Distributed DNN Inference with Fine-grained Model Partitioning in Mobile Edge Computing Networks

Hui Li, *Student Member, IEEE,* Xiuhua Li, *Member, IEEE,* Qilin Fan, *Member, IEEE*, Qiang He, *Senior Member, IEEE,* Xiaofei Wang, *Senior Member, IEEE,* Victor C. M. Leung, *Life Fellow, IEEE*

Abstract—Model partitioning is a promising technique for improving the efficiency of distributed inference by executing partial deep neural network (DNN) models on edge servers (ESs) or Internet-of-Things (IoT) devices. However, due to heterogeneous resources of ESs and IoT devices in mobile edge computing (MEC) networks, it is non-trivial to guarantee the DNN inference speed to satisfy specific delay constraints. Meanwhile, many existing DNN models have a deep and complex architecture with numerous DNN blocks, which leads to a huge search space for fine-grained model partitioning. To address these challenges, we investigate distributed DNN inference with fine-grained model partitioning, with collaborations between ESs and IoT devices. We formulate the problem and propose a multi-task learning based asynchronous advantage actor-critic approach to find a competitive model partitioning policy that reduces DNN inference delay. Specifically, we combine the shared layers of actor-network and critic-network via soft parameter sharing, and expand the output layer into multiple branches to determine the model partitioning policy for each DNN block individually. Experiment results demonstrate that the proposed approach outperforms state-of-the-art approaches by reducing total inference delay, edge inference delay and local inference delay by an average of 4.76%, 10.04% and 8.03% in the considered MEC networks.

Index Terms—Mobile edge computing, distributed DNN inference, model partitioning, multi-task learning, asynchronous advantage actor-critic.

1 Introduction

- *This work is supported in part by National Key R & D Program of China (Grants No. 2022YFE0125400), National NSFC (Grants No. 62372072, 62102053, 62072060, 92067206 and 61972222), Chongqing Research Program of Basic Research and Frontier Technology (Grant No. cstc2022ycjhbgzxm0058), Key Research Program of Chongqing Science & Technology Commission (Grant No. cstc2021jscx-dxwtBX0019), Science and Technology Plan Project of Chongqing Economic and Information Commission (Grant No. 2211R49R03), Haihe Lab of ITAI (Grant No. 22HHXCJC00002), the Natural Science Foundation of Chongqing, China (Grant No. CSTB2022NSCQ-MSX1104), the General Program of Chongqing Science & Technology Commission (Grant No. CSTB2022TIAD-GPX0017, CSTB2022TIAD-STX0006), Key Laboratory of Big Data Intelligent Computing, Chongqing University of Posts and Telecommunications (Grant No. BDIC-2023-B-003), Regional Innovation Cooperation Project of Sichuan Province (Grant No. 2023YFQ0028), Regional Science and Technology Innovation Cooperation Project of Chengdu City (Grant No. 2023-YF11-00023-HZ), and Guangdong Pearl River Talent Recruitment Program (Grants No. 2019ZT08X603 and 2019JC01X235). (Corresponding author: Xiuhua Li).*
- *H. Li, X. Li, and Q. Fan are with the School of Big Data & Software Engineering, Chongqing University, Chongqing, China 400000 (e-mail: h.li@cqu.edu.cn, lixiuhua1988@gmail.com, fanqilin@cqu.edu.cn).*
- *Q. He is with the National Engineering Research Center for Big Data Technology and System, Services Computing Technology and System Lab, Cluster and Grid Computing Lab, School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, China, and the Department of Computing Technologies, Swinburne University of Technology, Melbourne, VIC 3122, Australia (e-mail: hqiang@hust.edu.cn).*
- *X. Wang is with the College of Intelligence and Computing, Tianjin University, China 300072 (e-mail:xiaofeiwang@tju.edu.cn).*
- *V. C. M. Leung is with the College of Computer Science & Software Engineering, Shenzhen University, Shenzhen 518060, China, and also with the Department of Electrical and Computer Engineering, The University of British Columbia, Vancouver, BC V6T1Z4, Canada (e-mail: vleung@ieee.org).*

With the rapid advancement of deep learning (DL) technology and Internet-of-Things (IoT) devices, applications in a wide range of industries are generating massive deep neural network (DNN) inference tasks, such as virtual/augmented reality, video streaming services, etc [1]–[5]. The traditional cloud-based architecture deploys resources on centralized cloud servers far away from IoT devices. It is hard for them to satisfy the real-time requirements when processing large-scale concurrent DNN tasks. Fortunately, the emergence of mobile edge computing (MEC) networks offers a possible solution to this problem. It is a novel technology that partially sinks computing resources to the network edges (e.g., base stations), where real-time distributed DNN inference can be facilitated [6]–[8]. The deployment of DNN inference tasks at the network edge can improve the quality of service/experience (QoS/QoE) for IoT devices and in the meantime, alleviate the pressure on the backbone network significantly [9]–[11].

Although MEC networks offer unique advantages in reducing inference delay, it suffer from potential performance bottlenecks when facilitating distributed DNN inference [12]–[14]. First, it is non-trivial to guarantee that all the DNN inference tasks can be executed on time under specific delay constraints when processing large-scale concurrent DNN inference tasks [15], [16]. In addition, it is a grand challenge to coordinate the resources of IoT devices and ESs in a distributed manner. They usually have constrained computing power and storage capacity, making it difficult to drive inferences based on the entire DNN model, especially large ones. The heterogeneous resource of ESs and IoT devices complicates the process of distributed DNN inference,

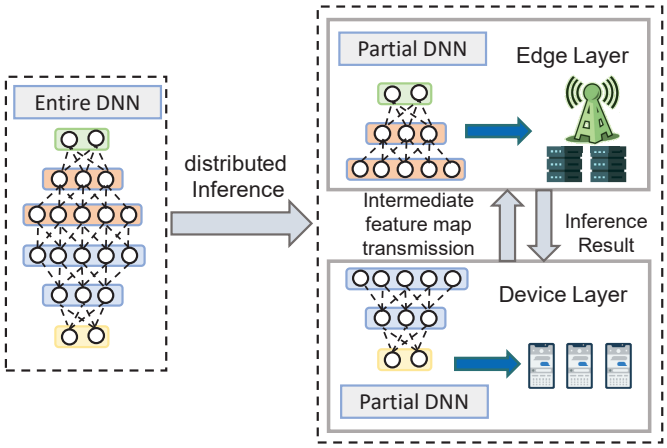


Fig. 1. Paradigm of model partitioning of distributed DNN inference in MEC networks.

and may lead to extra inference delay [17], [18].

Recently, model partitioning is widely acknowledged as a promising solution to the above challenges. With model partitioning, we can divide an entire DNN model into multiple blocks or fragments, and execute these blocks or fragments individually on different ESs or IoT devices for distributed inference, as shown in Fig. 1. This technique [19]–[21] is effective in resource-constrained MEC networks where ESs or IoT devices may not have sufficient resources to execute entire DNN model, especially when the sizes of new DNN models are increasing rapidly [22], [23]. However, conventional model partitioning approaches only utilize a single model partition point that divides the entire DNN model into cloud servers and IoT devices locally for distributed inference. Despite the potential for dynamic adjustment of the model partitioning point, its flexibility remains inherently limited. Fine-grained model partitioning [24]–[26] adopts multi-partition points and splits entire DNN model into multiple DNN fragments or blocks for distributed inference on different IoT devices and ESs. During distributed DNN inference, the intermediate feature map of previous DNN block is transmitted to the location of subsequent DNN block. This approach can effectively reduce DNN inference delay and energy consumption overheads [27], [28]. Combining MEC and fine-grained model partitioning offers new opportunities for improving the performance of distributed DNN inference.

To improve the performance of distributed DNN inference with fine-grained model partitioning in MEC networks, there is a fundamental challenge to overcome: **How to determine a fine-grained model partitioning policy for an efficient distributed DNN inference?**

Our goal is to search for an appropriate model partitioning policy that incurs the lowest inference delay. However, many existing DNN models (e.g., AlexNet and VGG) have a deep and complex architecture with numerous DNN blocks. We need to partition these DNN blocks for execution on different ESs to speed up the DNN inference process. There is a large search space M^N [24], [25](M, N is the number of DNN blocks and ESs, respectively) for finding the optimal model partitioning policy. Actually, the fine-grained DNN model partitioning is a combinatorial optimization problem, which is complicated by the potentially-exponential scale of DNN blocks. It is non-trivial to find the optimal solution within a polynomial time. This highlights the strong need for efficient approaches to tackle this issue.

