer neural network, partition it with different methods, and finally deploy it on a three-tier edge network. The results are shown in Fig. 8.

The upper subgraph is the running time of the neural network under the three partition methods, and the subgraph below is the corresponding data throughput. For edge network collaborative computing, the partition solution of the neural network is very important. A good partition solution should ensure as little data throughput as possible between tiers to minimize the overhead of communication. By comparing the left and right subgraphs, we can see that the runtime is basically the same as the change of data throughput. The larger the throughput, the longer the runtime. In addition, by comparing the three partitioning methods, we can see that the proposed partitioning method can obtain a better partition solution. Evenly partition presents a randomness, i.e., the inability to control data throughput. Binary partition method can get better partition solution than evenly partition method, but overall is not as good as OEGA partition method because it is easy to fall into local optimal solution, while OEGA partition method is easier to jump out of local optimal solution.

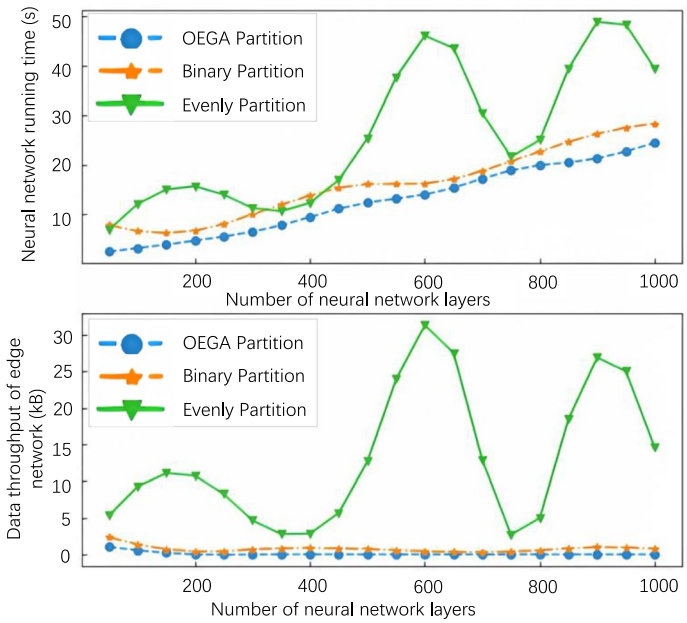

Fig. 8 Comparison of delay and data throughput between OEGA partition method and other partition methods. The upper subgraph is the running time of the neural network under the three partition methods, and the subgraph below is the corresponding data throughput. By comparing the left and right subgraphs, we can see that the runtime is basically the same as the change of data throughput. The larger the throughput, the longer the runtime. In addition, by comparing the three partitioning methods, we can see that the proposed partitioning method can obtain a better partition solution. Evenly partition presents a randomness, i.e., the inability to control data throughput. Binary partition method can get better partition solution than evenly partition method, but overall is not as good as OEGA partition method because it is easy to fall into local optimal solution, while OEGA partition method is easier to jump out of local optimal solution

