Collaborative Cloud-Edge Service Cognition Framework for DNN Configuration Toward Smart IIoT

Wenjing Xiao, *Student Member, IEEE*, Yiming Miao, *Member, IEEE*, Giancarlo Fortino, *Fellow, IEEE*, Di Wu, Min Chen, *Fellow, IEEE*, and Kai Hwang, *Life Fellow, IEEE*

***Abstract*—With the widespread application of artificial intelligence and the Internet of Things, the intellectualization of the industrial Internet of Things (IIoT) has received more and more attention. However, in the application scenario with numerous sensors, the contradiction between massive requests of computing tasks and high requirements of inference quality affects the operation efficiency and service reliability. Moreover, due to the heterogeneity of computing resources and the randomness of communication environments of the cloud-edge system, how to compute and deploy deep learning models in a cloud-edge collaborative environment has also become a challenging problem. Therefore, this article presents a collaborative cloud-edge service cognitive framework for deep neural network (DNN) model service configuration to provide dynamic and flexible computing services. In order to adapt to different service requirements, we explored the tradeoffs between accuracy, latency, and energy consumption indicators, and a revenue target is established, which considers the quality of service experience and the system energy consumption to improve resource utilization efficiency. By transforming the optimization of the revenue target into a partially observable DNN configuration reinforcement learning problem, a dueling deep Q-learning network-based self-adaptive DNN configuration algorithm is proposed. Experimental results show that the proposed mechanism can effectively learn from external experience, adapt to the dynamic network environment, and reduce delay and energy consumption while meeting the service requirements.**

***Index Terms*—Cloud-edge collaboration, deep neural network (DNN) configuration, deep learning model, industrial Internet of Things (IIoT).**

Manuscript received November 29, 2020; revised February 12, 2021 and April 20, 2021; accepted August 4, 2021. Date of publication August 18, 2021; date of current version July 11, 2022. This work was supported in part by the China National Natural Science Foundation under Grant 62176101, in part by the Shenzhen Institute of Artificial Intelligence and Robotics for Society (AIRS) under Grant AC01202107002, and in part by the Technology Innovation Project of Hubei Province of China under Grant 2019AHB061. The work of Giancarlo Fortino was supported in part by Italian MIUR, PRIN 2017 through Project "Fluidware" under Grant CUP H24I17000070001. The work of Di Wu was supported by the National Natural Science Foundation of China under Grant U1911201 and Grant U2001209. Paper no. TII-20-5391. *(Corresponding authors: Min Chen; Kai Hwang.)*

Wenjing Xiao and Min Chen are with the School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, China (e-mail: wenjingx@hust.edu.cn; minchen2012@hust.edu.cn).

Yiming Miao and Kai Hwang are with the School of Data Science, Chinese University of Hong Kong (Shenzhen), Shenzhen 518172, China, and also with the Shenzhen Institute of Artificial Intelligence and Robotics for Society, Shenzhen 518129, China (e-mail: yimingmiao@hust.edu.cn; hwangkai@cuhk.edu.cn).

Giancarlo Fortino is with the Department of Informatics, Modeling, Electronics, and Systems, University of Calabria, 87036 Rende, Italy (e-mail: g.fortino@unical.it).

Di Wu is with the School of Computer Science and Engineering, Sun Yat-Sen University, Guangzhou 510006, China, and also with the Guangdong Key Laboratory of Big Data Analysis and Processing, Guangzhou 510006, China (e-mail: wudi27@mail.sysu.edu.cn).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TII.2021.3105399.

Digital Object Identifier 10.1109/TII.2021.3105399

I. Introduction

THE next generation of IIoT requires integrating sensors or controllers with perceptual capabilities into industrial production processes and then using intelligent analysis algorithms to achieve industrial automation, thereby improving manufacturing efficiency. Among them, one of the most critical links is to utilize deep learning models and intelligent algorithms to analyze industrial data to support automated production lines and industrial control. Therefore, how to deploy artificial intelligence algorithm is one of the key problems to realize the landing of Internet of Things (IoT) terminal to intelligent application.

Nowadays, deep neural network (DNN) has been widely applied in many fields, including computer vision [1], speech recognition [2], and natural language processing [3]. In the intelligent service, a well-trained DNN model deployed on the server uses the data from the user as the model input and then returns the inference results to the user. However, for industrial IoT application scenarios, ultra-large-scale AI services mean massive data transmission and frequent model reasoning, which will lead to higher-standard resource requirements and more flexible service configuration.

Cloud computing technology with sufficient resources can support the parallel reasoning and intensive computation of multiple servers [4]. The existing cloud-based system research has achieved excellent works, such as the optimization of network management [5] and DNN computing [6] to improve

the efficiency and reliability. Nevertheless, even though cloud technology provides an outstanding computational paradigm for perceiving, analyzing, and communicating the terminal data, the ultra-high data transmission causes immense traffic pressure [7]. At the same time, edge computing migrates some computing services and functions to the locations closer to the terminal, providing computing, storage, network, and communication resources at the edge of the network. In Wi-Fi and LTE networks, edge computing platforms can significantly improve the latency of interactive applications [8], [9], which can solve the problems of resource constraints and excessive time delay in cloud computing. However, the limited computing and storage ability of edge nodes cannot support the demand for high-precision reasoning continuously.

For fully utilizing cloud servers and edge servers, this article proposes a novel deep learning model and service deployment solution, that is collaborative cloud-edge service cognition framework (CoCE-SC). The DNN models are efficiently loaded between cloud center and edge nodes to provide dynamic and elastic storage and computing services in this framework.

In addition, to ensure that the edge cloud network can cover each IoT node, a large number of edge cloud servers need to be deployed. Therefore, the heterogeneity and complexity of cloud and multiedge cloud computing environments contribute to DNN model computation and deployment difficulty. Furthermore, considering the requirements for response delay and reasoning accuracy of different users, this article analyzes the DNN configuration model in the collaborative cloud-edge computing environment. In order to maximize the revenue target, we apply the dueling deep Q-learning network (DQN) algorithm and propose a self-adaptive DNN configuration algorithm. In a word, the main contributions of this article are as follows.

1) This article presents a collaborative cloud-edge service cognition framework, which integrates the hierarchical structure of cloud-edge collaboration into industrial-grade AI application services.
2) This article considers the accuracy, latency, and energy consumption indicators, and the DNN service configuration model of the CoCE-SC framework is established where the target is described as finding the tradeoffs between the quality of service experience and the system energy consumption.
3) This article transforms the optimization of the revenue target into a partially observable DNN configuration reinforcement learning problem, and a dueling DQN-based self-adaptive DNN configuration algorithm is proposed.

The rest of this article is organized as follows. Section II discusses different computing paradigms, and then introduces collaborative cloud-edge service cognition framework. In Section III, the DNN configuration model of the CoCE-SC framework is established and analyzed. Section IV introduces the design process of dueiling-DQN-based self-adaptive DNN configuration algorithm. Section V elaborates experiment results and analysis. Finally, Section VI concludes this article.

