End-Edge Collaborative Inference of Convolutional Fuzzy Neural Networks for Big Data-Driven Internet of Things

Yuhao Hu, Xiaolong Xu†, Li Duan, Muhammad Bilal, Qingyang Wang, Wanchun Dou

***Abstract*—Deep Neural Networks (DNN) has been widely applied in big data-driven Internet of Things (IoT) for excellent learning ability, while the black-box nature of DNN leads to uncertainty of inference results. With higher interpretability, Convolutional Fuzzy Neural Network (CFNN) becomes an alternative choice for the model of IoT applications. IoT applications are often latency-sensitive. By jointly utilizing computing power of IoT devices and edge servers, end-edge collaborative CFNN inference improves the insufficiency of local computing resources and reduces the latency of computing-intensive CFNN inference. However, the calculation amount of fuzzy layers is hard to get directly, bringing difficulty to CFNN partition. Additionally, the profit of service providers is often ignored in existing work on distributed inference. In this paper, an end-edge collaborative inference framework of CFNNs for big data-driven IoT, named DisCFNN, is proposed. Specifically, a novel CFNN structure and a method of fuzzy layer calculation amount assessment are designed at first. Next, computing resource allocation and CFNN partition decisions are generated on each edge server based on deep reinforcement learning. Then, each IoT device sends the request of CFNN inference service to a certain edge server or infer the whole CFNN locally according to the task offloading strategy obtained through many-to-one matching game. Finally, the effectiveness of DisCFNN is evaluated through extensive experiments.**

***Index Terms*—Convolutional Fuzzy Neural Network, big data-driven, Internet of Things, end-edge collaborative inference.**

I. Introduction

NOWADAYS, Internet of Things (IoT) is developing fast with wide application of various network techniques, such as Cellular Network and WiFi. In IoT, diverse devices are connected to the Internet and communicate with each other, bringing the flourish of big data-based IoT applications including automatic driving, smart healthcare, etc [1]. According to the survey, IoT will become one of the most important technologies impacting human's life and the number of interconnected IoT devices will reach an unprecedented magnitude in 2025 [2].

Deep Neural Networks (DNN) has been widely used as the model of IoT applications because of their excellent ability to learn potential relations between different features in input [3]. For instance, Convolutional Neural Network (CNN) has been widely used in computer vision applications like virtual reality and face recognition [4], [5]. However, the black-box nature makes the interpretability of DNN bad and results in hardness to understand the detailed internal working mechanism of DNN. This causes that when facing unknown conditions in real world, prediction results of DNN may be bad time to time and it is difficult to find reasons for the appearance of those bad results. So DNN is not absolutely reliable in reality. Conversely, fuzzy logic possesses high interpretability. Through merging fuzzy logic into the DNN, a novel hybrid system called Deep Neuro-Fuzzy System (DNFS) appears. As one of the most significant DNFS models, Convolutional Fuzzy Neural Network (CFNN) combines excellent learning ability of CNN and great interpretability of fuzzy logic, efficiently reducing the uncertainty when inferring input from real word and improving reliability of IoT applications.

Nevertheless, inference of CFNN is computing-intensive. An IoT device with limited computing resources is hard to independently infer a whole CFNN in a short time, which cannot meet the fast-response requirement of the IoT application. The traditional solution to this problem is inferring the whole CFNN on the edge or cloud server. Although stronger computing power of the edge or cloud server helps reduce CFNN inference latency, the huge network traffic and limited channel bandwidth between IoT devices and servers in big data-driven IoT leads to ultra-high data transmission latency, increasing the final response time of the service [6]. Comprehensively considering the advantage of local computing without data transmission latency and the advantage of edge computing with stronger computing power, end-edge collaborative CFNN inference is hopeful to efficiently decrease CFNN inference latency. Note that the cloud is not used in this work for the unbearable data transmission latency caused by the ultra-remote distance between cloud and edge. Existing studies on distributed inference empowered by edge computing mainly focus on reduction of service response latency [7] or IoT device energy consumption [8], where only customers' profit is taken into account and service providers' profit is ignored. Also, all existing studies focus on distributed inference of DNN. The fuzzy layer in CFNN is different from other neural

Yuhao Hu and Xiaolong Xu are with the School of Software, Nanjing University of Information Science and Technology, Nanjing 210044, China. E-mail: 202212210036@nuist.edu.cn; xlxu@nuist.edu.cn.

Li Duan is with the Beijing Key Laboratory of Security and Privacy in Intelligent Transportation, Beijing Jiaotong University, Beijing 100044, China. E-mail: duanli@bjtu.edu.cn.

Muhammad Bilal is with the School of Computing and Communications, Lancaster University, United Kingdom. E-mail: m.bilal@ieee.org.

Qingyang Wang is with the School of Reading, Nanjing University of Information Science and Technology, Nanjing 210044, China. E-mail: 202183710046@nuist.edu.cn.

Wanchun Dou is with the State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing 210023, China. E-mail: douwc@nju.edu.cn.

† Xiaolong Xu is the corresponding author.

Manuscript received April 19, 2021; revised August 16, 2021.

layers like convolution layers and fully connected layers, and the calculation amount of fuzzy layers is difficult to be directly computed. Furthermore, former studies mainly consider the scenario with only one edge server, which is not applicable to real scenarios where multiple edge servers are located and reasonable task offloading strategies should be made.

To tackle challenges mentioned above, an end-edge collaborative inference framework of CFNN for big data-driven IoT, named DisCFNN, is proposed in this paper. Specifically, at first, a novel CFNN structure is designed offline and the calculation amount of the fuzzy layer is predicted based on polynomial regression. The CFNN designed in this work converges faster during training and presents less uncertainty during validation compared with traditional CNN. Next, computing resource allocation and CFNN partition decisions are generated on each edge server based on deep reinforcement learning (DRL). At last, optimal task offloading strategies are obtained through many-to-one matching game.

The main contributions of this paper are shown as follows:

- Design a novel CFNN to improve training efficiency and reduce uncertainty during validation, and predict calculation amount of the fuzzy layer based on polynomial regression.
- Propose a DRL-based algorithm to dynamically dispatch computing resources and partition CFNN to maximize the total IoT customer utility.
- Design a matching game-based algorithm to offload CFNN inference tasks from IoT devices with the goal of balancing profit of IoT customers and service providers.
- The effectiveness of the proposed framework is confirmed through extensive experiments.

The remaining parts of this paper are presented as follows. Section II illustrates the related work. In Section III, the system model is described and the optimization problem is formulated. Section IV presents the design of DisCFNN. In Section V, experiment settings are elaborated and simulation results are analyzed. In Section VI, we conclude the paper.

II. Related Work

In this section, the existing researches related to our work are reviewed from the aspects of distributed inference empowered by edge computing and studies on DNFS.

Considering limited resources on end devices and long data transmission distance between end devices and edge or cloud servers, either local computing or pure edge/cloud computing cannot efficiently reduce response time of intelligent applications for the intolerable computing latency on end devices or huge transmission latency of raw input between end devices and servers. So distributed inference empowered by edge computing becomes a promising method to reduce DNN inference latency and attracts considerable attention. Hu et al. [9] proposed a dynamic adaptive model partition scheme based on the min-cut algorithm to reduce inference latency. The proposed scheme can be used to partition DNN with a structure of directed acyclic graph, and this scheme supports automatic adjustment of partition strategies according to network fluctuation between devices. Yang et al. [10] designed an edge-cloud collaborative inference system which joint model compression with model partition together. The designed system partitions a DNN into three parts to adequately use resources on end device, edge server and the cloud. Additionally, this system reduces the size of transmitted data between devices through model compression to further decrease the total inference latency. Chen et al. [8] minimized the system energy consumption while ensuring reasonable service response time for each customer through making appropriate model partition strategies based on Genetic Particle Swarm Optimization algorithm. Zeng et al. [11] proposed a data parallelism system for parallel inference of different parts in original input on heterogeneous devices. The proposed system efficiently CNN inference latency and energy consumption. However, these optimization schemes for inference are all designed for DNN, which possess black-box nature and lack interpretability.

Combining excellent learning ability of DNN and high interpretability of fuzzy logic, DNFS models converge faster during training and show less instability during validation, which present higher reliability and more suitable to be used as models of intelligent services in big data-based IoT. In order to enhance machines' ability to interact with human, Chen et al. [12] designed a novel deep FNN based on fuzzy C-means and deep neural network with sparse autoencoder for intention understanding according to people's emotions. Guan et al. [13] proposed a tailored CFNN to achieve accurate human lip region segmentation for lip pictures with various kinds of noise. Yeganejou et al. [14] designed a deep convolutional fuzzy classifier based on CNN and fuzzy Rocchio's algorithm to improve accuracy of classification results. Yazdinejad et al. [15] proposed a mized fuzzy deep learning model based on Non-Dominated Sorting Genetic Algorithm II to improve classification accuracy for pictures with a large number of unrelated and redundant features.

However, to the best of our knowledge, existing studies on DNFS models all focus on the accuracy improvement of output. To this end, we propose an end-edge collaborative inference framework of CFNNs for big data-driven IoT to balance profit of customers and service providers while ensuring acceptable CFNN inference latency and reasonable edge server energy consumption.

III. System Model and Problem Formulation

In this section, the system model is introduced first. Then, the models of IoT customer utility and edge server utility are established. At last, the weighted-sum optimization is formulated to balance the total utility between IoT customers and service providers while ensuring acceptable CFNN inference latency and reasonable edge server energy consumption.

A. System Model

As shown in Fig. 1, the big data-driven IoT is divided into two layers: end layer and edge layer. The end layer is a set of IoT devices, each corresponding to an IoT customer. In an IoT application, raw data from the corresponding customer is

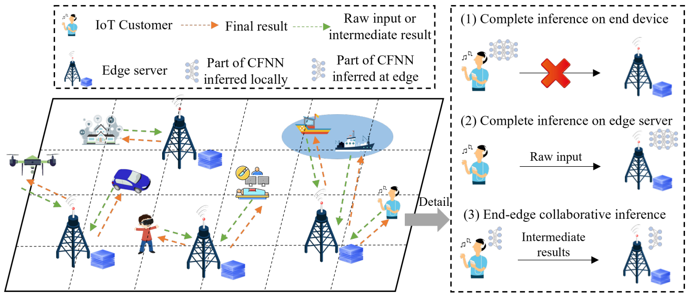

IoT Customer
Final result
Raw input or intermediate result
Edge server
Part of CFNN inferred locally
Part of CFNN inferred at edge
Detail
(1) Complete inference on end device
(2) Complete inference on edge server
Raw input
(3) End-edge collaborative inference
Intermediate results

Fig. 1. The system model of end-edge collaborative CFNN inference for big-data driven IoT.

used as input of a CFNN which then returns the inference result. Each IoT device has a few computing resources, and can infer a part of a CFNN or a whole CFNN. The edge layer consists of a set of edge servers which are service providers and provide CFNN inference services for IoT customers. The computing resources on edge servers are more than those on IoT devices.