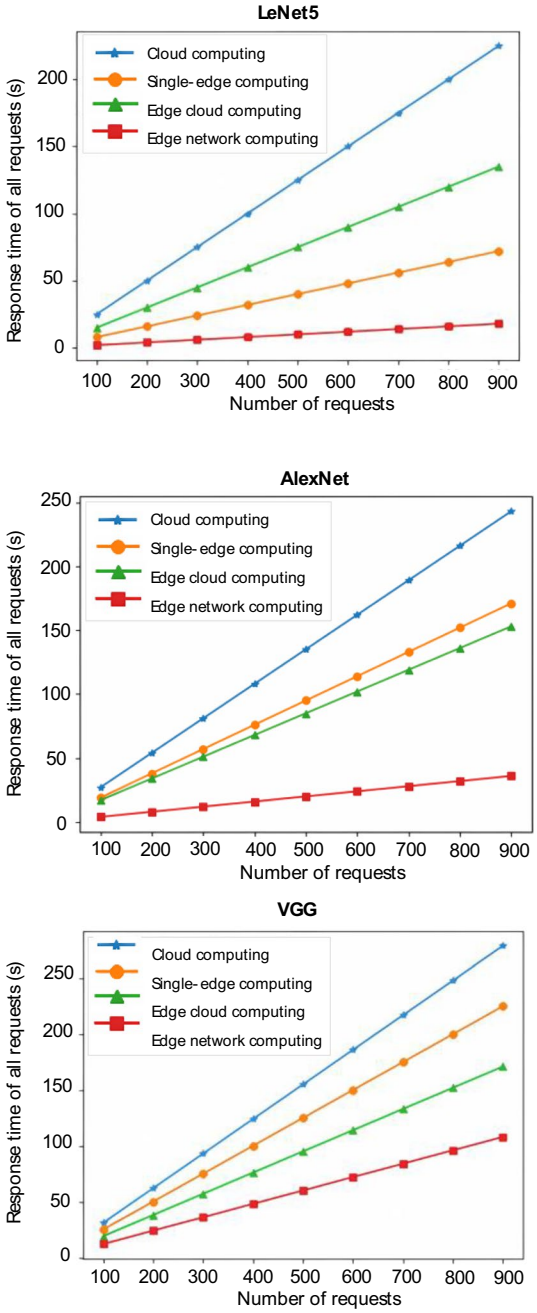

Fig. 9 Response time changes as the number of batch task requests increases. One advantage of multi-layer edge network collaborative computing is pipelined collaborative computing. When there are batch requests to be processed, the logic layers are processed in parallel like the factory pipeline. network collaborative computing. We still use LeNet5, AlexNet, VGG network in this experiment and set the device bandwidth as 100 KB. It shows the experimental results of the response time of each computing method when processing requests in batch. When the number of requests in each batch is small, the gap between computing methods is not obvious. With the increase of each batch of requests, the processing time of cloud computing and single-edge device computing increases faster. This is because these two computing modes cannot rely on pipeline for parallel processing, and the request accumulation is serious. The edge-cloud collaborative computing and multi-layer edge network collaborative computing process faster than the first two methods

4.6 Comparison of batch request processing

In the previous subsections, we mentioned that one advantage of multi-layer edge network collaborative computing is pipelined collaborative computing. When there are batch requests to be processed, the logic layers are processed in parallel like the factory pipeline. In this way, the accumulation of tasks can be alleviated when processing batch tasks, so as to improve the operation efficiency. Theoretically, the more requests per batch, the more obvious the advantages of multi-layer edge network collaborative computing.

In this experiment, we still use LeNet5, AlexNet, VGG network. When the device bandwidth is 100 KB, we count the response time of each computing method when processing requests in batch. The results are listed in Fig. 9.

When the number of requests in each batch is small, the gap between computing methods is not obvious. With the increase of each batch of requests, the processing time of cloud computing and single-edge device computing increases faster. This is because these two computing modes cannot rely on pipeline for parallel processing, and the request accumulation is serious. The edge-cloud collaborative computing and multi-layer edge network collaborative computing process faster than the first two methods.

5 Conclusion

In this paper, a method of collaborative computing of neural network under multi-layer edge network is presented. Firstly, this method selects a communication path in the edge network from edge to cloud and carries out neural network collaborative computing along this path. Secondly, this paper explores the problem of partitioning the neural network. To ensure the running time of partitioned neural network as small as possible, we establish a delay optimization mathematical model for the computing process to evaluate the advantages and disadvantages of different partition solutions. Finally, to find the optimal partition solution, we designed OEGA, which has better convergence speed and effect than traditional algorithms. For the proposed algorithm and computing method, we use LeNet5, AlexNet and VGG to do experimental validation. The experimental results show that the proposed algorithm has better computational efficiency when bandwidth is limited. In addition, multi-layer edge network collaborative computing has pipeline characteristics compared to other traditional computing methods, so it has more advantages in dealing with many tasks, making full use of computing resources and parallelism of edge device clusters.

The work of this paper is limited to the analysis of static edge networks. In the future, we will further explore the problem of collaborative computing of neural networks on edge network with dynamic changes of connectivity, bandwidth, and data size, etc.

Abbreviations

EGA	Elitist genetic algorithm
IoT	Internet of things
OEGA	Ordered elitist genetic algorithm
SGA	Simple genetic algorithm

Author contributions

QL and M-TZ contributed to the conceptualization; T-FR was involved in the formal analysis; QL acquired the funding; C-BJ contributed to the investigation; M-TZ contributed to the methodology; QL was involved in the project administration; T-FR contributed to the software; QL and M-TZ assisted in the supervision; T-FR contributed to the validation; YC

contributed to the visualization; QL, M-TZ and T-FR were involved in writing—original draft; M-TZ contributed to writing—review and editing.

Funding
This research was funded by the National Key Research and Development Program of China (grant Nos. 2020YFB2104500), Research and Application Demonstration of Intelligent IoT and Control Technology for Urban Integrated Energy.

Data availability statement
Not applicable.