II. System Architecture and Description

A. Service Senior in IIoT

The intelligent applications of existing IoT systems rely on offline data processing and analysis modes. In order to establish a reliable application architecture, it is necessary to consider issues including communication methods, data processing, and model selection. For the deep learning model, cloud computing can provide a good computing platform. And edge computing can directly process and respond to IoT requests, and has the characteristics of low latency and high energy efficiency. Collaborative cloud-edge computing can make full use of resources and establish a more reliable AI service model. In this section, we discuss different computing paradigms for intelligent application scenarios of IoT.

Cloud Service Senior: Is a traditional machine learning service architecture, that is, the reasoning process of DNN model is carried out on the cloud. When the device terminal needs reasoning service, the device transmits the industrial data from the IoT device to the cloud for analysis. In this case, sufficient computing and storage resources on the cloud can realize the training of computing intensive tasks with high reasoning accuracy.

On-Edge Service Senior: Places the DNN model from the cloud to the edge. When a user requests a service, the data will be sent to the edge node, and then the data will be input to the edge model to predict and get the inference result, and finally quickly return to the user. On-edge reasoning is an effective solution for delay-sensitive IoT applications. The edge end with limited resources must be equipped with a GPU processor, and this greatly increases the cost of edge intelligence.

Collaborative Cloud-Edge Service Senior: Refers to a multi-level service execution environment integrating edge and cloud. DNN model is deployed in the scene of collaborative edge and cloud, rather than limited to only cloud server or only edge server. This scenario can make full use of the available resources of edge and cloud, and optimize DNN reasoning service system. A collaborative cloud-edge service architecture means that the model does not infer completely on the edge or cloud, but is executed in cloud or edge system dynamically.

B. CoCE-SC Architecture

The proposed CoCE-SC framework mainly includes two parts, a perception engine and a control engine, as shown in Fig. 1. The perception engine observes and perceives various resources (communication methods, network traffic, business requests, DNN models, and other environmental parameters) of the cloud center and edge servers, and then feeds back global resource information to the control engine. The control engine uses intelligent algorithms to realize the dynamic scheduling and allocation of system resources and optimize business processing strategies.

The CoCE-SC framework provides AI-as-a-service function, that is DNN models are used as a requestable service. The execution environment of AI services is deployed on the cloud and edge servers, including resource configuration and deep learning

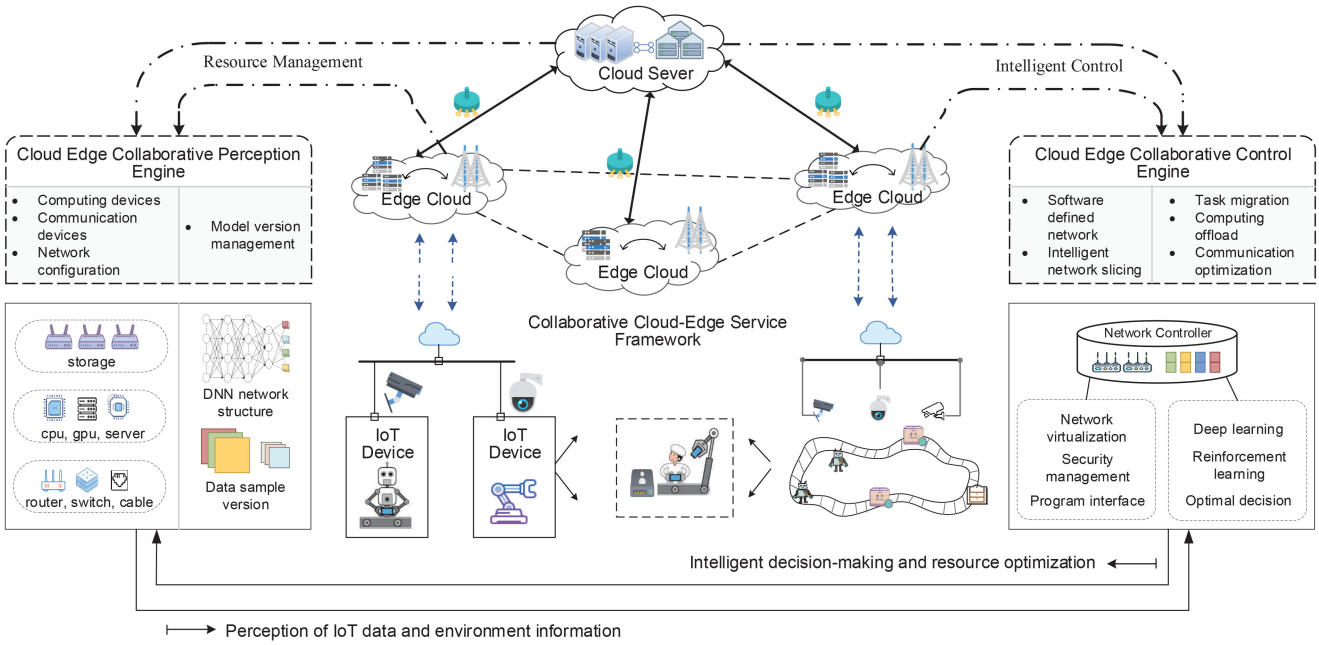


Fig. 1. Architecture of collaborative cloud-edge service cognition framework.

mirroring. In industrial tasks, when the IoT device detects the target object, the system will request DNN reasoning services from the service providers. In this process, The perception engine first perceives the request of the DNN model. These requests have different requirements for latency, reliability, and flexibility. Then, the control engine uses machine learning and deep learning algorithms to analyze the current resource distribution and real-time requests of users. The perception engine will report dynamic resource information to the control engine. Then, the control engine dynamically decides the model configuration according to the comprehensive revenue target, including model version and calculation location.

III. System Model and Problem Formulation

In the CoCE-SC framework, the DNN model is an available resource that can be requested in the cloud or edge server. When the DNN reasoning service is executed, it will take up a lot of computing resources and generate high computing costs. In addition, the size of the data sample is sensitive to the impact of communication costs under immense requests. Taking into the heterogeneity of cloud-edge resources and the diversity of service requests, the CoCE-SC framework designs the DNN models with different network versions (such as different model compression ratios) and different data versions (such as different sample resolutions). The configuration of different model versions can provide a greater optimization space and meet different inference requests. Under the configuration with adaptive DNN models, this section discusses the tradeoffs between accuracy, latency, and energy consumption indicators, and describes the DNN service configuration model of the CoCE-SC framework.

The symbol meaning in this model is defined as follows. Assume that M, U, E, and C represent DNN model set, users set, edge-server set, and remote cloud, respectively; and let I be the number of network version and J be the number of input version. The number of users and edge-server are denoted as H and F, respectively.

A. Model Version Selection

Assume that the system is an inference service provider for a certain type of DNN model. This DNN inference service contains multiple model versions, denoted as $M = \{m_{i,j} | i \in I, j \in J\}$, $m_{i,j}$ denotes the DNN model with the ith network version and the jth data version, where the number of network versions is I and the number of data versions is J. And $o_{i,j}$ denotes the data size of the $m_{i,j}$, $s_{i,j}$ denotes the accuracy of the model $m_{i,j}$ in the public dataset, which is used to evaluate the inference performance of a DNN model. Meanwhile, the batch calculation bits per second of $m_{i,j}$ at the cloud and edge with a batch size of one is $v_{i,j}$ and $u_{i,j}$, respectively.

