

Chapter 12

Collaborative Deep Neural Network Inference via Mobile Edge Computing



Wen Wu, Yujie Tang, Peng Yang, Weiting Zhang, and Ning Zhang

12.1 Introduction

Advanced neural network techniques and ubiquitous Internet of Things (IoT) devices enable deep neural network (DNN) inference as a key technology in next generation wireless networks. In recent years, DNNs have been applied in many intelligent applications, ranging from facility monitoring, fault diagnosis, to object detection [1, 2]. For example, IoT devices in industrial applications, such as vibration sensors, can sense the industrial operating environment. Then, the sensing data is sent to a pre-trained DNN via wireless communication links, and the DNN processes the sensing data and renders inference results. Such a process

W. Wu (✉)

Pengcheng Laboratory, Shenzhen, P.R. China

e-mail: wuw02@pcl.ac.cn

Y. Tang

School of Computer Science and Technology, Algoma University, Sault Ste. Marie, ON, Canada

e-mail: yujie.tang@algomau.ca

P. Yang

School of Electronic Information and Communications, Huazhong University of Science and Technology, Wuhan, P.R. China

e-mail: yangpeng@hust.edu.cn

W. Zhang

School of Electronic and Information Engineering, Beijing Jiaotong University, Beijing, P.R. China

e-mail: 17111018@bjtu.edu.cn

N. Zhang

Department of Electrical and Computer Engineering, University of Windsor, Windsor, ON, Canada

e-mail: ning.zhang@uwindsor.ca

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2022

L. Cai et al. (eds.), *Broadband Communications, Computing, and Control*

for *Ubiquitous Intelligence, Wireless Networks*,

https://doi.org/10.1007/978-3-030-98064-1_12

is referred to as *DNN inference* [3]. A large number of experimental results indicate that DNN inference can achieve high inference accuracy as compared to traditional alternatives, such as decision trees in classification tasks.

Executing DNN inference tasks is computation intensive. Tremendous numbers of multiply-and-accumulation operations are conducted in a DNN inference task [4]. A *device-only* solution that purely executes DNN inference tasks at resource-constrained mobile devices becomes intractable, due to prohibitive energy consumption and a high service delay. For instance, processing an image using AlexNet incurs up to 0.45 W energy consumption even in a tailored energy-efficient chip [5]. An *edge-only* solution that purely offloads large-volume sensing data to resource-rich edge nodes, e.g., access point (AP), suffers from an unpredictable service delay due to time-varying wireless channels [6]. Therefore, neither a device-only nor an edge-only solution can effectively support low-delay DNN inference services.

Collaborative DNN inference, which coordinates resource-constrained mobile devices and the resource-rich AP, is a potential framework to provide low-delay and high-accuracy inference services [7]. Within the collaborative inference framework, sensing data from mobile devices can be either processed locally or offloaded to the AP. At mobile devices, light-weight *compressed* DNNs, i.e., neural networks are compressed without significantly decreasing their performance, are deployed due to constrained on-board computing capability, which saves computing resources at the cost of inference accuracy [8, 9]. At the AP, *uncompressed* DNNs are deployed to provide high-accuracy inference services at the cost of network resources including computing and communication resources. The overall service performance can be enhanced through the task offloading between mobile devices and the AP.

However, the *sampling rate adaption* technique that dynamically configures the sampling rates of mobile devices, is seldom investigated in the collaborative DNN inference framework. The sampling rates of mobile devices can be dynamically adjusted based on mobile devices' real-time channel conditions and the AP's computation workloads. As such, the sensing data from mobile devices can be compressed, thereby reducing not only the offloaded data volume but also the task computation workload. On the one hand, when the mobile device's channel condition is poor or the AP's computation workload is heavy, the sampling rate is decreased to reduce the offloaded data volume and the requested computation workload. As a result, the service delay is reduced at the cost of limited inference accuracy. Our experimental results show that the reduction of inference accuracy is acceptable in harsh network environments. On the other hand, when the mobile device's channel condition is good and the edge computation workload is light, the sampling rate can be increased to help deliver a high-accuracy service with an acceptable service delay. Therefore, sampling rate adaption can effectively reduce the service delay, which should be considered as an important component in the collaborative DNN inference framework.

In this chapter, we present the collaborative DNN inference technology in wireless networks. *Firstly*, we give a comprehensive overview of DNN inference, mobile edge computing (MEC), and machine learning. *Secondly*, we study a detailed case on collaborative DNN inference via device-edge orchestration. The problem

is formulated as a constrained Markov decision process (CMDP) taking time-varying channel conditions and random task arrivals into account. Specifically, three decisions, i.e., sampling rates of mobile devices, task offloading, and edge computation resource allocation, are jointly optimized to achieve the minimum average service delay while guaranteeing the long-term accuracy requirements of multiple DNN inference services. *Thirdly*, since traditional RL algorithms target at optimizing a long-term reward without considering policy constraints, it is difficult to directly apply them to solve the formulated CMDP with long-term constraints. To address the issue, we propose a *three-step* solution: (1) the Lyapunov optimization technique is leveraged to transform the CMDP into an MDP; (2) to solve the MDP, a learning-based algorithm is developed based on the deep deterministic policy gradient (DDPG) algorithm; and (3) the edge computing resource allocation can be directly solved via an optimization subroutine, and then the optimization subroutine is incorporated in the learning-based algorithm to reduce the training complexity. Extensive simulations are conducted to validate the effectiveness of the proposed algorithm in reducing the average service delay while preserving the long-term accuracy requirements.

The remainder of this chapter is organized as follows. Section 12.2 presents a comprehensive overview of three key technologies, including DNN inference, MEC, and machine learning. The considered scenario, the system model, the formulated problem, and the proposed learning-based solution are presented in Sect. 12.3. Simulation results are given in Sect. 12.4. Finally, Sect. 12.5 concludes this chapter.

12.2 Background

12.2.1 DNN Inference

Recently, DNN inference for mobile devices has attracted much attention from academia. A device-only solution resorts to on-board computing resources to facilitate DNN inference services. DNN compression techniques are applied to reduce the computational complexity at the mobile devices. Typical techniques include weight pruning [8] and knowledge distillation [10]. The authors in [4] designed a light-weight DNN inference model, which can dynamically compress the model size in order to balance inference accuracy and energy efficiency, taking the widely equipped energy-harvesting functionality in IoT devices into account. In another line of research, by utilizing powerful edge computing servers, edge-assisted DNN inference solutions can provide high-accuracy inference services. The authors in [11] proposed an online video quality and computing resource allocation strategy to maximize video analytic accuracy, thereby facilitating low-delay and accurate DNN-based video analytics. Another important work proposed a novel device-edge collaborative inference scheme [7]. In this work, the DNN model is partitioned and deployed at both the device and the edge, and intermediate results are transferred via wireless links. The above works can offer potential resource

allocation solutions to enhance DNN inference performance. In comparison with the existing works, the following case study in this chapter takes the sampling rate adaptation of IoT devices into account, aiming at providing accuracy-guaranteed inference services in dynamic network environments.

12.2.2 *Mobile Edge Computing*

In the current wireless networks, a large volume of computing demands are generated by mobile devices to support emerging applications, such as intelligent path planning, safety applications, and on-board entertainments. Taking the autonomous driving service as an example, when an autonomous vehicle is on the road, a large number of computation-intensive tasks are required to be processed [12]. Processing such computation-intensive tasks by mobile devices requires expensive on-device computing facilities and degrades energy efficiency. As a remedy to these limitations, a potential solution is to explore the MEC paradigm. In the MEC paradigm, mobile devices can offload these computation tasks to nearby radio access networks (RANs) with computation-powerful edge servers for prompt processing. Extensive experiments show that the task processing delay can be significantly reduced by leveraging the MEC paradigm.

Recently, MEC problems have been widely investigated from many perspectives in wireless networks. In high-mobility vehicular networks, the roadside MEC servers judiciously collaborate with each other to provide low-latency services for autonomous vehicles [13]. Also, in the context of vehicular networks, a dynamic RAN slicing framework taking roadside MEC servers into account is proposed to guarantee the quality of service requirements of autonomous driving services [14]. In recent emerging unmanned aerial vehicle (UAV) networks, a UAV endowed with an MEC server is dispatched to collect and then process tasks from a large number of IoT devices in the remote area [15]. In this chapter, the MEC server at the AP is applied to handle the computation-intensive DNN inference task.

