DisCOV: Distributed COVID-19 Detection on X-Ray Images With Edge-Cloud Collaboration

Xiaolong Xu, Hao Tian, Xuyun Zhang, Lianyong Qi, Qiang He, *Member, IEEE*, and Wanchun Dou

Abstract—Currently, the world is experiencing the rapid spread of Coronavirus Disease 2019 (COVID-19). Since the epidemic continues to take a devastating impact on the society, economy, and healthcare, the real-time detection of COVID-19 is essential for fast and cost-effective diagnosis services. Fortunately, deep learning (DL), as a promising technology, enables the COVID-19 diagnosis services on chest X-ray (CXR) images. The training task of DL model is generally implemented at the centralized cloud. However, due to the geo-distributed data sources and the transmission of large amounts of raw data to the centralized cloud, the transmission latency becomes a bottleneck of the COVID-19 diagnosis model training. In this paper, we propose a Distributed COVID-19 detection model training method on CXR images with edge-cloud collaboration, named DisCOV. Specifically, to improve the training efficiency and guarantee the model accuracy, a distributed lightweight model-based training algorithm is designed with the cooperation of edge computing and cloud computing. In addition, a resource allocation algorithm is developed during the training to jointly minimize the time cost and energy consumption. Extensive experiments based on real-world CXR image datasets demonstrate that DisCOV is better performed and more promising than the existing baselines.

Index Terms—COVID-19, deep learning, CXR image classification, edge computing, edge-cloud collaboration

1 Introduction

An ongoing outbreak of Coronavirus Disease 2019 (COVID-19) is spreading rapidly with over 119 million confirmed cases and 2 million deaths worldwide reported by World Health Organization (WHO) as of 15 March 2021. WHO has declared that the COVID-19 is regarded as a public health emergency of international concern (PHEIC) [1], [2]. The COVID-19, caused by Severe Acute Respiratory Syndrome Coronavirus 2 (SARSCoV-2), has some typical symptoms including fever, dry cough, fatigue, pain, and soreness, which is confirmed with the human-to-human spread and extremely infectious [3]. To slow down the spread of COVID-19 and control the epidemic, it is necessary to conduct an early detection of the suspected patients accurately and efficiently. According to clinical experience, the Reverse Transcription-Polymerase Chain Reaction (RT-PCR) is currently utilized to detect viral nucleic acid for screening the suspected cases of COVID-19 [4].

Although RT-PCR is the gold standard for the clinical diagnosis of COVID-19, the lack of convenient equipment and strict testing environment make it challenging for the RT-PCR-based COVID-19 diagnosis to meet the high accuracy and real-time requirements [5]. In contrast, chest X-ray (CXR), as an effective complement to RT-PCR testing, is capable of executing a rapid and convenient COVID-19 detection, which is superior to RT-PCR and other radiological imaging techniques like computed tomography (CT) [6]. But it is very difficult for the traditional way to detect COVID-19 efficiently because a specialist is always demanded to inspect CXR images manually in order to offer professional insights. Fortunately, with the advancements of deep learning (DL) technology in medical image processing, the utilization of DL in CXR images to achieve rapid, efficient, and accurate COVID-19 detection is actively explored [7], [8].

Generally, considering that the training of the DL model for COVID-19 diagnosis is resource-intensive and time-consuming, cloud computing (CC) is a proper paradigm to implement the model training tasks, where all the raw CXR data are transmitted to the resource-rich cloud data centers. However, uploading large amounts of data to the centralized cloud through the backbone link may cause significant bandwidth congestion, which increases the unexpected transmission

- *Xiaolong Xu is with the School of Computer and Software, Nanjing University of Information Science and Technology, Nanjing 210044, China, and with the Engineering Research Center of Digital Forensics, Ministry of Education, Nanjing University of Information Science and Technology, Nanjing 211544, China, and also with the Provincial Key Laboratory for Computer Information Processing Technology, Soochow University, Suzhou, Jiangsu 215006, China. E-mail: xlxu@ieee.org.*
- *Hao Tian is with the School of Computer and Software, Nanjing University of Information Science and Technology, Nanjing 210044, China. E-mail: withhaotian@gmail.com.*
- *Xuyun Zhang is with the Department of Computing, Macquarie University, Sydney, NSW 2109, Australia. E-mail: xuyun.zhang@mq.edu.au.*
- *Lianyong Qi is with the School of Information Science and Engineering, Qufu Normal University, Rizhao 276826, China, and also with the State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing 210023, China. E-mail: lianyongqi@gmail.com.*
- *Qiang He is with the School of Software and Electrical Engineering, Swinburne University of Technology, Melbourne, VIC 3122, Australia. E-mail: qhe@swin.edu.au.*
- *Wanchun Dou is with the State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing 210023, China. E-mail: douwc@nju.edu.cn.*

Manuscript received 30 Mar. 2021; revised 4 Dec. 2021; accepted 8 Jan. 2022. Date of publication 13 Jan. 2022; date of current version 15 June 2022.
This work was supported by the National Key R&D Program of China under Grant 2019YFE0190500. Dr. Xuyun Zhang is the recipient of an ARC DECRA Project No. DE210101458 funded by the Australian Government.
(Corresponding author: Lianyong Qi.)
Digital Object Identifier no. 10.1109/TSC.2022.3142265

latency. Moreover, considering the increasing growth of medical data generated at the network edge, offloading enormous original data to the remote cloud servers for processing may not be feasible because the required network resources and incurred delay when the data size is huge and geographically distributed. Currently, edge computing (EC) is emerging as a distributed computing paradigm working well with CC [9], [10]. By sinking the computing and storage resources from the centralized cloud to the edge of networks near the end devices, EC enables data processing locally at edge servers (ESs) deployed at base stations (BSs) rather than uploading the raw data to the cloud for execution [11]. Thereby, the pressure of the core network is alleviated.

Furthermore, to unleash the potential of EC and CC for training a high-performance COVID-19 detection model, it is naturally required to promote the cooperation of EC and CC, called edge-cloud collaboration [12], [13]. In edge-cloud collaboration empowered DL training, the CXR data from end devices are distributed in the ESs to train DL models, and model aggregation is conducted in the centralized cloud [14]. Therefore, the time efficiency of the training is improved. However, since the training process of the DL model is resource-intensive, as well as the model parameters and required computational resources are generally large, ESs with limited capacities on resources (e.g., processor, memory, and bandwidth) hardly undertake multiple training tasks, which leads to the increase of unexpected latency and power consumption [15]. In addition, with the growing number of training tasks requested from the end devices, each ES may perform one or more training tasks in parallel. It is highly probable that the inappropriate resource allocation strategies result in longer training delay and higher energy of some tasks, which not only increases the cost of the model learning, but also degrades the training performance.

Inspired by these observations, in this paper, we propose a Distributed COVID-19 diagnosis model training method on CXR images, named DisCOV. Specifically, taking full advantages of the advanced structure of edge-cloud collaboration, we formulate a hierarchical framework of COVID-19 detection model training. Then, to improve the training efficiency and guarantee the model accuracy, a lightweight model-based distributed training algorithm is designed for the COVID-19 diagnosis. Furthermore, to achieve the tradeoff between the training delay and the energy cost, a deep reinforcement learning (DRL)-based resource allocation algorithm is developed to realize the optimal strategies of dynamic computing and communication resource dispatching.

It is worth noting that our proposed distributed training scheme based on edge-cloud collaboration differs from the centralized training (i.e., training on cloud servers) and traditional federated learning (i.e., on-device training). First of all, the centralized training is a classical training scheme where all the raw data are transmitted from end devices to cloud servers to learn a machine learning model. However, with the tremendous growth of data generated at the network edge, the centralized training becomes impractical for real-time service demands because of the unpredictable transmission latency. Besides, the traditional federated learning is initially introduced to perform on-device training to preserve the data privacy without original data sharing (note that the focus of this paper is orthogonal to the data privacy issues which have been investigated in many other works [16]). Considering the limited computation and storage capabilities of CXR devices, it is challenging to perform training tasks locally to meet the requirements from fast-response diagnosis services. Therefore, an edge-cloud collaboration oriented distributed training architecture is proposed in this paper, where the original data are transmitted to ESs in proximity to the end devices to perform model training tasks rather than processing locally or at cloud servers.

The main contributions of this paper are summarized as

- Formulate a novel hierarchical architecture of COVID-19 detection model training empowered by edge-cloud collaboration.
- Propose a lightweight model-based distributed COVID-19 detection model training algorithm (LDT) to improve the training efficiency and guarantee the model accuracy.
- Design a DRL-based resource allocation algorithm (DRA) to dynamically dispatch the computing and communication resources to minimize the training latency and power consumption for the training tasks.
- Conduct extensive experiments based on the real-world COVID-19 CXR image datasets, called COVIDx [17], to evaluate the performance of the LDT and DRA.

The reminder of this paper is presented as follows. Section 2 illustrates the related work. In Section 3, we propose the system model and formulate the optimization problem. Section 4 presents the design of DisCOV. Section 5 conducts the evaluation performance based on extensive experiments in the real-world datasets. In Section 6, we conclude the paper.

2 Related Work

In this section, the existing researches related to our works are reviewed from the aspect of DL for COVID-19 diagnosis and EC.