Besides, the DNN model must be instantiated on a container or virtual machine before executing inference tasks, the cost of model instantiation is denoted as $c_{i,j}$. The location where the model can be instantiated includes the cloud server C or the edge server E (denoted as $w \in \{0, 1\}$). If $w = 0$, it means that the DNN model is executed in the cloud; otherwise, it means that the DNN model is executed in the edge cloud of the base station to which the user belongs. In our scenario, every base station deploys one edge server.

B. DNN Inference Service Model

IoT terminals device are users who request services from the system. In each service request, the user can specify the performance requirements of the service time delay and model accuracy. Since this is an industrial-level service request, in a

period of time, most of the requested IP addresses are from the same geographic area where their performance requirements are the same. Therefore, we regard such a set of service requests as the same set of requests.

Assume that there are users $U = \{1, \ldots, H\}$ who request services in the system, correspondingly, there are edge clouds $E = \{1, \ldots, F\}$. Each edge cloud only receives a request from one user in a time period, that is, $H = F$. The $r_{f,h} = \{a_{f,h}, t_{f,h}, g_{f,h}\}$ represents the service request of user h under the edge server f. $a_{f,h}$ denotes the requirements for inference accuracy, that is, the request $r_{f,h}$ needs to select the accuracy rate not less than $a_{f,h}$. $t_{f,h}$ denotes that the inference time cannot exceed $t_{f,h}$. And $g_{f,h}$ denotes the number of samples that need to be used to reason. In the CoCE-SC framework, once the task is completed, the agent will send the inference result back to the user. Therefore, the selection result of model version and placement location for each user is expressed as $x^w_{f,h,i,j}$.

C. Analysis of Time Delay and Energy Consumption

Edge Sever Layer: If the DNN model decides to execute DNN inference tasks on the edge server, the industrial data from IoT device needs to be transmitted to the edge server through the uplink channel. So, the total end-to-end delay of the system mainly includes the transmission delay of data upload and the computing delay of DNN inference service. In the wireless network environment, the bandwidth of the base station is $B^e = \{b^e_f | f \in E\}$. Thus, the data transmission delay $t^{e\to u}_{f,h,i,j}$ from the edge cloud f to the user h is expressed as

$$t^{e\to u}_{f,h,i,j} = \frac{g_{f,h} * o_{i,j}}{b^e_f}. \tag{1}$$

At the same time, the computing delay of DNN inference on edge server is denoted as $t^e_{f,h,i,j}$, and calculated as $t^e_{f,h,i,j} = g_{f,h} * v_{i,j}$. If the agent chooses the model version $m_{i,j}$, the time delay is $d^e_{f,h,i,j} = t^{e\to u}_{f,h,i,j} + t^e_{f,h,i,j}$. Thus, the total delay for user h to perform the DNN inference task on the edge server is expressed as

$$d^e_{f,h} = \sum_{i}^{I}\sum_{j}^{J} x^0_{f,h,i,j} d^e_{f,h,i,j}. \tag{2}$$

The system energy consumption includes communication and computing energy consumption. Assume that the transmission power from user to the edge server is $P^e = \{p^e_f | f \in E\}$, and computing power is $\Gamma^e = \{\varsigma^e_f | f \in E\}$. Therefore, the communication energy consumption of data transmission is expressed as $\varepsilon^{e\to u}_{f,h,i,j} = t^{e\to u}_{f,h,i,j} * p^e_f$. In the same way, the computing energy consumption is expressed as $\varepsilon^e_{f,h,i,j} = t^e_{f,h,i,j} * \varsigma^e_f$. For the service request of user h, the energy consumption of performing DNN inference tasks on the edge server is $k^e_{f,h,i,j} = \varepsilon^e_{f,h,i,j} + \varepsilon^{e\to u}_{f,h,i,j}$. Thus, assume that $c_{i,j}$ is the energy consumption of the DNN model instantiated on the edge server, the total energy consumption $k^e_{f,h}$ is calculated as follows:

$$k^e_{f,h} = \sum_{i}^{I}\sum_{j}^{J} x^0_{f,h,i,j}(k^e_{f,h,i,j} + c_{i,j}). \tag{3}$$

Cloud Sever Layer: If the DNN model is executed on the cloud server, the user device h will upload data to the cloud through the 5G core network. Ignoring the network connection time, the communication time from the user to the cloud is proportional to the data size, and b^c represents the cloud transmission rate. $t^{c\to u}_{f,h,i,j}$ represents the data transmission delay from users to the cloud, denoted as $t^{c\to u}_{f,h,i,j} = b^c(g_{f,h} * o_{i,j})$. Assume that $t^c_{f,h,i,j}$ represents the computing delay on the cloud, and its expression is $t^c_{f,h,i,j} = g_{f,h} * u_{i,j}$. Compared with edge servers, cloud server has more sufficient computing and storage resources and parallel computing capabilities [10], where multiple DNN models can be executed at the same time. Thus, for the process of computing the DNN model $m_{i,j}$ on the cloud, the time delay is $d^c_{f,h,i,j} = t^{c\to u}_{f,h,i,j} + t^c_{f,h,i,j}$. $d^c_{f,h}$ denotes the total delay of user h executing inference service on the cloud, as follows:

$$d^c_{f,h} = \sum_{i}^{I}\sum_{j}^{J} x^1_{f,h,i,j} d^c_{f,h,i,j}. \tag{4}$$

The cloud server configuration is clustered GPU equipments with definite computing power. Assume that ς^c denotes the computing power of the cloud server, which is calculated according to the preconfigured GPU parameters [11]. The computing energy consumption of executing the DNN model on the cloud is $\varepsilon^c_{f,h,i,j} = t^c_{f,h,i,j} * \varsigma^c$. Assume that p^c represents the transmission power of the communication channel from users to the cloud server. For the process of computing the DNN model $m_{i,j}$ on the cloud, the communication energy consumption is expressed as $\varepsilon^{c\to u}_{f,h,i,j} = t^{c\to u}_{f,h,i,j} * p^c$. Thus, the total energy consumption $k^c_{f,h}$ is expressed as

$$k^c_{f,h} = \sum_{i}^{I}\sum_{j}^{J} x^1_{f,h,i,j} k^c_{f,h,i,j}. \tag{5}$$

D. System Revenue Target Model

When the users request DNN inference service, the agent will comprehensively consider the system cost and quality of service to determine the model version and placement of DNNs. Based on the abovementioned modeling, assume that the decision variables of this system are $x^w_{f,h,i,j}$, the system energy consumption cost G_{energy} is expressed as follows:

$$G_{\text{energy}}(\mathbf{x}) = \sum_{f=1}^{F}\sum_{h=1}^{H}\sum_{i=1}^{I}\sum_{j=1}^{J} (x^0_{f,h,i,j}(k^e_{f,h,i,j} + c_{i,j}) + x^1_{f,h,i,j} k^1_{f,h,i,j}). \tag{6}$$

And the inference performance of the system is expressed as G_{service}, as follows:

$$G_{\text{service}}(\mathbf{x}) = \sum_{f=1}^{F}\sum_{h=1}^{H}\sum_{i=1}^{I}\sum_{j=1}^{J} x^0_{f,h,i,j} s_{i,j} + x^1_{f,h,i,j} s_{i,j} \tag{7}$$

where $x = \{x^w_{f,h,i,j} | w \in \{0,1\}, f \in E, h \in U, i \in I, j \in J\}$. And if $x^0_{f,h,i,j} = 1$, it means model $m_{i,j}$ is executed on the edge server, and when $x^1_{f,h,i,j} = 1$, the agent chooses the model $m_{i,j}$ to execute on the cloud platform.