In big data-driven IoT, let $E = \{e_1, e_2, \cdots, e_M\}$ denote the set of IoT devices, and let $S = \{s_1, s_2, \cdots, s_N\}$ denote the set of edge servers. M and N respectively represent the total number of IoT devices and edge servers in the IoT. The task offloading strategy for IoT device e_i is denoted as $Z(e_i) \in \{-1, S\}$, and $Z(e_i) = -1$ means that IoT device e_i completely infers the CFNN locally. Similarly, the set of IoT devices sending CFNN inference service requests to edge server s_j is denoted as $Z(s_j) \in E$. The CFNN is mainly made up of the fuzzy layer, convolution layers and fully connected layers. The CFNN where the IoT application on device e_i is based is denoted as $D_i = \{d_{i,0}, d_{i,1}, d_{i,2}, \cdots, d_{i,L_i}\}$. L_i is total number of layers in CFNN D_i, and $d_{i,l}$ is calculation amount of the lth layer in CFNN D_i. Note that $d_{i,0} = 0$.

Assume that all kinds of CFNN possible to be used in the big-data driven IoT have already been deployed on each edge server in advance.

B. Customer Utility Model

The utility of an IoT customer is dependent on CFNN inference latency and cost on 'purchasing' computing resources from the corresponding server where the CFNN inference service request is offloaded.

1) CFNN Inference Latency: CFNN inference latency includes three parts, which are computing latency on the IoT device, data transmission latency between end and edge, and computing latency on the edge server. Inference latency of CFNN D_i is given by

$$t_i^{end} = \frac{c_i^{req}}{f_i}, \tag{1}$$

where c_i^{req} is the floating-point operations (FLOPs) of the part of CFNN D_i inferred on IoT device e_i, f_i is computing resources of IoT device e_i. Similarly, inference latency of CFNN D_i on edge server s_j is given by

$$t_j^{server} = \frac{c_{j,i}^{req}}{f_{j,i}}, \tag{2}$$

where $c_{j,i}^{req}$ is FLOPs of the part of CFNN D_i inferred on edge server s_j, $f_{j,i}$ is computing resources allocated to IoT device e_i by edge server s_j. We assume that all IoT devices and edge servers in the IoT perform computing tasks based on CPU cores [16] whose computing capability is measured by floating-point operations per second (FLOPS), and a unit of computing resource is abstracted as one FLOPS here. Moreover, c_i^{req} and $c_{j,i}^{req}$ are given by

$$c_i^{req} = \sum_{k=0}^{p_i} d_{i,k}, \tag{3}$$

$$c_{j,i}^{req} = \sum_{k=p_i}^{L} d_{i,k}, \tag{4}$$

where p_i is the partition point of CFNN D_i. The FLOPs of convolution layers and fully connected layers are easy to calculate. If layer l is a convolution layer, $d_{i,l}$ is given by

$$d_{i,l} = 2 \cdot (c_{in} \cdot w \cdot h) \cdot (c_{out} \cdot w_{out} \cdot h_{out}), \tag{5}$$

where c_{in}, w, h are number of channels, width and height of convolution layer l, respectively. c_{out}, w_{out}, h_{out} are number of channels, width and height of output from convolution layer l, respectively. If layer l is a fully connected layer, $d_{i,l}$ is given by

$$d_{i,l} = 2 \cdot n_{in} \cdot n_{out}, \tag{6}$$

where n_{in}, n_{out} are feature number of input and output of fully connected layer l, respectively. However, because of complex arithmetical operation such as exponent in the fuzzy layer, the FLOPs of the fuzzy layer are hard to be directly calculated like convolution layers and fully connected layers.

During the process of end-edge collaborative CFNN inference, the transmission latency of raw input or intermediate results from IoT device e_i to edge server s_j is given by

$$t_{j,i}^{tran} = \frac{o_{j,i}}{r_{j,i}}, \tag{7}$$

where $o_{j,i}$ is size of data transmitted from IoT device e_i to edge server s_j, $r_{j,i}$ is data transmission speed between IoT device e_i and edge server s_j. It is emphasized that considering the small size of final results, transmission latency from edge to end is neglected here [17].

2) Customer Utility: To compute the utility of customers, we introduce an economic concept named 'satisfaction level', which is measured by a logarithmic function. Satisfaction level has been widely used to assess the quality of task offloading in former studies [18]–[20]. Based on those studies, normative satisfaction level is used to assess the relative speed of CFNN inference for IoT device e_i, which is presented as

$$\Phi_i^{Z(e_i)} = \frac{log(1 + T_i^{max} - T_i^{Z(e_i)})}{log(1 + T_i^{max})}, \tag{8}$$

where T_i^{max} is the permitted maximum inference latency of CFNN D_i. $T_i^{Z(e_i)}$ is the total inference latency of CFNN D_i, which is obtained through Eqs. (1), (2) and (7), and is presented as

$$T_i^{z_i} = \begin{cases} t_i^{end} + t_{Z(e_i)}^{server} + t_{Z(e_i),i}^{tran} & Z(e_i) \in S, \\ t_i^{end} & Z(e_i) = -1, \end{cases} \tag{9}$$

If a customer sends request of CFNN inference service to an edge server, the server must allocate a number of computing resources to infer the whole CFNN or a part of it, which causes cost for the customer. The cost for IoT device e_i is presented as

$$\Theta_i^{Z(e_i)} = \begin{cases} \frac{k_{Z(e_i),i} \cdot f_{Z(e_i),i}}{K_i^{max}} & Z(e_i) \in S, \\ 0 & Z(e_i) = -1, \end{cases} \tag{10}$$

where $k_{Z(e_i),i}$ is the unit price of computing resources charged by edge server $Z(e_i)$, K_i^{max} is the budget of IoT device e_i for the cost payed to edge servers.

Therefore, the utility of IoT device e_i is given by

$$U_i^{Z(e_i)} = \alpha_i^{Z(e_i)} \cdot \Phi_i^{Z(e_i)} - (1 - \alpha_i^{Z(e_i)}) \cdot \Theta_i^{Z(e_i)}, \tag{11}$$

where $\alpha_i^{Z(e_i)} \in [0, 1]$ is the weight coefficient.

C. Server Utility Model

The utility of an edge server to a customer is dependent on revenue the server gets through 'selling' computing resources to the customer and the energy consumption of the server when processing the CFNN inference service request from the customer.

1) Server Energy Consumption: Referring to former study [20], the energy consumption of edge server s_j to infer CFNN D_i is given by

$$E_j^i = \mu \cdot (f_{j,i})^{\tau - 1} \cdot c_{j,i}^{req}, \tag{12}$$

where μ is the effective switched capacitance of CPU [21] and is set as 7.8^{-21} in this work, τ is a constant and is set as 3 in this work [22].

2) Server Utility: An edge server sets a unit price of computing resources for each customer, and obtains revenue from payment of customers. The normative revenue of edge server s_j to infer service a_i is given by

$$R_j^i = \frac{k_{j,i} \cdot f_{j,i}}{K_j^{max} \cdot f_j^{max}}, \tag{13}$$

where K_j^{max} is the pre-set maximum unit price of computing resources in server s_j, which is always larger than the upper bound of the actual value range of unit resource price $k_{j,i}$. f_j^{max} is total number of computing resources in server e_j.

The normative energy consumption of edge server s_j to infer CFNN D_i is given by

$$\Theta_j^i = \frac{E_j^i}{E_j^{max}}, \tag{14}$$

where E_j^{max} is the permitted maximum energy consumption of edge server s_j to infer a CFNN.

The utility of edge server s_j to infer CFNN D_i is given by

$$U_j^i = \beta_j^i \cdot R_j^i - (1 - \beta_j^i) \cdot \Theta_j^i, \tag{15}$$

where $\beta_j^i \in [0, 1]$ is the weight coefficient.

D. Problem Formulation

The objective of this work is balancing the total utility between IoT customers and service providers while ensuring acceptable CFNN inference latency and reasonable edge server energy consumption, which is achieved by jointly optimizing computing resource allocation, CFNN partition, and task offloading strategies. The optimization problem is formulated as

$$\max_{(S,p,f,c)} \gamma \cdot \sum_{i=1}^{M} U_i^{Z(e_i)} + (1 - \gamma) \cdot \sum_{j=1}^{N} \sum_{i=1}^{M} U_j^i \cdot \Omega(Z(e_i)). \tag{16}$$

$$s.t. \quad p_i \in [0, L], \forall i \in [1, M], \tag{16a}$$

$$\sum_{i=1}^{M} f_{j,i} \cdot \Omega(Z(e_i)) \le f_j^{max}, j \in [1, N], \tag{16b}$$

$$k_{Z(e_i),i} \cdot f_{Z(e_i),i} \le K_i^{max}, \forall i \in [1, M], \tag{16c}$$

$$T_i^{Z(e_i)} \le T_i^{max}, \forall i \in [1, M], \tag{16d}$$

$$E_j^i \le E_j^{max}, \forall i \in [1, M], Z(e_i) = j, \forall j \in [1, N], \tag{16e}$$

where $\gamma \in [0, 1]$ is the weight coefficient, and $\Omega(Z(e_i)) = \begin{cases} 1 & Z(e_i) = j, \\ 0 & else, \end{cases}$. Constraint (16a) means that the value of partition point of a CFNN is not negative and not larger than total number of layers in the CFNN. Constraint (16b) means that the number of computing resources allocated by an edge server does not exceed the total number of computing resources on that server. Constraint (16c) ensures that the cost payed by an IoT device is not larger than its maximum budget when 'purchasing' computing resources from an edge server. Constraint (16d) guarantees that the inference latency of a CFNN does not exceed its maximum permitted latency. Constraint (16e) guarantees that the energy consumption of an edge server to infer a CFNN does not exceed the corresponding permitted maximum energy consumption.

IV. DisCFNN Design

In this section, the implementation of DisCFNN is presented. As illustrated in Fig. 2, DisCFNN composes of three parts. First, a novel CFNN structure is designed. Next, a DRL-based algorithm, named DRAP, is presented to allocate computing resources and make CFNN partition strategies on each edge server. Finally, a matching game-based algorithm is proposed to offload CFNN inference tasks from IoT customers to different edge servers.

A. CFNN Structure Design

Design of the CFNN structure is an offline process. As shown in the left part of Fig. 2, the designed CFNN is mainly made up of three parts, which are a fuzzy layer, several convolution layers and and several fully connected layers. The functions of convolutions layers and fully connected layers are the same with those in convolutional neural networks. The fuzzy layer is described in detail below.