Several pioneers attempted to facilitate distributed DNN inference with model partitioning based on deep reinforcement learning (DRL) [29], [30]. Dong *et al.* [29] proposed a DNN inference framework and developed a DRL-based algorithm to learn the model partitioning policy that minimizes the inference delay. Kim *et al.* [30] proposed a soft actor-critic based DNN inference strategy that considers the IoT devices' differentiated levels of heterogeneous resources. However, when these DRL approaches are applied to DNN model partitioning directly, the number of nodes in the output layer of DRL agent (i.e., the size of the action space) will grow exponentially [31], [32] with the number of DNN blocks, thereby resulting in high training time. Multi-task learning based DRL approach can significantly reduce action space and make decisions for multiple related tasks in parallel by sharing representation information [33]–[36]. Considering the partitioned DNN blocks are a series of DNN subtasks with shared characteristics, multi-task learning based DRL approach can separate the output layer of the action space to generate a fine-grained model partitioning policy for each DNN block. Recently, a number of studies have tried to use this approach to solve the corresponding optimization problems in MEC networks. Dong *et al.* [33] developed a joint slicing and network routing mechanism via multi-task learning based DRL algorithm to satisfy service requirements in 6G mobile networks. Chen *et al.* [34] proposed a multi-task learning based transfer DRL scheme to train the DRL agent that proficiently handles multiple tasks simultaneously. These studies have demonstrated that the multi-task learning based DRL approach has advantages over traditional DRL in reducing the action space of policy-making, thereby reducing the model training time.

In this paper, we are inspired to investigate the problem of distributed DNN inference with fine-grained model partitioning to minimize the inference delay in MEC networks. To reduce the action space of model partitioning systematically, this paper presents a novel multi-task learning based asynchronous advantage actor-critic (A3C) approach that facilitates distributed DNN inference with fine-grained model partitioning. It expands the conventional actor-critic network into a multi-task learning manner and integrates the hidden layer of actor-network and critic-network into a shared layer via soft parameter sharing. In this way, it allows agents to share experiences efficiently, which can significantly reduce the action space of policy-making, thereby reducing the model training time. The major contributions of this paper are summarized as follows:

- We present a fine-grained model partitioning mechanism that supports distributed DNN inference with the collaboration of ESs and IoT devices, for significantly reducing the DNN inference delay with specific delay constraints. We formulate the optimization problem as a markov decision process (MDP) with the objective to maximize the long-term discounted cumulative reward of distributed DNN inference.
- We propose a novel multi-task learning based A3C

approach to search for an appropriate fine-grained model partitioning policy. Specifically, we employ soft parameter sharing to integrate the shared layers of both the actor-network and critic-network, and expand the output layer into multiple branches to determine the fine-grained model partitioning policy for each individual DNN block. It can significantly reduce the action space of DRL agents, thereby reducing the training time of the proposed approach.

- We evaluate the performance of the proposed approach through extensive experiments conducted on widely-used datasets in MEC networks. The experimental results validate the effectiveness of the proposed approach, which achieves an average performance improvement of 4.76%, 10.04% and 8.03% in terms of minimizing the total inference delay, edge inference delay and local inference delay in the considered MEC networks, respectively.

This paper is organized as follows. The related work is presented in Section II. Section III discusses the system model and the problem formulation. The proposed approach is introduced in Section IV. We present and analyze the experimental results in Section V. Finally, we conclude this paper in Section VI.

2 Related Work

2.1 DNN Inference in MEC Networks

Recently, many pioneers have studied the issue of DNN inference in MEC networks [22], [37], [38]. The authors in [37] proposed an efficient distributed DNN inference framework in MEC networks. Different DNN block is partitioned and offloaded to different locations for inference. The authors in [22] studied an energy-aware distributed DNN inference framework to minimize energy consumption and proposed an approximate algorithm for solving the DNN inference in MEC networks. Zeng *et al.* [38] proposed a distributed DNN inference framework, named CoEdge, to utilize available computation resources at network edges and orchestrate cooperative DNN inference over heterogeneous IoT devices. However, these studies only considered DNN inference under the condition of sufficient computing resources, ignoring the restriction of specific tolerated delay. Actually, due to the real-time requirements of delay-sensitive DNN inference tasks, it is necessary to study the issue of distributed DNN inference under specific delay constraints.

To deal with scenarios with specific delay constraints, many researchers try to improve DNN inference performance while ensuring the maximum tolerated delay [15], [16], [39], [40]. Li *et al.* [15] studied a novel delay-aware DNN inference problem to speed up inference parallelism by DNN model partitioning. To satisfy the specific delay requirement of DNN inference, the authors proposed an online algorithm for dynamically improving the distributed DNN inference process. The authors in [16] optimized the model partitioning policy to reduce long-term DNN inference delay while guaranteeing delay or energy overheads of ESs. Zhang *et al.* [39] proposed a DRL-based resource management algorithm to maximize average DNN accuracy while satisfying the specific delay requirements of DNN tasks. Fan *et al.* [40] investigated the impact of joint DNN inference and task offloading in industrial MEC networks. To solve the formulated problem, the authors proposed a DRL-based algorithm for finding the near-optimal solution. These studies have been evidenced to improve the efficiency of distributed DNN inference performance significantly.

2.2 Common Approaches for DNN Inference

Extensive studies have been conducted and achieved excellent results for DNN inference with model partitioning based on DL in MEC networks [24], [41], [42]. Ren *et al.* [24] developed a fine-grained DNN model partitioning framework for distributed DNN inference towards mobile services by coordinating the hierarchical computing resources. Then the authors proposed a DRL approach for learning the optimal model partitioning policy. The authors in [41] proposed an adaptive collaborative inference framework and developed an actor-critic based DRL algorithm to search the model partitioning policies of DNN inference. Wu *et al.* [42] investigated the distributed DNN inference problem in industrial MEC networks. To solve the formulated problem, the authors proposed a DRL-based algorithm to obtain the near-optimal distributed DNN inference policy. Actually, if DRL approaches are applied directly to DNN inference, the size of the action space of the DRL agent will grow exponentially with the number of DNN blocks.

Some studies have begun to investigate the performance of multi-task learning on DNN inference [43]–[45]. Multi-task learning can improve performance by sharing information, so that the experience between DRL agents can complement each other. The authors in [43] proposed an on-device multi-task DNN inference framework that supports typical DNN layers (convolutional layers, fully connected layers) and applies to DNN tasks with different input domains. Zhang *et al.* [44] proposed a novel multi-task learning based approach to reduce the bandwidth requirement for DNN inference of IoT devices in MEC networks. The multi-task learning method is used to balance the rate control and inference tasks. The authors in [45] proposed a multi-task federated learning based algorithm for personalized DNN inference in MEC networks. In contrast, our study focuses on how to design the distributed DNN inference scheme with fine-grained model partitioning via multi-task learning A3C approach under specific delay constraints, which can be applied in realistic MEC environments.

3 System Model and Problem Formulation

In this section, we first discuss the considered MEC architecture, then detailly introduce the DNN task model and DNN inference model at the IoT devices and ESs, respectively. Finally, we formulate the total inference delay minimization problem of distributed inference in MEC networks. All symbols used in this paper are listed in Table 1.

3.1 MEC Architecture

As shown in Fig. 2, the architecture of the considered MEC network for distributed DNN inference with fine-grained model partitioning mainly includes two layers from the left to the right: device layer and edge layer.

TABLE 1
Summary of Important Notations

Notation	Definition
$\mathcal{N}(N)$	The set (number) of ESs
$\mathcal{T}(t)$	The set (index) of timeline
$\mathcal{V}(t)$	The DNN inference task generated at time slot t
$v_m(t)$	m-th inference block of DNN model
$I_m(t)$	The input data size in bits of block $v_m(t)$
$c_m(t), d_m(t)$	The required number of CPU cycles and data size
$a_m^0(t), a_m^n(t)$	The indicator for whether DNN block $v_m(t)$ is processed at IoT devices or ES_n
f^l	The local computation capability of IoT devices
F_n^C	The computation capability of ES_n
$R_m^n(t)$	The achievable transmission rate from IoT device to the associated ES
R_{max}	The transmission rate between different ESs
B	The allocated bandwidth resource for each sub-channel
p_m	The transmit power of IoT device
h_m	The channel power gain
N_0	The spectral density of noise power
F_n^C	The computation capability of ES_n
$E_m^l(t)$	The energy consumption to complete DNN block $v_m(t)$ of local inference
$E_m^n(t)$	The energy consumption of transmitting the intermediate feature map of adjacent DNN blocks
$L_m(t)$	The total inference delay
$L_{m,n}^{trans}(t)$	The transmission delay of intermediate feature map of adjacent DNN blocks $v_m(t)$ and $v_{m+1}(t)$
$L_{m,n}^{calc}(t)$	The calculation delay for DNN block $v_m(t)$ at ES_n
$\tau_m(t)$	The maximum tolerated delay for block $v_m(t)$
$\mathcal{X}^t, \mathcal{A}^t$	The system state and action
$r^t(\mathcal{X}^t, \mathcal{A}^t)$	The reward function
$V\left(\mathcal{X}^t; \theta_v\right)$	The state value function
$H\left(\pi\left(\mathcal{A}^t; \theta\right)\right)$	The entropy to encourage exploration
G^t	The discounted accumulated reward
ψ, α	The penalty term and training momentum
β	The coefficient used to balance the total inference delay and penalty term
θ	The gradients of the loss function
η, γ	The learning rate and discount factor

The device layer is mainly composed of different types of IoT devices (e.g., cameras, smartphones, smartpads and wearable devices) and these IoT devices are randomly distributed in different geographic locations through wireless connections to communicate with the ESs. These IoT devices usually have limited computing resources and can frequently generate computation-intensive DNN inference tasks with advanced services (e.g., virtual/augmented reality applications). These resource-demanding DNN inference tasks executed in resource-constrained IoT devices is difficult to meet real-time requirements. Therefore, we can partition these DNN inference tasks by dividing them into several DNN blocks and complete the distributed DNN inference process through edge-device collaboration. In actual MEC networks, we can use containerization and virtualization technologies to deploy different DNN inference tasks in independent containers and virtual machines to ensure the stability of the distributed inference process.