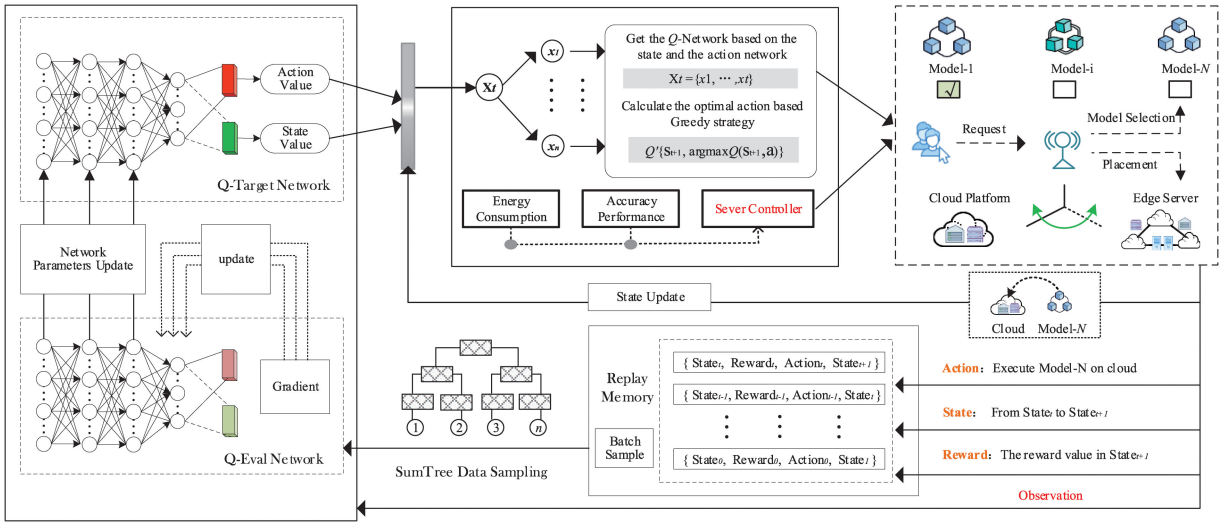


Fig. 2. Dueling-DQN-based model selection mechanism of the CoCE-SC framework.

In other words, the expected goal is to minimize the energy consumption of the system $\min\{G_{\text{service}}\}$ and maximize the quality of experience of users $\max\{G_{\text{energy}}\}$. Thus, this article makes a tradeoff between the system cost and inference accuracy, the overall revenue goal of the system is

$$\max_{\mathbf{x}}\{G\} = \max\{\varepsilon_1 * G_{\text{energy}}(\mathbf{x}) - \varepsilon_2 * G_{\text{service}}(\mathbf{x})\} \tag{8}$$

$$\begin{aligned} &subject\ to \\ &\sum_{f}^{F}\sum_{h}^{H}\sum_{i}^{I}\sum_{j}^{J} x^1_{f,h,i,j} \\ &\le \mathcal{N}, f \in E, h \in U, i \in I, j \in J \\ &\sum_{i}^{I}\sum_{j}^{J}\left(x^0_{f,h,i,j} + x^1_{f,h,i,j}\right) = 1 \\ &\sum_{i}^{I}\sum_{j}^{J}\left(x^0_{f,h,i,j} d^e_{f,h,i,j} + x^1_{f,h,i,j} d^c_{f,h,i,j}\right) \le t_{f,h} \\ &\sum_{i}^{I}\sum_{j}^{J}\left(x^0_{f,h,i,j} s_{i,j} + x^1_{f,h,i,j} s_{i,j}\right) \ge a_{f,h} \end{aligned} \tag{9}$$

where, ε_1 and ε_2 are weight coefficients; $constraint.1$ limits the number of DNN models executed in the cloud. In the abovementioned selection and placement problem of DNN model, resources of cloud server are limited, and users will compete to use them. $constraint.2$ ensures that every request is processed. $constraint.3$ and $constraint.4$, respectively, ensure that the model selection deployment scheme meets the inference performance requirements of users.

According to the abovementioned analysis, the objective optimization problem of DNN configuration with 0–1 variables is a mixed integer nonlinear programming problem, which is a *NP*-hard problem. When the scale of users is N, the agent needs to iteratively explore the optimal solution in a 2^N decision space. And the variable dimension N increases linearly, and the time and energy consumption cost for finding the optimal solution will increase exponentially. Especially in IIoT scenarios with large-scale service requests, it is difficult to handle the randomness of link states and computing resources, and quickly obtain superior decision-making solutions using traditional optimization methods [12]. Thus, we consider feasible deep reinforcement learning methods to solve this problem.

IV. Dueiling-DQN-Based Self-Adaptive DNN Configuration Decision

In our case, the system environment will continue to change over time, so massive memory and computing resources will be consumed to analyze such dynamic information. Deep reinforcement learning combines the intelligent perception capabilities of deep learning and the real-time decision-making capabilities of reinforcement learning, which can generate decisions in a dynamic IoT context. Thus, this article establishes a reinforcement learning solution to solve the problem (8). And the DNN model configuration of the CoCE-SC framework is described from a partially observable Markov decision process (MDP). Especially, the decision of DNN configuration is described by state space $\mathcal{S} = \{s_t, t \in T\}$, action set $\mathcal{A} = \{a_t, t \in T\}$, and reward function $\mathcal{R}$. Then, dueiling-DQN algorithm, as strategy optimization mechanism, utilizes the reward signal to update the parameters θ_{DQN} of DQN. Fig. 2 shows the Dueling-DQN-based model selection mechanism of the CoCE-SC framework.

A. MDP Observation Space

MDP observation space consists of four elements, denoted as $\mathcal{M} = \{\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}\}$, where $\mathcal{S} = \{s_t, t \in T\}$ represents the state space. At time t, the decision to select the model version and placement location is set to the current state value $s_t = \{s^i_t, i \in H\}$. The action space is represented as $\mathcal{A} = \{a_t, t \in T\}$, and the action value a_t is an $|M| * (|E| + |C|)$-dimensional vector. And $\mathcal{P}$ represents the transition probability from s^i_t to s^i_{t+1}. The $\mathcal{R}$

denotes the reward value after performing an action. The reward value is the revenue value of the decision-making of the agent. In this problem, this revenue value is a tradeoff between the revenue of system energy consumption and the revenue of users inference performance. At time-step t, DRL agent observes the current state of the environment s_t and choose an available action a_t. After the action is executed, the environment of MDP makes a transition from state s_t to s_{t+1} with a transition probability $p(s_{t+1}|s_t, a_t) \in \mathcal{P}$. Then, the agent calculates the current reward value $r_t \in \mathcal{R}$ based on the observed new state s^{t+1}. The specific settings of the elements are as follows.

State Space $\mathcal{S}$: In our problem, each state component s_t is a 2-D matrix of size $H \times M$, s_t^i represents the state value of the i user in the time-step t state value, which is the i column vector in s_t. After continuous exploration by the agent, the state space $\mathcal{S}$ will include all possible decision-making solutions. If $s_t^i = \{0, \cdots\}$, it represents that the DNN model of the user i is deployed in the cloud. Otherwise, this model is executed on edge server.