With the rapid spread of COVID-19 among most countries over the world, the society, economy, and healthcare system are affected extremely. DL becomes a promising technology to advance the efficiency of medical imaging analysis. To this end, the COVID-19 diagnosis with DL technology attracts considerable attention. Roy *et al.* [18] proposed a DL network based on the spatial transformer networks to predict the severity of COVID-19 according to the lung ultrasonography images and identify the location of the lesion in a weakly-supervised manner. Compared with the traditional convolutional neural network, the 3D convolutional neural network enables the greater capabilities to capture spatial and temporal features. Wang *et al.* [2] developed a weakly-supervised DL model based on the 3D convolutional neural network via CT images for the COVID-19 detection and the location of the lesion. At the same time, it is also an effective way to analyze the CXR images to executing the COVID-19 image classification. With the aid of CXR images, Oh *et al.* [8] presented a DL-based approach with a patch-based convolutional neural network taking into account the limited training data sets of disease, which could train the proposed network by the finite trainable parameters. To early assess COVID-19, Ahmed *et al.* [19] proposed an Internet of Things (IoT) based DL framework, utilizing a region proposal network to execute the COVID-19 detection over CXR images. The proposed framework relieved the pressure of the medical experts or radiologists greatly and helped to control the outbreak of COVID-19. Minaee *et al.* [20] gave a COVID-19 detection framework on CXR images by utilizing the fine-tuning technique to four classical DL models, e.g., SqueezeNet, which conducts a binary classification evaluation with four models on about 5000 CXR images. To help experts with initial screening, Ozturk *et al.* [21] developed a fully automated end-to-end COVID-19 detection model, called DarkCovidNet, to diagnose positive cases for binary classification and multi-class classification. However, these centralized model training methods endure an unexpected degradation of training performance, when the massive data are transmitted to the cloud through backbone link inducing the network congestion.

EC is an emerging computing paradigm by pushing the computation, storage, and communication resources to the network edge near to the end customers. Therefore, the network latency is reduced greatly with the increasing mobile devices, as well as the data security and privacy are guaranteed due to geo-distributed data processing instead of handling all data in a centralized cloud. To cater to the demand of several delay-sensitive services, Alameddine *et al.* [22] formulated the task offloading and scheduling problems as a joint optimization problem. Furthermore, a new thoughtful decomposition scheme was proposed benefiting by the logic-based benders decomposition technique. In practice, the coordination of the communication, computing, and storage resources, along with the network control, is essential for the implementation of the efficient EC. Ndikumana *et al.* [23] took the aforementioned coordination into consideration in the EC environment and presented a proximal upper bound problem based on the original problem solved by the block successive upper bound minimization approach. Considering the decentralized data distribution, the data privacy issues from the centralized cloud were mitigated. Ma *et al.* [24] developed two privacy preserving reputation management methods aiming to guarantee the privacy and tackle malicious users. At the same time, the blockchain can also protect the data security with its distributed structure. Li *et al.* [25] presented a blockchain and certificateless cryptography-based method for the decentralized data storage in the EC-enabled IoT. Xu *et al.* [26] proposed the big data sharing framework with the blockchain technique for multiple applications at the edge, which aimed to construct a trust EC environment.

However, to the best of our knowledge, only few works in the literature accounted for the COVID-19 detection model learning in a distributed manner by taking edge-cloud collaboration into consideration. To this end, we propose a distributed COVID-19 diagnosis model training method to meet the minimization of the training latency and energy consumption while guaranteeing the training efficiency and model accuracy.

3 System Model and Problem Formulation

In this section, the system architecture is introduced first. Then, we describe the system model of local data transmission and edge-cloud based learning. In the end, the weighted-sum optimization is formulated to minimize the delay and energy consumption in the training process.

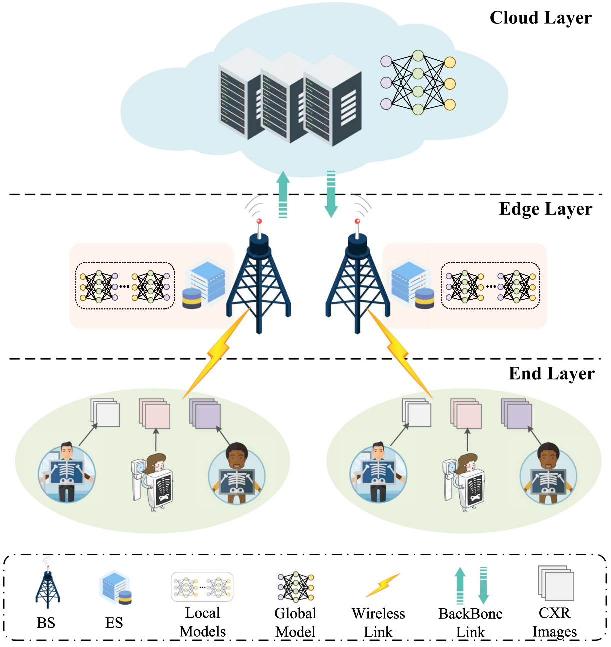


Fig. 1. The architecture of the COVID-19 detection model training system with edge-cloud collaboration.

3.1 Network Model

As illustrated in Fig. 1, we consider a three-layer hierarchical system of COVID-19 detection model training with edge-cloud collaboration, which consists of multiple CXR devices in the end layer, several ESs in the edge layer near to CXR devices, and a centralized cloud in the cloud layer. Assume that CXR devices are endowed with storage capabilities to store the collected CXR images from patients, and ESs are endowed with both computing and storage capabilities for model training and diagnosing the COVID-19 cases. In addition, the cloud data center has near-infinite resources to afford the computation and storage requirements. Let $\mathcal{M} = \{1, 2, \ldots, M\}$ denote the set of CXR devices, and let $\mathcal{N} = \{1, 2, \ldots, N\}$ denote the set of ESs. In addition, let $Q = \{Q_1, \ldots, Q_M\}$ denote the M-dimensional decision vector of CXR devices, where $Q_m \in \{0, 1\} (\forall m \in \mathcal{M})$. Then Q_m is given by

$$Q_m = \begin{cases} 1, & \text{if CXR device } m \text{ has a training request }, \\ 0, & \text{otherwise.} \end{cases} \tag{1}$$

Assume that the local models with training requests are distributed in ESs, which enables the model training to execute in proximity to the CXR devices. Meanwhile, the global model is deployed in the cloud as an aggregator to collect training parameters from each local model. After updating the global parameters, the cloud distributes the latest model parameters to each local model.

3.2 Local Data Uploading Model

Generally, the data-driven DL technology needs numerous training data to learn more features of data for constructing a high-performance DL model. We assume that each CXR device owns local datasets of CXR images. Let $D^l_m (m \in \mathcal{M})$ denote the local dataset. Before ESs perform the model training, each CXR device uploads its local dataset to the corresponding ES. Note that each CXR device can only upload the data to one ES, where each data sample is labeled. For example, $D^l_i = \{(x_i, y_i)\}$ is one of a data sample in local datasets, where x_i is the input data and y_i is the desired output of input data (i.e., the label of image). CXR devices are capable to connect to the ESs via wireless channels and transmit data to the ESs without interference based on the Orthogonal Frequency Division Multiplexing (OFDM) access technology. For the available accessing ES n, the CXR device m uploads the local dataset over the wireless communication. Let $x_{m,n} \in \{0,1\}$ be the association value between CXR device m and ES n, which is given by

$$x_{m,n} = \begin{cases} 1, & \text{if CXR device } m \text{ accesses to ES } n, \\ 0, & \text{otherwise.} \end{cases} \tag{2}$$

Additionally, assume that each CXR device can only be served by one ES at the same time. Therefore, the uploading latency from CXR devices to ESs is given by

$$T^l_{upload} = \sum_{n=1}^{N} \sum_{m=1}^{M} x_{m,n} Q_m \frac{s_m}{B^l_m \log_2(1+r)}, \tag{3}$$

where s_m is the size of datasets in CXR device m, B^l_m is the bandwidth of CXR device m, and r is the signal-noise ratio value. Meanwhile, define p^l_m as the transmission power of CXR device m. Then, the energy cost of the transmission of datasets is calculated as

$$E^l_{upload} = p^l_m T^l_{upload}. \tag{4}$$

3.3 Distributed Edge Learning Model

After the transmission of datasets from CXR devices, each ES serves the collected data as its training datasets, which are represented as $D_{m,n} (\forall m \in \mathcal{M}, n \in \mathcal{N})$. For ES n, it performs the model training to learn the optimal parameters. Loss function is defined as the error of the model and the goal of the training process is to minimize the loss function by updating model parameters. According to [27], the loss function of ES n for CXR device m is given by

$$Loss_{m,n}(w) = \frac{1}{|D_{m,n}|} \sum_{k \in D_{m,n}} loss_k(w), \tag{5}$$

where $|D_{m,n}|$ is the size of the datasets, w is the model parameter, and $loss_k(w)$ is the loss function on each data sample $k \in D_{m,n}$. At global iteration $t+1$, the model on ES n utilizes the gradient descent method to update its local parameters $w_{m,n}$, which is given by

$$w_{m,n}(t+1) = w_{m,n}(t) - \alpha \nabla Loss_{m,n}\big(w_{m,n}(t)\big), \tag{6}$$

where α is the learning rate and $\nabla Loss_{m,n}\big(w_{m,n}(t)\big)$ denotes the gradient of $Loss_{m,n}\big(w_{m,n}(t)\big)$. In the training process, the computing resources are consumed to meet the computation requirements, which leads to the computation latency and energy overheads. We let L_n be the number of local iterations of ES n, ξ_m be the number of required CPU cycles for one data sample processing from CXR device m, and $f_{m,n}$ be the allocated computing resources of ES n for CXR device m. To obtain the execution time, the ESs with allocated computing resources (computation capacity of processing one CPU cycle per second) have to process each data sample that requires several CPUs cycles. Since the computation amount of DL model mainly depends on the number of Floating Point Operations (FLOPs) of each network layer, we use the number of FLOPs to quantify the computations in the DL network. Moreover, it is associated with the number of required CPU cycles. According to the work in [28], the number of FLOPs per CPU cycle denoted as Π can be calculated by

$$\Pi = \varpi \cdot \rho \cdot \chi, \tag{7}$$

where ϖ denotes the number of FLOPs in the instruction's semantic, ρ indicates the number of operations performed simultaneously in an instruction, and χ is the number of instructions performed simultaneously in one cycle. Note that the calculation of Eq. (7) is highly hardware-independent and complicated due to the hardware evolution. The value of Π used in this paper is set according to [28], which will be described in Section 5.1. Besides, the number of required FLOPs of network layers (e.g., convolutional layers and fully connected layers) is denoted as γ_m. Therefore, the required CPU cycles of training task from CXR device m is calculated as

$$\xi_m = \frac{\gamma_m}{\Pi}. \tag{8}$$

Based on [29] and [30], the computation latency of ES n for the training task from CXR device m in one global iteration is given by

$$T^{comp}_{m,n} = x_{m,n} Q_m L_n \big|D_{m,n}\big| \frac{\xi_m}{f_{m,n}}. \tag{9}$$