The edge layer consists of a large number of ESs located at the network edges close to the IoT devices. These ESs are connected with each other by wired links (e.g., Industrial Ethernet and Optical Fiber). The ESs can adopt the fine-grained model partitioning mechanism to divide the entire DNN model into multiple suitable DNN blocks, and then assign these DNN blocks to IoT devices and ES for distributed execution according to the computing and storage capabilities of the edge devices and network conditions. Specifically, each ES and IoT device can execute a partial DNN model. When the inference process of the previous DNN block is completed, it will transmit the intermediate feature map to the ESs or IoT devices where the next DNN block is located. The device layer and edge layer are cooperative to build a stable distributed DNN inference framework through the intelligent model partitioning controller.

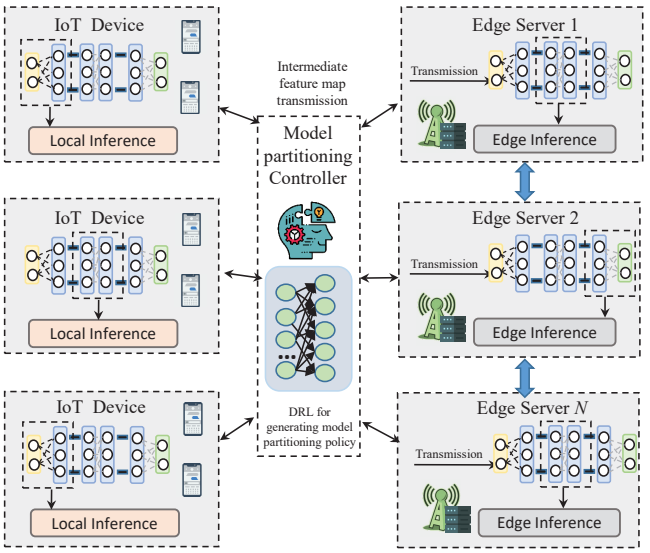
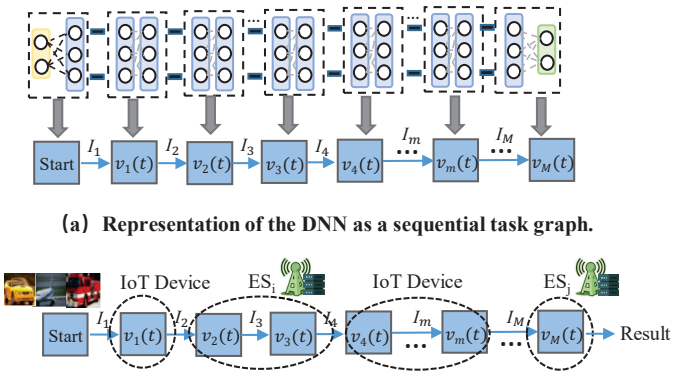


Fig. 2. Topology of distributed DNN inference with fine-grained model partitioning in MEC networks.

3.2 DNN Task Model

In particular, we consider the scenario of MEC networks consisting of N ESs (denoted by $\mathcal{N} = \{\text{ES}_1, \text{ES}_2, ..., \text{ES}_N\}$) and several IoT devices. Each IoT device is randomly distributed within the service area of the ESs and connects with ES via wireless links (e.g., 5G and WiFi). The ESs can provide distributed DNN inference services for IoT devices. Aiming to capture the distributed DNN inference process more effectively, we assume that the MEC networks operate in a slotted manner and the timeline set $\mathcal{T} = \{1, 2, ..., t, ..., T\}$ is equally divided into several time slots. Each time slot has a reasonable time span, which can be determined by DNN inference task generation patterns and system measurement.

The DNN inference task (e.g., AlexNet, NiN and VGG) consists of multiple DNN blocks and each block contains several layers (e.g., convolutional (conv) layer, pooling layer or fully connected (FC) layer). We model the DNN inference task as a sequential task graph, as shown in Fig. 3(a). Each vertex in the sequential task graph represents a DNN inference block. Mathematically, the DNN inference task generated at time slot t can be expressed as $\mathcal{V}(t) = \{v_1(t), v_2(t), ..., v_M(t)\}$, where $v_m(t)$ refers to m-th inference block of DNN model and M is total number of the blocks. Each block $v_m(t)$ can be characterized by a two-tuple $\{c_m(t), d_m(t)\}$, where $c_m(t)$ is the required



(a) Representation of the DNN as a sequential task graph.

(b) Distributed inference with fine-grained model partitioning.

Fig. 3. Representation of DNN task graph and a case of distributed inference with fine-grained model partitioning.

number of CPU cycles, $d_m(t)$ is the data size of DNN block $v_m(t)$. Considering the real-time constraints of distributed inference, we assume that each DNN block has a maximum tolerable delay $\tau_m(t)$. Each edge in the sequential task graph represents the input data of DNN block $v_m(t)$, which is the output of preceding block $v_{m-1}(t)$. We define the input data size in bits of block $v_m(t)$ as $I_m(t)$. Note that DNN block $v_m(t)$ runs in serial order, therefore, the sequential DNN inference task $\mathbb{J}(t)$ at time slot t can be expressed as

$$\mathbb{J}(t) = \{v_1(t), ..., I_m(t), v_m(t), I_{m+1}(t), ..., v_M(t)\}. \quad (1)$$

In this paper, we adopt a fine-grained model partitioning mechanism with multi-partition points. Denote set $\mathcal{A} = \{a_m^0(t), a_m^1(t), ..., a_m^N(t) | \forall m \in \mathcal{M}, \forall t \in \mathcal{T}\}$ as DNN model partitioning policy, where $a_m^0(t), a_m^n(t) \in (0, 1)$ indicates whether DNN block $v_m(t)$ is processed at IoT devices or ES$_n$, respectively. Note that each DNN block can only be executed in one IoT device or ES either, thus, the DNN model partitioning policy can be constrained by

$$a_m^0(t) + \sum_{n=1}^{N} a_m^n(t) = 1, \forall m \in \mathcal{M}, \forall t \in \mathcal{T}. \quad (2)$$

To better explain the fine-grained model partitioning mechanism with multi-partition points, Fig. 3(b) describes a case of distributed DNN inference with fine-grained model partitioning. We have a DNN model that consists of several DNN blocks. To improve scalability and enable efficient distributed Inference, we intend to partition different DNN blocks across multiple computing resources of IoT devices and ESs for execution. First, we employ a block-wise model partitioning approach, where we divide the entire DNN model into separate multiple DNN blocks at each block boundary. This allows us to distribute the computational workload across multiple compute resources of ESs or IoT devices. Each DNN block is responsible for executing the calculation operations within its assigned layer. Image data is the initial input of the DNN inference task. DNN block $v_1(t)$ is executed at IoT device while $v_2(t)$, $v_M(t)$ is executed at ES$_i$, ES$_j$ at different layers of MEC networks. All DNN blocks cooperate to produce the final inference result.

3.3 DNN Inference Model

In this section, we calculate the DNN inference delay and energy consumption in terms of local inference at IoT devices and edge inference at ESs, respectively.

1) Local Inference Model: local inference refers to executing DNN inference task on the IoT devices, and the delay and energy consumption mainly comes from the local calculation of DNN tasks. Denote f^l as the local computation capability of IoT devices. Therefore, the local inference delay for DNN block $v_m(t)$ can be calculated as

$$L_m^l(t) = \frac{c_m(t)}{f^l}, \forall m \in \mathcal{M}, \forall t \in \mathcal{T}. \quad (3)$$

The energy consumption to complete DNN block $v_m(t)$ of local inference at time slot t can be expressed as

$$E_m^l(t) = \kappa (f^l)^2 c_m(t), \forall m \in \mathcal{M}, \forall t \in \mathcal{T}, \quad (4)$$

in which $\kappa = 10^{-26}$ is the coefficient of energy consumption, which is associated with the chip architecture [46].