12.2.3 *Machine Learning*

Recently, machine learning (ML) has achieved great success in a number of research fields, ranging from computer vision, gaming, natural language processing, object detection, and traffic prediction [16]. The machine learning methods can be classified into three categories: (1) *supervised learning*, in which the training data structure includes both feature and label. For example, the support vector machine algorithm is supervised learning; (2) *unsupervised learning*, in which the training structure only includes feature without label, e.g., K-means algorithm; and (3) *RL*, in which the data structure is defined by state, action, and reward. As shown in Fig. 12.1, the action can be the control decisions, the state can be the

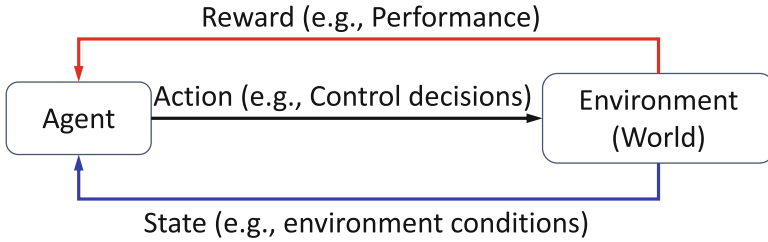


Fig. 12.1 An illustrative example of RL algorithms

environment conditions, and the observed reward from the environment can be system performance. The objective of RL is to learn a good policy in a sequential decision-making problem, such that the learning agent can take appropriate actions based on the current state. A typical example of RL algorithm is the deep Q learning algorithm. Seeing the great benefits of different machine learning methods, it is expected that ML will be widely applied in future wireless networks. The potential ML applications in next generation wireless networks, i.e., 6G networks, are investigated in [17–19], ranging from network slicing, traffic prediction, to digital twin management.

The main benefits of ML methods can be summarized as follows: (1) model-free, which makes ML methods different from traditional model-based approaches. It learns from the data and does not suffer from complicated modelling and strong assumptions; and (2) flexible, which means that ML methods can adaptively adjust the decision based on the current network environment. By training the learning modules properly offline, ML methods can make quick online decisions in highly complex scenarios.

Among different categories of ML algorithms, RL has attracted great attention from both academia and industry in the field of wireless communications. RL algorithms have been widely applied in network resource allocation, such as service migration in vehicular networks [20], network slicing in cellular networks [17], content caching in edge networks [21, 22], and beam alignment in mmWave networks [23, 24]. Hence, RL algorithms can be considered as potential solutions to manage network resources for DNN inference services. In this chapter, we propose a deep RL-based algorithm to deal with resource allocation and sampling rate selection issues in the collaborative DNN inference problem.

12.3 Collaborative DNN Inference via Device-Edge Orchestration

In this section, we introduce a case study on collaborative DNN inference, in which mobile devices and the network edge are orchestrated to provide DNN inference services. The collaborative DNN inference framework is presented in Sect. 12.3.1,

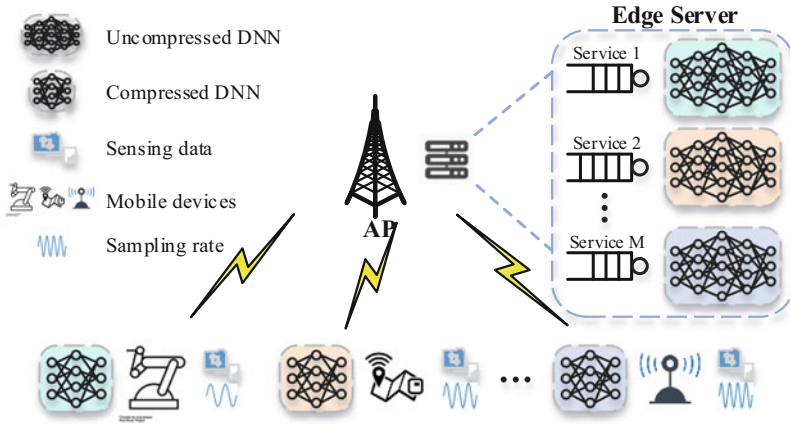


Fig. 12.2 An illustrative example of the collaborative DNN inference framework

and the corresponding detailed performance analysis on service delay and accuracy is provided in Sect. 12.3.2. Based on the system model, the problem is presented in Sect. 12.3.3, which is solved via a learning-based algorithm in Sect. 12.3.4.

12.3.1 Collaborative DNN Inference Framework

We consider a wireless network with one AP to serve multiple types of mobile devices, as illustrated in Fig. 12.2. In the network, the AP collects network information and then conducts resource orchestration decisions. Let \mathcal{M} denote a set of M types of supported inference services, e.g., facility fault diagnosis and facility monitoring services [25]. The set of mobile devices subscribed to service m is denoted by \mathcal{N}_m , and the set of all mobile devices is denoted by $\mathcal{N} = \cup_{m \in \mathcal{M}} \mathcal{N}_m$.

Consider industrial facility monitoring services as an example. In a smart factory, wireless sensors are equipped to measure the status of the industrial facility. Vibration sensors can sense the operation condition of a facility with a certain sampling rate, e.g., 24 KHz. Mobile devices send the sensing data to a DNN for a specific inference service, and then DNN processes the sensing data and conducts inference, e.g., fault diagnosis.

In the collaborative inference framework, two kinds of DNNs are deployed:

- **Compressed DNN**, which is deployed on mobile devices. The compressed DNN can be implemented via the weight pruning technique, which prunes less-important weights to reduce computational complexity while maintaining similar inference accuracy [8].

- **Uncompressed DNN**, which is deployed at the AP. As such, M types of uncompressed DNNs share the edge computing resource to serve different kinds of inference requests.

The collaborative DNN inference framework operates in a time-slotted manner. The procedure consists of the following two steps:

- **Step 1: Sampling rate selection.** Mobile devices select their sampling rates based on channel conditions and computation workloads. The set of candidate sampling rates is denoted by $\mathcal{K} = \{\theta_1, \theta_2, \dots, \theta_K\}$, where θ_K denotes the raw sampling rate. We assume the sampling rate in \mathcal{K} increases linearly with the index, i.e., $\theta_k = k\theta_K/K$. Let t denote the time index, where $t \in \mathcal{T} = \{1, 2, \dots, T\}$. Let \mathbf{X}^t denote the sampling rate decision matrix in time slot t , whose element $x_{n,k}^t = 1$ indicates the mobile device $n \in \mathcal{N}$ selects the k -th sampling rate.
- **Step 2: Task processing.** The sensing data from mobile devices within a time slot is deemed as a computation task, which can be either offloaded to the AP or executed locally. Let $\mathbf{o}^t \in \mathbb{R}^{|\mathcal{N}| \times 1}$ denote the offloading decision vector in time slot t , whose element $o_n^t = 0$ indicates offloading the computation task from mobile device n . Otherwise, $o_n^t = 1$ indicates executing the computation task locally.

12.3.2 Service Delay and Accuracy Analysis of Collaborative DNN Inference

In this subsection, we analyze the inference delay and accuracy performance in the considered collaborative DNN inference framework.

12.3.2.1 Inference Delay Analysis

In the considered framework, a computation task can be either processed locally or offloaded to the AP. In the following, we analyze the service delay in these two cases, i.e., executing tasks locally or offloading tasks to AP.

Case 1: Executing Tasks Locally The task arrival rate of the n -th mobile device in time slot t is denoted by λ_n^t . We assume that the task arrival follows a general random distribution. Let $\xi_n^t = \lambda_n^t v_m, \forall n \in \mathcal{N}_m$ denote the raw data size of the generated tasks at the n -th device. Here, v_m denotes the raw data size of a task for service m . When the sampling rate is selected, we can represent the data size of the generated task by:

$$\zeta(\mathbf{x}_n^t) = \sum_{k=1}^K \frac{x_{n,k}^t \xi_n^t k}{K}. \quad (12.1)$$

Here, $\mathbf{x}_n^t = \{x_{n,k}^t\}_{k \in \mathcal{K}}$ is the n -th device's sampling rate selection decision vector. If the inference task is processed via a compressed DNN in the local mobile device, the service delay should consist of two parts: the queuing delay in the local computing queue and the task processing delay. The detailed calculation of the two parts is given by:

$$d_{n,l}^t = \frac{o_n^t \eta_{m,c} (B_n^t + \zeta(\mathbf{x}_n^t))}{f_n}, \forall n \in \mathcal{N}_m. \quad (12.2)$$

Here, f_n denotes the n -th mobile device's central processing unit (CPU) frequency, and $\eta_{m,c}$ represents the computation intensity of the compressed DNN for the m -th service. Let B_n^t denote the backlogged computation tasks (in bits) in the local computing queue, which is updated via

$$B_n^{t+1} = \min \left\{ \left[B_n^t + o_n^t \zeta(\mathbf{x}_n^t) - \frac{f_n \tau}{\eta_{m,c}} \right]^+, B_n^{max} \right\}, \quad (12.3)$$

where $[x]^+ = \max\{x, 0\}$. Here, B_n^{max} represents the local computing queue capacity, and τ denotes a time slot duration. It is worth noting that tasks have to be dropped if the local computing queue is full. The amount of the dropped tasks in the local computing queue of device n can be represented by:

$$\Psi_{b,n}^t = \max \left\{ B_n^t + o_n^t \zeta(\mathbf{x}_n^t) - \frac{f_n \tau}{\eta_{m,c}} - B_n^{max}, 0 \right\}. \quad (12.4)$$

Here, $\Psi_{b,n}^t > 0$ indicates that a local computing queue overflow event occurs at the n -th device. Then, a corresponding penalty will be incurred to avoid queue overflow.