After ESs complete one iteration training process, the local parameters are delivered to the cloud for global parameter update. To this end, define c_m as the size of local model parameter from CXR device m and $b_{m,n}$ as the allocated bandwidth of ES n for the training task from CXR device m. The uploading latency is presented as

$$T^{upload}_{m,n} = x_{m,n} Q_m \frac{c_m}{b_{m,n} \log_2(1+r)}. \tag{10}$$

Considering the training tasks are conducted in each edge server concurrently, the total latency based on the computation and uploading delay is affected by the last finished training task. Therefore, the total latency at the edge is derived as

$$T^e_{total} = G \cdot \max_{m \in \mathcal{M}, n \in \mathcal{N}} \Big\{ T^{comp}_{m,n} + T^{upload}_{m,n} \Big\}, \tag{11}$$

where G is the number of global iterations. For instance, Fig. 2 elaborates the process of global aggregation in one global iteration. It is observed that the training task 3 endures the longest time consumption of model training in one global iteration. Although other three tasks complete its one global iteration

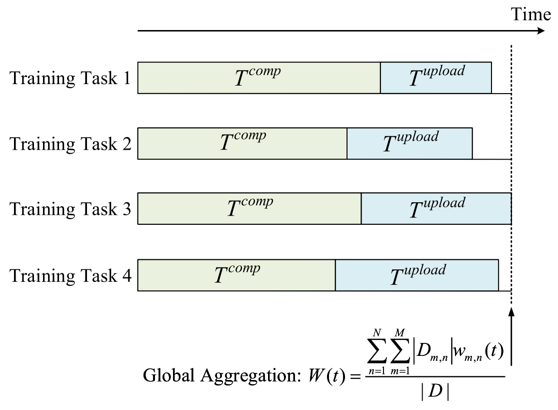

Time
Training Task 1
Training Task 2
Training Task 3
Training Task 4
T^{comp}
T^{upload}
Global Aggregation: $W(t)=\frac{\sum_{n=1}^{N}\sum_{m=1}^{M}|D_{m,n}|w_{m,n}(t)}{|D|}$

Fig. 2. An example of global aggregation with four training tasks in one global iteration.

before the task 3, the global aggregation of model parameters only happens when the last task is finished.

The energy cost is affected by the computation and uploading delay, which is given by

$$E^e_{total} = G \cdot \sum_{n=1}^{N}\sum_{m=1}^{M} p^c_n T^{comp}_{m,n} + p^u_n T^{upload}_{m,n}, \tag{12}$$

where p^c_n is the computation power and p^u_n is the parameter transmission power of ES n.

3.4 Cloud Aggregation Model

As the cloud data center receives the uploaded local parameters from each ES, the global aggregation is conducted to update the global model by the global parameters. Afterward, the remote cloud delivers the global parameters to all ESs which serve the global parameters as its local parameters at each local iteration. For any global iteration t, the global parameters are calculated by

$$W(t) = \frac{\sum_{n=1}^{N}\sum_{m=1}^{M}|D_{m,n}|w_{m,n}(t)}{|D|}, \tag{13}$$

where $D = \bigcup_{m\in\mathcal{M},n\in\mathcal{N}} D_{m,n}$ is all the data sample collected from the ESs. Considering the sufficient resources in the remote cloud, for ease of analysis, we neglect the computation delay of parameter aggregation and the transmission delay of parameter distribution to each local model.

3.5 Problem Formulation

As the system model presented above, the minimization of latency and energy cost is taken into consideration in this paper. The system latency considers the transmission delay both on the end layer and the edge layer, as well as the computation delay on the ESs, so the system latency is given by

$$T_{total} = T^l_{upload} + T^e_{total}. \tag{14}$$

Meanwhile, the system energy cost is affected by the local data and parameters on the edge, as well as the training process at the ESs. The system energy is presented as

$$E_{total} = E^l_{upload} + E^e_{total}. \tag{15}$$

Furthermore, the system-wide optimization problem with respect to the weighted-sum latency and energy cost problem. Let η_T and η_E be the weighted indicators, where $\eta_T + \eta_E = 1$ and $\eta_T, \eta_E \in [0,1]$. The optimization problem is presented as

$$\min_{(f,b)} \quad \eta_T T_{total} + \eta_E E_{total}. \tag{16}$$

$$\text{s.t.} \quad Q_m \in \{0,1\}, \forall m \in \mathcal{M}, \tag{17a}$$

$$x_{m,n} \in \{0,1\}, \forall m \in \mathcal{M}, n \in \mathcal{N}, \tag{17b}$$

$$\sum_{m=1}^{M} x_{m,n} = 1, \forall m \in \mathcal{M}, n \in \mathcal{N}, \tag{17c}$$

$$0 \le f_{m,n} \le F_n, \forall m \in \mathcal{M}, \tag{17d}$$

$$0 \le \sum_{m=1}^{M} x_{m,n} Q_m f_{m,n} \le F_n, \forall n \in \mathcal{N}, \tag{17e}$$

$$0 \le b_{m,n} \le B_n, \forall m \in \mathcal{M}, \tag{17f}$$

$$0 \le \sum_{m=1}^{M} x_{m,n} Q_m b_{m,n} \le B_n, \forall n \in \mathcal{N}, \tag{17g}$$

where (17a) denotes the training decision variable. (17b) and (17c) show the association value for CXR device m to choose only one ES n. The constraint (17d) and (17e) guarantee that the allocated computation resource does not exceed the limit of computation resources. Besides, the constraint (17f) and (17g) ensure the total bandwidth of ESs does not exceed the bandwidth capacity. Intuitively, the formulated problem is a mixed-integer nonlinear programming (MINLP) problem, which is generally NP-hard and requires exponential computational time [29].

4 DisCOV Design

In this section, we present the implementation of the DisCOV. Fig. 3 illustrates the architecture of DisCOV. First, a lightweight model-based distributed COVID-19 detection model training algorithm, named LDT, is proposed. Furthermore, we formulate the detailed Markov decision process (MDP) model of the distributed model training problem. Finally, a DRL-based resource allocation algorithm, named DRA, is presented to solve the formulated MDP.

4.1 Implementation of LDT

In the detection of COVID-19 based on CXR images, DL technology effectively grasps the lung CXR characteristics of the detector, and then assists in the diagnosis of COVID-19 cases. However, considering that the training of the DL model needs a large number of computing resources, transferring a large amount of raw CXR data to the cloud for training reduces the training efficiency. In addition, because the computing and storage capacity of CXR devices is limited, it is difficult to deal with the training task of the

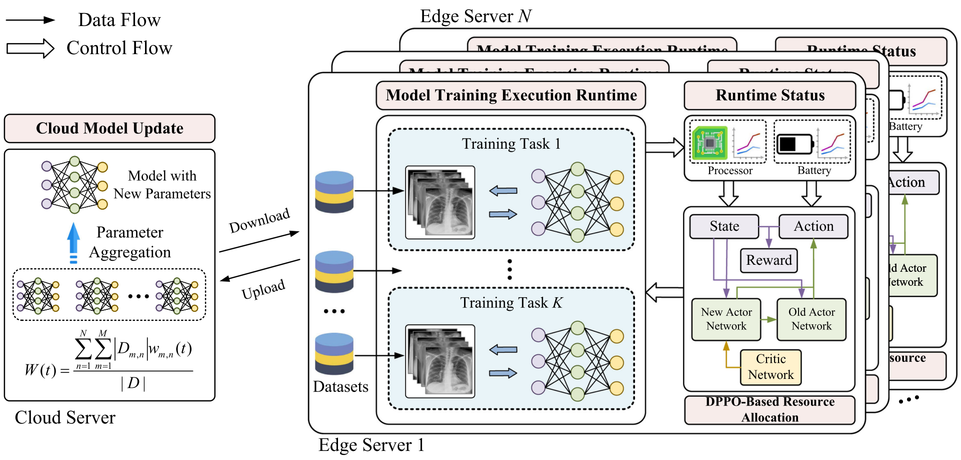


Fig. 3. The framework of DisCOV.

COVID-19 detection model. In general, the DL model consumes a lot of memory and computing resources. To this end, when the models are deployed on the ESs, the training delay is increased and more power is consumed. Therefore, to improve the training efficiency, the distributed model-based training algorithm needs to place the training task on the ES near the CXR device. In recent years, with the development of the lightweight model, the DL model with fewer parameters, less computation, and shorter reasoning time has been realized. Hence, it is feasible to deploy the lightweight model on ES to detect the COVID-19 cases compared with the traditional DL model which is computationally intensive and has large model parameters.