2) Edge Inference Model: edge inference needs to offload these DNN blocks to different ESs for inference. The inference delay is composed of two aspects: transmission delay and calculation delay. The transmission delay occurs at the transmission of the intermediate feature map of adjacent DNN blocks [47], [48]. We mainly calculate the transmission delay in three forms when considering different inference locations of adjacent DNN blocks. Hence, the transmission delay of the intermediate feature map of adjacent DNN blocks $v_{m-1}(t)$ and $v_m(t)$ can be calculated as

$$L_m^{trans}(t) = \begin{cases} 0, \\ \frac{I_m(t)}{R_m^n(t)}, \forall m \in \mathcal{M}, \forall n \in \mathcal{N}, \\ \frac{I_m(t)}{R_{max}}, \forall m \in \mathcal{M}. \end{cases} \quad (5)$$

where $R_m^n(t) = B \log_2 \left(1 + \frac{p_m |h_m|^2}{N_0 B}\right)$ is the achievable transmission rate from IoT devices to the associated ES, B is the allocated bandwidth resource for each subchannel, p_m is the transmit power of IoT device, h_m represents the channel power gain and N_0 is the noise power spectral density, R_{max} is the transmission rate between different ESs. To explain it, "0" means that two adjacent DNN blocks $v_{m-1}(t)$ and $v_m(t)$ are simultaneously processed at IoT devices or offloaded to the same ES. "$\frac{I_m(t)}{R_m^n(t)}$" refers to that one of the two adjacent DNN blocks is executed at IoT device locally, and the other is offloaded to ES for inference. "$\frac{I_m(t)}{R_{max}}$" means that two adjacent DNN blocks are executed at two different ESs. Please note that $I_1(t)$ is the size of the original data. In the image classification task, it represents the size of the original image that needs to be recognized.

The calculation delay refers to the time consumption that data input to the DNN block for calculations. Denote F_n^C as the computation capability of ES$_n$. Therefore, the calculation delay for DNN block $v_m(t)$ at ES$_n$ can be expressed as

$$L_{m,n}^{calc}(t) = \frac{c_m(t)}{F_n^C}, \forall m \in \mathcal{M}, \forall n \in \mathcal{N}. \quad (6)$$

In the edge inference model, we only consider the energy consumption caused by transmitting the intermediate feature map of DNN blocks. We ignore the energy consumption

of the DNN blocks executed at the ESs, because the ESs can be powered by some renewable energy (e.g., solar energy and wind energy). Therefore, the energy consumption for transmitting the intermediate feature map of adjacent DNN blocks $v_{m-1}(t)$ and $v_m(t)$ can be calculated as

$$E_m^n(t) = p_m L_{m,n}^{trans}(t). \tag{7}$$

As mentioned earlier, the edge inference delay is mainly composed of the transmission delay and calculation delay. Therefore, the edge inference delay for DNN block $v_m(t)$ can be calculated as

$$L_{m,n}^{O}(t) = L_{m,n}^{trans}(t) + L_{m,n}^{calc}(t). \tag{8}$$

Define $\mathbf{1}(\cdot)$ as the judgment function to determine whether the internal conditions are true. Furthermore, the total inference delay for DNN block $v_m(t)$ can be expressed as

$$L_m(t) = \mathbf{1}(a_m^0(t) = 1)L_m^l(t) + \mathbf{1}(a_m^n(t) = 1)L_{m,n}^{O}(t). \tag{9}$$

3.4 Problem Formulation

In the considered MEC networks, our target to minimize the total inference delay of DNN tasks by optimizing the model partitioning policy $\mathcal{A} = \{a_m^0(t), a_m^1(t), ..., a_m^N(t) | \forall m \in \mathcal{M}, \forall t \in \mathcal{T}\}$. Therefore, the corresponding distributed DNN inference optimization problem can be formulated as

$$\mathcal{P}: \min_{\mathcal{A}} \sum_{t=1}^{T} \sum_{m=1}^{M} L_m(t) \tag{10a}$$

$$s.t.\ \mathrm{C1}: a_m^0(t) + \sum_{n=1}^{N} a_m^n(t) = 1, \forall m \in \mathcal{M}, \forall t \in \mathcal{T}, \tag{10b}$$

$$\mathrm{C2}: L_m(t) \leq \tau_m(t), \forall m \in \mathcal{M}, \tag{10c}$$

$$\mathrm{C3}: a_m^0(t), a_m^1(t), ..., a_m^N(t) \in \{0, 1\}, \forall m \in \mathcal{M}, \forall t \in \mathcal{T}. \tag{10d}$$

Constraint C1 states that each DNN block can only be executed at one IoT device or ES either. Constraint C2 restricts the DNN inference delay of each DNN block cannot exceed the maximum tolerated delay $\tau_m(t)$. Constraint C3 means that the model partitioning policy of DNN is a 0-1 variable. The origin problem $\mathcal{P}$ is a non-convex optimization problem, which is NP-hard. It is generally challenging to solve by traditional optimization methods. Considering the fact that massive IoT devices and huge number of system parameters in the considered MEC networks, our goal is to design a DRL-based approach to find a competitive sub-optimal fine-grained model partitioning policy.

4 Framework Design

In this section, we present the multi-task learning based A3C approach for solving the corresponding distributed DNN inference problem in the considered MEC networks. Specifically, we discuss the MDP formulation of system state, action and reward design at first. Then, we explain the feasibility of integrating the A3C algorithm with multi-tasking learning. Finally, we provide the complexity analysis of the proposed approach.

4.1 MDP Formulation

We model the formulated distributed DNN inference problem with fine-grained model partitioning as a MDP process. Specifically, the MDP process consists of three key elements, i.e., system state design, model partitioning action design and system reward design.

1) System state design: at the beginning of each time slot, the agents can observe the system state information, including generated DNN inference task $\mathcal{V}(t)$ and the current state of the IoT devices. Specifically, the state information x_m^t can be defined as the set $x_m^t = \{c_m(t), d_m(t), R_m^n(t)\}$, in which $c_m(t), d_m(t), R_m^n(t)$ is the required number of CPU cycles, the data size of DNN block and the achievable transmission rate, respectively. Besides, we should consider enough energy to complete distributed DNN inference for each DNN block at the considered time slot. Denote $E^r(t)$ as the remaining energy before completing the DNN inference task $\mathcal{V}(t)$ at time slot t, which can be calculated as

$$E^r(t) = E^{max} - \sum_{i=1}^{t-1} \sum_{m=1}^{M} (\mathbf{1}(a_m^0(i) = 1)E_m^l(t) + \mathbf{1}(a_m^n(i) = 1)E_m^n(t)). \tag{11}$$

The system state needs to contain all DNN blocks information in the considered MEC networks, hence, we define the whole system state as $\mathcal{X}^t = (x_1^t, x_2^t, \ldots, x_M^t, E^r(t))$. To explain it, remaining energy $E^r(t)$ can make the DRL agent pay more attention to the energy consumption of the inference process, so that the model partitioning action generated by the agent can meet the remaining energy guarantee in the considered MEC networks.

2) Model partitioning action design: After receiving the system state information, the agent selects an appropriate DNN model partitioning action for each DNN block. The DNN model partitioning action of block $v_m(t)$ can be described as $\boldsymbol{a}_m(t) = [a_m^0(t), a_m^1(t), ..., a_m^N(t)]$. In multi-task learning based DRL approach, we expand the output layer into multiple branches to determine the DNN model partitioning policy for each DNN block separately. Each output unit makes the DNN model partitioning policy independently. Hence, the action space for distributed DNN inference can be defined as $\mathcal{A}^t = [a_1(t), a_2(t), ..., a_M(t)]$.

3) System reward design: when the DRL agent takes action $\mathcal{A}_t$ under the current state $\mathcal{X}^t$, the system will transition to the new state $\mathcal{X}^{t+1}$ and it will obtain an instantaneous feedback reward $r^t(\mathcal{X}^t, \mathcal{A}^t)$ from the environment. In this paper, our target is to minimize the total inference delay as much as possible, so the reward function is negatively related to the objective function. Based on this rule, we intend to design the reward function as the negative sum of the total inference delay, which can be defined as

$$r^t(\mathcal{X}^t, \mathcal{A}^t) = -\sum_{m=1}^{M} L_m(t). \tag{12}$$

Moreover, to guarantee each DNN task can be executed on time under the specific delay constraints, we will add a certain penalty term if the DNN inference task cannot be completed within the specified time. The penalty term is calculated as the difference between the DNN task inference

This article has been accepted for publication in IEEE Transactions on Mobile Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TMC.2024.3357874