Case 2: Offloading Tasks to AP If a task is offloaded to the AP, the task will be processed by an uncompressed DNN. The service delay consists of three components: task offloading delay, queuing delay in the edge computing queue, and task processing delay, which are analyzed respectively as follows.¹

Task Offloading Delay Component The offloading delay of the n -th mobile device is given by:

$$d_{n,o}^t = \frac{(1 - o_n^t) \zeta(\mathbf{x}_n^t)}{R_n^t}. \quad (12.5)$$

¹ Note that we assume free transmission backlog in this chapter.

Here, the transmission rate between the n -th mobile device and the AP, R_n^t is represented by:

$$R_n^t = \frac{W}{N} \log_2 \left(1 + \frac{P_T G(H_n^t)}{N_f \sigma^2} \right). \quad (12.6)$$

In the above equation, W , P_T , $G(H_n^t)$, and N_f represent the system bandwidth, transmit power, channel gain, and noise figure, respectively. Here, the background noise is denoted by $\sigma^2 = N_o W/N$, where N_o is thermal noise spectrum density. In this chapter, we assume that channel gain $G(H_n^t)$ varies in terms of channel state H_n^t . Based on extensive real-time measurements, a finite set of channel states \mathcal{H} can be used to model channel state H_n^t [26]. A discrete-time and ergodic Markov chain model can be used to characterize the evolution of channel states. The evolution is given by a transition matrix $\mathbf{P} \in \mathbb{R}^{|\mathcal{H}^t| \times |\mathcal{H}^t|}$.

Task Processing Delay Component The tasks from all mobile devices subscribed to the m -th service are placed in the edge computing queue for the m -th service. Here, $\sum_{n \in \mathcal{N}_m} (1 - o_n^t) \zeta(\mathbf{x}_n^t)$ represents the amount of aggregated tasks. The computing resource is dynamically allocated among multiple services at the AP based on service task arrivals. The dynamic resource allocation can be implemented via a number of existing containerization techniques, such as Dockers and Kubernetes [27]. The computing resource allocation decision vector in time slot t is denoted by $\mathbf{c}^t \in \mathbb{R}^{M \times 1}$, whose each element $0 \leq c_m^t \leq 1$ represents the portion of the allocated computing resource to the m -th service. As such, the processing delay can be calculated by:

$$d_{n,p}^t = \frac{\eta_{m,u} (1 - o_n^t) \zeta(\mathbf{x}_n^t)}{c_m^t f_b}, \forall n \in \mathcal{N}_m. \quad (12.7)$$

Here, f_b represents the computing server's CPU frequency at the AP. The computation intensity of processing the m -th service task by the uncompressed DNN is represented by $\eta_{m,u}$. It is worth noting that $\eta_{m,u} > \eta_{m,c}$. The underlying reason is that the uncompressed DNN consumes more computing resource.

Queuing Delay Component The queuing delay consists of the following two parts:

- The first part is the time taken to process backlogged tasks in the edge computing queue, which is given as follows:

$$d_{n,q}^t = \frac{Q_m^t \eta_{m,u}}{c_m^t f_b}, \forall n \in \mathcal{N}_m. \quad (12.8)$$

In the above equation, Q_m^t represents the edge computing queue backlog for the m -th service in time slot t . The task arrival can be represented by $a_m^t = \sum_{n \in \mathcal{N}_m} (1 - o_n^t) \zeta(\mathbf{x}_n^t)$, and hence the edge computing queue backlog is updated

according to

$$Q_m^{t+1} = \min \left\{ \left[Q_m^t + a_m^t - \frac{c_m^t f_b \tau}{\eta_{m,u}} \right]^+, Q_m^{max} \right\}. \quad (12.9)$$

Similar to that in local computing queues, tasks have to be dropped once the edge computing queue is full, As such, the amount of dropped tasks for the m -th edge computing queue is given as follows:

$$\Psi_{q,m}^t = \max \left\{ Q_m^t + a_m^t - \frac{c_m^t f_b \tau}{\eta_{m,u}} - Q_m^{max}, 0 \right\}. \quad (12.10)$$

In the above equation, $\Psi_{q,m}^t > 0$ indicates that an edge computing queue overflow event occurs.

- The second part is the average waiting time among all newly arrived tasks until all the tasks of mobile device n are processed, which is given as follows:

$$d_{n,w}^t = \frac{\eta_{m,u} \sum_{i \neq n, i \in \mathcal{N}_m} (1 - o_i^t) \zeta(\mathbf{x}_i^t)}{2c_m^t f_b}, \quad (12.11)$$

where $\sum_{i \neq n, i \in \mathcal{N}_m} (1 - o_i^t) \zeta(\mathbf{x}_i^t)$ represents the amount of the aggregated tasks excluding the task of mobile device n .

Overall, taking both local execution and task offloading into consideration, the inference delay of the collaborative DNN in time slot t is calculated as follows:

$$D^t = \sum_{n \in \mathcal{N}} \left(d_{n,l}^t + d_{n,o}^t + d_{n,p}^t + d_{n,q}^t + d_{n,w}^t \right) + w_p \left(\sum_{n \in \mathcal{N}} \mathbb{1}_{\{\Psi_{b,n}^t > 0\}} + \sum_{m \in \mathcal{M}} \mathbb{1}_{\{\Psi_{q,m}^t > 0\}} \right). \quad (12.12)$$

Here, $\mathbb{1}_{\{x\}}$ the indicator function, which takes a value of 1 when the event x is true, and $w_p > 0$ is the positive unit penalty cost for queue overflow. In the above equation, the first term indicates the required delay of completing all the tasks in time slot t , and the second term indicates the penalty for the local and edge computing queues overflow events.

12.3.2.2 Inference Accuracy Analysis

The achieved DNN inference accuracy is determined by two factors: the sampling rate of a task and the type of DNN that executes a task. To obtain the inference accuracy, the following two steps are conducted:

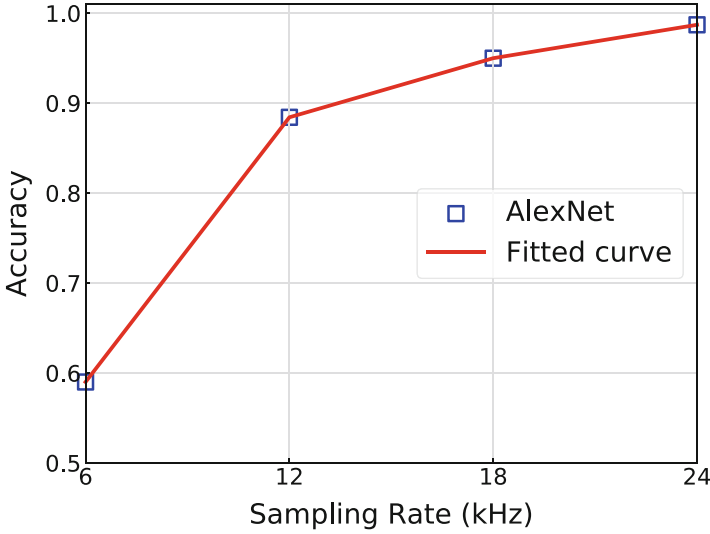


Fig. 12.3 Inference accuracy in terms of different sampling rates on the bearing vibration dataset [29]

- Firstly, we characterize the relationship between the inference accuracy and the sampling rate. The relationship is specified by accuracy function $g(\theta_k), \forall \theta_k \in \mathcal{K}$. To obtain the function, we first implement a DNN inference algorithm, i.e., AlexNet [28]. Then, we use the AlexNet to diagnose facility fault type according to the collected bearing vibration signal [29]. This adopted bearing vibration dataset in the experiment collects the vibration signal of drive end bearings at a sampling rate of 48 KHz, and there are 10 types of possible faults. As shown in Fig. 12.3, inference accuracy grows sub-linearly with the sampling rate. For example, when the sampling rate increases from 18 to 24 KHz, the accuracy increases from 95 to 98.7%. Finally, we measure the accuracy function values in terms of the sampling rates, and the accuracy function is plotted in Fig. 12.3. We can see that the inference accuracy increases with the sampling rate, while the accuracy performance gain decreases at a high sampling rate.
- Secondly, we characterize the relationship between the inference accuracy and the type of DNN via experiments. Here, for the m -th service, the inference accuracy of the compressed DNN is represented by $h_{m,c}$, and that of the uncompressed DNN is represented by $h_{m,u}$. It is worth noting that we have $h_{m,c} < h_{m,u}$. The underlying reason is that an uncompressed DNN achieves higher fault diagnosis accuracy than a compressed DNN.