Action Set $\mathcal{A}$: This article defines the actions of each step as a decision of model version and placement location. When the agent selects a solution for each user, that is, to determine the model version and placement location, a round of exploration process is completed. During every round of exploration, the processing order of users is random. Dynamic optimization strategy is used for action selection, and the selected action will make the current state transfer to the new state with the maximum transition probability.

Reward Function $\mathcal{R}$: an appropriate reward function is the key to the application of reinforcement learning algorithms. In our problem, the reward function is jointly determined by inference performance and energy consumption of the system, denoted as r_e and r_s, respectively. $\Sigma = w_s r_s - w_r r_e$ represents the overall benefits of the system, w_r and w_s are the weighting factors of the two returns in the target function. In summary, the reward function is set to (10). And Ω is a constant, which is used to ensure that the reward value is always positive. $\mathcal{S}_T$ represents the end state of the system.

$$r_{t+1}(s_t, a_t) = \begin{cases} \Omega + \Sigma, & if\ s_{t+1}\ is\ \mathcal{S}_T \\ 0, & others \end{cases}. \tag{10}$$

B. Strategy Optimization Mechanism

The task of agent is to select the appropriate model version and placement location. However, in industrial-grade DNN model inference services, the state space and action set will expand rapidly, and it will increase the difficulty of solving. Therefore, we choose DQN [13] to store the state space and action set of the system. Moreover, a DQN model with a dueling structure is built to obtain a more effective decision-making plan.

DQN Exploration Network: DQN uses the neural network that replaces the traditional Q-table to record the states and actions. This is composed of a main network and a target network, which are used to select and estimate actions, respectively. When the agent receives a service request, the agent will use the main network to select the action with the maximum Q value, and this decision will also be stored as an experience sample (s_t, a_t, r, s_{t+1}) in the experience replay. If the samples in experience replay reach a certain number, he will randomly take out a certain number of samples for training. This random sampling scheme solves the problems of sample correlation and nonstatic distribution, and improves the stability of model performance.

Dueling Decision Strategy: in the network evaluation process, we use the dueling Q-network algorithm [14] to select an action. This framework has two networks for state estimation and action selection, respectively, called the state estimation network and the action selection network. The dueling Q-network enables that the network converges faster than the single network, especially when there is numerous industrial data.

Our goal is to obtain a decision strategy for model selection and service placement, $\rho_{\theta^h} : \mathcal{S}_h \times \mathcal{A}_h \rightarrowtail \{0, 1\}$ to find an optimal compromise strategy, that is reducing the system delay and improving the inference accuracy. And θ^h is the strategy parameter of the user h. The expected reward $Q^{\rho_{\theta^h}}(s_h, a_h)$ represents the discount reward expectation accumulated from the initial state s_h^0, and s_h^0 obeys the original distribution: $s_h^0 \sim \kappa_h^0$. Thus, the expected reward is expressed as follows:

$$\begin{aligned} Q_t^{\rho_{\theta^h}}\left(s_t^h, a_t^h; \theta, \alpha, \beta\right) &= \mathcal{V}\left(s_t^h; \theta, \beta\right) + \mathcal{A}\left(s_t^h, a_t^h; \theta, \alpha\right) \\ &= \mathcal{V}\left(s_t^h; \theta, \beta\right) + \left(\mathcal{A} - \frac{1}{\mathbb{A}} \sum \mathcal{A}\left(s_t^h, a_t^{h+1}; \theta, \alpha\right)\right) \end{aligned} \tag{11}$$

where $\mathcal{A}(s_t^h, a_t^h; \theta, \alpha)$ function as the action selection network is used to select action, $\mathcal{V}(s_t; \theta, \beta)$ represents the state estimation network. And discount factor is denoted as $\gamma \in [0, 1]$, the $\mathcal{V}(s_t^h; \theta, \beta)$ is defined as

$$\mathcal{V}\left(s_t^h; \theta, \beta\right) = \mathbb{E}\left(\sum_0^{t \to \triangleright\triangleleft} \gamma_t r_{t+1}^h \mid s_0^h = s^h, \rho_{\theta^h}, \rho_{\theta^{h-1}}\right). \tag{12}$$

And the action strategy of $\mathcal{A}(s_t^h, a_t^h; \theta, \alpha)$ refers to the environmental state of historical decisions and gives the probability of feasible action decisions that meet the task requirements of the current user. The evaluation strategy of $\mathcal{V}(s_t^h; \theta, \beta)$ generates the state of the next round by observing the current environment, and then combines the output of the action strategy and the corresponding estimated reward to evaluate the action strategy. In the stage of a user's action selection, the agent must consider the global environment before making action decision. So there is a dedicated data pool to store historical sample information related to the current user. Because the strategy optimization mechanism learns in a continuous sequence of samples, batch samples, and timing difference [15] are utilized for the loss function of state estimation strategy

$$\mathcal{L}^h\left(\boldsymbol{\omega}^h\right) = \frac{1}{|\mathcal{G}|} \sum_{\sigma \in \mathcal{G}} \left(\left(r_{h_\sigma} + \gamma \mathcal{V}_{\sigma_h}\left(s_{h'_\sigma} - \mathcal{V}_{\sigma_h}\left(s_{h_\sigma}\right)\right)\right)^2. \tag{13}$$

Different from the randomizing the samples sequence in Q-learning method, our DNN service configuration mechanism requires policy learning on an ordered sample set. Thus, this article uses an external memory method to effectively learn continuous input samples. Moreover, according to (9), there are

Algorithm 1: Dueiling-DQN-Based Self-Adaptive DNN Configuration Algorithm.

Input: Experience Sample $\mathcal{D}(\mathcal{S}, \mathcal{A})$;
Network Exploration $\mathcal{T}$;
Experience Replay Capacity $\mathcal{B}$;
Output: Selected Action Set $\mathcal{X}$
Initialize:
the Q_{eval} with random weights θ_{eval};
the Q_{target} with random weights $\theta_{\text{target}} = \theta_{\text{eval}}$;
Create the reply memory pool $\mathcal{M}$;
For $step = 1$ to $\mathcal{T}$ **Do**:
  Set $step = 0$ and Initialize the state space S_0;
  While Ture **Do**:
    Calculate the transition probability value p;
    Decide the model version and placement location of DNN model based on ε-greedy strategy;
    Select $a_t = \underset{x}{\text{argmax}} Q(s_t, a; \theta)$;
    Calculate the reward r_{t+1} and observe the new state s_{t+1};
    Store transition sample $(s_t, a_t, r_{t+1}, s_{t+1})$ in $\mathcal{M}$;
    Sample random minibatches of transitions $(s_i, a_i, r_{i+1}, s_{i+1})$ from $\mathcal{M}$;
    Set $y_i = \begin{cases} r_{i+1}, & if\ s_{t+1}\ is\ terminal\ state \\ Y_i, & otherwise \end{cases}$
    If $step \bmod \Delta = 0$ **Then**
      Update θ_{target}, $\theta_{\text{target}} = \theta_{\text{eval}}$;
End For

some constraints in the target function. If these constraints are defined as the penalty item and is set to a negative reward value, this will cause the loss function to oscillate sharply. Therefore, no penalty term is established in the reward function. Instead, we select a set of actions Δ_a of the current state, which meets these restrictions. In the decision-making process, only select $a \in \Delta_a$. Through the abovementioned strategy design, the agent first continuously learns experience in a dynamic environment. After many iterations, the network parameters are updated. The proposed strategy is shown in Algorithm 1: *Dueiling-DQN-based self-adaptive DNN configuration algorithm.*

V. Experiments and Analysis

In this section, we evaluate the proposed DNN configuration mechanism. In order to perform large-scale performance analysis, this article establishes a parameter scheme based on real application scenarios, and builds a simulation system to verify system performance.