In this section, a distributed training algorithm is proposed to improve the training efficiency of the COVID-19 detection model in an edge-cloud collaboration environment. Specifically, inspired by a lightweight model, called MobileNet V2 [31], we develop the LDT algorithm based on the MobileNet V2. Such lightweight model proposes a novel layer structure, named inverted residual with linear bottleneck. The inverted residual with linear bottleneck allows the input to be extended in high dimensions, followed by the data filter based on a lightweight deep convolution operation. Then, the feature is projected in low dimensions using a linear bottleneck. Such a process ensures that the reasoning process does not need to achieve a large tensor calculation, and effectively reduces the physical device access time to memory. In addition, replacing conventional convolution with depthwise separable convolutions makes the computation more efficient and reduces the number of parameters and computation. Therefore, in this paper, a distributed lightweight model training algorithm is implemented to train the COVID-19 detection model.

Algorithm 1 shows the details of LDT algorithm. Before starting the model training, system variables such as global and local parameters are initialized (lines 1 to 2). For each global iteration training, the CXR device selects an ES to perform the training task, and each training task is executed in parallel. In the local training process, the training task on ES calculates the loss function according to each training data sample, and then updates the local parameters (lines 5 to 9). After the local iteration, local parameters for each training task are transferred from the ES to the cloud and wait for global aggregation (lines 11). However, the global parameter update cannot take place until the local iteration of the last training task is completed and transferred to the cloud. After all the training tasks finished, the cloud performs the global aggregation (lines 15). Overall, the last task to complete the training influences the performance of the training.

Algorithm 1. LDT

```
    Input: $\mathcal{M}, \mathcal{N}, G, L_n, D_{m,n}, W, w_{m,n}, Loss_{m,n}(w)$;
    Output: Global parameter $W$;
    // Initialization
 1: Initialize $G$ and $L_n$;
 2: Initialize $W, w_{m,n}$, and $Loss_{m,n}(w)$;
 3: for iteration = 1 to $G$ do
 4:     for each training task in parallel do
          // Edge training
 5:       for $l = 1$ to $L_n$ do
 6:           for $k \in D_{m,n}$ do
 7:               ES $n$ chooses one local sample $k \in D_{m,n}$ from
                  CXR device $m$;
 8:               Calculate the loss function $loss_k(w)$ of one
                  sample $k$;
 9:           end
10:           Calculate the loss function by (5);
11:           Update the local parameters $w_{m,n}$ by (6);
12:       end
13:       Transmit the local parameters $w_{m,n}$ from ES $n$ to the
          cloud;
14:     end
        // Cloud aggregation
15:     Update the glocal paramaters of the aggregation by
        (13);
16: end
17: return $W$;
```

4.2 MDP Model Formulation

To deal with the model learning problem under a dynamic environment, it is attempted to utilize DRL-based approach to obtain optimal resource allocation decisions to minimize the overall delay and power consumption. First, to implement the DRL method, the problem is formulated as an MDP. In this paper, the MDP is represented as a three-tuple

(S, A, R), where S is the set of environment states, A is the set of actions, and R is the set of reward functions. Moreover, the term $R(s(t), a(t))$ is defined as the reward function when the state is $s \in S$ and action $a \in A$. Specifically, three main elements are given as follows.

State. The state space reflects the observation from the constructed multi-ES model learning environment, including the training data size from CXR devices, the required computing resources for each training task, the size of model parameters, and the available resources of ESs. The state $s(t)$ is obtained as

$$s(t) = \{C(t), F(t), K(t), R(t), B(t)\}, \tag{18}$$

where $C(t) = [C_1(t), \ldots, C_M(t)]$ represents the size of training data from CXR devices, $F(t) = [F_1(t), \ldots, F_M(t)]$ represents the required computations for training tasks, $K(t) = [K_1(t), \ldots, K_M(t)]$ represents the size of model parameters for training task from CXR devices, $R(t) = [R_1(t), \ldots, R_N(t)]$ represents the available computing resources of ESs, and $B(t) = [B_1(t), \ldots, B_N(t)]$ represents the available bandwidth resources of ESs.

Action. Based on the observed states, the DRL agent decides the allocation of computing and bandwidth resources for training tasks from CXR devices. The action $a(t)$ is defined as

$$a(t) = \{f(t), b(t)\}, \tag{19}$$

where $f(t) = [f_{11}(t), \ldots, f_{1N}(t), \ldots, f_{MN}(t)]$ denotes the computation resources that ES allocates to training task from CXR device, $b(t) = [b_{11}(t), \ldots, b_{1N}(t), \ldots, b_{MN}(t)]$ denotes the bandwidth resources that ES allocates to training task from CXR device.

Reward. According to the state and action, the agent calculates the reward values based on the reward function $R(s(t), a(t))$. Specifically, the reward function corresponds with the Eq. (16). Since the objective of the optimization problem is to minimize the delay and energy consumption jointly, the reward function is presented as

$$R(s(t), a(t)) = -(\eta_T T_{total} + \eta_E E_{total}). \tag{20}$$

4.3 DRA Algorithm

To solve the above MDP problem, this section designed a resource allocation algorithm inspired by a DRL algorithm, called distributed proximal policy optimization (DPPO) [32]. Specifically, the DPPO is a parallel optimization algorithm based on the traditional proximal policy optimization (PPO) [33]. PPO is a model-free, actor-critic, and strategy gradient-based deep reinforcement learning algorithm. To effectively reduce the large policy update and the complexity of computation, the PPO uses the rate of change to express the difference between the new policy and the old policy. Moreover, the change rate is limited to a certain range, which ensures the updating range of the policy is small. Local optimization of the new strategy parameter π_θ based on the first-order approximation of the old strategy. The ratio between the old and new policy is defined as

$$\varrho(\theta) = \frac{\pi_\theta(a(t) \mid s(t))}{\pi_{\theta'}(a(t) \mid s(t))}, \tag{21}$$

where $\pi_\theta(a(t)|s(t))$ and $\pi_{\theta'}(a(t)|s(t))$ are the new policy and the old policy respectively. If $\varrho(\theta) > 1$, then the new policy is more likely to happen than the old policy, otherwise, the old policy is more likely to happen. However, in the absence of constraints, there may be a problem of over-updating of policies. Therefore, the PPO optimizes the objective function utilizing a clipped surrogate objective, in which the objective function is defined as

$$L_C(\theta) = E[\min(\varrho(\theta)A_{\theta'}, clip(\varrho(\theta), 1-\delta, 1+\delta)A_{\theta'}))], \tag{22}$$

where $A_{\theta'}$ is expressed as the advantage function, $clip(\varrho(\theta), 1-\delta, 1+\delta)$ represents the pruning operation. When $\varrho(\theta)$ is less than $1-\delta$, return $1-\delta$. If $\varrho(\theta)$ is greater than $1+\delta$, return $1+\delta$. If $1-\delta < \varrho(\theta) < 1+\delta$, $clip(\varrho(\theta), 1-\delta, 1+\delta)$ returns $\varrho(\theta)$. Through such clipping operation, the gap between the policy parameters θ and θ' is avoided, to prevent the policy from falling into the local optimal solution.

Algorithm 2. DRA

```
    Input: $C(0)$, $F(0)$, $K(0)$, $R(0)$, $B(0)$;
    Output: Resource allocation policy $f, b$;
    // Initialization
1:  Initialize the parameter $\theta$ of policy $\pi_\theta(a(t)|s(t))$ and $\theta'$ based on $\theta$;
2:  Initialize the number of agents;
3:  Initialize the number of inner iterations $V$;
    // Iteration
4:  for each agent in parellel do
5:    for each episode do
6:      Initialize the state $s$;
7:      for $i$=1 to $G$ do
8:        Collect the $s(t)$ ;
9:        Select the $a(t)$ based on the actor network;
10:       Calculate the $R(s(t)), a(t))$ and obtain the $s(t+1)$ based on $a(t)$;
11:       Update the loss value of critic network;
12:       for $v$=1 to $V$ do
13:         Calculate the ratio by (21) based on the new and old actor network;
14:         Update the actor network by the loss function (22);
15:       end
16:       Update the parameter of old policy by $\theta' \leftarrow \theta$;
17:     end
18:   end
19: end
20: return $f, b$;
```

To improve the sampling efficiency, the PPO uses the importance sampling technique to select the sample expectation from the old policy $\pi_{\theta'}(a(t)|s(t))$, and optimizes the new policy $\pi_\theta(a(t)|s(t))$. However, as the new policy changes, the differences between the two policies will become larger and larger, increasing the estimated variance. Therefore, the new policy needs to be updated periodically through the old policy. In addition, to ensure that the ratio is between $[1-\delta, 1+\delta]$, it is necessary to ensure the approximation of the state transition function between the two policies. DPPO is an improved version to accelerate the learning speed of the PPO. By interacting with the environment independently, multiple agents eliminate the correlation of resource allocation policy

TABLE 1
Main Parameters

Parameter	Value
Number of ESs	[5, 20]
Number of CXR devices	[5, 50]
Computing capacity of ESs	5 GHz [34]
Bandwidth of ESs	20 MHz [35]
Transmission power	23 dBm [36]
Computing power of ESs	0.1 W [37]
Number of FLOPs/cycle	16 [28]
Training period	3000 seconds
Learning rate	1e-2
Batch size	128
Episode	1e3

brought by a single agent, and accelerate the convergence speed of learning. A global PPO policy is generated when the allocation policy is generated, and multiple agents are used to generate the resource allocation policy in parallel, and the reward value of the system is calculated.