As the sampling rate selection and the DNN model selection (i.e., task offloading decision) are independent, DNN inference accuracy can be calculated via the product of the accuracy value in terms of the selected sampling rate and the accuracy value in terms of the selected DNN type, i.e.,

$$g \left(\sum_{k \in \mathcal{K}} x_{n,k}^t \theta_k \right) (o_n^t h_{m,c} + (1 - o_n^t) h_{m,u}).$$

As such, in time slot t , the average inference accuracy for the m -th service can be calculated as follows:

$$A_m^t = \sum_{n \in \mathcal{N}_m} \frac{1}{|\mathcal{N}_m|} g \left(\sum_{k \in \mathcal{K}} x_{n,k}^t \theta_k \right) \cdot (o_n^t h_{m,c} + (1 - o_n^t) h_{m,u}). \quad (12.13)$$

The above calculation takes both executing locally and offloading to the AP cases into consideration.

The above DNN inference model can be easily extended and applied to cases when other inference methods are adopted. The reason is that the accuracy values in terms of sampling rates and DNN types can be acquired via practical experiments rather than theoretical models.

12.3.3 Joint Sampling Rate Selection and Resource Allocation Problem

12.3.3.1 Constrained Markov Decision Process

In the DNN inference services, not only the service delay is required to be minimized, but also their long-term accuracy requirements should be guaranteed. The CMDP is a class of problems that target at maximizing the long-term reward while satisfying the constraints on the long-term cost [30]. Hence, such problem is suitable to be modeled as a CMDP. We define the action, state, reward, and state transition matrix of the CMDP as follows:

Action The action of the CMDP includes the sampling rate selection, task offloading, and edge computing resource allocation decisions, i.e.,

$$\hat{a}^t = \{\mathbf{X}^t, \mathbf{o}^t, \mathbf{c}^t\}.$$

It is worth noting that the action's components should satisfy following constraints:

- The sampling rate selection decision is constrained by $x_{n,k}^t \in \{0, 1\}$.
- The binary task offloading decision is required, i.e., $o_n^t \in \{0, 1\}$.

- The continuous computing resource allocation decision is constrained by $\sum_{m \in \mathcal{M}} c_m^t \leq 1$ and $0 \leq c_m^t \leq 1$.

The constraint of each action component is satisfied via projecting it into a feasible action set.

State The state of the CMDP includes four components: local computing queues backlog of mobile devices B_n^t , edge computing queues backlog Q_m^t , channel conditions of mobile devices H_n^t , and the raw data size of the generated tasks at mobile devices ξ_n^t . Hence, we have

$$\hat{s}^t = \{\{B_n^t\}_{n \in \mathcal{N}}, \{Q_m^t\}_{m \in \mathcal{M}}, \{H_n^t\}_{n \in \mathcal{N}}, \{\xi_n^t\}_{n \in \mathcal{N}}\}. \quad (12.14)$$

In the above state, both queue backlogs, including $\{B_n^t\}_{n \in \mathcal{N}}$ and $\{Q_m^t\}_{m \in \mathcal{M}}$, adopt a unit in bits. As such, it results in a large state space, especially when the number of mobile devices is large.

Reward The reward of the CMDP is designed to achieve the service delay minimization, as shown in (12.12) in time slot t . In this way, the reward is defined as

$$\hat{r}^t(\hat{s}^t, \hat{a}^t) = -D^t.$$

State Transition Probability State transition probability of the CMDP is given as follows:

$$\begin{aligned} \Pr(\hat{s}^{t+1} | \hat{s}^t, \hat{a}^t) &= \prod_{n \in \mathcal{N}} \Pr(B_n^{t+1} | B_n^t, x_{n,k}^t, o_n^t) \cdot \\ &\quad \prod_{m \in \mathcal{M}} \Pr(Q_m^{t+1} | Q_m^t, \mathbf{X}^t, \mathbf{o}^t) \cdot \prod_{n \in \mathcal{N}} \Pr(H_n^{t+1} | H_n^t) \cdot \\ &\quad \prod_{n \in \mathcal{N}} \Pr(\xi_n^{t+1} | \xi_n^t). \end{aligned} \quad (12.15)$$

The above equality holds since different state terms are independent. Specifically, the first two terms are controlled by the evolution of both local computing queues and edge computing queues, as detailed in (12.3) and (12.9), respectively. The third term is evolved based on the discrete-time Markov chain of channel conditions as mentioned above. The last term is determined by the memoryless task arrival pattern. It is worth noting that each of those state terms only depends on its previous state terms. Such behavior indicates the state transition is Markovian.

In our case, we aim to find a stationary policy $\pi \in \mathcal{I}$ that can dynamically configure sampling rates selection \mathbf{X}^t , task offloading \mathbf{o}^t , and edge computing resource allocation \mathbf{c}^t based on state \hat{s}^t . The policy can minimize the service delay and guarantee long-term inference accuracy requirements $\{A_m^{th}\}_{m \in \mathcal{M}}$ simultaneously. To acquire the policy, the optimization problem is formulated as follows:

$$\mathbf{P}_0 : \min_{\pi \in \Pi} \lim_{T \rightarrow \infty} \frac{1}{T} \sum_{t=1}^T \mathbb{E}_{\pi} [D^t] \quad (12.16a)$$

$$\text{s.t. } \lim_{T \rightarrow \infty} \frac{1}{T} \sum_{t=1}^T A_m^t \geq A_m^{th}, \forall m \in \mathcal{M}. \quad (12.16b)$$

The above problem can be deemed as a CMDP.

It is challenging to directly solve the above CMDP via dynamic programming solutions [30]. The reasons are two-fold:

- Firstly, the state transition probability is unknown due to the lack of statistical information on the channel condition variation and task arrival patterns of all mobile devices.
- Secondly, even if the state transition probabilities are known, large action space and state space that grow with respect to the number of mobile devices incur an extremely high computational complexity, which makes dynamic programming solutions intractable.

To solve the CMDP in dynamic environments, we aim to adopt a deep RL-based algorithm. The benefit is that RL-based algorithm can be applied in large-scale networks without requiring statistical information of network dynamics.

However, the existing RL algorithms, such as DDPG, are designed to solve MDP problems without considering policy constraints. Due to the underlying differences between CMDP and MDP, CMDP cannot be solved via traditional RL algorithms.

To solve the problem, we propose a novel learning-based solution for CMDP in the following.

12.3.4 Deep RL-Based Solution

The proposed deep RL-based solution consists of the following three steps:

- **Step 1:** We leverage the Lyapunov optimization technique to deal with the long-term constraints and transform the problem into an MDP, which is suitable to be solved by RL algorithms.

- **Step 2:** We develop a deep RL-based algorithm to solve the MDP.
- **Step 3:** We embed an optimization subroutine in the proposed RL algorithm to directly obtain the optimal edge computation resource allocation.

These three steps are detailed in the following.

12.3.4.1 Markov Decision Process Transformation (Step 1)

To solve problem \mathbf{P}_0 , the major challenge is to handle the long-term constraints. To address this challenge, we leverage the Lyapunov technique [31, 32].

The basic idea of the step is to construct accuracy deficit queues to characterize the satisfaction status of the long-term accuracy constraints, thereby guiding the learning agent to meet the long-term accuracy constraints.

As such, the problem is transformed in the following way:

- Firstly, inference accuracy *deficit queues* are constructed for all services. The dynamics of the queue evolve as follows:

$$Z_m^{t+1} = \left[A_m^{th} - A_m^t + Z_m^t \right]^+, \forall m \in \mathcal{M}. \quad (12.17)$$

Here, the deviation of the achieved instantaneous accuracy from the long-term accuracy requirement is represented by Z_m^t . Its initial state is set to $Z_m^0 = 0$. Next, we introduce a Lyapunov function to characterize the satisfaction status of the long-term accuracy constraint. The Lyapunov function is defined as [31–33]

$$L(Z_m^t) = \frac{(Z_m^t)^2}{2}.$$

In the above equation, a smaller value of $L(Z_m^t)$ means better long-term accuracy constraint satisfaction.