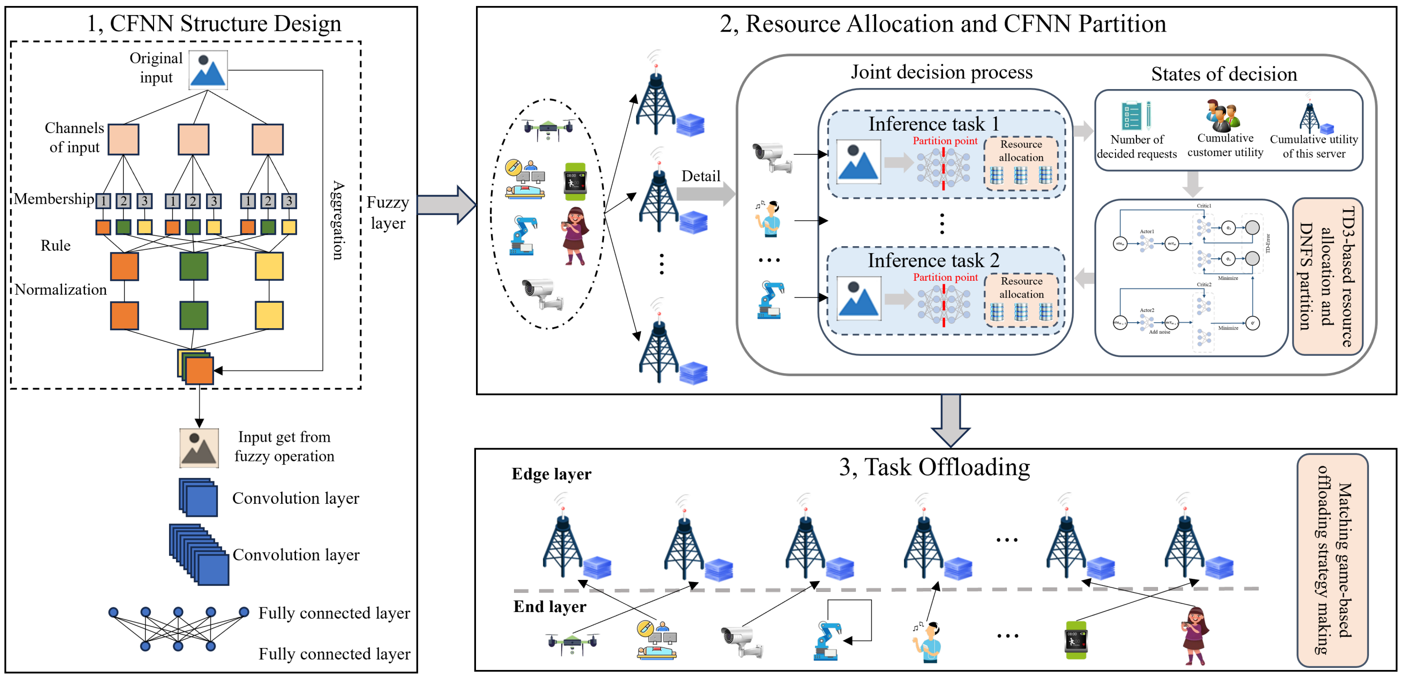


Fig. 2. The architecture of DisCFNN.

1) Working Mechanism of Fuzzy Layer: Function of the fuzzy layer in the CFNN designed in this paper is adding 'fuzzy features' to the original input, by which the training efficiency is improved and the interpretability of network is enhanced. Based on the theory of fuzzy neural network (FNN), the designed fuzzy layer consists of four parts, i.e. membership layer, rule layer, normalization layer and aggregation layer. Referring to the former study [23], for each channel of the original input, the membership of each pixel is calculated using membership functions in the membership layer. The three membership functions for all channels of input is the same and are given by

$$N = e^{-\frac{(x+\xi)^2}{2\sigma^2}}, \tag{17}$$

$$Z = e^{-\frac{x^2}{2\sigma^2}}, \tag{18}$$

$$P = e^{-\frac{(x-\xi)^2}{2\sigma^2}}, \tag{19}$$

where x is the value of a pixel. The center point ξ and width σ are fixed values. It is obvious that three new channels are produced when each input channel is processed by the membership layer. In the rule layer of classical FNN, such as Takagi-Sugeno FNN [24], memberships of all input features are mutually multiplied in all possible combinations, whose time complexity is $O(v^g)$, where v is the number of each feature's memberships and g is the number of features. The time complexity of rule layer in classical FNN is so high, which brings huge computing load devices with limited computing resources. To reduce the computing amount of the rule layer, in our design, we group all new channels produced by the membership layer, and all new channels produced by the same membership function are in the same group. For the new channels in the same group, pixels at the same location of those channel are multiplied with each other. The time complexity of the fuzzy layer in our CFNN is $O(v \cdot (c_{in} - 1) \cdot \frac{g}{c_{in}})$, which is much lower than that of the fuzzy layer in classical FNN. The size output of the rule layer in our CFNN is the same with original input. After normalization, output of the fuzzy layer is aggregated with the original as 'fuzzy features', through addition operation. The obtained result is then processed by convolution layers and fully connected layers.

2) FLOPs of Fuzzy Layer: Different from convolution layers and fully connected layer which only include add and multiply operations, the fuzzy layer also includes exponent, so it is hard to directly calculate FLOPs of the fuzzy layer.

In order to estimate the computing amount of the fuzzy layer, we first infer the fuzzy layer on our server with various number of CPU cores. Under each tested number of CPU cores, we record the average inference latency of the fuzzy layer, through which a prediction model for each number of CPU cores possible to be allocated in an edge server is established using polynomial regression. By using the prediction model, we can get the average 'fake FLOPs' of the fuzzy layer d_{fake} with the formula below

$$d_{fake} = \frac{\sum_{n=1}^{maxi} t_n \cdot f_n}{n}, \tag{20}$$

where $maxi$ is the maximum number of CPU cores in servers, t_n is inference latency of the fuzzy layer when n CPU cores are allocated, f_n is number of computing resources, i.e. FLOPS, in the n CPU cores. Note that the 'fake FLOPs' of the fuzzy layer d_{fake} is far from the actual FLOPs, because utilization ratio of resources in each CPU core is so hard to reach 100%, and the actual inference latency t_n is usually much higher than the theoretical value. So, to get the approximate actual FLOPs of the fuzzy layer, we can get the average 'fake

FLOPs' of convolution layers and fully connected layers, and calculate the average ratio of actual FLOPs to 'fake FLOPs' of convolution layers and fully connected layers, which is then multipled with the 'fake FLOPs' of the fuzzy layer d_{fake}.

B. Resource Allocation and CFNN Partition Based on DRL

We consider a big data-driven IoT with several IoT customers and a fixed number of edge servers. Whenever the network condition or the customer number in the IoT changes, each IoT device sends the customer information, including CFNN to be inferred and permitted maximum inference latency, to all edge servers. After receiving all customer information, each edge server temporarily assumes that all CFNN are partially inferred on itself. In this case, each edge server jointly generates resource allocation and CFNN partition decisions based on DRL for all customers.

Note that the final task offloading strategies have not been obtained, which means that the resource allocation strategies of edge servers cannot be directly made. To solve this problem, a concept named 'resource pricing' is introduced. The resource allocation problem is converted to making a reasonable unit price for resources allocated to each IoT device, then the number of computing resources allocated to each IoT device is achieved through the corresponding unit resource price and maximization of the customer's utility. The detailed derivation for the number of computing resources allocated to IoT device e_i by edge server s_j is shown below.

From Eqs. (11), we know that the utility of IoT device e_i when service a_i is partially offloaded to server z_i is given by

$$\begin{aligned}
&U_i^{Z(e_i)}\\
=&\alpha_i^{Z(e_i)} \cdot \Phi_i^{Z(e_i)} + (1-\alpha_i^{Z(e_i)}) \cdot \Theta_i^{Z(e_i)}\\
=&\alpha_i^{Z(e_i)} \cdot \frac{log(1+T_i^{max}-(t_i^{end}+t_{Z(e_i)}^{server}+t_{Z(e_i),i}^{tran}))}{log(1+T_i^{max})}\\
&\quad + (1-\alpha_i^{Z(e_i)}) \cdot \frac{k_{Z(e_i),i}\cdot f_{Z(e_i),i}}{K_i^{max}}\\
=&\alpha_i^{Z(e_i)} \cdot \frac{log(1+T_i^{max}-(\frac{c_i^{req}}{f_i}+\frac{c_{Z(e_i),i}^{req}}{f_{Z(e_i),i}}+\frac{o_{Z(e_i),i}}{r_{Z(e_i),i}}))}{log(1+T_i^{max})}\\
&\quad + (1-\alpha_i^{Z(e_i)}) \cdot \frac{k_{Z(e_i),i}\cdot f_{Z(e_i),i}}{K_i^{max}},
\end{aligned}$$

the derivative of $U_i^{Z(e_i)}$ with respect to allocated computing resources $f_{Z(e_i),i}$ is calculated as

$$\begin{aligned}
&\frac{\partial U_i^{Z(e_i)}}{\partial f_{Z(e_i),i}}\\
=&\alpha_i^{Z(e_i)} \cdot \frac{c_{Z(e_i),i}^{req}}{f_{Z(e_i),i}^2}\\
&\cdot \frac{1}{(1+T_i^{max}-\frac{c_i^{req}}{f_i}-\frac{o_{Z(e_i),i}}{r_{Z(e_i),i}}-\frac{c_{Z(e_i),i}^{req}}{f_{Z(e_i),i}})\cdot log(1+T_i^{max})}\\
&+ (1-\alpha_i^{Z(e_i)}) \cdot \frac{k_{Z(e_i),i}}{K_i^{max}},
\end{aligned}$$

further, the second-order derivative of $U_i^{Z(e_i)}$ with respect to allocated computing resources $f_{Z(e_i),i}$ is calculated as