Algorithm 2 elaborates the process of DRA algorithm. The actor network parameters for all agents are initialized (line 1) before the algorithm executes. In addition, set the number of agents, perform independent interaction between the multi-agent and the environment, and learn in parallel from each agent (lines 2 to 3). In each episode, the agent selects an action based on the states and actor network, and obtains a reward and the next state based on the selected action (lines 8 to 10). In addition, the loss value of the critic network is calculated and the critic network is updated. On lines 12 to 15, the agent performs V times to calculate the ratio of the new policy to the old policy and updates the actor network according to the loss function. Finally, the agent updates the old policy based on the parameters of the new policy.

5 Experimental Performance

In this section, extensive experiments are carried out to evaluate the performance of our proposed method. The experiments are conducted on Manjaro Linux 21.1.0 with Intel i5-10600KF CPU (4.1 GHz, 6 cores), 64 GB DRAM, and 2 NVIDIA RTX 3090 GPUs. The Python 3.9.1 is implemented as the simulation environment. Main parameters used in the experiments are presented in Table 1.

5.1 Experiment Settings

5.1.1 Datasets

In the experiments, an open-access COVID-19 CXR dataset, called COVIDx [17], is utilized. Specifically, COVIDx consists of almost 15,000 CXR images, including 617 COVID-19 positive cases, 6069 pneumonia cases, and 8851 normal cases. In addition, the COVIDx is split with 80% as the training set and 20% as the testing set. Considering the small number of COVID-19 positive images, data augment techniques (e.g., rotation and clip) to enlarge the COVID-19 positive dataset are adopted before model training, which increases the training set of COVID-19 positive cases to 1480. The total size of the training data used in the experiment is about 5.5 GB (over 15,000 CXR images). For each training task, it samples the same number of images (no overlap between the training data) by shuffling the training data randomly. For example, if there are 50 training tasks in the ESs, based on the distribution of the CXR dataset, $\sim$110 MB of data are allocated to each training task on average. Each CXR image is converted to 224$\times$224 pixels in the training phase. Besides, based on the 224$\times$224 pixels of images, the total number of model parameters of the utilized MobileNet V2 is about 3.4 MB.

5.1.2 Baselines

First, to demonstrates the effectiveness of LDT in CXR image classification, three baselines are selected to compare the LDT as follows.

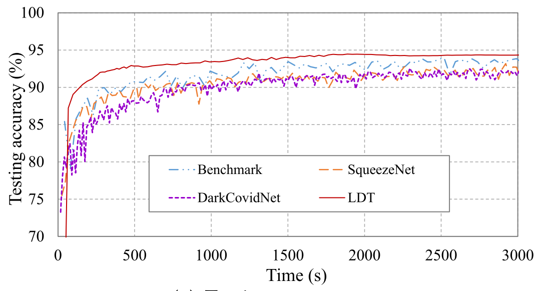
- *Benchmark*. In the Benchmark, the MobileNet V2 is adopted to perform the model training in a centralized manner. Specifically, the whole training data are collected in the cloud to train a COVID-19 detection model with MobileNet V2.
- *SqueezeNet* [20]. SqueezeNet is a lightweight and efficient CNN model with 50 times fewer parameters than AlexNet, but model accuracy is close to that of AlexNet. In the experiments, we conduct the training procedure with SqueezeNet in a centralized manner, i.e., the whole training data are collected in the cloud.
- *DarkCovidNet* [21]. DarkCovidNet is a CNN model for automatic diagnosis of COVID-19, which can be used for accurate diagnosis of binary classification (COVID-19 and normal) and multi-classification (COVID-19, pneumonia, and normal). In the experiments, the DarkCovidNet is conducted for the three-classification on the COVIDx and the training procedure is performed in a centralized manner, i.e., the whole training data are collected in the cloud.

In addition, the performance of DRA for resource allocation on the delay and energy cost is compared with the other two existing schemes as follows.

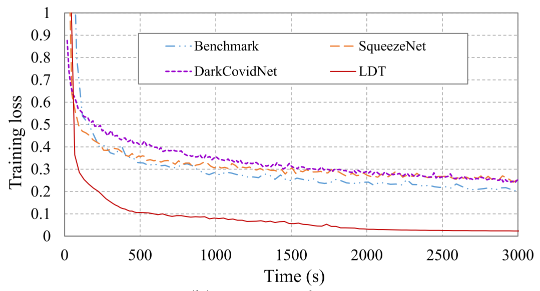
- *Deep Deterministic Policy Gradient (DDPG)* [38]. DDPG is a DRL algorithm with the combination of the policy-based policy gradient and the action value-based DQN. It updates the policy in a single step via the off-policy method, predicts the deterministic strategy, and then maximizes the reward value.
- *Actor-Critic (AC)* [39]. AC is a classical actor-critic based algorithm. The actor is responsible for the action selection based on the probability, and the critic evaluates the score of the selected action.

5.1.3 Implementation Details

In the experiments, we train the LDT, Benchmark, SqueezeNet, and DarkCovidNet on the same training and testing dataset with the stochastic gradient descent (SGD) optimizer with the momentum 0.5 and learning rate 1e-2. In addition, the DRA, DDPG, and AC are conducted in 1e3 episodes on the constructed environments with the defined states, actions, and reward function by (18), (19), and (20). We conduct the agent-based simulation method in the experiments. Furthermore, the numerical results are presented by utilizing average value of 10 times repetition based on the independent replication method [40]. In each



(a) Testing accuracy.



(b) Training loss.

Fig. 4. Training performance on different methods.

execution, the LDT and other three baselines are performed with the initialized model weights on the identical dataset and derive the trained weights, as well as the DRA, DDPG, and AC are performed in the initialized environments and agents to achieve the optimal resource allocation actions after 1000 iterations. Considering the hardware evolution, the calculation of Eq. (7) is sophisticated with much hardware-dependent details. In the simulations, we only consider the used Intel i5-10600KF. Based on Table 4 in [28], the peak performance of double-precision FLOPs, denoted as Π, for Intel i5-10600KF is set to 16 FLOPs per cycle (i.e., case of AVX+FMA(DP)) in the experiments. During every execution, the simulation parameters are fixed.

5.2 Performance Evaluation and Analysis

5.2.1 Performance on LDT

In this section, we focus on the training performance under the comparison between the LDT and baselines (i.e., Benchmark, SqueezeNet, and DarkCovidNet). Particularly, to implement the LDT, the ESs are employed to execute the training tasks in parallel, where each training task is with the initialized parameters of the model. ES performs the training tasks and all the ESs execute Algorithm 1 in a distributed manner to train a global model with parameter W. Besides, the total training set with three categories (COVID-19, Normal, and Pneumonia) is shuffled and averagely distributed to each ES according to the defined number of training tasks. Based on the dispense of the training set, each training task acquires the identical number of CXR images that do not overlap with other training tasks, which means the training set between each training task does not intersect. In addition, to evaluate the LDT, the following performance metrics are utilized as: 1) *training loss* (the value of loss function on training dataset); 2) *testing accuracy* (the prediction performance on testing dataset).

Fig. 4 depicts the training performance of different methods is evaluated. In Fig. 4, all the methods conduct the training phase and stop after 3000 seconds. First of all, the comparison of testing accuracy is shown in Fig. 4a. It is seen that the testing accuracy increases along with the increase of elapsed time. After 1210 seconds, Benchmark converges to 93% testing accuracy. However, we observe that the testing accuracy curve fluctuates slightly in the training process. On the contrary, the testing accuracy in LDT increases steadily and finally reaches 94%, where the SqueezeNet and DarkCovidNet only finally converge to 91% and 92%, respectively. In Fig. 4b, it is observed that the LDT outperforms the other three algorithms with lower training loss value. It is mainly because that the LDT is capable of achieving a lower training error than Benchmark, SqueezeNet, and DarkCovidNet with many participants to execute training task of locally trainng in a distributed manner. Moreover, compared with the Benchmark which is based on the MobileNet V2 in a centralized training manner, the LDT achieves a faster convergence in the training process from Figs. 4a to 4b. The main reason is that, based on the additional local training iterations in the ESs, the models at the edge are trained from their own training data.

As depicted in Table 2, the elapsed time and number of iterations with respect to different testing accuracy within the training period are illustrated based on Fig. 4. Each algorithm records the number of iterations and training time consumption when converges to 85%, 90%, 93%, and 94% testing accuracy, respectively. The number of training tasks in LDT is set to 20. Note that the notation "-" in Table 2 means that the algorithm cannot reach the expected testing accuracy value across the training process. In particular, when the testing accuracy is 93%, the LDT only achieves 30 iterations, whereas the other three algorithms gain at least 33 iterations. In addition, it is observed that LDT takes 53 iterative global training updates to finally reach 94%. Therefore, the total transmission volume of model parameters in LDT (~180MB) gains 64% communication reduction than that data transmission to the centralized servers (~280MB). In Table 2, the LDT gains a faster convergence speed than the other three methods with the same expected testing accuracy. Especially, LDT gains the 90% testing accuracy in 160.18s, which outperforms Benchmark, SqueezeNet, and DarkCovidNet by the acceleration of 2.8×, 3.54×, and 4.9×, respectively. Besides, to reach 93% accuracy, LDT consumes 679.5s with 30 iterations. It is noted that, through 1392.62 seconds of training, LDT eventually converges to 94% accuracy, however the other three algorithms cannot reach that level. Therefore, LDT cannot only realize a higher testing accuracy with same elapsed time but also converge faster during the training period.