- Secondly, to guarantee the long-term accuracy constraints, the Lyapunov function should be consistently pushed to a relatively low value. Therefore, a *one-shot Lyapunov drift* is introduced to capture the Lyapunov function's variation across two subsequent time slots [31]. When Z_m^t is given, we define the one-shot Lyapunov drift as follows: $\Delta(Z_m^t) = L(Z_m^{t+1}) - L(Z_m^t)$. We can obtain an upper bound as follows:

$$\begin{aligned}
\Delta(Z_m^t) &= \frac{1}{2} \left((Z_m^{t+1})^2 - (Z_m^t)^2 \right) \\
&\leq \frac{1}{2} \left((Z_m^t + A_m^{th} - A_m^t)^2 - (Z_m^t)^2 \right) \\
&= \frac{1}{2} (A_m^{th} - A_m^t)^2 + Z_m^t (A_m^{th} - A_m^t) \\
&\leq C_m + Z_m^t (A_m^{th} - A_m^t).
\end{aligned} \tag{12.18}$$

In the above equation, $C_m = (A_m^{th} - A_m^{min})^2 / 2$ is a constant. Here, A_m^{min} is the lowest inference accuracy, which can be required for service m . Due to the substitution of (12.17), the first inequality holds. The second inequality can be derived due to $A_m(\mathbf{X}^t, \mathbf{o}^t) \geq A_m^{min}$.

- Thirdly, leveraging the Lyapunov optimization theory, the original CMDP to minimize the service delay and guarantee the long-term accuracy requirements can be transformed to a problem of minimizing a *drift-plus-cost*. The transformed problem is given as follows:

$$\sum_{m \in \mathcal{M}} \Delta(Z_m^t) + V \cdot D^t \leq \sum_{m \in \mathcal{M}} C_m + \sum_{m \in \mathcal{M}} Z_m^t (A_m^{th} - A_m^t) + V \cdot D^t. \tag{12.19}$$

In the above equation, the inequality holds due to the upper bound in (12.18). Here V represents a positive parameter, which can adjust the tradeoff between the satisfaction status of the long-term accuracy constraints and the service delay minimization. The rationale behind this is that when the long-term accuracy constraint is violated, i.e., $Z_m^t > 0$, it is more urgent to stratify the long-term constraints via improving the instantaneous inference accuracy than to reduce the service delay.

Through this transformation, we reformulate the CMDP problem as a regular MDP problem. The objective of the MDP is to minimize the upper bound of drift-plus-cost as shown in (12.19). In such a reformulated MDP, we should modify the action, state, reward, and state transition matrix since the accuracy deficit queues are incorporated. The modified elements of the MDP are given as follows:

Modified Action The action is the same as that in the CMDP, i.e.,

$$a^t = \hat{a}^t = \{\mathbf{X}^t, \mathbf{o}^t, \mathbf{c}^t\}.$$

Modified State The accuracy deficit queue backlog of services $\{Z_m^t\}_{m \in \mathcal{M}}$ should be incorporated in the state space, as compared to the state of the CMDP. The modified state is given by:

$$s^t = \{\hat{s}^t, \{Z_m^t\}_{m \in \mathcal{M}}\}. \tag{12.20}$$

Modified Reward To minimize the drift-plus-cost in (12.19), the reward is modified as follows:

$$r^t = -V \cdot D^t - \sum_{m \in \mathcal{M}} Z_m^t (A_m^{th} - A_m^t). \quad (12.21)$$

It is worth noting that we ignore the constant term $\sum_{m \in \mathcal{M}} C_m$ in (12.19) in the reward for simplicity.

Modified State Transition Probability The evolution of state transition probability changes due to the incorporation of accuracy deficit queue backlogs in the state, which is detailed as follows:

$$\Pr(s^{t+1}|s^t, a^t) = \Pr(\hat{s}^{t+1}|\hat{s}^t, \hat{a}^t) \cdot \prod_{m \in \mathcal{M}} \Pr(Z_m^{t+1}|Z_m^t, \mathbf{X}^t, \mathbf{o}^t). \quad (12.22)$$

In the above equation, the second term represents the evolution of the accuracy deficit queue backlog based on (12.17). It is clear that the Markovian property holds for the overall state transition.

Based on the above reformulation and modification, we transform problem \mathbf{P}_0 into an MDP problem as follows:

$$\mathbf{P}_1 : \min_{\pi \in \Pi} \lim_{T \rightarrow \infty} \frac{1}{T} \sum_{t=1}^T \mathbb{E}_{\pi} \left[\sum_{m \in \mathcal{M}} Z_m^t (A_m^{th} - A_m^t) + V \cdot D^t \right]. \quad (12.23)$$

Similar to solving CMDP as mentioned above, using dynamic programming solutions to solve an MDP faces the curse of dimensionality issue since the state space is large. Therefore, we propose a deep RL-based algorithm to deal with the MDP in the following.

12.3.4.2 Optimization Subroutine for Resource Allocation (Step 3)

For better understanding, we first introduce the third step in the optimization subroutine and then introduce the second step in the RL algorithm design.

As mentioned above, problem \mathbf{P}_1 can be solved by RL algorithms. However, we can leverage an inherent property of edge computing resource allocation to reduce the training complexity of RL algorithms. Based on theoretical analysis on (12.23), we find that the edge computing resource allocation and the inference accuracy performance are independent. Specifically, the edge computing resource allocation only impacts the one-shot service delay performance. Therefore, in time slot t , when task offloading and sampling rate selection decisions are given, we can obtain the optimal computing resource allocation decision via solving the following optimization problem:

$$\mathbf{P}_2 : \min_{\mathbf{c}^t} D_t$$

$$\text{s.t. } \sum_{m \in \mathcal{M}} c_m^t \leq 1 \quad (12.24a)$$

$$0 \leq c_m^t \leq 1. \quad (12.24b)$$

Furthermore, an analysis of (12.12) demonstrates that the edge computing resource allocation only impacts the task processing delay and queuing delay at the AP, i.e., $\sum_{n \in \mathcal{N}} (d_{n,p}^t + d_{n,q}^t + d_{n,w}^t)$. In addition, we find that the aggregated delay from the perspective of all devices is equivalent to the aggregated delay from the perspective of all services. As such, we can rewrite the objective function in \mathbf{P}_2 as $\sum_{m \in \mathcal{M}} d_m^t$. As such, we have

$$d_m^t = \sum_{n \in \mathcal{N}_m} \left(\frac{\eta_{m,u} (1 - o_n^t) \zeta(\mathbf{x}_n^t)}{c_m^t f_b} + \frac{Q_m^t \eta_{m,u}}{c_m^t f_b} + \frac{\eta_{m,u} \sum_{i \neq n, i \in \mathcal{N}_m} (1 - o_i^t) \zeta(\mathbf{x}_i^t)}{2c_m^t f_b} \right). \quad (12.25)$$

The above equation represents the experienced delay of the m -th service. Through analysis, we show the convexity property of the problem. Then, the following theorem can be used to obtain the optimal edge computation resource allocation in each time slot.

Theorem 12.1 *The optimal edge computing resource allocation for problem \mathbf{P}_2 is given by:*

$$c_m^{t,*} = \frac{\sqrt{\Lambda_m^t}}{\sum_{m \in \mathcal{M}} \sqrt{\Lambda_m^t}}, \forall m \in \mathcal{M}, \quad (12.26)$$

where

$$\Lambda_m^t = \sum_{n \in \mathcal{N}_m} \left(\eta_{m,u} (1 - o_n^t) \zeta(\mathbf{x}_n^t) + Q_m^t \eta_{m,u} + \frac{\eta_{m,u}}{2} \sum_{i \neq n, i \in \mathcal{N}_m} (1 - o_i^t) \zeta(\mathbf{x}_i^t) \right). \quad (12.27)$$

Proof The theorem is proved via the following two steps:

- Firstly, we prove the problem to be a convex optimization problem. For simplicity, t is omitted in the proof. By the definition of Λ_m in (12.27), we can rewrite the objective function as $\sum_{m \in \mathcal{M}} \Lambda_m / (c_m f_b)$. The second-order derivative of the objective function can be derived as $2\Lambda_m / (f_b c_m^3) > 0$. In addition, we know that the inequality constraint is linear. Hence, the problem is a convex optimization problem.
- Secondly, we construct a Lagrange function for the problem by ignoring the inequality constraints, which is given as follows:

$$\mathcal{L}(\mathbf{c}, a) = \sum_{m \in \mathcal{M}} \frac{\Lambda_m}{c_m f_b} + a \left(\sum_{m \in \mathcal{M}} c_m - 1 \right). \quad (12.28)$$

Here, a represents the Lagrange multiplier. According to Karush–Kuhn–Tucker conditions for convex optimization [34], the following equation is obtained:

$$\frac{\partial \mathcal{L}(\mathbf{c}, a)}{\partial c_m} = -\frac{\Lambda_m}{f_b c_m^2} + a = 0, \forall m \in \mathcal{M}. \quad (12.29)$$

Here, $c_m^* = \sqrt{\Lambda_m / a f_b}$, $\forall m \in \mathcal{M}$, can be obtained by solving the above equation. Then, we substitute the above result into the complementary slackness condition $\sum_{m \in \mathcal{M}} c_m^* - 1 = 0$. Then, the optimal value of a can be given by $a^* = (\sum_{m \in \mathcal{M}} \sqrt{\Lambda_m})^2 / f_b$. Based on the above equation, a^* takes a positive value, and hence $\{c_m^*\}_{m \in \mathcal{M}}$ are positive values, which shows that constraint (12.24b), i.e., $c_m^t \geq 0$, $\forall m \in \mathcal{M}$, is automatically satisfied. We can then prove Theorem 12.1 by substituting a^* into the complementary slackness condition. \square

This optimization subroutine for the edge computing resource allocation is embedded in the following proposed deep RL-based algorithm. As such, we can reduce the training complexity of the proposed RL algorithm. The reason is that it is no longer necessary to train the neural networks to obtain an optimal edge computing resource allocation policy.