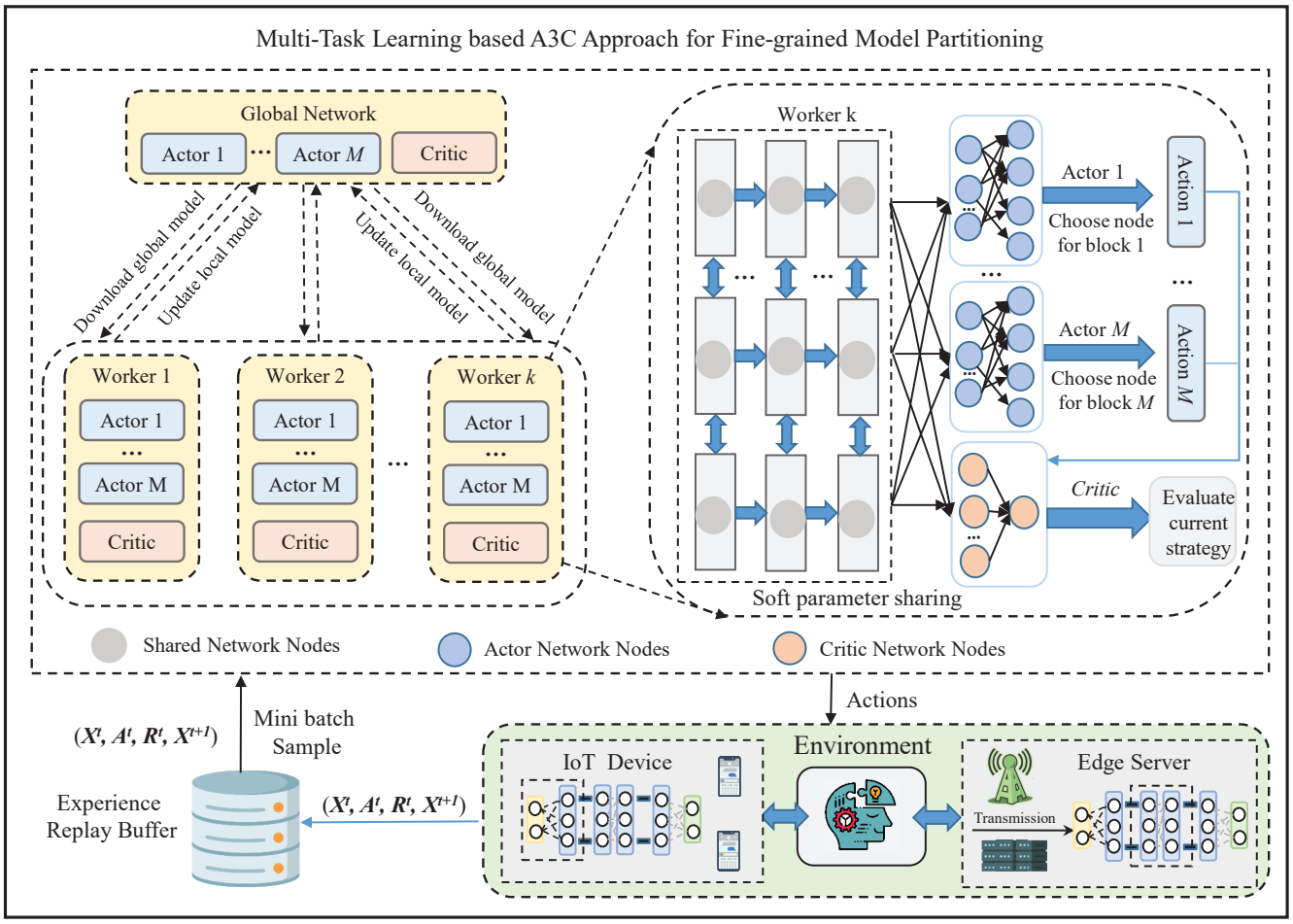


Fig. 4. An overview of the proposed multi-task learning based A3C approach for fine-grained model partitioning in the considered MEC networks.

delay $L_m(t)$ and the maximum tolerated delay $\tau_m(t)$. When the DNN task inference delay is much greater than the maximum tolerable delay $\tau_m(t)$, the value of the penalty term will be larger. Then the system will try to update the action to ensure a larger value of the reward function. Hence, the penalty term can be calculated as

$$\psi = \sum_{m=1}^{M} \max\{L_m(t) - \tau_m(t), 0\}, \tag{13}$$

To this end, taking the total inference delay and excess penalty term of DNN inference tasks into consideration, the reward function $r^t(\mathcal{X}^t, \mathcal{A}^t)$ can be rewritten as

$$r^t(\mathcal{X}^t, \mathcal{A}^t) = -(\sum_{m=1}^{M} L_m(t) + \beta\psi), \tag{14}$$

where β is the coefficient used to balance the total inference delay and penalty term. Intuitively, the system reward is always negative in the considered MEC networks.

4.2 Asynchronous Advantage Actor-Critic

A3C agent interacts with the MEC environment in multiple threads manner. Specifically, each thread aggregates the learned model weights in a central parameter server [49]–[51]. In the beginning, the central parameter server sends the parameters to worker nodes. Then, multiple worker nodes optimize the network and learn simultaneously via asynchronous gradient descent. After computing the gradient, the work nodes send their own updated parameters to the central parameters server for aggregation. A3C algorithm has significant advantages over the traditional AC algorithm. A3C uses asynchronous training. Multiple threads explore the environment independently at the same time, which improves sample efficiency and makes training faster. Besides, A3C encourages exploration, utilizes parallelism to achieve fast training, and effectively separates the policy and value network to improve stability.

At time slot t, system state $\mathcal{X}^t$ will be served as the input of the A3C approach. The actor-network receives the input of the system state $\mathcal{X}^t$ and generates a corresponding DNN model partitioning action $\mathcal{A}^t$. Meanwhile, an instantaneous reward $r^t(\mathcal{X}^t, \mathcal{A}^t)$ will be feedback to optimize the model partitioning policy. After that, the critic-network evaluates the performance of the actor-network based on the generated action and guides the actor-network in the next stage. Thus, the state value function of the A3C approach can be calculated as

$$\begin{aligned} V(\mathcal{X}^t; \theta_v) &= E\left[G^t \mid \mathcal{X} = \mathcal{X}^t, \pi\right] \\ &= E\left[\sum_{k=0}^{\infty} \gamma^k r^{t+k} \mid \mathcal{X} = \mathcal{X}^t, \pi\right], \end{aligned} \tag{15}$$

in which $G^t = \sum_{k=0}^{\infty} \gamma^k r^{t+k}$ is the accumulated discounted

reward of state $\mathcal{X}^t$, γ is the discount factor between the value of [0, 1]. The discount factor is used to measure how future rewards can influence the current state value.

A3C approach uses the k-step method to update the system reward. The policy and value function is updated until the terminal system state or after maximum step t_{max} actions, which can be given by

$$R^t = \sum_{i=0}^{k-1} \gamma^i r^{t+i} + \gamma^k V\left(\mathcal{X}^{t+k}; \theta_v\right), \tag{16}$$

where k is upper-bounded constrained by t_{max}.

Similar to actor-critic method, to reduce the variance, the A3C approach also adopts the advantage function to enhance the learning capacity. Its formula as

$$A\left(\mathcal{X}^t, \mathcal{A}^t; \theta, \theta_v\right) = R^t - V\left(\mathcal{X}^t; \theta_v\right), \tag{17}$$

where R^t is the real reward and $V\left(\mathcal{X}^t; \theta_v\right)$ is the estimated state value, θ and θ_v are the parameters of actor and critic-network, respectively. Based on the advantage function, the loss function of actor-network can be expressed as

$$\begin{aligned} f_\pi(\theta) = \log \pi\left(\mathcal{A}^t \mid \mathcal{X}^t; \theta\right)\left(R^t - V\left(\mathcal{A}^t; \theta_v\right)\right) \\ +\beta H\left(\pi\left(\mathcal{A}^t; \theta\right)\right), \end{aligned} \tag{18}$$

in which $H\left(\pi\left(\mathcal{A}^t; \theta\right)\right)$ is the entropy to encourage exploration, β is the hyperparameter that is used to control the strength of entropy regularization term and facilitates the trade-off between exploitation and exploration. Similarly, the loss function of the critic-network can be given by

$$f_v(\theta_v) = \left(R^t - V\left(\mathcal{X}^t; \theta_v\right)\right)^2, \tag{19}$$

which can be used to update the value function $V\left(\mathcal{X}^t; \theta_v\right)$.

4.3 Multi-Task Learning based Approach

A3C is a DRL algorithm primarily designed to train a single agent for solving specific tasks. However, practical scenarios often necessitate the simultaneous handling of multiple interrelated tasks. The extension of multi-task learning based A3C can support agents to exchange acquired policies and experiences among various tasks. This sharing mechanism enhances the algorithm's generalization capabilities, reduces data requirements, and facilitates knowledge transfer between tasks. Multi-task learning based A3C can improve performance by sharing information, so that the experience between agents can complement each other. Multi-task learning has a faster inference speed. In this paper, we integrate the advantages of multi-task learning and DRL to form a multi-task learning based A3C framework. An overview of the proposed multi-task learning based A3C approach can be shown in Fig. 4.

Particularly, we expand the A3C approach into a multi-task learning manner. We integrate the hidden layer of the actor-network and critic-network into a soft parameter shared layer. In this manner, the experience of each agent can be indirectly obtained by other agents. The reduction of neural network parameters also reduces the training time. Since the output of each actor-network is the policy for each specific DNN block, we need to expand the output layer of actor-network into M branches that correspond to M DNN blocks. Meanwhile, the output dimension of each branch of actor-network is determined by the size of DNN model partitioning action space. The output of the critic-network is the evaluation value of the current state, we can use only one critic-network to calculate the state value function. Therefore, the proposed multi-task learning based A3C approach has $M + 1$ branches to generate the DNN model partitioning policy for distributed DNN inference.

Algorithm 1: Multi-Task Learning based A3C Approach for Fine-grained Model Partitioning.