Declarations

Competing interests
The authors declare no conflict of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript; or in the decision to publish the results.

Received: 16 January 2023 Accepted: 7 August 2023
Published online: 19 August 2023

References

1. M. Chiang, Z. Tao, Fog and iot: an overview of research opportunities. IEEE Internet Things J. **3**(6), 854–864 (2017)
2. W. Shi, C. Jie, Z. Quan, Y. Li, L. Xu, Edge computing: vision and challenges. Internet Things J. IEEE **3**(5), 637–646 (2016)
3. 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**(4), 2322–2358 (2017)
4. L.A. Hai, B. Sz, A. Zc, C. Hl, D. Lw, A survey on computation offloading modeling for edge computing—sciencedirect. J. Netw. Comput. Appl. 2020, 169.
5. X. Wang, Y. Han, V. Leung, D. Niyato, X. Chen, Convergence of edge computing and deep learning: a comprehensive survey. IEEE Commun. Surv. Tutorials **22**(99), 869–904 (2020)
6. X. Chen, Q. Shi, L. Yang, J. Xu, Thrifty edge: resource-efficient edge computing for intelligent iot applications. IEEE Netw. **32**(1), 61–65 (2018)
7. X. Chen, W. Li, S. Lu, Z. Zhi, X. Fu, Efficient resource allocation for on-demand mobile-edge cloud computing. IEEE Trans. Veh. Technol. **67**(9), 8769–8780 (2018)
8. C. Xu, R. Ju, D. Zhang, Y. Zhang, Distilling at the edge: a local differential privacy obfuscation framework for iot data analytics. IEEE Commun. Mag. **56**(8), 20–25 (2018)
9. Y. Ding, Y. Feng, W. Lu et al., Online edge learning offloading and resource management for UAV-assisted MEC secure communications. IEEE J. Select. Topics Signal Process. **17**(1), 54–65 (2023). https://doi.org/10.1109/JSTSP.2022.3222910
10. W. Lu, Y. Mo, Y. Feng et al., Secure transmission for multi-UAV-assisted mobile edge computing based on reinforcement learning. IEEE Trans. Netw. Sci. Eng. **10**(3), 1270–1282 (2023). https://doi.org/10.1109/TNSE.2022.3185130
11. Z. Zhou, X. Chen, E. Li, L. Zeng, K. Luo, J. Zhang, Edge intelligence: paving the last mile of artificial intelligence with edge computing. Proc. IEEE **107**(8), 1738–1762 (2019)
12. S. Han, H. Mao, W.J. Dally, Deep compression: compressing deep neural networks with pruning, trained quantization and huffman coding. Fiber **56**(4), 3–7 (2015)
13. Y.D. Kim, E. Park, S. Yoo, T. Choi, L. Yang, D. Shin, Compression of deep convolutional neural networks for fast and low power mobile applications. Computer Science **71**(2), 576–584 (2015)
14. N. Lane, S. Bhattacharya, A. Mathur, C. Forlivesi, F. Kawsar,*DXTK: Enabling Resource-efficient Deep Learning on Mobile and Embedded Devices with the DeepX Toolkit*. Eai International Conference on Mobile Computing. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering). 2016.
15. S. Liu, Y. Lin, Z. Zhou, K. Nan, J. Du. *On-Demand Deep Model Compression for Mobile Devices: A Usage-Driven Model Selection Framework*. MobiSys '18: Proceedings of the 16th Annual International Conference on Mobile Systems, Applications, and Services, 2018, 389–400.
16. Oh, Y. H., Quan, Q., Kim, D., Kim, S., & Lee, J. W., *A Portable, Automatic Data Qantizer For Deep Neural Networks,* PACT '18: Proceedings of the 27th International Conference on Parallel Architectures and Compilation Techniques,November 2018, 1–14.
17. Y. Kang, J. Hauswald, G. Cao, A. Rovinski, T. Mudge, J. Mars et al., Neurosurgeon: collaborative intelligence between the cloud and mobile edge. Acm Sigplan Notices **52**(1), 615–629 (2017)
18. E. Li, L. Zeng, Z. Zhou, X. Chen, Edge ai: on-demand accelerating deep neural network inference via edge computing. IEEE Trans. Wireless Commun. **19**(1), 447–457 (2020)
19. C. Hu, W. Bao, D. Wang and F. Liu, in *Dynamic Adaptive DNN Surgery for Inference Acceleration on the Edge*, IEEE INFOCOM 2019 - IEEE Conference on Computer Communications, 2019, pp. 1423–1431.
20. M. Liu, Y. Li, Y. Zhao, H. Yang, J. Zhang, Adaptive DNN model partition and deployment in edge computing-enabled metro optical interconnection network. Optic Fiber Commun Conf Exhibit **2020**, 1–3 (2020)
21. H. Li, C. Hu, J. Jiang, Z. Wang, Y. Wen and W. Zhu, JALAD: Joint accuracy-and latency-aware deep structure decoupling for edge-cloud execution, *2018 IEEE 24th international conference on parallel and distributed systems (ICPADS)*, 2018, pp 671–678
22. M. Xu, F. Qian, M. Zhu, F. Huang, S. Pushp, X. Liu, DeepWear: adaptive local offloading for on-wearable deep learning. IEEE Trans. Mob. Comput. **19**(2), 314–330 (2020)