Fig. 5 illustrates the training results of different numbers of local iterations with 3000s training period, where the number of training tasks is 20. It is seen that the higher performance is achieved when the number of local iterations increases. The more local iterations make the model train the better parameters. However, considering the data overfitting, we should set the number of iterations appropriately. For example, when the number of iterations is 5, the over-fitting happens. The reason is that the limited local iterations may affects the convergence performance. In Fig. 5a, the testing accuracy is higher than other cases as the number of local iterations reaches 20. As depicted in Fig. 5b,

TABLE 2
Comparison on Elapsed Time and Number of Iterations Under Different Expected Testing Accuracy

Expected testing accuracy	85%		90%		93%		94%	
	Elapsed time	Iterations	Elapsed time	Iterations	Elapsed time	Iterations	Elapsed time	Iterations
Benchmark	118.45s	3	447.44s	12	1210.84s	33	-	-
SqueezeNet	116.97s	5	566.49s	26	2921.37s	126	-	-
DarkCovidNet	200.08s	16	785.56s	60	-	-	-	-
LDT	**66.98s**	**2**	**160.18s**	**5**	**679.5s**	**30**	**1392.62s**	**53**

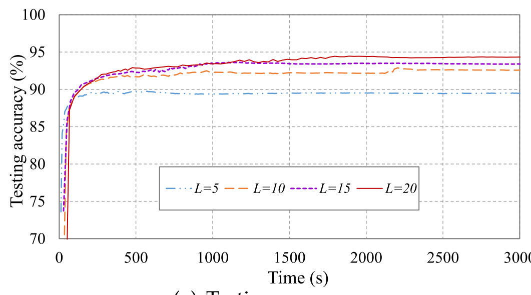
the training loss of four curves is very close. Besides, it can be observed that the testing accuracy finally converges to 94.4%, 93.4%, 92.6%, and 89.5% across the training process when number is 20, 15, 10, and 5, respectively. Therefore, the number of local iterations is an important factor in the performance of LDT. However, considering the additional computing resources consumed with local iterative training, it is worth noting that we should take this number reasonably.

For the dataset partition, since the amount of data for each training task is the same (i.e., with the balanced data partition), we conduct the impact of the different sizes of the dataset. As elaborated in Fig. 6, the training performance of different fractions of the dataset, i.e., 20%, 40%, 60%, 80%, and 100%, is evaluated. From Figs. 6a to 6b, it is observed that as the training loss decreases, the average testing accuracy increases along with the growth of the size of the dataset. The numerical results of average testing accuracy are shown in Fig. 6a. It is observed that the training tasks with abundant data samples make the increase of average testing accuracy. When the full data is obtained, the average testing accuracy in LDT is improved by 1.45%, 4.42%, and 7.26% compared with Benchmark, SqueezeNet, and DarkCovidNet, respectively. In Fig. 6b, we see that the LDT gains the best performance of average training loss on the different fractions of the dataset. Specifically, when the fraction of the dataset is 20%, the LDT outperforms Benchmark, SqueezeNet, and DarkCovidNet by the improvements of 3.5×, 4×, and 5×, respectively.

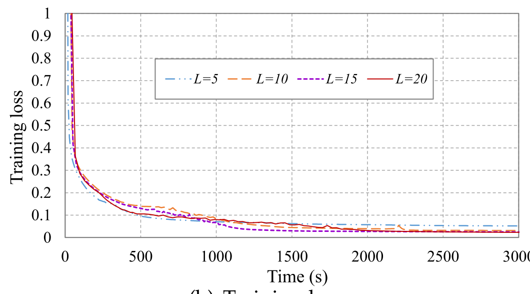
Fig. 7 shows the diagnosis results for different testing cases. The horizontal axis in the figure represents the actual labels on the testing set, and the vertical axis represents the results from the trained model. From Fig. 7a, it is seen that the LDT correctly predicted the results for all the 4 normal cases in the test set. Correct diagnosis of normal cases is essential to ensure that a normal case is not diagnosed with pneumonia or COVID-19. In Fig. 7b, we can see that the horizontal axis and vertical axis of the 4 pictures of pneumonia cases are consistent, which indicates that the predicted results are matched with the real values. In addition, Fig. 7c shows the predicted results of the COVID-19 cases on the testing set, and it can be observed that the predicted values of the 4 COVID-19 cases are the same as the real values. Overall, the model with high predicting accuracy is very critical for diagnosing the COVID-19 cases.

5.2.2 *Performance on DRA*

We then evaluate the performance of the DRA in this section. Specifically, the number of agents in the DRA equals the number of ESs, and we employ the agent to ES respectively

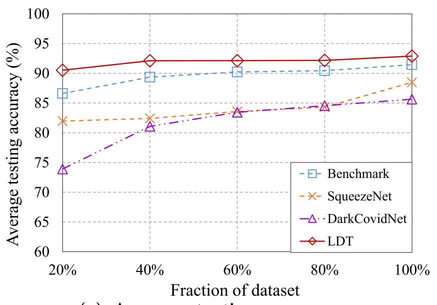


(a) Testing accuracy.

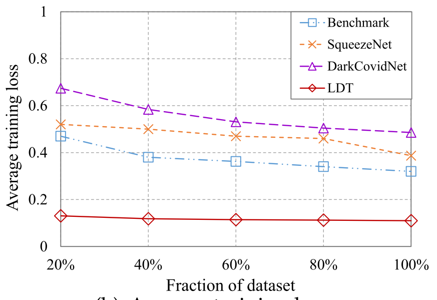


(b) Training loss.

Fig. 5. Training performance under different numbers of local iterations where training tasks = 20.



(a) Average testing accuracy.



(b) Average training loss.

Fig. 6. Training performance under different fractions of dataset.

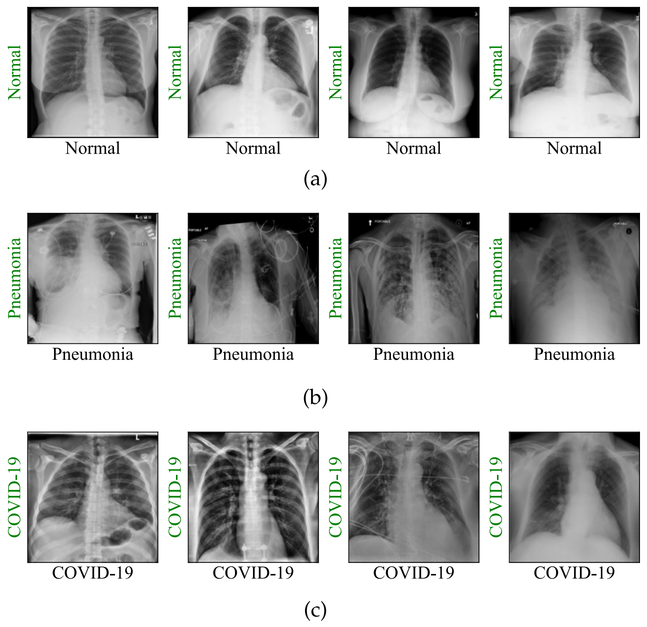


Fig. 7. The diagnosis results of different testing cases: (a) Normal cases; (b) Pneumonia cases; (c) COVID-19 cases.

to perform the training procedure in parallel. Such that, each agent executes a part number of learning episodes and interacts with the DRL environment independently (according to the collected observations $s(t)$ and reward value $R(s(t), a(t))$ to learn the optimal actions $a(t)$). When the sum of accumulated episodes for all agents reaches the defined episodes (i.e., 1e3), the DRL learning is finished. Besides, the main storage cost for agents is to store the states, actions, and rewards, and the computation overhead is to perform the backpropagation updating the actor network weights to minimize the loss function (22).

Fig. 8 presents the convergence performance of different DRL-based methods. It is seen that the DRA gains a faster convergence speed than DDPG and AC. The reason is that the DRA is capable of computing the new strategy in each episode to minimize the loss function and ensure that each latest computed strategy is similar to the original strategy. In comparison, the curve of the DDPG-based method is fluctuated in the first 200 episodes. In the DRA, the actor is responsible for the selection of action by the importance sampling on the generated distribution, as well as the critic estimates the performance of the selected action. Moreover,

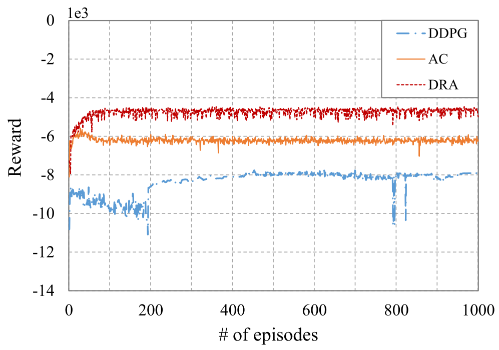


Fig. 8. The convergence performance of reward under different methods.

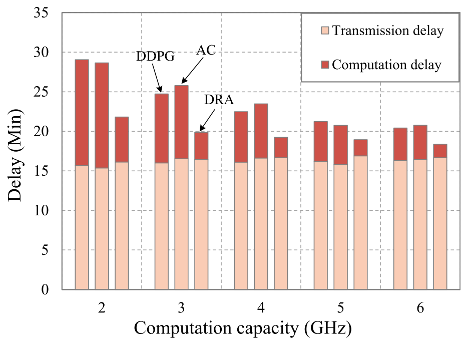


Fig. 9. The delay under different computation capacities.

the DRA outperforms DDPG and AC by the improvements of 70% and 32% in terms of the reward function value. The reward of the MDP model denotes the evaluation performance of the selected action, where the bigger reward value means the higher evaluation performance.