A. Experiment Setting

In order to analyze the influence of network and data version on inference accuracy and computational consumption, we do performance measurements on representative VGG networks. The task of crowd prediction, as an important research topic in smart cities, is a typical application of IoT. Thus, this article

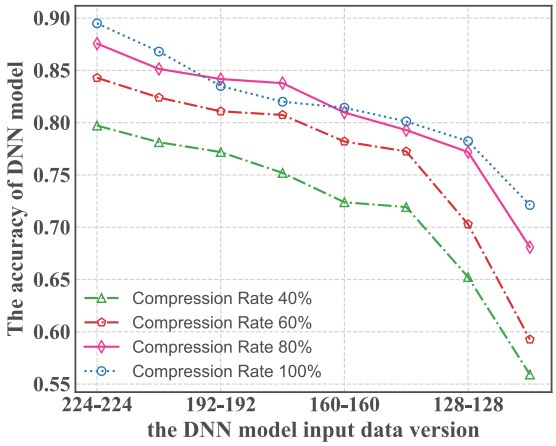


Fig. 3. Inference accuracy of the VGG network under different data versions (from 256@256 to 32@32) and different compression ratios (from 100% to 40%).

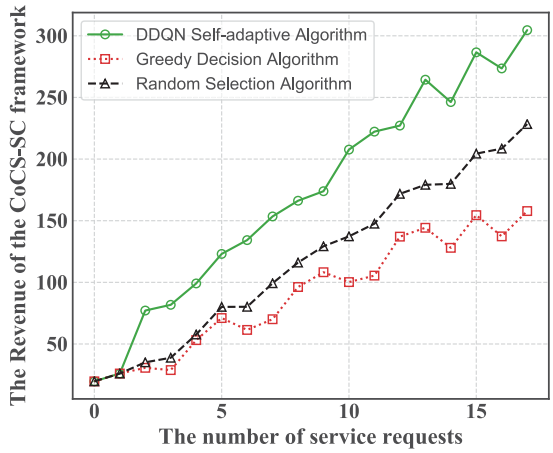


Fig. 4. System performance comparison of the DDQN self-adaptive strategy, greedy decision strategy, and random selection strategy.

uses the crowd counting task as an application case of this experiment. Especially, we use autoMC framework [16] to remove unimportant parts of DNN models, including filters, weight parameters, and channel. And four different compression ratios are used to generate different network versions of DNN models, which are 100%, 80%, 60%, and 40%. For the dataset, we select ShanghaiTech [17]. And there are eight data versions (input image size), ranging from 256@256 to 32@32 with a step length of 32@32. The measurement result is shown in Fig. 3. As the data size decrease, the accuracy rate will decrease significantly. On the other hand, when the compression ratio is 80% or 60%, the accuracy of DNNs decreases slightly. And if the compression ratio is 40%, the accuracy of the model is reduced by about 15%.

For other parameter settings, assume that the cloud computing bandwidth b^c is $1M$, and the transmission power and computing power are, respectively, p^c and ς^c. The transmission power of edge computing is p^e and the computing power is ς^e. In addition, for the service request $r_{f,h}$, the accuracy requirements and sample size are all generated using random probability distributions. In addition, the number of network version I is 4, and the

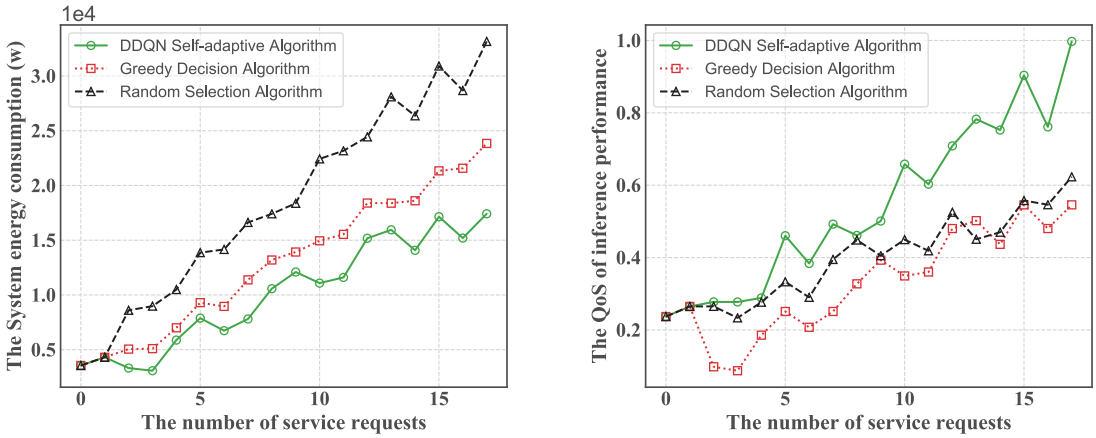


Fig. 5. System performance comparison of the self-adaptive strategy, greedy selection strategy, and random selection strategy. (a) System energy consumption changes with the number of IoT device requests. (b) Users QoS changes with the number of IoT device requests.

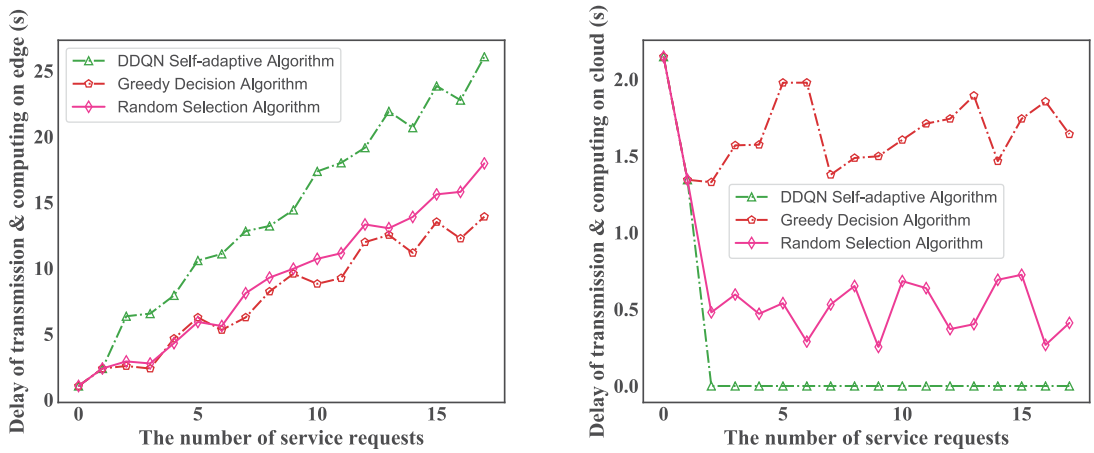


Fig. 6. Relationship between the capacity of the service requests pool and the total delay during the DNN model inference service.

number of input version J is 8. And we set b^c to $1M$, ς^c to $0.9W$, and $\varepsilon^{c\to u}_{f,h,i,j}$ to $1.3W$. Meanwhile, the Gaussian channel noise (ι^2) is set to $10^{-9}W$ and the channel power gain (ℓ^n_h) is 10^{-5}.