Initialize: initialize the actor-network and critic-network with parameters $\boldsymbol{\theta} = \{\theta_1, \theta_2, ..., \theta_M\}$ and θ_v, respectively;
while Episode is not terminated **do**
  for *each work node* **do**
    Initialize the gradients of global agent $d\boldsymbol{\theta} = 0$ and $d\theta_v = 0$, respectively:
    Synchronous parameters of each worker node with global parameters $\boldsymbol{\theta}' = \boldsymbol{\theta}$, $\theta'_v = \theta_v$;
  end
  for t in the set $\mathcal{T} = \{1, 2, 3, ..., T\}$ **do**
    Get the system state $\mathcal{X}^t$ from environment;
    Perform action $\mathcal{A}^t$ under the policy $\pi\left(\mathcal{A}^t|\boldsymbol{\theta}'\right)$;
    Obtain reward $r^t(\mathcal{X}^t, \mathcal{A}^t)$ according to Eq. (14) and new state $\mathcal{A}^{t+1}$;
    Get the estimated value from critic-network;
    Store the tuple $(\mathcal{X}^t, \mathcal{A}^t, r^t, \mathcal{A}^{t+1})$ into replay memory D;
  end
  Randomly sample a minibatch from D;
  Update the model parameters via policy gradient according to the loss function defined in (21);
end

Correspondingly, the loss function of proposed multi-task learning based A3C approach can be expressed as

$$f(\theta_1, \theta_2, ..., \theta_m, \theta_v) = \frac{\sum_{i=1}^{M} f_\pi(\theta_i)}{M} + f_v(\theta_v). \tag{20}$$

Meanwhile, the gradient descent formula to update the loss function $f_\pi(\theta)$ of actor-network can be expressed as

$$\begin{aligned} \nabla_\theta f_\pi(\theta) = \nabla_\theta \log \pi\left(\mathcal{A}^t \mid \mathcal{X}^t; \theta\right)\left(R_t - V\left(\mathcal{A}^t; \theta_v\right)\right) \\ +\beta \nabla_\theta H\left(\pi\left(\mathcal{A}^t; \theta\right)\right). \end{aligned} \tag{21}$$

Similarly, the gradient descent formula for updating the loss function of critic-network can be given by

$$\nabla_{\theta_v} f_v(\theta_v) = 2\left(R^t - V\left(\mathcal{X}^t; \theta_v\right)\right) \nabla_{\theta_v} V\left(\mathcal{X}^t; \theta_v\right), \tag{22}$$

We minimized the loss function by adopting the RMSProp algorithm, which has been widely used in many DL scenarios. Then, the estimate of the gradient under the RMSProp strategy can be expressed as

$$g = \alpha g + (1 - \alpha)\Delta\theta^2, \tag{23}$$

in which α is the coefficient of training momentum and θ is the accumulated gradients of the loss function. Based on the g just obtained, the RMSProp strategy can be updated according to the following formula as

$$\theta \leftarrow \theta - \eta \frac{\Delta\theta}{\sqrt{g + \epsilon}}, \tag{24}$$

This article has been accepted for publication in IEEE Transactions on Mobile Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TMC.2024.3357874

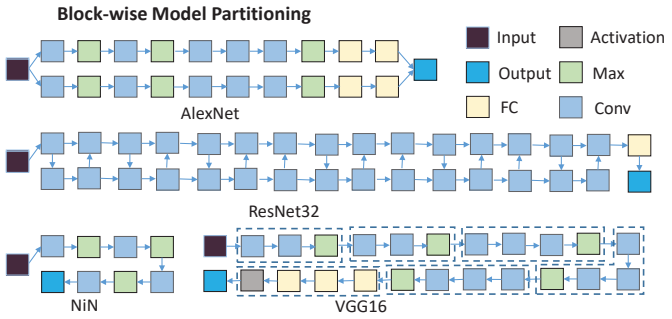


Fig. 5. Four well-known DNN models for the experiment in the considered MEC networks: Alexnet, ResNet32, NiN and VGG16.

where coefficient η represents the learning rate, and ϵ is a tiny small positive number more than zero. The detail of the multi-task learning based A3C approach for fine-grained model partitioning is shown in Algorithm 1.

4.4 Complexity Analysis

The computational complexity of the proposed multi-task learning based A3C approach mainly relies on network structure and the number of neurons. Network structure mainly includes a shared layer, actor-network with M branches, and a critic-network. Furthermore, the computation of the shared layer can be calculated as the product of the number of neurons in each layer and the number of neural network layers. Denote n_x^{share} as the number of neurons at the x-th layer at the shared layer. hence, the computation complexity is $O(\sum_{x=0}^{X-1} n_x^{share} n_{x+1}^{share})$, where X is the number of the fully connected layer in the shared layer. Similarly, we can obtain the computation complexity of each actor-network and critic-network as $O(\sum_{y=0}^{Y-1} n_y^{actor} n_{y+1}^{actor})$ and $O(\sum_{z=0}^{Z-1} n_z^{critic} n_{z+1}^{critic})$, where Y and Z is the number of the fully connected layer at actor-network and critic-network, respectively. In the training phase, the proposed approach incorporates the parameter updating of all layers, so the computation complexity is $O(\sum_{x=0}^{X-1} n_x^{share} n_{x+1}^{share} + M \cdot \sum_{y=0}^{Y-1} n_y^{actor} n_{y+1}^{actor} + \sum_{z=0}^{Z-1} n_z^{critic} n_{z+1}^{critic})$.

5 Performance Evaluation

5.1 Experiment Setup

The inference process uses a server with Intel(R) Xeon(R) Silver 4214 2.20GHz processor, 24GB of video memory GeForce RTX 3090 and RTX 2080Ti with 12GB memory and a Windows with Intel(R) Core(TM) i7-9700 CPU @3.00GHz (8CPUs), 16384MB RAM. The edge servers are distributed randomly in the considered MEC network.

Parameter Settings. We evaluate the performance of distributed DNN inference with fine-grained model partitioning through extensive experiments on MEC environments. We consider that the MEC network is composed of [2, 9] ESs. Each ES can serve the maximum coverage with a radius of 750 meters. The achievable transmission rate from IoT devices to the associated ES is set in the range of [0.8, 2] MB/s. At the beginning of each time slot, each IoT device generates DNN inference tasks and each DNN task consists of several DNN blocks. The size of the DNN block is based on the model parameter size. The maximum tolerated delay $\tau_m(t)$ of each block is set in [1, 7] seconds. Besides, we simulate T = 600 time slots and the default batchsize is set to 32. The default learning rate of actor-network and critic-network is both set as 0.01. The DNN task released rates are set in the range of [30, 110]/s. The probability of random exploration is decreasing from 1 to 0.01 gradually. The discount factor is set to 0.95.

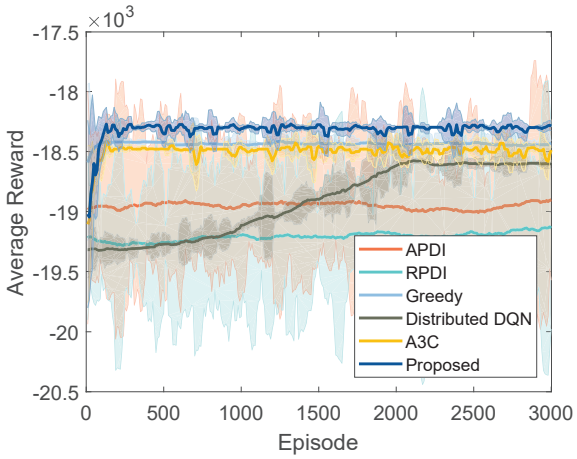


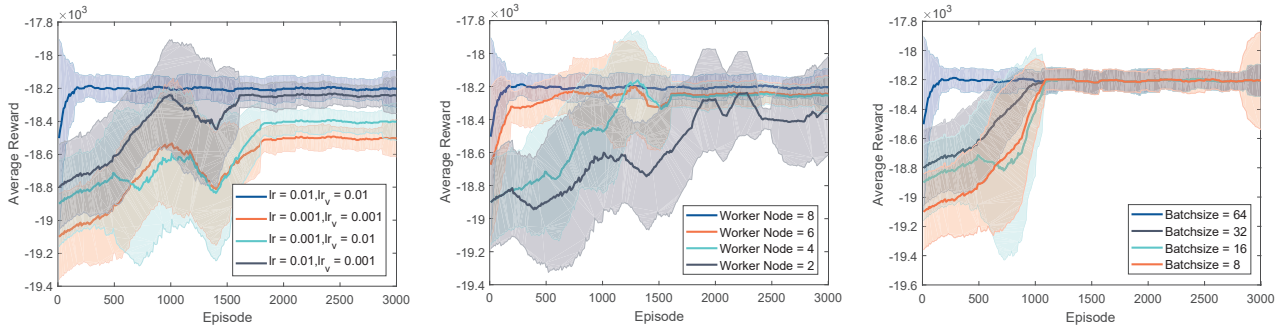
Fig. 6. Convergence of different algorithms with the default setting in the considered MEC networks.

DNN Models and Dataset. We employ four well-known DNN models for the experiments: Alexnet, ResNet32, NiN and VGG16, as shown in Fig. 5. These four DNN models have been widely benchmarked in the DL community. In this paper, we employ a block-wise partitioning approach, where we divide the entire DNN model into separate multiple DNN blocks at each block boundary. A feasible model partitioning method of VGG16 is shown at the lower right corner of Fig. 5. In terms of dataset, we use CIFAR10 as the standard dataset. CIFAR-10 is a widely acknowledged dataset in computer vision, which consists of 60,000 32x32 color images in 10 classes. Each class has 6,000 images. The dataset is divided into a training set with 50,000 images and a test set with 10,000 images for distributed DNN inference.