$$\begin{aligned}
&\frac{\partial^2 U_i^{Z(e_i)}}{\partial^2 f_{Z(e_i),i}}\\
=&\alpha_i^{Z(e_i)} \cdot \frac{1}{log(1+T_i^{max})}\\
&\cdot (\frac{\frac{-2c_{Z(e_i),i}^{req}}{f_{Z(e_i),i}^3}\cdot(1+T_i^{max}-\frac{c_i^{req}}{f_i}-\frac{o_{Z(e_i),i}}{r_{Z(e_i),i}}-\frac{c_{Z(e_i),i}^{req}}{f_{Z(e_i),i}})}{(1+T_i^{max}-\frac{c_i^{req}}{f_i}-\frac{o_{Z(e_i),i}}{r_{Z(e_i),i}}-\frac{c_{Z(e_i),i}^{req}}{f_{Z(e_i),i}})^2}\\
&- \frac{\frac{(c_{Z(e_i),i}^{req})^2}{f_{Z(e_i),i}^4}}{(1+T_i^{max}-\frac{c_i^{req}}{f_i}-\frac{o_{Z(e_i),i}}{r_{Z(e_i),i}}-\frac{c_{Z(e_i),i}^{req}}{f_{Z(e_i),i}})^2})\\
=&\alpha_i^{Z(e_i)} \cdot \frac{1}{log(1+T_i^{max})}\\
&\cdot \frac{\frac{-2c_{Z(e_i),i}^{req}}{f_{Z(e_i),i}^3}\cdot(1+T_i^{max}-\frac{c_i^{req}}{f_i}-\frac{o_{Z(e_i),i}}{r_{Z(e_i),i}})+\frac{(c_{Z(e_i),i}^{req})^2}{f_{Z(e_i),i}^4}}{(1+T_i^{max}-\frac{c_i^{req}}{f_i}-\frac{o_{Z(e_i),i}}{r_{Z(e_i),i}}-\frac{c_{Z(e_i),i}^{req}}{f_{Z(e_i),i}})^2}\\
=&\alpha_i^{Z(e_i)} \cdot \frac{1}{log(1+T_i^{max})}\\
&\cdot (\frac{c_{Z(e_i),i}^{req}\cdot(-2f_{Z(e_i),i}\cdot(1+T_i^{max}-\frac{c_i^{req}}{f_i}-\frac{o_{Z(e_i),i}}{r_{Z(e_i),i}}))}{f_{Z(e_i),i}^4\cdot(1+T_i^{max}-\frac{c_i^{req}}{f_i}-\frac{o_{Z(e_i),i}}{r_{Z(e_i),i}}-\frac{c_{Z(e_i),i}^{req}}{f_{Z(e_i),i}})^2}\\
&- \frac{c_{Z(e_i),i}^{req}\cdot(-c_{Z(e_i),i}^{req})}{f_{Z(e_i),i}^4\cdot(1+T_i^{max}-\frac{c_i^{req}}{f_i}-\frac{o_{Z(e_i),i}}{r_{Z(e_i),i}}-\frac{c_{Z(e_i),i}^{req}}{f_{Z(e_i),i}})^2}),
\end{aligned}$$

according to $log(1+T_i^{max}-T_i^{Z(e_i)})$, we can get that

$$\begin{aligned}
&1+T_i^{max}-T_i^{Z(e_i)}>0\\
\Rightarrow & T_i^{Z(e_i)}<1+T_i^{max}\\
\Rightarrow & \frac{c_i^{req}}{f_i}+\frac{o_{Z(e_i),i}}{r_{Z(e_i),i}}+\frac{c_{Z(e_i),i}^{req}}{f_{Z(e_i),i}}<1+T_i^{max}\\
\Rightarrow & \frac{o_{Z(e_i),i}}{r_{Z(e_i),i}}<1+T_i^{max}-\frac{c_i^{req}}{f_i}-\frac{c_{Z(e_i),i}^{req}}{f_{Z(e_i),i}}\\
&\qquad\quad <1+T_i^{max}-\frac{c_i^{req}}{f_i}-\frac{c_{Z(e_i),i}^{req}}{2f_{Z(e_i),i}}\\
\Rightarrow & \frac{o_{Z(e_i),i}}{r_{Z(e_i),i}}-1-T_i^{max}+\frac{c_i^{req}}{f_i}<\frac{c_{Z(e_i),i}^{req}}{-2f_{Z(e_i),i}}\\
\Rightarrow & -2f_{Z(e_i),i}\cdot(\frac{o_{Z(e_i),i}}{r_{Z(e_i),i}}-1-T_i^{max}+\frac{c_i^{req}}{f_i})>c_{Z(e_i),i}^{req}\\
\Rightarrow & -2f_{Z(e_i),i}\cdot(-\frac{o_{Z(e_i),i}}{r_{Z(e_i),i}}+1+T_i^{max}-\frac{c_i^{req}}{f_i})<-c_{Z(e_i),i}^{req},
\end{aligned}$$

note that in $\frac{\partial^2 U_i^{Z(e_i)}}{\partial^2 f_{Z(e_i),i}}$, $\alpha_i^{Z(e_i)} \geq 0$, $c_{Z(e_i),i}^{req} \geq 0$, $f_{Z(e_i),i}^4 \cdot (1+T_i^{max}-\frac{c_i^{req}}{f_i}-\frac{o_{Z(e_i),i}}{r_{Z(e_i),i}}-\frac{c_{Z(e_i),i}^{req}}{f_{Z(e_i),i}})^2 \geq 0$, $-2f_{Z(e_i),i}\cdot(-\frac{o_{Z(e_i),i}}{r_{Z(e_i),i}}+1+T_i^{max}-\frac{c_i^{req}}{f_i})-(-c_{Z(e_i),i}^{req}) \leq 0$. So $\frac{\partial^2 U_i^{Z(e_i)}}{\partial^2 f_{Z(e_i),i}} \leq 0$, which means that $U_i^{Z(e_i)}$ is a concave function. As a result, the maximum value of $U_i^{Z(e_i)}$ exists. In order to calculate the value of $f_{Z(e_i),i}$ where $U_i^{Z(e_i)}$ gets the maximum value, we let $\frac{\partial U_i^{Z(e_i)}}{\partial f_{Z(e_i),i}}$ be equal to 0. Through the quadratic formula, the optimal number of computing resources $f^*_{Z(e_i),i}$ allocated by server $Z(e_i)$ to IoT device e_i is given by

$$f^*_{Z(e_i),i} = \frac{\zeta \cdot c^{req}_{Z(e_i),i} \pm \sqrt{\zeta^2 \cdot (c^{req}_{Z(e_i),i})^2 + 4\zeta \cdot \delta}}{2\zeta \cdot (1 + T_i^{max} - \frac{c_i^{req}}{f_i} - \frac{o_{Z(e_i),i}}{r_{Z(e_i),i}})}, \tag{21}$$

where $\delta = (1 + T_i^{max} - \frac{c_i^{req}}{f_i} - \frac{o_{Z(e_i),i}}{r_{Z(e_i),i}}) \cdot \alpha_i^{Z(e_i)} \cdot \frac{K_i^{max} \cdot c^{req}_{Z(e_i),i}}{k_{Z(e_i),i} \cdot log(1+T_i^{max})}$, $\zeta = 1 - \alpha_i^{Z(e_i)}$. Considering that $\zeta \cdot c^{req}_{Z(e_i),i} > 0$ and $\zeta \cdot \delta > 0$, it can be obtained that $\zeta \cdot c^{req}_{Z(e_i),i} = \sqrt{\zeta^2 \cdot (c^{req}_{Z(e_i),i})^2} < \sqrt{\zeta^2 \cdot (c^{req}_{Z(e_i),i})^2 + 4\zeta \cdot \delta}$. As a result, to ensure $f^*_{Z(e_i),i}$ is not negative, we get $f^*_{Z(e_i),i} = \frac{\zeta \cdot c^{req}_{Z(e_i),i} + \sqrt{\zeta^2 \cdot (c^{req}_{Z(e_i),i})^2 + 4\zeta \cdot \delta}}{2\zeta \cdot (1 + T_i^{max} - \frac{c_i^{req}}{f_i} - \frac{o_{Z(e_i),i}}{r_{Z(e_i),i}})}$.

Through customer utility U_i^j and server utility U_j^i, the range of unit price of computing resources $k_{j,i}$ allocated to IoT device e_i by edge server s_j can be achieved. Specifically, in order to make customer utility U_i^j greater than 0, we get

$$\begin{aligned} &\alpha_i^j \cdot \Phi_i^j - (1 - \alpha_i^j) \cdot \Theta_i^j > 0 \\ \Rightarrow &\alpha_i^j \cdot \frac{log(1 + T_i^{max} - T_i^j)}{log(1 + T_i^{max})} > (1 - \alpha_i^j) \cdot \frac{k_{j,i} \cdot f_{j,i}}{K_i^{max}} \\ \Rightarrow &k_{j,i} < \frac{\alpha_i^j \cdot \frac{log(1+T_i^{max} - T_i^j)}{log(1+T_i^{max})} \cdot K_i^{max}}{(1 - \alpha_i^j) \cdot f_{j,i}} < \frac{\alpha_i^j \cdot K_i^{max}}{(1 - \alpha_i^j) \cdot f'_{j,i}}, \end{aligned}$$

where $f'_{j,i}$ is the total computing resources in a single CPU core. Note that $\frac{\alpha_i^j \cdot K_i^{max}}{(1-\alpha_i^j) \cdot f'_{j,i}} < K_{j,i}^{max}$ is always tenable. Similarly, in order to make server utility U_j^i greater than 0, we get

$$\begin{aligned} &\beta_j^i \cdot R_j^i - (1 - \beta_j^i) \cdot \Theta_j^i > 0 \\ \Rightarrow &\beta_j^i \cdot \frac{k_{j,i} \cdot f_{j,i}}{K_j^{max} \cdot f_j^{max}} > (1 - \beta_j^i) \cdot \frac{E_j^i}{E_j^{max}} \\ \Rightarrow &k_{j,i} > \frac{(1 - \beta_j^i) \cdot E_j^i \cdot K_j^{max} \cdot f_j^{max}}{\beta_j^i \cdot E_j^{max} \cdot f_{j,i}} \\ \Rightarrow &k_{j,i} > \frac{(1 - \beta_j^i) \cdot \mu \cdot (f_{j,i})^{\tau - 2} \cdot c_{j,i}^{req} \cdot K_j^{max} \cdot f_j^{max}}{\beta_j^i \cdot E_j^{max}} > 0. \end{aligned}$$

DRPA, the DRL-based algorithm for resource allocation and CFNN partition, is described now.

State. The state space presents the observation of the resource allocation and CFNN partition decision making process, including the number of IoT devices for which the resource allocation and CFNN partition decisions have been generated, cumulative customer utility and cumulative edge server utility. For an edge server, the state sta_i is given by

$$sta_i = \{n_i, U_i^{cus}, U_i^{ser}\}, \tag{22}$$

where n_i is the number of services for which the resource allocation and CFNN partition decisions have been made, U_i^{cus} is cumulative customer utility, U_i^{ser} is cumulative utility of the edge server.