12.3.4.3 Deep RL-Based Algorithm (Step 2)

In the following, we propose a deep RL-based algorithm to solve problem \mathbf{P}_1 . The proposed algorithm is extended from the well-known DDPG algorithm [35]. However, the DDPG algorithm and the proposed algorithm are different. The main difference is that we embed the above optimization subroutine for computing resource allocation into the RL algorithm to reduce the training complexity. The proposed algorithm can be deployed at the AP that is in charge of collecting the network state information and enforcing the policy to all connected mobile devices.

In the proposed algorithm, the learning agent consists of an actor network that determines the action based on the current state and a critic network that evaluates the determined action based on the reward feedback from the environment. The actor network and the critic network are denoted by $\mu(s|\phi^\mu)$ and $Q(s, a|\phi^Q)$, respectively. The corresponding neural network weights are represented by ϕ^μ and ϕ^Q , respectively. The details of the deep RL-based algorithm are shown in Algorithm 1.

Algorithm 1 Deep RL-based algorithm for sampling rate adaption and resource allocation

Initialize all neural networks and the experience replay memory;

for each episode **do**

 Reset the environment and obtain initial state s_0 ;

for time slot $t \in \mathcal{T}$ **do**

 Determine sampling rate selection and task offloading actions $\{\mathbf{X}^t, \mathbf{o}^t\}$ according to s^t ;

 Determine edge computing resource allocation action \mathbf{c}^t by (12.26);

 Send joint action $a^t = \{\mathbf{X}^t, \mathbf{o}^t, \mathbf{c}^t\}$ to all mobile devices by the AP;

 Execute the joint action at mobile devices;

 Observe reward r^t and new state s^{t+1} ;

 Store transition $\{s^t, a^t, r^t, s^{t+1}\}$ in the experience replay memory;

 Sample a random minibatch transitions from the experience replay memory;

 Train the critic and actor network by (12.30) and (12.31), respectively;

 Update target networks by (12.32).

end for

end for

The proposed algorithm operates in a time-slotted manner, which consists of the following three stages:

- **Stage 1: Obtain experience by interacting with the environment.** The actor network generates the task offloading and sampling rate selection decisions based on the current network state s^t . The decisions are generated with an additive policy exploration noise that follows a Gaussian distribution $\mathcal{N}(0, \sigma^2)$. Additionally, the edge computation resource allocation action is generated by the optimization subroutine. Next, the joint action is executed at all mobile devices, and the corresponding reward r^t is obtained. In addition, we can observe the next state s^{t+1} from the environment. The state transition tuple $\{s^t, a^t, r^t, s^{t+1}\}$ is stored in the experience replay memory for actor and critic network training.
- **Stage 2: Train the actor and critic network based on the stored experience.** A minibatch of transitions are randomly sampled from the experience replay memory to break experience correlation, thereby avoiding the divergence issue caused by DNN. By minimizing the following loss function, the critic network is trained:

$$Loss(\phi^Q) = \frac{1}{N_b} \sum_{i=1}^{N_b} (y_i - Q(s_i, a_i | \phi^Q))^2, \quad (12.30)$$

where

$$y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1} | \phi^{\mu'}) | \phi^Q).$$

Here, N_b represents the minibatch size $\mu'(s | \phi^{\mu'})$ and $Q'(s, a | \phi^Q)$ indicate actor and critic target networks with weights $\phi^{\mu'}$ and ϕ^Q , respectively. The actor network is trained via the following policy gradient:

$$\nabla_{\phi^\mu} \approx \frac{1}{N_b} \sum_{i=1}^{N_b} \nabla_a Q(s_i, a | \phi^Q) |_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu} \mu(s_i | \phi^\mu) |_{s_i}. \quad (12.31)$$

- **Stage 3: Update target networks.** The actor and critic target networks are softly updated according to the following equations to ensure network training stability, i.e.,

$$\begin{aligned} \phi^{Q'} &= \delta \phi^Q + (1 - \delta) \phi^{Q'} \\ \phi^{\mu'} &= \delta \phi^\mu + (1 - \delta) \phi^{\mu'}. \end{aligned} \quad (12.32)$$

In the above equations, $0 < \delta \ll 1$ is the target network update ratio.

Remark Traditional RL algorithms, e.g., DDPG, can be applied to solve MDP problems, in which learning agents seek to optimize a long-term reward without policy constraints, while they cannot deal with constrained long-term optimization problems [30, 36]. Our proposed deep RL-based algorithm can address long-term constraints within the RL framework by the modification of reward based on the Lyapunov optimization technique. In addition, an optimization subroutine is embedded in our algorithm to further reduce the training complexity.

12.4 Performance Evaluation

12.4.1 Experiment Setup

We consider a smart factory in which mobile devices such as vibration sensors are randomly scattered. Those devices mounted on industrial facilities (e.g., robot arms) capture the operating information. Those sensing data are then either locally processed or offloaded to an AP in the factory.

DNN Inference Services We consider two kinds of DNN inference services:

- *Type I Service:* A facility fault diagnosis service that identifies the type of fault according to the collected bearing vibration signal dataset [29]. Because the period of a time slot is one second, we configure the task data size to be the data volume of a one-second signal, given by the multiplication of the raw sampling rate and the signal quantization parameter. The bearing vibration signal is captured at 48 KHz sampling rate and 16 bit quantization. The resulting task data size is 768 Kb. For this type of service, we set the long-term accuracy threshold to be 0.8.

- *Type II Service*: An extended service from the Type I that diagnoses facility fault based on a low-grade bearing vibration dataset at higher inference accuracy requirement, 0.9. The low-grade dataset senses the vibration at a lower sampling rate of 32 KHz, and the resulting task data size is 512 Kb.

We assume the task arrival rates of both services at each device in each time slot form a uniform distribution. Four potential sampling rates for each device are considered in the simulation, which are 25%, 50%, 75%, and 100% of the raw sampling rate. Accordingly, based on extensive experiments on the dataset [29], the required accuracy to those sampling rates are 0.59, 0.884, 0.950, and 0.987, and the balance parameter, V , is set to be 0.05.

Neural Network Structure To train the proposed deep RL-based algorithm, we set the learning rate of the actor and the critic to be 10^{-4} and 10^{-3} , respectively. The hidden units of both the actor and the critic are set to be (64, 32), while the ReLU function is employed for hidden activation. Note that the Tanh function is used for actor output activation. The training process lasts for 1000 episodes, each of which consists of 200 time slots.

Benchmark We consider the following two benchmark algorithms for performance comparison:

- *Delay myopic*: Each device dynamically determines the sampling rate and task offloading decisions, to maximize the one-step reward in (12.21) based on the network state.
- *Static configuration*: Each device follows a fixed configuration on the sampling rate and the task offloading, which satisfy the services' accuracy requirements.

12.4.2 Convergence Performance

Figure 12.4 shows the performance comparison of service delay in the training stage. The average service delay drops as the training continues, which suggests the convergence of the proposed RL-based algorithm. In addition, Fig. 12.5 illustrates the accuracy performance for both services with training episodes. The accuracy performance fluctuates at the beginning of the training. But after around 1000 episodes of training, the average accuracy converges to the required level.