Performance Metrics. To evaluate the effectiveness of distributed DNN inference and the proposed approach, we use three common performance metrics in the considered MEC networks as: *1) Total Inference Delay*, including local inference delay and edge inference delay two parts; *2) Edge Inference Delay*, denoting the inference delay of executing DNN blocks at the selected ESs; *3) Local Inference Delay*, denoting the delay of executing DNN blocks at IoT devices.

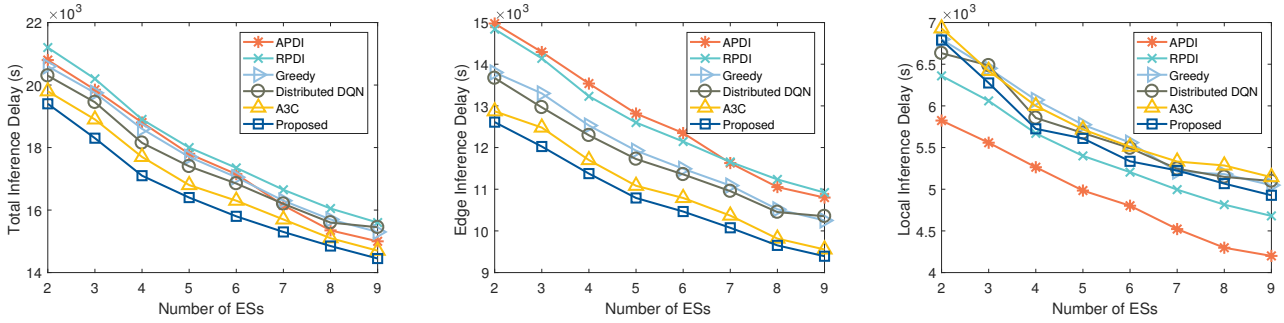
Baseline Schemes. We compare the proposed approach with the following five baseline schemes as: *1) A3C* [52]: a deep reinforcement learning approach to address the same case in this paper; *2) Distributed DQN* [53]: we use the distributed DQN algorithm to partition DNN blocks; *3) Greedy* [54]: each DNN block selects the ES with a lower load for DNN inference; *4) Average Partitioning of DNN Inference (APDI)*: all DNN blocks are equally partitioned to the ESs; *5) Random Partitioning of DNN Inference (RPDI)*: all DNN blocks are inferenced at the IoT devices and ESs randomly.

This article has been accepted for publication in IEEE Transactions on Mobile Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TMC.2024.3357874



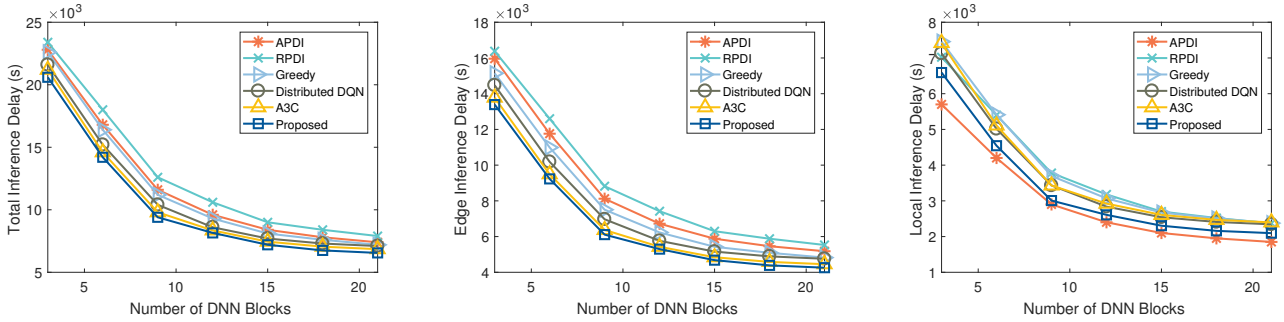
(a) Average Reward. (b) Average Reward. (c) Average Reward.

Fig. 7. The average reward versus different learning rates, work nodes and batchsize at different episodes in the considered MEC networks.



(a) Total Inference Delay. (b) Edge Inference Delay. (c) Local Inference Delay.

Fig. 8. The total inference delay, edge inference delay and local inference delay versus different numbers of ESs in the considered MEC networks (The number of DNN blocks is set as 6 and the DNN task released rate is set as 60/s).



(a) Total Inference Delay. (b) Edge Inference Delay. (c) Local Inference Delay.

Fig. 9. The total inference delay, edge inference delay and local inference delay versus different numbers of DNN blocks in the considered MEC networks (The number of ESs is set as 4 and the DNN task released rate is set as 60/s).

5.2 Algorithm Convergence

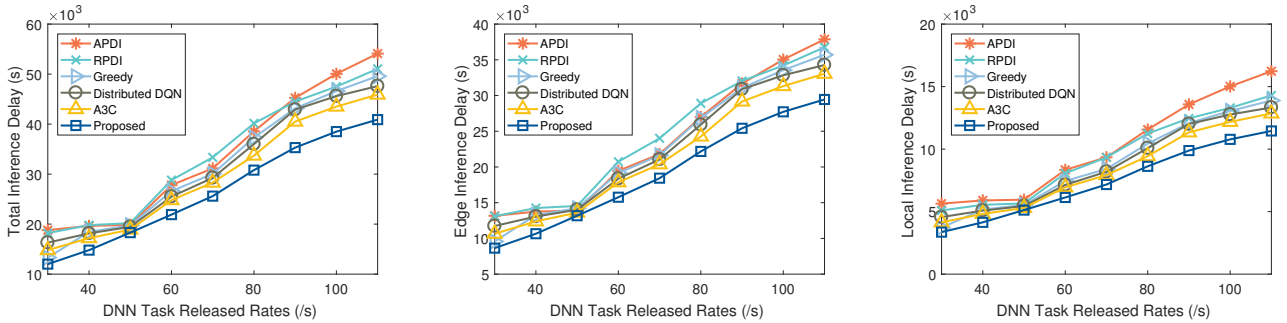
Fig. 6 shows the convergence curves of different algorithms with the default setting in the considered MEC networks. The solid curves and shaded areas are the mean and standard deviation of the average rewards, respectively. We can clearly observe that the reward value of the proposed approach increases with the number of iterations. Specifically, the reward value of the proposed approach converges to -18.30×10^3, while the reward value of A3C and distributed DQN algorithm converges to -18.49×10^3 and -18.56×10^3, respectively. The reward value of greedy, APDI and RPDI algorithms remain almost unchanged as the number of iterations increases. This is because the three baseline schemes lack the learning ability to perform adaptive fine-grained DNN model partitioning. In terms of convergence speed, the proposed approach starts to converge gradually at 400 episodes, while the distributed DQN starts to converge at about 2000 episodes. To explain it, the proposed multi-task learning based A3C approach can softly share some network parameters and system information can be better recognized. The distributed inference process between adjacent DNN blocks can be efficiently coordinated.

5.3 Impacts of Different Hyperparameter Settings

In Fig. 7, we investigate the impacts of average reward of the proposed approach in terms of different learning rates, work nodes and batchsize at different episodes.

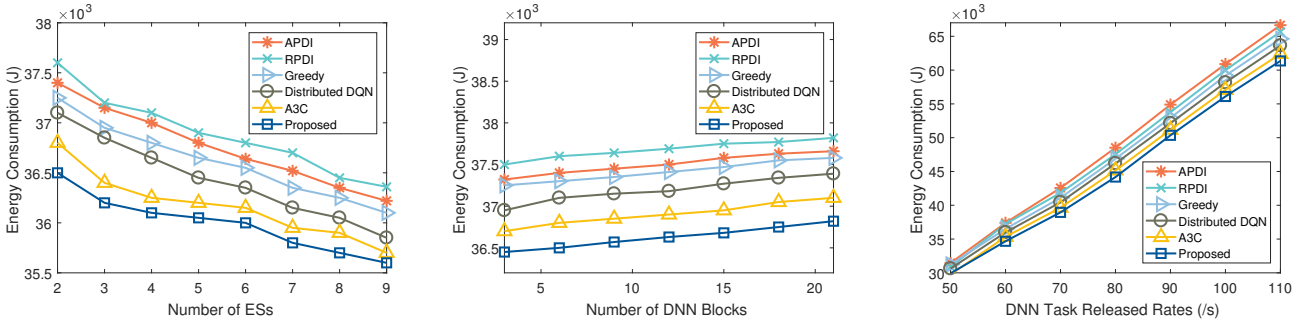
Fig. 7(a) illustrates the learning efficiency of the proposed approach in the considered MEC networks. We conduct multiple experiments and further display the reward in terms of different learning rates of actor-network and critic-network. Specifically, we alternately set the learning

This article has been accepted for publication in IEEE Transactions on Mobile Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TMC.2024.3357874



(a) Total Inference Delay. (b) Edge Inference Delay. (c) Local Inference Delay.