Action. Based on states, the server generates reasonable computing resource price and CFNN partition decision for each IoT device. The action act_i is given by

$$act_i = \{r_{k_{j,i}}, r_{p_i}\}, \tag{23}$$

where $r_{k_{j,i}}$ is resource pricing ratio, r_{p_i} is CFNN partition ratio.

Reward. Based in Eqs. 16, the reward of the DRL-based algorithm in server s_j is given by

$$reward = \gamma \cdot U_i^j \cdot \Gamma(U_i^j) + (1 - \gamma) \cdot U_j^i \cdot \Gamma(U_j^i), \tag{24}$$

where $\Gamma(x) = \begin{cases} \eta & x < 0, \\ 1 & x \geq 0, \end{cases}$, and Γ is a penalty term which helps improve the possibility of U_i^j and U_j^i to be greater than 0.

It is obvious that $n_i = n_{i-1} + 1$, $U_i^{cus} = U_{i-1}^{cus} + U_i^j$, $U_i^{ser} = U_{i-1}^{ser} + U_j^i$, from which we learn that the current state is only decided by the last state and is not related to earlier states. It means that the Markov property is met and DRL is applicable here. DRAP is based on Twin Delayed Deep Deterministic policy gradient algorithm (TD3) in DRL [25], [26], and is elaborated in Algorithm 1.

Algorithm 1: DRAP.

```
Input : Initial state $sta_0$, number of services $n$
Output: Resource allocation decision $f$, CFNN
        partition decision $p$
1  Initialize critic networks $Q_{\theta_1}$, $Q_{\theta_2}$, actor network $\pi_\emptyset$,
     target critic networks $Q_{\theta'_1}$, $Q_{\theta'_2}$, target actor network
     $\pi_{\emptyset'}$, replay buffer $\rho$;
2  for $episode = 1$ to $T$ do
3      for $i = 1$ to $N$ do
4          if $episode < \tau$ then
5              $act_i = \varepsilon$;
6          end
7          else
8              $act_i = \pi_\emptyset(sta_i)$;
9          end
10         Observe reward $r_i$ and the next state $sta_{i+1}$
             and store transition $(sta_i, act_i, r_i, sta_{i+1})$ to
             $\rho$;
11         if $episode \geq \tau$ then
12             Update critics $\theta_1$, $\theta_2$;
13             if $episode$ % $v == 0$ then
14                 Update actor $\emptyset$;
15                 Update target networks $Q_{\theta'_1}$, $Q_{\theta'_2}$, $\pi_{\emptyset'}$;
16             end
17         end
18     end
19 end
20 return $f$, $p$;
```

As illustrated in Algorithm 1, parameters of critics, actor, target critics and target actor, as well as the replay buffer, are initialized in the beginning (lines 1). In terms of the producing of actions, random values are endowed during the former τ training episodes in order to encourage the agent to explore (lines 4 to 9). Note that the two elements in the action space, which are computing resource pricing ratio and CFNN partition ratio, both range from 0 to 1. When pricing for each unit of computing resources, $k_{j,i}$ is calculated as $r_{k_{j,i}} \cdot \frac{\alpha_i^j \cdot K_i^{max}}{(1-\alpha_i^j)\cdot f'_{j,i}} \cdot \kappa$, where κ is a discount factor. The partition point p_i of CFNN D_i is calculated as $[r_{p_i} \cdot L_i]$. Based on the obtained resource price and partition point, the reward r_i and the next sta_{i+1} is achieved, transition $(sta_i, act_i, r_i, sta_{i+1})$ is stored to the replay buffer (line 10). After the τth episode of training, two critics start to be updated (lines 11 to 12). Delayed update is used for the actor, which is updated together with the parameters of target networks (lines 13 to 19).

Algorithm 2: Matching game-based task offloading.

```
Input : The set of IoT devices $E$, the set of edge servers $S$
Output: The optimal matching strategy $Z$
1  Initialize $E_{rejected}$ with $E$, $E_{rejected\_}$ with $\emptyset$;
2  for $e_i \in E$ do
3      for $s_j \in S$ do
4          Get $f^*_{j,i}$, $p_i$ based on Algorithm 1;
5          Calculate preference value of $s_j$ on $e_i$;
6          Calculate preference value of $e_i$ on $s_j$;
7      end
8  end
9  Construct preference lists of IoT devices and edge servers;
10 while $E_{rejected}$ is not empty do
11     if $E_{rejected}$ is equal to $E_{rejected\_}$ then
12         break;
13     end
14     for $e_i \in E_{rejected}$ do
15         Select the most preferred server $s_{j^*}$;
16         Update the matching list of $e_i$;
17         Update the matching list of $s_{j^*}$;
18     end
19     $E_{rejected\_}$ = $E_{rejected}$;
20     for $s_j \in S$ which receives new service requests do
21         Adjust matching list of $s_j$;
22         Update $E_{rejected}$;
23         for $e_i \in E_{rejected}$ do
24             Update the preference list and the matching list of $e_i$;
25         end
26     end
27 end
28 return $Z$;
```

C. Task Offloading Based on Matching Game

According to resource allocation and CFNN partition decisions generated on each server, task offloading strategies are made. In the big data-driven IoT, each IoT device sends CFNN inference request to one edge server while each edge server processes service requests from several IoT devices. So the task offloading in the big data-driven IoT is regarded as a many-to-one matching game.

The matching game is described as a a triplet of (X, Y, Z). $X = (E_{rejected}, S)$ is two disjoint sets of players in the matching game, where $E_{rejected}$ denotes the set of IoT devices whose task offloading strategy has not been obtained, and S is the set of edge servers. $Y = (Y_i, Y_j)$ is preference lists of IoT devices and edge servers. Each IoT device $e_i \in E_{rejected}$ has a descending ordered preference list Y_i on edge servers, where $Y_i = \{s_j | s_j \in S, s_j \succ_i s_{j'}\}$ and $\succ_i$ denotes the preference of IoT device e_i towards edge servers. Likewise, each edge server $s_j \in S$ has a descending ordered preference list Y_j on IoT devices, where $Y_j = \{e_i | e_i \in E_{rejected}, e_i \succ_j e_{i'}\}$. $Z = (Z(e_i), Z(s_j))$ is the many-to-one matching strategy, where $Z(e_i) \in \{-1, S\}$ and $Z(s_j) \subseteq \{e_i | e_i \in E_{rejected}\}$.

The preference lists of IoT devices and edge servers are constructed as follows:

- The optimal computing resource allocation and CFNN partition decisions are generated on each server for all IoT devices based on Algorithm 1.
- Calculate the preference value for each edge server s_j on IoT devices $e_i \in E_{rejected}$ as $\ell_j(e_i) = U_j^i$.
- Construct the preference list Y_j for each server s_j by ranking the preference values as a descending order.
- Calculate the preference value for each IoT device $e_i \in E_{rejected}$ on edge server s_j as $\ell_i(s_j) = U_i^j$.
- Construct the preference list Y_i for each IoT device e_i by ranking the preference values as a descending order.

As illustrated in Algorithm 2, at start, $E_{rejected}$ is initialized with E, and $E_{rejected\_}$ is initialized with $\emptyset$ (line 1). The optimal resource allocation and CFNN partition decisions are obtained from Algorithm 1, based on which the preference lists of IoT devices and edge servers are constructed (lines 2 to 9). The matching game begins and ends until the $E_{rejected}$ is empty or each IoT device in $E_{rejected}$ cannot send service request to any edge server (lines 10 to 13). For each IoT device e_i in $E_{rejected}$, the most preferred server s_{j^*} is selected. Then, the matching list of e_i is updated as $Z(e_i) = Z(e_i) \bigcup s_{j^*}$ and the matching list of s_{j^*} is updated as $Z(s_{j^*}) = Z(s_{j^*}) \bigcup e_i$ (lines 14 to 18). For servers s_j which receive new service requests, the updated matching list $Z(s_j)$ is adjusted (lines 20 to 21). In detail, the top n IoT devices in $Z(s_j)$ are remained according to the total number of computing resources in server s_j, and other IoT devices in $Z(s_j)$ are removed. The IoT devices remained in $Z(s_j)$ are removed from $E_{rejected}$ (line 22). The preference lists Y_i and matching lists $Z(e_i)$ of IoT devices e_i remained in $E_{rejected}$ are updated as $Y_i = Y_i \backslash \{s_{j^*}\}$ and $Z(e_i) = Z(e_i) \backslash \{s_{j^*}\}$ (lines 23 to 27).

Now, stability of the many-to-one matching game is proved.

Definition 1: Blocking pair. Z is blocked by the blocking pair (e_i, s_j) if and only if e_i and s_j prefer each other to $s_{j'} = Z(e_i)$ and $e_{i'} = Z(s_j)$, while i and j are not matched with each other under matching strategy Z in fact. [20].

Definition 2: Stable matching. A matching game is stable if and only if there is no blocking pair. [27].

Theorem 1: The many-to-one matching game in this work is stable for each IoT device $e_i \in E_{rejected}$ and edge server $s_j \in S$.

Proof: Proof by contradiction is used here. We assume the many-to-one matching game is not stable, which means at least one blocking pair exists. It is assumed that IoT device e_i and edge server s_j prefer each other to their current matching strategies, but in reality e_i and s_j are not matched with each other. So the following formulas are obtained:

$$s_j \succ_{e_i} Q(e_i), \tag{25a}$$
$$e_i \succ_{s_j} Q(s_j), \tag{25b}$$
$$s_j \neq Q(e_i), \tag{25c}$$
$$e_i \notin Q(s_j), \tag{25d}$$

□

where $Z(e_i)$ is the current server matched to IoT device e_i, and $Z(s_j)$ is current IoT devices matched to server s_j. If formula (25a) holds, it is obvious that IoT device e_i prefers server s_j to its currents matching partners. Nevertheless, it can be learnt from formula (25d) that server s_j does not prefer IoT device e_i, which is denoted as $Q(s_j)i \succ_{s_j} e_i$. It contradicts formula (25b). As a result, (e_i, s_j) will not become a blocking pair. Similar conclusions can be obtained through starting from other formulas in (25). So the many-to-one matching game in this work is stable.

V. Experimental Performance

In this section, the experiment settings are described first. Next, training performance of the CFNN designed in this work is assessed through comparing it with other two models. Then, the convergence of DRAP, which is based on TD3, is analyzed. Finally, effectiveness of the end-edge collaborative CFNN inference decision obtained by DisCFNN is evaluated through comparing it with two baselines.

A. Experiment Settings

1) Device Configuration: The experiments are conducted on a server with Intel(R) Xeon(R) Platinum 8352V CPU. The total number of CPU cores in the server is 120, and computing resources in each CPU core is 67.2G FLOPS. Note that on a real machine, the resources are allocated to each IoT device in units of CPU cores according to $[\frac{f_{j,i}}{f\_}]$, where $f_ = 67.2$G here. During CFNN inference with multiple CPU cores, we split the input set into many groups in average and each CPU core is responsible for inferring a group of input, which uses the multiprocess function of Python.