23. L. Zeng, E. Li, Z. Zhou, X. Chen, Boomerang: on-demand cooperative deep neural network inference for edge intelligence on the industrial internet of things. IEEE Network **33**(5), 96–103 (2019)
24. H. J. Jeong, H. J. Lee, C. H. Shin, S. M. Moon. *IONN: Incremental Offloading of Neural Network Computations from Mobile Devices to Edge Servers*, in the ACM Symposium. ACM. 2018.
25. G. Li, L. Liu, X. Wang, X. Dong, P. Zhao, X. Feng. *Auto-tuning neural network quantization framework for collaborative inference between the cloud and edge*, in ICANN 2018: Artificial Neural Networks and Machine Learning, 2018, pp. 402–411.
26. S. Teerapittayanon, B. McDanel and H.T. Kung, *BranchyNet: Fast inference via early exiting from deep neural networks*, 2016 23rd International Conference on Pattern Recognition (ICPR), 2016, pp. 2464–2469.
27. 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 2017 IEEE International Conference on Computer Design (ICCD), 2017, pp. 273–280
28. Rapp, J. W., Jackson, L., Jones, M., & Cherasaro, T., Pipeline accelerator for improved computing architecture and related system and method. 2004, CA, US-2004136241-A1
29. Harlap, A., Narayanan, D., Phanishayee, A., Seshadri, V., Devanur, N., & Ganger, G., et al,. Pipedream: fast and efficient pipeline parallel dnn training, 2018, eprint arXiv:1806.03377.
30. Seyedzadegan, M., Othman, M., Ali, B. M., & Subramaniam, S, Wireless Mesh Networks: WMN Overview, 2011, WMN Architecture.
31. Ranjini, A., & Zoraida, B., Preemptive appliances scheduling in smart home using genetic algorithm. Adv. Intell. Syst. Comput. 2015, 387–393.
32. D. Bhandari, C.A. Murthy, Pal SK (1996) Genetic algorithm with elitist model and its convergence. Int J Pattern Recogn Artific Intell **10**(06), 731–747 (1996)
33. Koonce, B., VGG Network, in *Convolutional Neural Networks with Swift for Tensorflow* (Apress, Berkeley, CA, 2021).
34. Alom, M. Z., Taha, T. M., Yakopcic, C., Westberg, S., & Asari, V. K., The history began from alexnet: a comprehensive survey on deep learning approaches, 2018, eprint arXiv:1803.01164.
35. P. Wiener, P. Zehnder and D. Riemer, in *Towards Context-Aware and Dynamic Management of Stream Processing Pipelines for Fog Computing*. 2019 IEEE 3rd International Conference on Fog and Edge Computing (ICFEC), Larnaca, Cyprus, 2019, pp. 1–6, doi: https://doi.org/10.1109/CFEC.2019.8733145.
36. M.-T. Zhou, F.-G. Shen, T.-F. Ren and X.-Y. Feng, Blockchain-based Volunteer Edge Cloud for IoT Applications. In: 2021 IEEE 93rd Vehicular Technology Conference (VTC2021-Spring), Helsinki, Finland, 2021, pp 1–6, https://doi.org/10.1109/VTC2021-Spring51267.2021.9449041
37. K. Suzaki, K. Nakajima, T. Oi, A. Tsukamoto, TS-Perf: general performance measurement of trusted execution environment and rich execution environment on intel SGX, Arm TrustZone, and RISC-V Keystone. IEEE Access **9**, 133520–133530 (2021). https://doi.org/10.1109/ACCESS.2021.3112202
38. Q. Li, M.-T. Zhou, T.-F. Ren, et al., Optimized Edge-Network Collaborative Computing for Artificial Intelligence, submitted, May 2023

Publisher's Note