12.4.3 Impact of Task Arrival Rate

After the algorithm is well-trained offline, the performance of the proposed inference algorithm is evaluated in an online scenario. Figure 12.6 gives the comparison on the average service delay with respect to different task arrival rates for $W =$

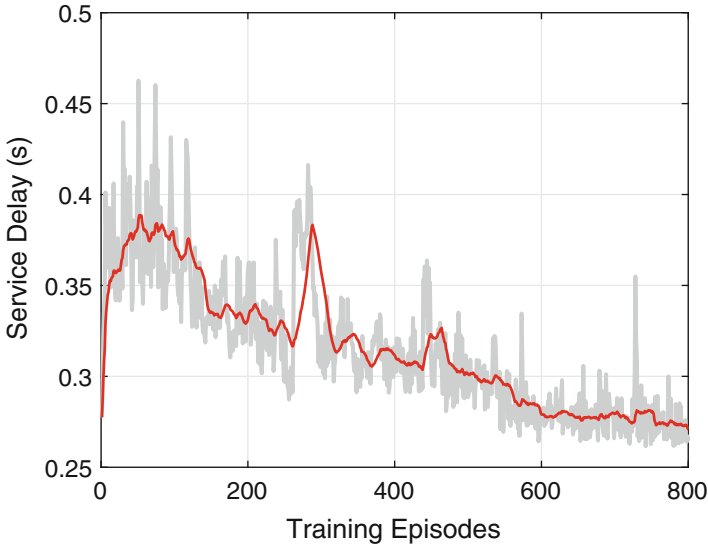


Fig. 12.4 Average delay performance of the proposed algorithm with respect to training episodes in the training stage

20 MHz. Each arrival rate is added up with a 95% confidence interval. It is shown that the service delay grows with the task arrival rate of the constrained communication and computing resources. Meanwhile, the proposed RL-based algorithm gives significantly lower service delay than the benchmark schemes. This is because the proposed algorithm can capture network dynamics, including the pattern of task arrival and channel condition variation, by continuously interacting with the environment. Such knowledge is learnt and utilized by the algorithm to make online decisions that improves long-term performance. In contrast, benchmark schemes only focus on performance in the short term, and they cannot adapt to network dynamics either. In particular, the proposed algorithm reduces the average service delay by 19% and 25%, respectively, as compared with delay myopic and static configuration schemes.

We also give the boxplot accuracy distribution of both services with respect to different task arrival rates in Fig. 12.7. In this figure, the long-term accuracy requirements for both services are 0.8 and 0.9, respectively. The proposed algorithm is able to guarantee the long-term accuracy requirements of both services, with the maximum error probability less than 0.5%.

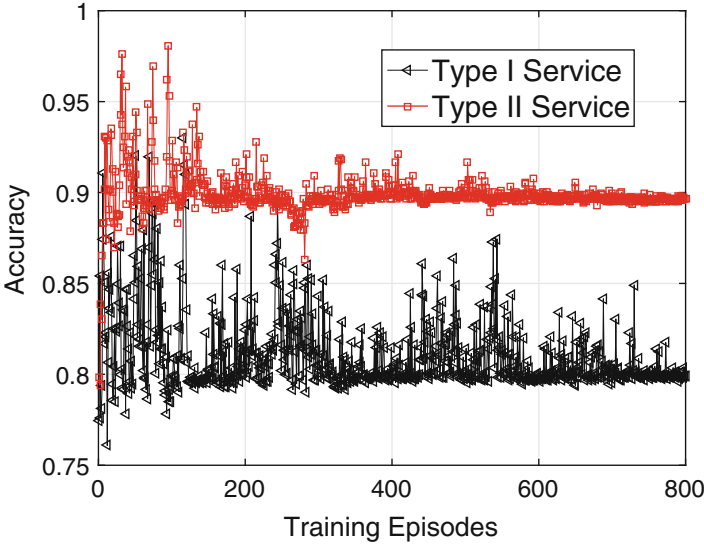


Fig. 12.5 Inference accuracy performance of the proposed learning algorithm with respect to training episodes in the training stage

12.4.4 Impact of Optimization Subroutine

We further evaluate the performance of the proposed algorithm with a fixed computing resource allocation strategy (referred to as proposed-fixed). This strategy allocates edge computing resource based on the average computing demand of two services. As shown in Fig. 12.8, the proposed algorithm provides significant performance gain in case of limited edge computing resource. Specifically, the performance gain in reducing the service delay decreases from $1.98\times$ at 1 GHz CPU frequency to only $1.02\times$ at 1.2 GHz CPU frequency. The underlying reason is that efficient resource allocation weighs higher in resource-constrained scenarios. The simulation curves confirm the effectiveness of the optimization subroutine of computing resource allocation. In light of this optimization subroutine, reducing the training complexity of the proposed RL algorithms can also be considered.

12.5 Conclusion

In this chapter, we have jointly investigated the collaborative DNN inference with sampling rate adaption and resource allocation problem in wireless networks. A deep RL-based algorithm has been devised to capture the pattern of the channel variation and the task arrival, which is then employed to deliver accuracy-guaranteed

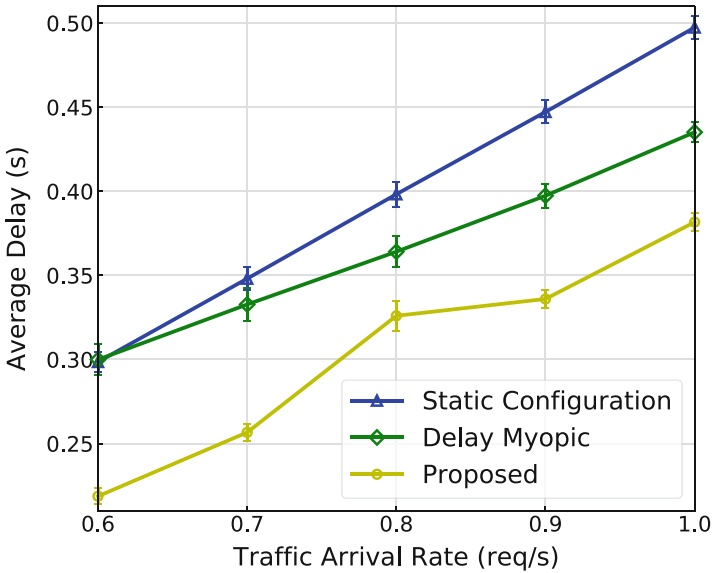


Fig. 12.6 Performance comparison of the service delay in terms of different task arrival rates

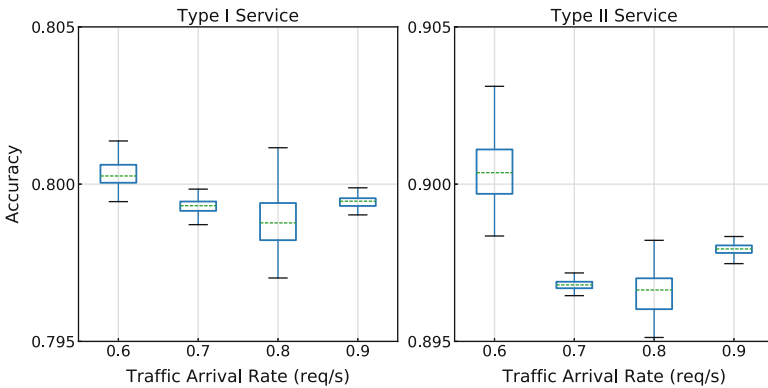


Fig. 12.7 Inference accuracy performance comparison in terms of different task arrival rates

DNN inference services. The proposed algorithm can dynamically reduce the service delay, without requiring prior information of network dynamics.

For DNN inference, further research is required in the following aspects: (1) DNN inference performance should be investigated in the mobile scenarios considering device mobility; and (2) instead of task offloading, the DNN model can be partitioned into a device-side model and a server-side model for collaborative inference. As such, the partition point and resource (computing and spectrum) allocation should be judiciously considered, especially in dynamic network environments.