TABLE I
Information of the proposed CFNN.

Layer number	Layer type	Size	Stride
1	Fuzzy layer	-	-
2	Convolution	$3 \times 3 \times 1 \times\ 4$	2
3	Convolution	$3 \times 3 \times 4 \times\ 8$	2
4	Convolution	$3 \times 3 \times 8 \times\ 16$	2
5	Convolution	$3 \times 3 \times 16 \times\ 32$	2
6	Convolution	$3 \times 3 \times 32 \times\ 64$	1
7	Fully connected	256×10	-

2) Dataset and CFNN Model: Dataset used in our experiments is FashionMNIST [28]. FashionMNIST includes 10 classes and there are 7000 pictures in each class. The pictures in FashionMNIST are gray, with size of $28 \cdot 28$. We partition the dataset into training set, validation set and test set with ratio of $8 : 1 : 1$. Then, in order to shorten training time and test time, 2000 samples and 500 samples are randomly selected from the training set and the test set. Each batch includes 50 samples during training and validation, and the episode number of training is set as 120. Besides, the initial size of input is transferred into $228 \cdot 228$.

The proposed CFNN used in our experiments includes 1 fuzzy layer, 5 convolution layers, and 1 fully connected layer. Detailed information of the proposed CFNN is presented in Table I.V-A3

3) Parameters of Simulation Environment: In order to evaluate the performance of DisCFNN, a virtual simulation environment is established. In the simulation environment, the number of edge servers is 3, with 36, 24, 48 CPU cores, respectively. The number of IoT devices is set as $[3, 6, 9, 12, 15]$. The number of CPU cores in each IoT device is set as a random value between 2 and 4. Transmission speed between each IoT device and edge server is set as a random value between 150M and 350M bits per second [29]. We set the maximum permitted inference latency of CFNN D_i as $rate_i \cdot t_i$, where t_i is latency needed by IoT device e_i to infer the whole CFNN and $rate_1$ is a random value ranging from 0.7 and 0.85. Likewise, the maximum permitted energy consumption of server s_j to infer CFNN D_i is set as $rate_{j,i} \cdot e_{j,i}$, where $e_{j,i}$ is energy consumed by the server to infer the whole CFNN and $rate_{j,i}$ is a random value ranging from 0.75 and 0.8. Budget K_i^{max} of IoT device e_i is set as 400G dollars and the pre-set maximum unit price of computing resources K_j^{max} on server s_j is set as 11.9 dollar per FLOPS [20].

TABLE II
Detailed Information of the Alexnet.

Layer number	Layer type	Size	Stride
1	Convolution	$3 \times 3 \times 1 \times\ 4$	2
2	Convolution	$3 \times 3 \times 4 \times\ 8$	2
3	Convolution	$3 \times 3 \times 8 \times\ 16$	2
4	Convolution	$3 \times 3 \times 16 \times\ 32$	2
5	Convolution	$3 \times 3 \times 32 \times\ 64$	1
6	Fully connected	256×10	-

TABLE III
Detailed Information of Comp-CFNN.

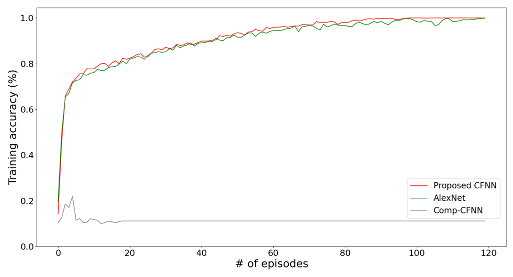
Layer number	Layer type	Size	Stride
1	Convolution	$3 \times 3 \times 1 \times\ 4$	2
2	Convolution	$3 \times 3 \times 4 \times\ 8$	2
3	Convolution	$3 \times 3 \times 8 \times\ 16$	2
4	Convolution	$3 \times 3 \times 16 \times\ 32$	2
5	Convolution	$3 \times 3 \times 32 \times\ 64$	1
6	Fully connected	256×10	-
7	Fuzzy layer	-	-

4) Baselines: In order to assess training performance of the CFNN designed by us, two other models are selected to compare with our model as follows.

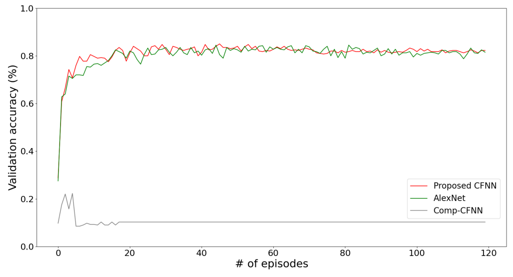
- *AlexNet [30].* AlexNet is a classical CNN model. The detailed information of AlexNet used in our experiments is shown in Table II.
- *Comp-CFNN [23].* In Comp-CFNN, the traditional FNN is merged between the last convolution layer and the first fully connected layer of a CNN, and the detailed information of the Comp-CFNN used in our experiments is shown in Table III.

In order to evaluate effectiveness of the end-edge collaborative CFNN inference decision generated by DisCFNN, we select two baselines which are shown as follows.

- *DisCFNN-NoPar.* DisCFNN-NoPar is a variant of DisCFNN, where all CFNN are either completely inferred on edge servers or completely inferred on IoT devices, without partition.
- *CRO-Offload [31].* In CRO-Offload, the edge server allocates computing resources to IoT devices using the algorithm based on Chemical Reaction Optimization (CRO) and makes CFNN partition decisions based on brute-force searching. Since CRO-Offload is designed for single-server scenarios, it is adjusted for the multi-server IoT in this paper.



(a) Training accuracy.



(b) Validation accuracy.

Fig. 3. Comparison of training performance.

B. Training Performance of Proposed CFNN

As illustrated in Fig. 3, training accuray and validation accuracy of the three models are compared. Training accuracy and validation accuracy of Comp-CFNN are much lower than those of other two models. This is because the FNN merged in Comp-CFNN disorganizes features extracted by convolution layers and the scalability of Comp-CFNN is bad. During training process, the convergence speed of Proposed CFNN, which is the CFNN designed in this work, is a little faster than that of classical AlexNet. Specifically, during training, Proposed CFNN's convergence speed exceeds that of AlexNet after the 3rd episode. In terms of the validation process, the fluctuation of Proposed CFNN's validation accuracy is smaller than that of AlexNet's validation accuracy after convergence, which shows less uncertainty of Proposed CFNN's output during validation.

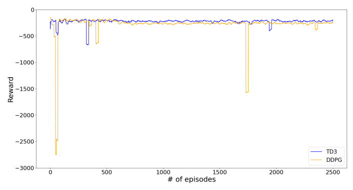
C. Convergence Performance of DRAP

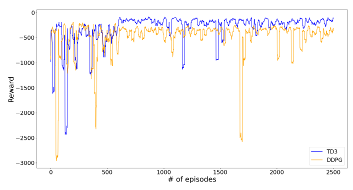
The DRAP designed by us is based on TD3, which is an improved version of Deep Deterministic Policy Gradient (DDPG). In TD3, two critics and two target critics are used, and in terms of the output of critics and target critics, the minimum ones are chosen, which effectively solves the problem of over estimation existing in DDPG. Additionally, delayed update of actor is used in TD3, which improves the learning efficiency. Fig. 4 presents the convergence performance of TD3 and DDPG in each servers in the big data-driven IoT with different numbers of customers. It is obvious that in each server, the rewards of TD3 is higher than that of DDPG regardless of the number of customers in the IoT. Note that a few great fluctuations exist in the reward functions of both DRL algorithms even when the convergence has been achieved because the penalty factor η is set as 300, 300 and 100, respectively, which largely pulled down the reward values if negative customer utility or server utility appears.

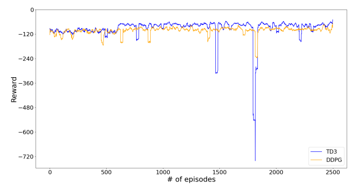
D. Effectiveness Analysis on End-Edge Collaborative CFNN Inference Decision Made by DisCFNN

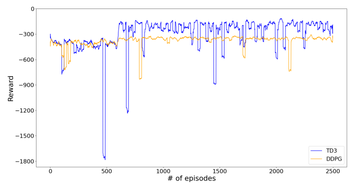
1) Analysis on Utility: Fig. 5 presents the performance evaluation of total customer utility in big data-driven IoT with different numbers of customers. It is seen that although DisCFNN always gets a lower total customer utility than DisCFNN-NoPar except when the number of customers is 3, the total customer utility of DisCFNN keeps positive when the number of customers is smaller than 15. In addition, with the increasing of customers in IoT, the total customer utility of the two methods reduces on the whole. This is because when there are more customers in the IoT, the computing load on servers becomes bigger, which means less computing resources are allocated to each IoT device in average and some IoT devices may even infer the whole CFNN locally. It greatly increases CFNN inference latency and more negative customer utility appears, which decrease the total customer utility.

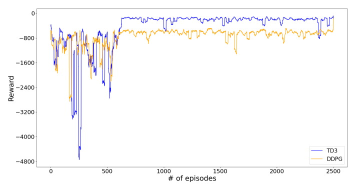
Fig. 6 shows the performance evaluation of total server utility in big data-driven IoT with different numbers of customers. The total server utility of DisCFNN is always higher than that of DisCFNN-NoPar. Specifically, when the number of customers rises from 3 to 15, DisCFNN outperforms DisCFNN-NoPar by the improvement of 0.31 times to 0.42 times. This is because most CFNN are partially inferred on edge servers in DisCFNN, while most CFNN are completely inferred on servers in DisCFNN-NoPar. It causes much more energy consumption and lower total server utility in DisCFNN-NoPar.

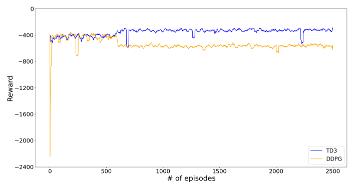

(a) Convergence performance on server 1 when there are 3 customers.

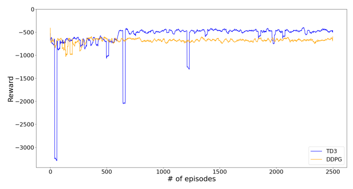

(b) Convergence performance on server 2 when there are 3 customers.

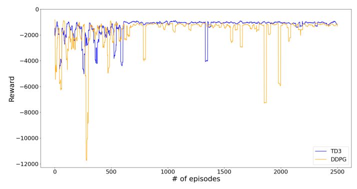

(c) Convergence performance on server 3 when there are 3 customers.