B. Comparison of Configuration Algorithms

This section compares the random-selection algorithm [18], the greedy-decision algorithm [19], and the proposed dueiling-DQN-based self-adaptive algorithm (termed as DDQN self-adaptive algorithm).

Random-Selection Algorithm: for each request $r_{f,h}$, the agent first randomly selects a model version m_t in M for service inference. Then, a value is randomly generated. if the value is an odd number, use cloud resources to perform inference services; otherwise, the model m_t is executed on the edge server.

Greedy-Decision Algorithm: the agent chooses the model version and placement location of DNN with the goal of maximizing the revenue target $G(\mathbf{x})$ of the current service request $r_{f,h}$, that is the request in request pools makes decisions one-by-one, and the revenue goal of each decision is $G(\mathbf{x}), H = 1$.

DDQN Self-Adaptive Algorithm: it considers the tradeoffs between accuracy, latency, and energy consumption indicators. This strategy transforms the optimization of the revenue target into a partially observable DNN configuration reinforcement learning problem to decide the model version and placement location.

It can be seen that the greedy-decision algorithm pays more attention to short-term revenues, and the random-selection algorithm focuses on the diversity of decision-making. As shown in Fig. 4, the configuration strategy proposed in this article is superior to other configuration schemes. The reason is that the greedy-decision algorithm only focuses on the revenue of the current service request and does not consider the overall revenue, and the configuration scheme of the random-selection algorithm is random. And neither of these two algorithms considers the configuration of heterogeneous computing resources and random communication environment. Thus, the DDQN self-adaptive algorithm has the better performance. In addition, with the increase in the number of service requests, the revenue difference between our strategy and other algorithms is greater. At this time, the advantages of this algorithm are more obvious.

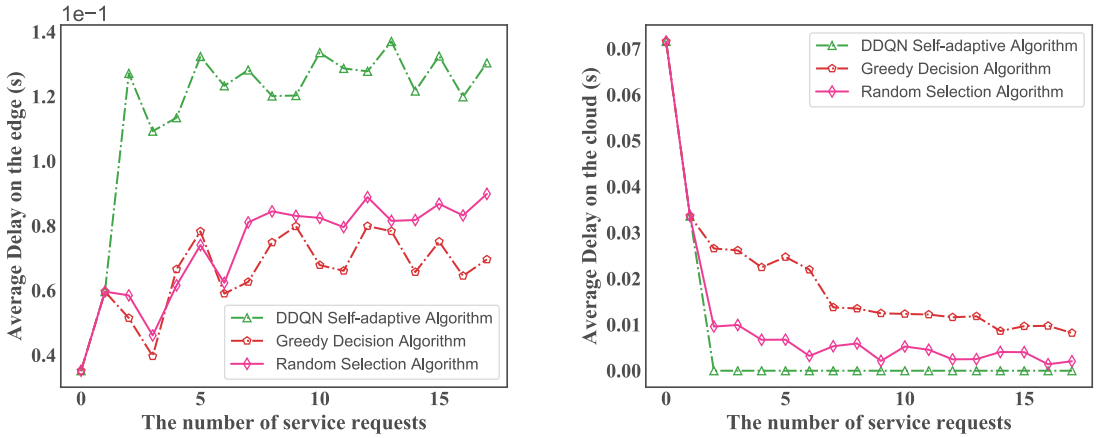


Fig. 7. Relationship between the capacity of the service request pool and the average delay during the DNN model inference service.

This also shows that the algorithm can well adapt to the environment of ultra-large-scale service requests in industrial scenarios.

C. Analysis of System Performance

Fig. 5 discusses the relationship between the number of service requests and the system energy consumption and the QoS of inference performance. As shown in the two figures, compared with the random selection algorithm, our algorithm can reduce system energy consumption by 50% and maintain superior inference performance. Compared with the greedy-decision algorithm, the DDQN self-adaptive algorithm proposed in this article can reduce system energy consumption by 30% and increase the overall inference accuracy by 20%.

Fig. 6 shows the relationship between the capacity of the service request pool and the total delay during the DNN model inference service. The abscissas of Fig. 6(a) and (b) represents the capacity of the service request pool. Fig. 6(a) shows the delay of inference services on the edge cloud. As the capacity of the service request pool increases, the delay in the whole inference process will also become longer. The reason is that the larger the capacity of service request pool, the more the requests that need to be processed in a round. Thus, it will take longer to complete all inference computing tasks, and the system will consume more energy. And Fig. 6(b) shows the delay of inference services on the cloud server. Unlike on the edge cloud, as the number of service requests increases, the time to perform inference tasks on the cloud decreases. The main reason is that the increase of base stations means that the resources of edge servers increase, and more reasoning service task can be placed on the edge. This result also reveals that the addition of more edge clouds can increase the robustness of the system.

Fig. 7(a) discusses the influence of the number of service requests on the average delay of the system. The abscissa of this figure still represents the capacity of the service request pool. In Fig. 7(b), the total delay of edge server and the cloud are calculated as $D^e_{\text{total}} = \sum_f^F \sum_h^H d^e_{f,h}$ and $D^c_{\text{total}} = \sum_f^F \sum_h^H d^c_{f,h}$, respectively. Thus, the average delay of edge server and the cloud are $D^e_{\text{ave}} = \frac{1}{N} D^e_{\text{total}}$ and $D^c_{\text{ave}} = \frac{1}{N} D^c_{\text{total}}$, respectively. It can be seen that at the beginning, as the number of service requests increase, the delay in executing inference service in the edge service increases linearly; in the later stages, the average delay is around a stable value. Therefore, it can be seen that with the expansion of industrial scale, the CoCE-SC framework can avoid the problem of dimensional disasters.

VI. Conclusion

This article proposed a collaborative cloud-edge service cognition framework for industrial IoT scenarios. Considering the requirements for response delay and reasoning accuracy of different users, the CoCE-SC framework designed the DNN models with different network versions (such as different model compression ratios) and different data versions (such as different sample resolutions). In order to solve the heterogeneity of computing resources and the randomness of communication environments of the cloud-edge system, we established the DNN service configuration model of the CoCE-SC framework. Then, a dueling DQN-based self-adaptive DNN configuration algorithm was proposed to optimize DNN-configuration decision.