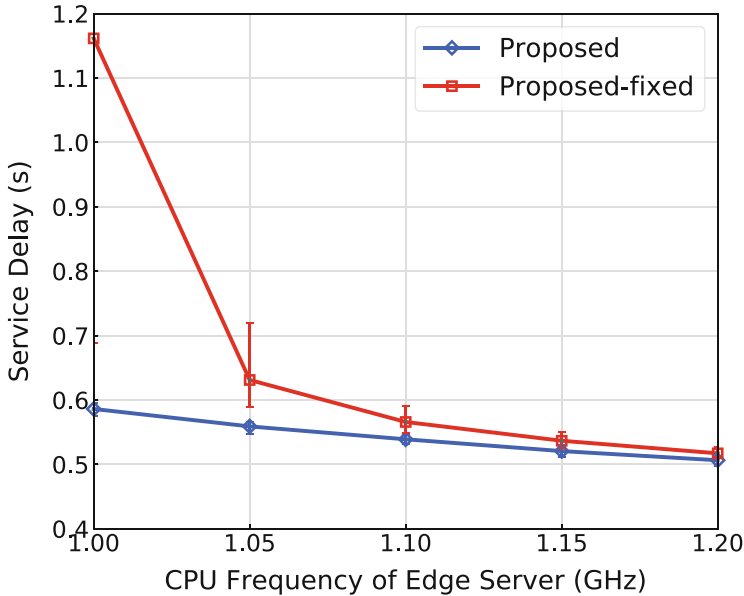


Fig. 12.8 Service delay in terms of CPU frequency of the edge server

References

1. W. Wu, P. Yang, W. Zhang, C. Zhou, X. Shen, Accuracy-guaranteed collaborative DNN inference in industrial IoT via deep reinforcement learning. *IEEE Trans. Ind. Inf.* **17**(7), 4988–4998 (2021)
2. H. Hu, B. Tang, X. Gong, W. Wei, H. Wang, Intelligent fault diagnosis of the high-speed train with big data based on deep neural networks. *IEEE Trans. Ind. Inf.* **13**(4), 2106–2116 (2017)
3. W. Zhang, D. Yang, H. Peng, W. Wu, W. Quan, H. Zhang, X. Shen, Deep reinforcement learning based resource management for DNN inference in industrial IoT. *IEEE Trans. Veh. Technol.* **70**(8), 7605–7618 (2021)
4. G. Gobieski, B. Lucia, N. Beckmann, Intelligence beyond the edge: Inference on intermittent embedded systems, in *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems* (2019), pp. 199–213
5. Y. Chen, T. Krishna, J. S. Emer, V. Sze, Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. *IEEE J. Solid-State Circuits* **52**(1), 127–138 (2017)
6. D.A. Chekired, L. Khoukhi, H.T. Mouftah, Industrial IoT data scheduling based on hierarchical fog computing: a key for enabling smart factory. *IEEE Trans. Ind. Inf.* **14**(10), 4590–4602 (2018)
7. E. Li, L. Zeng, Z. Zhou, X. Chen, EdgeAI: On-demand accelerating deep neural network inference via edge computing. *IEEE Trans. Wirel. Commun.* **19**(1), 447–457 (2020)
8. S. Han, H. Mao, W.J. Dally, Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding (2015). Preprint arXiv:1510.00149
9. S. Teerapittayanon, B. McDanel, H. Kung, Branchynet: Fast inference via early exiting from deep neural networks, in *Proceedings of the IEEE International Conference on Pattern Recognition* (2016), pp. 2464–2469

10. G. Chen, W. Choi, X. Yu, T. Han, M. Chandraker, Learning efficient object detection models with knowledge distillation, in *Proceedings of the Conference on Neural Information Processing Systems* (2017), pp. 742–751
11. P. Yang, F. Lyu, W. Wu, N. Zhang, L. Yu, X. Shen, Edge coordinated query configuration for low-latency and accurate video analytics. *IEEE Trans. Ind. Inf.* **16**(7), 4855–4864 (2020).
12. W. Zhuang, Q. Ye, F. Lyu, N. Cheng, J. Ren, SDN/NFV-empowered future IoV with enhanced communication, computing, and caching. *Proc. IEEE* **108**(2), 274–291 (2019)
13. M. Li, J. Gao, L. Zhao, X. Shen, Deep reinforcement learning for collaborative edge computing in vehicular networks. *IEEE Trans. Cogn. Commun. Netw.* **6**(4), 1122–1135 (2020)
14. W. Wu, N. Chen, C. Zhou, M. Li, X. Shen, W. Zhuang, X. Li, Dynamic RAN slicing for service-oriented vehicular networks via constrained learning. *IEEE J. Sel. Areas Commun.* **39**(7), 2076–2089 (2021)
15. C. Zhou, W. Wu, H. He, P. Yang, F. Lyu, N. Cheng, X. Shen, Deep reinforcement learning for delay-oriented IoT task scheduling in space-air-ground integrated network. *IEEE Trans. Wirel. Commun.* **20**(2), 911–925 (2021).
16. Y. Tang, N. Cheng, W. Wu, M. Wang, Y. Dai, X. Shen, Delay-minimization routing for heterogeneous VANETs with machine learning based mobility prediction. *IEEE Trans. Veh. Technol.* **68**(4), 3967–3979 (2019)
17. X. Shen, J. Gao, W. Wu, K. Lyu, M. Li, W. Zhuang, X. Li, J. Rao, AI-assisted network-slicing based next-generation wireless networks. *IEEE Open J. Veh. Technol.* **1**, 45–66 (2020)
18. X. Shen, J. Gao, W. Wu, M. Li, C. Zhou, W. Zhuang, Holistic network virtualization and pervasive network intelligence for 6G. *IEEE Commun. Surveys Tuts.* **24**(1), 1–30 (2022)
19. W. Wu, C. Zhou, M. Li, H. Wu, H. Zhou, N. Zhang, W. Zhuang, X. Shen, AI-native network slicing for 6G networks. *IEEE Wirel. Commun.* **29**(1), 96–103 (2022)
20. S. Wang, Y. Guo, N. Zhang, P. Yang, A. Zhou, X. Shen, Delay-aware microservice coordination in mobile edge computing: a reinforcement learning approach. *IEEE Trans. Mobile Comput.* **20**(3), 939–951 (2021)
21. P. Yang, N. Zhang, S. Zhang, L. Yu, J. Zhang, X. Shen, Content popularity prediction towards location-aware mobile edge caching. *IEEE Trans. Multimedia* **21**(4), 915–929 (2019)
22. W. Wu, N. Zhang, N. Cheng, Y. Tang, Aldubaikhy, K.X. Shen, Beef up mmWave dense cellular networks with D2D-assisted cooperative edge caching. *IEEE Trans. Veh. Technol.* **68**(4), 3890–3904 (2019)
23. W. Wu, N. Cheng, N. Zhang, P. Yang, W. Zhuang, X. Shen, Fast mmwave beam alignment via correlated bandit learning. *IEEE Trans. Wirel. Commun.* **18**(12), 5894–5908 (2019)
24. W. Wu, N. Cheng, N. Zhang, P. Yang, K. Aldubaikhy, X. Shen Performance analysis and enhancement of beamforming training in 802.11ad. *IEEE Trans. Veh. Technol.* **69**(5), 5293–5306 (2020)
25. W. Zhang, D. Yang, Y. Xu, X. Huang, J. Zhang, M. Gidlund, DeepHealth: a self-attention based method for instant intelligent predictive maintenance in industrial Internet of things. *IEEE Trans. Ind. Inf.* **17**(8), 5461–5473 (2021)
26. L. Lei, Y. Kuang, X. Shen, K. Yang, J. Qiao, Z. Zhong, Optimal reliability in energy harvesting industrial wireless sensor networks. *IEEE Trans. Wirel. Commun.* **15**(8), 5399–5413 (2016)
27. D. Bernstein, Containers and cloud: From LXC to Docker to Kubernetes. *IEEE Cloud Comput.* **1**(3), 81–84 (2014)
28. A. Krizhevsky, I. Sutskever, E. Hinton, ImageNet classification with deep convolutional neural networks, in *Proceedings of the Conference on Neural Information Processing Systems* (2012), pp. 1097–1105
29. International Data Corporation. [Online]. Available: <https://csegroups.case.edu/bearingdatacenter/pages/download-data-file>. Accessed: Jun. 2021
30. E. Altman, *Constrained Markov Decision Processes* (CRC Press, Boca Raton, 1999) Available via DIALOG
31. M.J. Neely, Stochastic network optimization with application to communication and queueing systems. *Synth. Lect. Commun. Netw.* **3**(1), 1–211 (2010)

32. J. Luo, F. Yu, Q. Chen, L. Tang, Adaptive video streaming with edge caching and video transcoding over software-defined mobile networks: A deep reinforcement learning approach. *IEEE Trans. Wirel. Commun.* **19**(3), 1577–1592 (2020)
33. J. Xu, L. Chen, P. Zhou, Joint service caching and task offloading for mobile edge computing in dense networks, in *Proceedings of the IEEE International Conference on Computer Communications* (2018), pp. 207–215
34. S. Boyd, S.P. Boyd, L. Vandenberghe, *Convex Optimization* (Cambridge University Press, Cambridge, Vandenberghe). Available via DIALOG
35. T.P. Lillicrap, J.J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, D. Wierstra, Continuous control with deep reinforcement learning, in *Proceedings of the International Conference on Learning Representations* (2016)
36. Q. Liang, F. Que, E. Modiano, Accelerated primal-dual policy optimization for safe reinforcement learning (2021). Available via DIALOG, <https://arxiv.org/abs/1802.06480>