Fig. 9 shows the performance evaluation of delay under different computation capacities. The increased computation capacity affects the delay. It is observed that when the computation capacity gets higher, the delay can be taken less. When the computation capacity increases, the DRA always gains a lower delay compared with the DDPG and AC. In addition, the increase in computational latency is more pronounced than the communication latency. As the computation capacity is 2 GHz, the DRA achieves 21.8 minutes training cost, which outperforms the DDPG and AC by the improvements of 33% and 31%. In addition, when the computation capacity grows from 2 to 6 GHz, the latency reduction ratio of the DRA compared with DDPG and AC minimizes from 33% to 11% and 31% to 13%, respectively. This is because that more computing resources can handle more training tasks with less computation delay at the edge.

Fig. 10 illustrates the performance evaluation of energy cost under different computation capacities. We can see that when the computation capacity increases from 2 to 6 GHz, the DRA outperforms the DDPG and AC by the improvements of 2 to 1.9 times and 1.9 to 1.27 times, respectively. It can also be seen from the figure that the energy consumption of communication transmission is almost constant as the computational power increases, but there is a significant reduction in the energy consumption of computation. This is because more computational resources are used for model training, which can reduce the computational latency. The

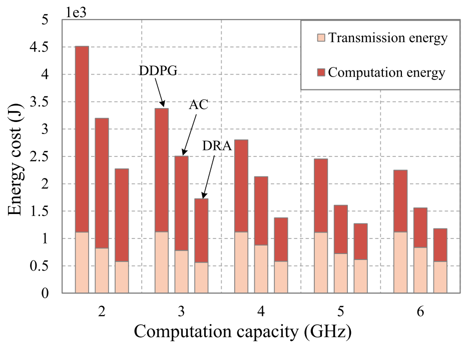


Fig. 10. The energy cost under different computation capacities.

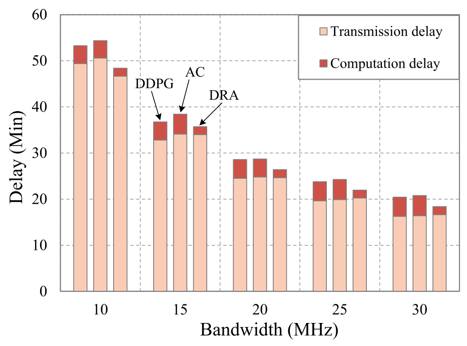


Fig. 11. The delay under different bandwidths.

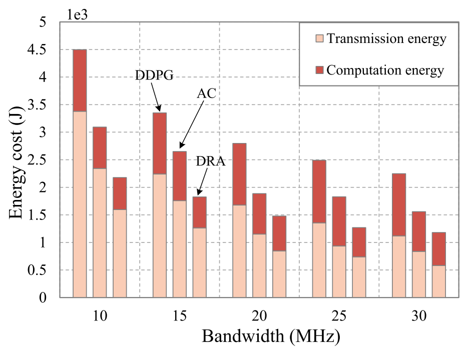


Fig. 12. The energy cost under different bandwidths.

energy cost decreases along with the increase of computation capacity since the minimization of computation delay. For example, as the computation capacity is 6 GHz, the energy cost of the DRA is 2248 J, which outperforms the baseline approaches. When the computation capacity increases from 2 to 6 GHz, the DRA reduces the total energy consumption by 1.9×. The reason is that when the more powerful computation capacity the ESs are, the lower computation latency is. In addition, the energy consumption of computation is reduced by 2.83× when the computing capacity goes from 2 to 6 GHz. Thus, the energy cost during the training process is minimized. Overall, the DRA always achieves the best performance of the delay and energy cost in diverse computation capacities.

Fig. 11 presents the performance evaluation of latency under different bandwidths. It is observed that the increased communication capacity affects the delay. As the bandwidth increases, the DRA always gains a lower delay compared with the DDPG and AC. Specifically, as the bandwidth is 10 MHz, the DRA outperforms the DDPG and AC by the improvements of 10% and 12% in the light of delay. It is seen that there is a significant reduction in transmission delay as the communication resources increase. This is because more bandwidth ensures fast data transmission. The DRA gains 48.39 minutes training time when the bandwidth is 10 MHz, where the DDPG and AC are 53.28 and 54.37, respectively. Also as the bandwidth gets to 30 MHz, the training delay of DRA achieves 18.37 minutes. Moreover, when the bandwidth grows from 10 to 30 MHz, the latency minimization ratio of the DRA compared with DDPG and AC minimizes from 10% to 11% and 12.4% to 13.1%, respectively. This is mainly because that the higher bandwidths make the less transmission time in the model parameters uploading from edge to cloud.

Fig. 12 elaborates the performance evaluation of energy cost under different bandwidths. It is seen that when the bandwidth gets higher, the energy cost can be taken less. As the bandwidth increases, the energy cost consumed to transmit data decreases linearly. When the bandwidth is 10 MHz, DRA consumes 2.07× and 1.42× less than DDPG and AC. As the bandwidth increases from 10 to 30 MHz, the DRA outperforms DDPG and AC by the improvements of 107% to 91% and 42% to 32%, respectively. The energy cost decreases along with the increase of bandwidth due to the minimization of execution latency. When the bandwidth is 30 MHz, the DRA consumes 1180 J energy during the training period. As the bandwidth increases from 10 to 30 MHz, the DRA reduces the total energy consumption and transmission energy consumption by 84% and 174%. This is mainly because that more communication resources lead to less transmission latency. Thereby, the power consumption during the training process is minimized. Overall, we can see that the DRA always gains the best performance of the training latency and energy cost under different bandwidths.

6 Conclusion

In this paper, we proposed DisCOV, a distributed COVID-19 diagnosis model training method over CXR images, taking into conseration of edge-cloud collaboration. Specifically, a novel architecture of COVID-19 detection model training was formulated first. Then, we proposed a optimization problem which was proven to be NP-hard to jointly minimize the training time and energy consumption. To overcome this issue, we developed DisCOV that includes two algorithms, i.e., LDT and DRA. To improve the training efficiency and guarantee the accuracy, we designed LDT, where the lightweight DL model is utilized to conduct the COVID-19 training tasks in a distributed manner with edge-cloud collaboration. Furthermore, DRA is given to dynamically allocate the computing and communication resources during the training phase. We finally conducted extensive experiments on an open-access COVID-19 CXR dataset to demonstrates the effectiveness of DisCOV. The experimental results shown that DisCOV achieves at least 2.8× training acceleration than baselines by reducing 64% of data transmission cost.

Acknowledgments

Xiaolong Xu and Hao Tian are co-first authors to this work.

References

[1] G. Wang *et al.*, “A noise-robust framework for automatic segmentation of COVID-19 pneumonia lesions from CT images,” *IEEE Trans. Med. Imag.*, vol. 39, no. 8, pp. 2653–2663, Aug. 2020.
[2] X. Wang *et al.*, “A weakly-supervised framework for COVID-19 classification and lesion localization from chest CT,” *IEEE Trans. Med. Imag.*, vol. 39, no. 8, pp. 2615–2625, Aug. 2020.
[3] E. Hosseini, K. Z. Ghafoor, A. S. Sadiq, M. Guizani, and A. Emrouznejad, “COVID-19 optimizer algorithm, modeling and controlling of coronavirus distribution process,” *IEEE J. Biomed. Health Inform.* , vol. 24, no. 10, pp. 2765–2775, Oct. 2020.
[4] X. Ouyang *et al.*, “Dual-sampling attention network for diagnosis of COVID-19 from community acquired pneumonia,” *IEEE Trans. Med. Imag.*, vol. 39, no. 8, pp. 2595–2605, Aug. 2020.

[5] D.-P. Fan *et al.*, "Inf-net: Automatic COVID-19 lung infection segmentation from CT images," *IEEE Trans. Med. Imag.*, vol. 39, no. 8, pp. 2626–2637, Aug. 2020.

[6] S. Tabik *et al.*, "COVIDGR dataset and COVID-sdnet methodology for predicting COVID-19 based on chest X-ray images," *IEEE J. Biomed. Health Inform.* , vol. 24, no. 12, pp. 3595–3605, Dec. 2020.

[7] F. Shi *et al.*, "Review of artificial intelligence techniques in imaging data acquisition, segmentation and diagnosis for COVID-19," *IEEE Rev. Biomed. Eng.*, vol. 14, pp. 4–15, 2021.

[8] Y. Oh, S. Park, and J. C. Ye, "Deep learning COVID-19 features on CXR using limited training data sets," *IEEE Trans. Med. Imag.*, vol. 39, no. 8, pp. 2688–2700, Aug. 2020.

[9] Y. Chen, Y. Zhang, Y. Wu, L. Qi, X. Chen, and X. Shen, "Joint task scheduling and energy management for heterogeneous mobile edge computing with hybrid energy supply," *IEEE Internet Things J.* , vol. 7, no. 9, pp. 8419–8429, Sep. 2020.

[10] L. Chen and J. Xu, "Seek common while shelving differences: Orchestrating deep neural networks for edge service provisioning," *IEEE J. Sel. Areas Commun.*, vol. 39, no. 1, pp. 251–264, Jan. 2020.

[11] T. Q. Dinh, B. Liang, T. Q. S. Quek, and H. Shin, "Online resource procurement and allocation in a hybrid edge-cloud computing system," *IEEE Trans. Wireless Commun.*, vol. 19, no. 3, pp. 2137–2149, Mar. 2020.

[12] M. Sun and Z. Zhou, "IoT services configuration in edge-cloud collaboration networks," in *Proc. IEEE Int. Conf. Web Services*, 2020, pp. 468–472.