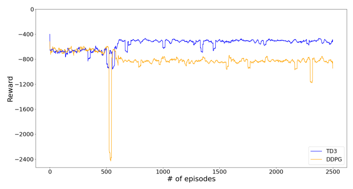

(d) Convergence performance on server 1 when there are 6 customers.

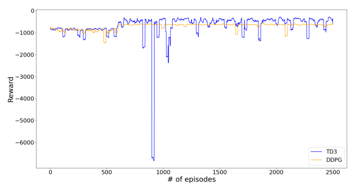

(e) Convergence performance on server 2 when there are 6 customers.

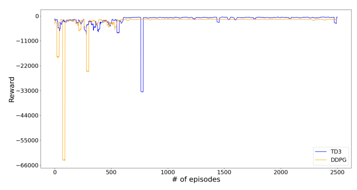

(f) Convergence performance on server 3 when there are 6 customers.

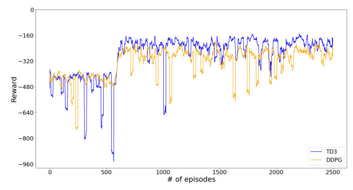

(g) Convergence performance on server 1 when there are 9 customers.

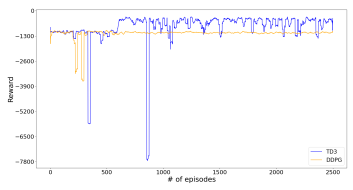

(h) Convergence performance on server 2 when there are 9 customers.

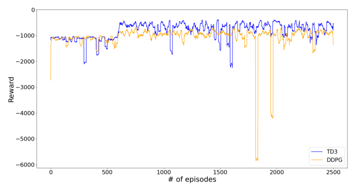

(i) Convergence performance on server 3 when there are 9 customers.


(j) Convergence performance on server 1 when there are 12 customers.


(k) Convergence performance on server 2 when there are 12 customers.


(l) Convergence performance on server 3 when there are 12 customers.


(m) Convergence performance on server 1 when there are 15 customers.


(n) Convergence performance on server 2 when there are 15 customers.

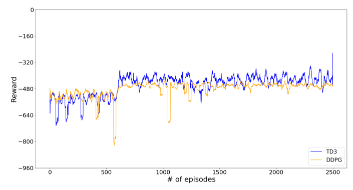

(o) Convergence performance on server 3 when there are 15 customers.

Fig. 4. Convergence performance of rewards under different DRL algorithms.

Fig. 7 illustrates the performance evaluation of total weighted-sum utility in big data-driven IoT with different numbers of customers. The weighted-sum utility is calculated based on Eqs. (16). From Fig. 7, we learn that the total weighted-sum utility of DisCFNN is higher than that of DisCFNN-NoPar except when the number of customers is 12, where the total weighted-sum utility of DisCFNN is negative while that of DisCFNN-NoPar is positive. This is caused by the weak instability of DRL algorithms. In detail, even when convergence is achieved, the fluctuation of reward values still exists. In our experiments, we select the result of the last episode of DRL algorithm's training. It is normal that extremely good or bad strategies are selected in occasion. However, on the whole, the performance of DisCFNN on total weighted-sum utility is better than that of DisCFNN-NoPar.

2) Analysis on Success Rate: Fig. 8 depicts the performance evaluation of the proportion of CFNN whose inference delay meets the maximum latency requirement in IoT with different numbers of customers. It is learnt that DisCFNN performs best except when the number of customers is 9, where the

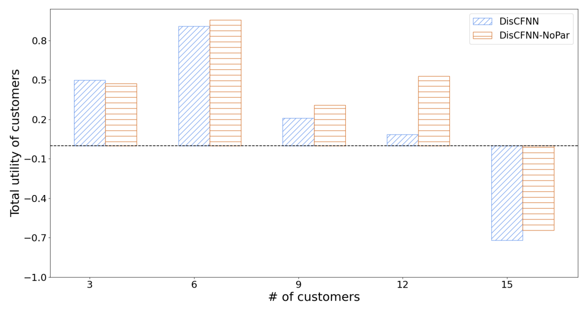


Fig. 5. Total customer utility in IoT with different numbers of customers.

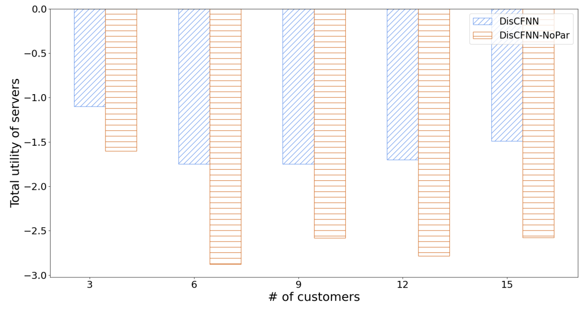


Fig. 6. Total server utility in IoT with different numbers of customers.

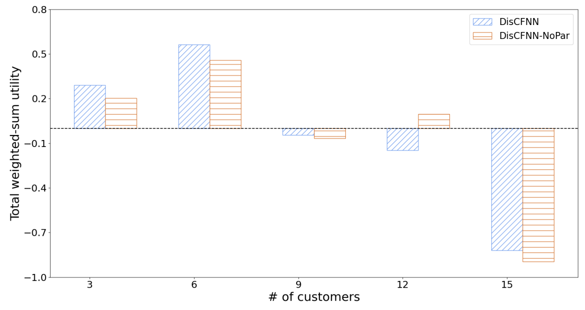


Fig. 7. Total weighted-sum utility in IoT with different numbers of customers.

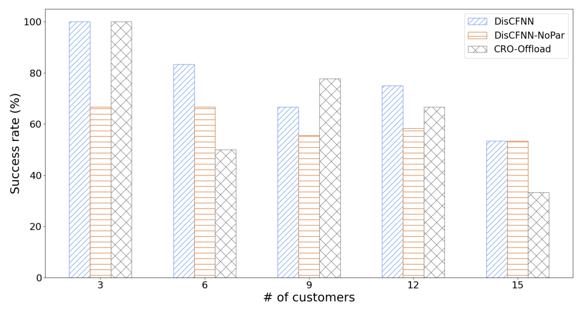


Fig. 8. Success rate in terms of CFNN inference latency in IoT with different numbers of customers.

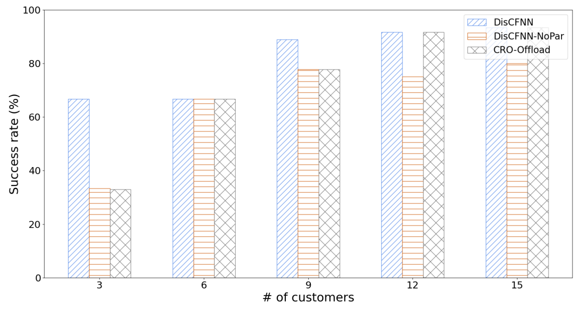


Fig. 9. Success rate in terms of server energy consumption in IoT with different numbers of customers.

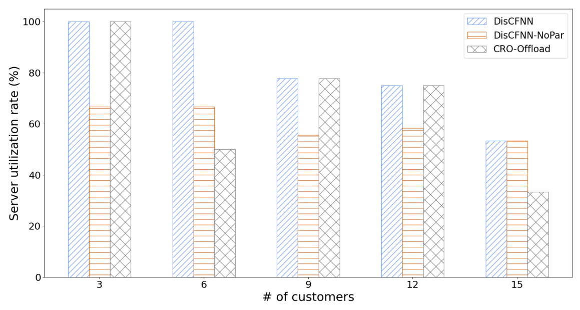


Fig. 10. Utilization rate of edge servers in IoT with different numbers of customers.

success rate of CRO-Offload is 16.7% higher than that of DisCFNN. This is caused by the great instability of CRO. Although extremely good results are occasionally obtained by CRO-Offload, DisCFNN outperforms CRO-Offload in most cases. Additionally, on the whole, the success rates of three methods present a downward trend as customers increases because when the number of customers rises, the computing load on servers becomes larger and less computing resources are allocated to each customer in average, which enhances the failure rate.

Fig. 9 elaborates the performance evaluation of the proportion of servers which meet the maximum energy consumption requirement in IoT with different numbers of customers. When the number of customers is 3, the success rate of DisCFNN is 1 time higher than those of DisCFNN-NoPar and CRO-Offload. While when the number of customers increased to 12, the success rate of DisCFNN is 14.3% higher than that of DisCFNN-NoPar and is equal to that of CRO-Offload. Furthermore, the success rates of three methods rise with the increasing of customers because when more customers enter the IoT, computing resources of servers may be not enough and the proportion of CFNN completely inferred on local goes up, which increases the success rate.

3) Analysis on Server Utilization Rate: Fig. 10 shows the performance evaluation of server utilization rate in IoT with different numbers of customers. When there are only 3 customers in the IoT, the server utilization rates of DisCFNN and CRO-Offload are both 100%, and that of DisCFNN-NoPar is 66.7%. When there are 15 customers in the IoT, the server utilization rates of DisCFNN and DisCFNN-NoPar are both 53.3%, and that of CRO-Offload is 33.3%. Server utilization rate of DisCFNN always maintains the highest level compard with those of two baselines. In addition, on the whole, server utilization rates of three methods present a decreasing trend, and the reason is that the proportion of CFNN completely or partially inferred on servers reduces as the the number of customers rises because of limited computing resources on edge servers.

VI. Conclusion

In this paper, we propose DisCFNN, an end-edge collaborative inference framework of CFNN for big data-driven IoT, to balance the profit of IoT customers and service providers while ensuring acceptable CFNN inference latency and reasonable edge server energy consumption. Specifically, a novel CFNN structure is designed offline and the calculation amount of the fuzzy layer is predicted based on polynomial regression at first. Next, computing resource allocation and CFNN partition decisions are generated on each server through DRL, based on which the IoT customer utility and edge server utility are calculated. Finally, according to the obtained utility of customers and servers, a many-to-one game is conducted to obtain optimal task offloading strategies. The effectiveness of the proposed framework is verified through comparison and analysis of experimental results. [32]

Acknowledgments

This work was supported in part by the Major Research plan of the National Natural Science Foundation of China under Grant (No. 92267104), Natural Science Foundation of Jiangsu Province of China under Grant (No. BK20211284), and Financial and Science Technology Plan Project of Xinjiang Production and Construction Corps under Grant (No. 2020DB005).

References

[1] M. Eskandari, Z. H. Janjua, M. Vecchio, and F. Antonelli, “Passban ids: An intelligent anomaly-based intrusion detection system for iot edge devices,” *IEEE Internet of Things Journal*, vol. 7, no. 8, pp. 6882–6897, 2020.