References

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proc. 25th Int. Conf. Neural Inf. Process. Syst.*, 2012, pp. 1097–1105.
[2] A. B. Nassif, I. Shahin, I. Attili, M. Azzeh, and K. Shaalan, "Speech recognition using deep neural networks: A systematic review," *IEEE Access*, vol. 7, pp. 19143–19165, 2019.
[3] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Proc. 26th Int. Conf. Neural Inf. Process. Syst.*, 2013, pp. 3111–3119.
[4] F. Barghikar, F. S. Tabataba, and M. N. Soorki, "Resource allocation for mmWave-NOMA communication through multiple access points considering human blockages," *IEEE Trans. Commun.*, vol. 69, no. 3, pp. 1679–1692, Mar. 2021.
[5] M. Zorzi, A. Zanella, A. Testolin, M. De Filippo De Grazia, and M. Zorzi, "Cognition-based networks: A new perspective on network optimization using learning and distributed intelligence," *IEEE Access*, vol. 3, pp. 1512–1530, 2015.
[6] M. Halpern, B. Boroujerdian, T. Mummert, E. Duesterwald, and V. Janapa Reddi, "One size does not fit all: Quantifying and exposing the accuracy-latency trade-off in machine learning cloud service APIs via tolerance tiers," in *Proc. IEEE Int. Symp. Perform. Anal. Syst. Softw.*, 2019, pp. 34–47.

[7] S. F. E. Harjula and A. Artemenko, "Edge computing for industrial IoT: Challenges and solutions," *Wireless Netw. Ind. IoT*, vol. 25, pp. 225–240, 2021, [Online]. Available: https://ojs.jctecs.com/index.php/com/article/view/293.

[8] Y. Hao, Y. Miao, M. Chen, H. Gharavi, and V. Leung, "6G Cognitive Information Theory: A Mailbox Perspective", *Big Data Cogn. Comput.*, vol. 5, no. 4, p. 56, 2021.

[9] M. Chen and V. C. Leung, "From cloud-based communications to cognition-based communications: A computing perspective," *Comput. Commun.*, vol. 128, pp. 74–79, 2018.

[10] Y. Liu, Q. Cui, J. Zhang, Y. Chen, and Y. Hou, "An actor-critic deep reinforcement learning based computation offloading for three-tier mobile computing networks," in *Proc. 11th Int. Conf. Wireless Commun. Signal Process.*, 2019, pp. 1–6.

[11] Y. Wen, W. Zhang, and H. Luo, "Energy-optimal mobile application execution: Taming resource-poor mobile devices with cloud clones," in *Proc. IEEE INFOCOM*, 2012, pp. 2716–2720.

[12] A. Szepietowski, "Closure properties of hyper-minimized automata," *RAIRO - Theor. Informat. Appl. - Informatique Théorique Et Appl.*, vol. 45, no. 4, pp. 459–466, 2011.

[13] O. Nachum, M. Norouzi, K. Xu, and D. Schuurmans, "Bridging the gap between value and policy based reinforcement learning," *Adv. Neural Inf. Proc. Syst.*, vol. 30, 2017.

[14] M. Chen, W. Li, G. Fortino, Y. Hao, L. Hu, and I. Humar, "A Dynamic Service-Migration Mechanism in Edge Cognitive Computing", *ACM Trans. Internet Technol.*, vol. 19, no. 2, 2019.

[15] Q. Liu, L. Shi, L. Sun, J. Li, M. Ding, and F. Shu, "Path planning for UAV-mounted mobile edge computing with deep reinforcement learning," *IEEE Trans. Veh. Technol.*, vol. 69, no. 5, pp. 5723–5728, May 2020.

[16] J. Wu, *et al.*, "PocketFlow: An automated framework for compressing and accelerating deep neural networks" in *NIPS Workshop Compact Deep Neural Netw. Representation Ind. Appl. (CDNNRIA)*.

[17] Y. Zhang, D. Zhou, S. Chen, S. Gao, and Y. Ma, "Single-image crowd counting via multi-column convolutional neural network," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 589–597.

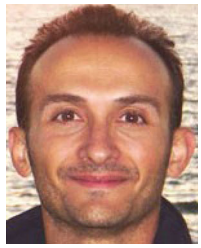
[18] K. Hwang, M., and Chen, "Big Data Analytics for Cloud/IoT and Cognitive Computing," Wiley, U.K., 2017.

[19] C. Savaglio, M. Ganzha, M. Paprzycki, C. Badica, M. Ivanovic, and G. Fortino, "Agent-based Internet of Things: State-of-the-art and research challenges", *Future Gener. Comput. Syst.*, vol. 102, pp. 1038–1053, 2020.

Wenjing Xiao (Student Member, IEEE) is currently working toward the bachelor-straight-to-doctorate degree in computer system architecture with the Embedded and Pervasive Computing (EPIC) Lab, School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan, China.

Her research interests include cloud computing, Internet of Things, and cognitive computing.

Yiming Miao (Member, IEEE) received the Ph.D. degree from the School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan, China in 2021. She is currently a Research Assistant Professor with the School of Data Science, The Chinese University of Hong Kong, Shenzhen, China. Her research interests include affective computing, edge computing, 5G mobile communication system.



Giancarlo Fortino (Senior Member, IEEE) received the Ph.D. degree and Laurea (M.Sc.+B.Sc.) degree in computer engineering from Unical. He is currently a Full Professor of Computer Engineering with the Department of Informatics, Modeling, Electronics, and Systems of the University of Calabria (Unical), Italy. He is Highly Cited Researcher (2002–2021) in Computer Science. His research interests include wearable computing systems, Internet of Things, and Cyber-security.

Prof. Fortino is High-end Foreign Expert of China (2015–2018) and Guest Professor at the Wuhan University of Technology (China). He is author of 550+ papers in international journals, conferences and books. He is the Founding Editor of the IEEE press book series on Human-Machine Systems and of the Springer Internet of Things series, and is AE of premier IEEE Transactions. He is cofounder and CEO of SenSysCal S.r.l., a Unical spinoff focused on innovative IoT systems. Fortino is currently member of the IEEE SMCS BoG and chair of the IEEE SMCS Italian Chapter.

Di Wu is currently a Professor and the Associate Dean of the School of Computer Science and Engineering with Sun Yat-sen University, Guangzhou, China.

Prof. Wu was the recipient of the IEEE INFOCOM 2009 Best Paper Award, IEEE Jack Neubauer Memorial Award, etc.

Min Chen (Fellow, IEEE) has been a Full Professor with the School of Computer Science and Technology, Huazhong University of Science and Technology (HUST), Wuhan, China, since February 2012. He worked as a Post-Doctoral Fellow in Department of Electrical and Computer Engineering at University of British Columbia (UBC) for three years.

Prof. Chen is the Director of Embedded and Pervasive Computing Lab, and the Director of Data Engieering Institute at HUST. He is the founding Chair of the IEEE Computer Society Special Technical Communities on Big Data. His Google Scholar Citations reached 32,800+ with an h-index of 87.

Kai Hwang (Life Fellow, IEEE) received the Ph.D. degree in electrical engineering and computer science from the University of California at Berkeley, Berkeley, CA, USA, in 1972.

Since 2018, he has been a Presidential Chair Professor with the Chinese University of Hong Kong (CUHK), Shenzhen, China. He was with Purdue University, IN, USA and the University of Southern California, CA, for many years prior joining the CUHK. He has authored or coauthored ten scientific books and more than 280 scientific papers.

Prof. Hwang was a recipient of the Outstanding Achievement Award in 2005 from China Computer Federation and the Lifetime Achievement Award from the IEEE CloudCom 2012, and the Tenth Wu Wenjun Artificial Intelligence Natural Science Award in 2020 from China's Artificial Intelligence Association for his recent work on AI-oriented clouds/datacenters.