[13] D. Wu, R. Bao, Z. Li, H. Wang, H. Zhang, and R. Wang, "Edge-cloud collaboration enabled video service enhancement: A hybrid human-artificial intelligence scheme," *IEEE Trans. Multimedia*, vol. 23, pp. 2208–2221, 2021.

[14] L. Wang, L. Jiao, T. He, J. Li, and H. Bal, "Service placement for collaborative edge applications," *IEEE/ACM Trans. Netw.*, vol. 29, no. 1, pp. 34–47, Feb. 2021.

[15] M. Chen *et al.*, "Joint data collection and resource allocation for distributed machine learning at the edge," *IEEE Trans. Mobile Comput.*, to be published, doi: 10.1109/TMC.2020.3045436.

[16] D. C. Nguyen, M. Ding, P. N. Pathirana, A. Seneviratne, J. Li, and H. Vincent Poor, "Federated learning for Internet of Things: A comprehensive survey," *IEEE Commun. Surv. Tuts.*, vol. 23, no. 3, pp. 1622–1658, Third quarter 2021.

[17] L. Wang, Z. Q. Lin, and A. Wong, "COVID-net: A tailored deep convolutional neural network design for detection of COVID-19 cases from chest x-ray images," *Sci. Rep.*, vol. 10, no. 1, pp. 1–12, 2020.

[18] S. Roy *et al.*, "Deep learning for classification and localization of COVID-19 markers in point-of-care lung ultrasound," *IEEE Trans. Med. Imag.*, vol. 39, no. 8, pp. 2676–2687, Aug. 2020.

[19] I. Ahmed, A. Ahmad, and G. Jeon, "An iot-based deep learning framework for early assessment of COVID-19," *IEEE Internet Things J.* , vol. 8, no. 21, pp. 15 855–15 862, Nov. 2021.

[20] S. Minaee, R. Kafieh, M. Sonka, S. Yazdani, and G. J. Soufi, "Deep-COVID: Predicting COVID-19 from chest X-ray images using deep transfer learning," *Med. Image Anal.*, vol. 65, p. 101794, 2020.

[21] T. Ozturk, M. Talo, E. A. Yildirim, U. B. Baloglu, O. Yildirim, and U. R. Acharya, "Automated detection of COVID-19 cases using deep neural networks with X-ray images," *Comput. Biol. Med.*, vol. 121, 2020, Art. no. 103792.

[22] H. A. Alameddine, S. Sharafeddine, S. Sebbah, S. Ayoubi, and C. Assi, "Dynamic task offloading and scheduling for low-latency IoT services in multi-access edge computing," *IEEE J. Sel. Areas Commun.*, vol. 37, no. 3, pp. 668–682, Mar. 2019.

[23] A. Ndikumana *et al.*, "Joint communication, computation, caching, and control in big data multi-access edge computing," *IEEE Trans. Mobile Comput.*, vol. 19, no. 6, pp. 1359–1374, Jun. 2020.

[24] L. Ma, X. Liu, Q. Pei, and Y. Xiang, "Privacy-preserving reputation management for edge computing enhanced mobile crowdsensing," *IEEE Trans. Serv. Comput.*, vol. 12, no. 5, pp. 786–799, Sep./Oct. 2019.

[25] R. Li, T. Song, B. Mei, H. Li, X. Cheng, and L. Sun, "Blockchain for large-scale Internet of Things data storage and protection," *IEEE Trans. Serv. Comput.*, vol. 12, no. 5, pp. 762–771, Sep./Oct. 2019.

[26] C. Xu *et al.*, "Making big data open in edges: A resource-efficient blockchain-based approach," *IEEE Trans. Parallel Distrib. Syst.*, vol. 30, no. 4, pp. 870–882, Apr. 2019.

[27] S. Wang *et al.*, "Adaptive federated learning in resource constrained edge computing systems," *IEEE J. Sel. Areas Commun.*, vol. 37, no. 6, pp. 1205–1221, Jun. 2019.

[28] R. Dolbeau, "Theoretical peak flops per instruction set: A tutorial," *The J. Supercomputing*, vol. 74, no. 3, pp. 1341–1377, 2018.

[29] B. Yang, X. Cao, J. Bassey, X. Li, and L. Qian, "Computation offloading in multi-access edge computing: A multi-task learning approach," *IEEE Trans. Mobile Comput.*, vol. 20, no. 9, pp. 2745–2762, Sep. 2021.

[30] C. Wang, F. R. Yu, C. Liang, Q. Chen, and L. Tang, "Joint computation offloading and interference management in wireless cellular networks with mobile edge computing," *IEEE Trans. Veh. Technol.*, vol. 66, no. 8, pp. 7432–7445, Aug. 2017.

[31] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "MobileNetV2: Inverted residuals and linear bottlenecks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 4510–4520.

[32] N. Heess *et al.*, "Emergence of locomotion behaviours in rich environments," 2017, *arXiv:1707.02286*.

[33] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017, *arXiv:1707.06347*.

[34] X. Zheng, M. Li, Y. Chen, J. Guo, M. Alam, and W. Hu, "Blockchain-based secure computation offloading in vehicular networks," *IEEE Trans. Intell. Transp. Syst.*, vol. 22, no. 7, pp. 4073–4087, Jul. 2021.

[35] A. Asheralieva and D. Niyato, "Learning-based mobile edge computing resource management to support public blockchain networks," *IEEE Trans. Mobile Comput.*, vol. 20, no. 3, pp. 1092–1109, Mar. 2021.

[36] Z. Zhou, J. Feng, Z. Chang, and X. Shen, "Energy-efficient edge computing service provisioning for vehicular networks: A consensus admm approach," *IEEE Trans. Veh. Technol*, vol. 68, no. 5, pp. 5087–5099, May 2019.

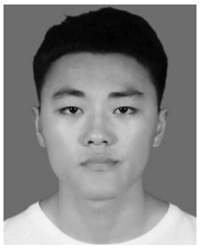
[37] Q. Li, S. Wang, A. Zhou, X. Ma, F. Yang and A. X. Liu, "QoS driven task offloading with statistical guarantee in mobile edge computing," *IEEE Trans. Mobile Comput.*, vol. 21, no. 1, pp. 278–290, Jan. 2022.

[38] T. P. Lillicrap *et al.*, "Continuous control with deep reinforcement learning," 2015, *arXiv:1509.02971*.

[39] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. Cambridge, MA, USA: MIT press, 2018.

[40] H. Kang *et al.*, "Diagnosis of coronavirus disease 2019 (COVID-19) with structured latent multi-view representation learning," *IEEE Trans. Med. Imag.*, vol. 39, no. 8, pp. 2606–2614, Aug. 2020.

Xiaolong Xu received the PhD degree in computer science and technology from Nanjing University, China, in 2016. He was a research scholar with Michigan State University, USA, from April 2017 to May 2018. He is currently a full professor with the School of Computer and Software, Nanjing University of Information Science and Technology. He has published more than 100 peer-review articles in international journals and conferences, including *IEEE Transactions on Parallel and Distributed Systems*, *IEEE Transactions on Services Computing*, *IEEE Transactions on Intelligent Transportation Systems*, *IEEE Transactions on Industrial Informatics*, *ACM Transactions on Sensor Networks*, *ACM Transactions on Multimedia Computing, Communications, and Applications*, *ACM Transactions on Intelligent Systems and Technology*, IEEE ICWS, ICSOC, etc. He was selected as the Highly Cited Researcher of Clarivate 2021. He received the best paper awards from IEEE CBD 2016, IEEE CPSCom 2020, SPDE 2020, IEEE CyberSciTech2021, CENET 2021, and EAI Cloudcomp 2021, and the Distinguish Paper Award from IEEE Smart City 2021. His research interests include edge computing, the Internet of Things (IoT), cloud computing, and Big Data.



Hao Tian received the BS degree in computer science and technology engineering from the Nanjing University of Information Science and Technology, in 2020, where he is currently working toward the master's degree in software engineering. His research interests include mobile edge computing, Big Data, Internet of Things, and machine learning.

Xuyun Zhang received the BS and ME degrees in computer science from Nanjing University, Nanjing, China, in 2011 and 2008, respectively, and the PhD degree from the University of Technology Sydney, Ultimo, NSW, Australia, in 2014, He worked as a postdoctoral fellow with the Machine Learning Research Group, NICTA (currently, Data61, CSIRO), Sydney, NSW, Australia. He is currently a senior lecturer with the Department of Computing, Macquarie University, Sydney. His primary research interests include Internet of Things and smart cities, Big Data, cloud computing, scalable machine learning and data mining, data privacy and security, and Web service technology.

Lianyong Qi received the PhD degree from the Department of Computer Science and Technology, Nanjing University, China, in 2011. He is currently a full professor with the School of Information Science and Engineering, Chinese Academy of Education Big Data, Qufu Normal University, China. He has already published more than 50 articles, including *IEEE Journal on Selected Areas in Communications*, *IEEE Transactions on Cloud Computing*, TBD, FGCS, *Journal of Computational Social Science*, CCPE, ICWS, and ICSOC. His research interests include services computing, Big Data, and the Internet of Things.

Qiang He (Member, IEEE) received the first PhD degree from the Swinburne University of Technology, Australia, in 2009, and the second PhD degree in computer science and engineering from the Huazhong University of Science and Technology, China, in 2010. He is currently a senior lecturer with Swinburne. His research interests include edge computing, cloud computing, software engineering, and service computing. He is a member of the IEEE. For more information, please visit https://sites.google.com/site/heqiang/

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