[2] N. Council, N. Nic, and S. Intelligence, “Disruptive civil technologies: Six technologies with potential impacts on us interests out to 2025,” in *Conference Report CR*, vol. 2007. SRI Consulting Business Intelligence Menlo Park, CA, USA, 2008.

[3] X. Chen, M. Li, H. Zhong, Y. Ma, and C.-H. Hsu, “Dnnoff: offloading dnn-based intelligent iot applications in mobile edge computing,” *IEEE transactions on industrial informatics*, vol. 18, no. 4, pp. 2820–2829, 2021.

[4] X. Xu, S. Tang, L. Qi, X. Zhou, F. Dai, and W. Dou, “Cnn partitioning and offloading for vehicular edge networks in web3,” *IEEE Communications Magazine*, 2023.

[5] X. Hou, Y. Guan, T. Han, and N. Zhang, “Distredge: Speeding up convolutional neural network inference on distributed edge devices,” in *2022 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 2022, pp. 1097–1107.

[6] X. Xu, Z. Liu, M. Bilal, S. Vimal, and H. Song, “Computation offloading and service caching for intelligent transportation systems with digital twin,” *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 11, pp. 20 757–20 772, 2022.

[7] H. Yang, S. Sun, M. Liu, Q. Zhang, and Y. Wang, “Mjoa-mu: End-to-edge collaborative computation for dnn inference based on model uploading,” *Computer Networks*, p. 109801, 2023.

[8] X. Chen, J. Zhang, B. Lin, Z. Chen, K. Wolter, and G. Min, “Energy-efficient offloading for dnn-based smart iot systems in cloud-edge environments,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 3, pp. 683–697, 2021.

[9] C. Hu, W. Bao, D. Wang, and F. Liu, “Dynamic adaptive dnn surgery for inference acceleration on the edge,” in *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*. IEEE, 2019, pp. 1423–1431.

[10] S. Yang, Z. Zhang, C. Zhao, X. Song, S. Guo, and H. Li, “Cnnpc: End-edge-cloud collaborative cnn inference with joint model partition and compression,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 12, pp. 4039–4056, 2022.

[11] L. Zeng, X. Chen, Z. Zhou, L. Yang, and J. Zhang, “Coedge: Cooperative dnn inference with adaptive workload partitioning over heterogeneous edge devices,” *IEEE/ACM Transactions on Networking*, vol. 29, no. 2, pp. 595–608, 2020.

[12] L. Chen, W. Su, M. Wu, W. Pedrycz, and K. Hirota, “A fuzzy deep neural network with sparse autoencoder for emotional intention understanding in human–robot interaction,” *IEEE Transactions on Fuzzy systems*, vol. 28, no. 7, pp. 1252–1264, 2020.

[13] C. Guan, S. Wang, and A. W.-C. Liew, “Lip image segmentation based on a fuzzy convolutional neural network,” *IEEE Transactions on Fuzzy Systems*, vol. 28, no. 7, pp. 1242–1251, 2019.

[14] M. Yeganejou, S. Dick, and J. Miller, “Interpretable deep convolutional fuzzy classifier,” *IEEE transactions on fuzzy systems*, vol. 28, no. 7, pp. 1407–1419, 2019.

[15] A. Yazdinejad, A. Dehghantanha, R. M. Parizi, and G. Epiphaniou, “An optimized fuzzy deep learning model for data classification based on nsga-ii,” *Neurocomputing*, vol. 522, pp. 116–128, 2023.

[16] J. Li, W. Liang, Y. Li, Z. Xu, X. Jia, and S. Guo, “Throughput maximization of delay-aware dnn inference in edge computing by exploring dnn model partitioning and inference parallelism,” *IEEE Transactions on Mobile Computing*, 2021.

[17] J. Ren, G. Yu, Y. He, and G. Y. Li, “Collaborative cloud and edge computing for latency minimization,” *IEEE Transactions on Vehicular Technology*, vol. 68, no. 5, pp. 5031–5044, 2019.

[18] J. Zhang, H. Guo, J. Liu, and Y. Zhang, “Task offloading in vehicular edge computing networks: A load-balancing solution,” *IEEE Transactions on Vehicular Technology*, vol. 69, no. 2, pp. 2092–2104, 2019.

[19] J. Zhao, Q. Li, Y. Gong, and K. Zhang, “Computation offloading and resource allocation for cloud assisted mobile edge computing in vehicular networks,” *IEEE Transactions on Vehicular Technology*, vol. 68, no. 8, pp. 7944–7956, 2019.

[20] Z. Sun, G. Sun, Y. Liu, J. Wang, and D. Cao, “Bargain-match: A game theoretical approach for resource allocation and task offloading in vehicular edge computing networks,” *IEEE Transactions on Mobile Computing*, 2023.

[21] Y. Pan, C. Pan, K. Wang, H. Zhu, and J. Wang, “Cost minimization for cooperative computation framework in mec networks,” *IEEE Transactions on Wireless Communications*, vol. 20, no. 6, pp. 3670–3684, 2021.

[22] G. Zhang, W. Zhang, Y. Cao, D. Li, and L. Wang, “Energy-delay tradeoff for dynamic offloading in mobile-edge computing system with energy harvesting devices,” *IEEE Transactions on Industrial Informatics*, vol. 14, no. 10, pp. 4642–4655, 2018.

[23] M.-J. Hsu, Y.-H. Chien, W.-Y. Wang, and C.-C. Hsu, “A convolutional fuzzy neural network architecture for object classification with small training database,” *International Journal of Fuzzy Systems*, vol. 22, pp. 1–10, 2020.

[24] J. Wang, Q. Chang, T. Gao, K. Zhang, and N. R. Pal, “Sensitivity analysis of takagi–sugeno fuzzy neural network,” *Information Sciences*, vol. 582, pp. 725–749, 2022.

[25] Q. Shi, H.-K. Lam, C. Xuan, and M. Chen, “Adaptive neuro-fuzzy pid controller based on twin delayed deep deterministic policy gradient algorithm,” *Neurocomputing*, vol. 402, pp. 183–194, 2020.

[26] L. Yao, X. Xu, M. Bilal, and H. Wang, “Dynamic edge computation offloading for internet of vehicles with deep reinforcement learning,” *IEEE Transactions on Intelligent Transportation Systems*, 2022.

[27] D. Gusfield and R. W. Irving, *The stable marriage problem: structure and algorithms*. MIT press, 1989.

[28] H. Xiao, K. Rasul, and R. Vollgraf, “Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms,” *arXiv preprint arXiv:1708.07747*, 2017.

[29] B. Lin, Y. Huang, J. Zhang, J. Hu, X. Chen, and J. Li, "Cost-driven offloading for dnn-based applications over cloud, edge, and end devices," *IEEE Transactions on Industrial Informatics*, vol. 16, no. 8, pp. 5456–5466, 2019.

[30] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size," *arXiv preprint arXiv:1602.07360*, 2016.

[31] Q. Wang, Z. Li, K. Nai, Y. Chen, and M. Wen, "Dynamic resource allocation for jointing vehicle-edge deep neural network inference," *Journal of Systems Architecture*, vol. 117, p. 102133, 2021.

[32] X. Zheng and M. Amemiya, "Method for applying crowdsourced street-level imagery data to evaluate street-level greenness," *ISPRS International Journal of Geo-Information*, vol. 12, no. 3, p. 108, 2023.

Yuhao Hu received the B.S. degree in mechanical engineering from Nanjing Tech University in 2022. He is currently pursuing the master's degree in software engineering. His research interests include mobile edge computing, big data, Internet of Things, and deep learning.

Xiaolong Xu received the Ph.D. degree in computer science and technology from Nanjing University, China, in 2016. He was a Research Scholar with Michigan State University, USA,from April 2017 to May 2018. He is currently a Full Professor with the School of Software, Nanjing University of Information Science and Technology. He has published more than 100 peer-review articles in international journals and conferences, including the IEEE TPDS, IEEE T-ITS, IEEE TII, ACM TOSN, ACM TOMM, ACM TIST, IEEE ICWS, ICSOC, etc. He was selected as the Highly Cited Researcher of Clarivate 2021 and 2022. His research interests include Big data, Internet of Things and Deep learning.

Li Duan received the Ph.D. degree in computer science and technology from the Beijing University of Posts and Telecommunications, Beijing, China, in 2016. She was a Research Fellow with Nanyang Technological University, Singapore, and the University of Science and Technology Beijing, Beijing. She is currently an Assistant Professor with the School of Computer and Information Technology, Beijing Jiaotong University, Beijing. Her research interests are services computing and the Internet of Things, data security and privacy protection, and blockchain security and applications. Dr. Duan received the National Natural Science Foundation of China, the Post-Doctoral Fund, and the Basic Scientific Research Project.

Muhammad Bilal received the Ph.D. degree from the Korea University of Science and Technology. He served as a Postdoctoral Research Fellow at Korea University's Smart Quantum Communication Center. In 2018, he joined Hankuk University of Foreign Studies, South Korea, where he was as an Associate Professor with the Division of Computer and Electronic Systems Engineering. In 2023, he joined Lancaster University, where he is working as a Senior Lecturer (Associate Professor) with the School of Computing and Communications. With over 14 years of experience, Dr. Bilal has actively participated in numerous research projects funded by prestigious organizations like IITP, MOTIE, NRF, and NSFC, etc. He is the author/co-author of 140+ articles published in renowned journals and holds multiple US and Korean patents. His research interests include Network Optimization, Cyber Security, the Internet of Things, Vehicular Networks, Information-Centric Networking, Digital Twin, Artificial Intelligence, and Cloud/Fog Computing. Dr. Bilal actively contributes to the academic community as an editorial board member for esteemed journals, including IEEE Transactions on Intelligent Transportation Systems, IEEE Internet of Things Journal, Alexandria Engineering Journal, and more. He has also served on the Technical Program Committees of prestigious conferences like IEEE VTC, IEEE ICC, ACM SigCom, and IEEE CCNC.

Qingyang Wang is currently pursuing the bachelor's degree in data science and big data technology from Nanjing University of Information Science and Technology. His research interests are in the areas of big data, deep learning and reinforcement learning.

Wanchun Dou is a lecturer in the received the Ph.D. degree in mechanical and electronic engineering from the Nanjing University of Science and Technology, China, in 2001. He is currently a Full Professor at the State Key Laboratory for Novel Software Technology, Nanjing University. From April 2005 to June 2005 and from November 2008 to February 2009, he visited the Departments of Computer Science and Engineering, Hong Kong University of Science and Technology, Hong Kong, respectively, as a Visiting Scholar. He has published more than 100 research papers in international journals and international conferences. His research interests include workflow, cloud computing, and service computing.