Fig. 10. The total inference delay, edge inference delay and local inference delay versus different DNN task released rates in the considered MEC networks (The number of ESs is set as 4 and the number of DNN blocks is set as 6).



(a) Total Inference Delay. (b) Edge Inference Delay. (c) Local Inference Delay.

Fig. 11. The energy consumption versus different number of ESs, DNN blocks, different DNN task released rates in the considered MEC networks.

rates of the actor-network and critic-network as 0.01 and 0.001, respectively. It can be easily observed that the convergence performance of the proposed approach is better when both the actor-network and critic-network learning rates are set as 0.01 (i.e., $\mathrm{lr} = 0.01, \mathrm{lr}_v = 0.01$). This is because a relatively small learning rate can easily cause the network to fall into a local optimal. Fig. 7(b) compares the performance of the reward versus the number of work nodes at different episodes. We can clearly see that the speed of the convergence curve becomes faster as the work node increases. Specifically, when the work node is 8, the convergence of the proposed framework is fastest compared with the other situations. Fig. 7(c) shows the performance of different batchsize of the proposed approach. We find that the trend of the convergence curve rises at first and then remains almost unchanged. When the batchsize is set to 64, the performance of the proposed approach works better.

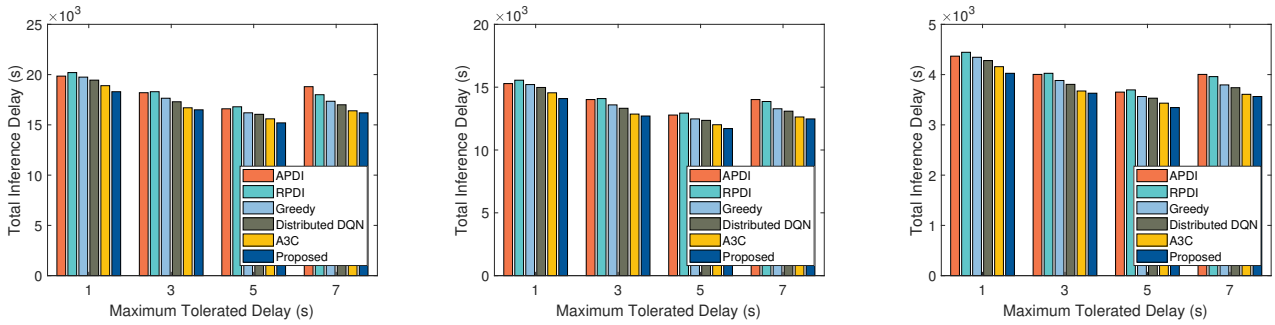
5.4 Impacts of Different Numbers of ESs, DNN Blocks and DNN Task Released Rates

In this subsection, we investigate the impact of different numbers of ESs, DNN blocks and DNN task released rates. Specifically, DNN task released rates refer to the number of DNN inference tasks generated per time slot. We use the DNN model of VGG16 for the experiments.

Fig. 8 shows the performance of total inference delay, edge inference delay and local inference delay versus different numbers of ESs in the considered MEC networks. The number of DNN blocks is set as 6 and DNN task released rates is set as 60/s. Fig. 8(a) shows that the proposed approach has an excellent performance in reducing the total inference delay compared with the other five baseline schemes. Especially when the number of ESs is equal to 6, it can achieve up to 4.61%, 6.88%, and 7.53% improvements compared with the A3C, distributed DQN and greedy baseline schemes, respectively. This is because as the number of available ESs increases, each DNN block has multiple optional ESs and the calculation delay at each ES decreases significantly. This is because the proposed multi-task learning based A3C approach can reduce the edge inference latency more effectively. Fig. 8(b) evaluates the performance in terms of the edge inference delay. We can easily observe that the gap between different algorithms becomes larger. From Fig. 8(c), the proposed multi-task learning based A3C approach can reduce 8.40%, 6.21%, 6.48% and 4.12% of the local inference delay at the number of ESs in [3, 4, 6, 7] compared with the distributed DQN baseline scheme, respectively.

Fig. 9 shows the impacts of different numbers of DNN blocks in the considered MEC networks. The number of ESs is set as 4 and DNN task released rates is set as 60/s. From Fig. 9(a), we can clearly observe that the total inference delay gradually decreases with the number of DNN blocks. This is because different DNN blocks can be partitioned to different ESs for inference, which can significantly improve available resource utilization in the considered MEC networks. Fig. 9(b) shows the trend of edge inference delay with the number of DNN blocks, the proposed approach can achieve 7.25%, 10.52%, 15.61% edge inference delay reduction when the number of DNN blocks set as 3, 6, 9 compared with the distributed DQN baseline scheme, respectively. From Fig. 9(c), the proposed multi-task learning based A3C approach

This article has been accepted for publication in IEEE Transactions on Mobile Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TMC.2024.3357874



(a) Total Inference Delay. (b) Edge Inference Delay. (c) Local Inference Delay.

Fig. 12. The total inference delay, edge inference delay and local inference delay versus different maximum tolerated delay in the considered MEC networks (The number of ESs is set as 4, the number of DNN blocks is set as 6 and DNN task released rate is set as 60/s).

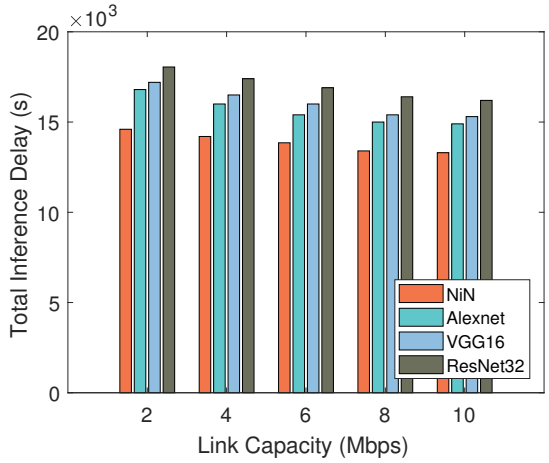


Fig. 13. The total inference delay versus different link capacity with different DNN models in the considered MEC networks.

can achieve 9.45%, 10.77% local inference delay reduction when the number of DNN blocks is set as 15, 18 compared with the greedy algorithm baseline scheme, respectively.

Fig. 10 shows the performance of different DNN task released rates in terms of the total inference delay, edge inference delay and local inference delay. The number of ESs is set as 4 and number of DNN blocks is set as 6. From Fig. 10(a), it can be easily observed that as the release rate increases, the performance gap between the proposed approach and other baseline schemes becomes larger. When the DNN task release rate is set as 50, the performance gap is the smallest. This is because the proposed multi-task learning based A3C approach can softly share network parameters and the model partitioning policy can be determined reasonably. Fig. 10(b) and Fig. 10(c) display the edge inference delay and local inference delay at different DNN task release rates, respectively. We can also observe that edge inference delay and local inference delay also maintain the growth trend. The performance gap of the local inference delay and edge inference delay fluctuates in a similar small range. Therefore, the proposed multi-task learning based A3C approach has better performance in terms of different DNN task released rates in the considered MEC networks.

5.5 Energy Consumption Under Different Numbers of ESs, DNN Blocks and DNN Task Released Rates

In this subsection, we investigate the energy consumption of VGG16 under the conditions of different numbers of ESs, DNN blocks and different DNN task released rates.

Fig. 11 shows the energy consumption versus different number of ESs, DNN blocks, different DNN task released rates in the considered MEC networks, respectively. In Fig. 11(a), we set the number of ESs in the range of [2, 9] and study the energy consumption of DNN inference. We can observe that as the number of ESs increases, the energy consumption gradually decreases. This is because as the number of ESs increases, the proposed multi-task learning based A3C approach can offload some DNN blocks to more suitable ES, which can significantly reduce energy consumption. From Fig. 11(b), we can observe that as the number of DNN blocks increases, the energy consumption increases slightly. This is because the increase of DNN blocks results in more transmission of intermediate feature maps, which results in increased energy consumption to a certain extent. Fig. 11(c) shows the energy consumption of different DNN task released rates in the considered MEC networks. We can clearly observe that as the DNN task released rates increases, the energy consumption increases more significantly. To explain it, a higher DNN task released rates means more DNN inference tasks will be generated per time slot. These DNN inference tasks will be processed in the MEC network in an edge-cloud collaboration manner, resulting in higher energy consumption. Overall, we can find that our proposed algorithm has advantages in reducing energy consumption compared to other five baseline schemes.

5.6 Impacts of Different Maximum Tolerated Delay and Link Capacity

In this subsection, we investigate the impact of different maximum tolerated delay of DNN inference tasks and link capacity in terms of the total inference delay, edge inference delay and local inference delay.

Fig. 12 shows the performance of different maximum tolerated delays in the considered MEC networks. The number of ESs is set as 4, the number of DNN blocks is set as 6 and DNN task released rates is set as 60/s. From Fig. 12(a), we can easily observe that the total inference delay first decreases and then increases with the maximum tolerated delay. To explain it, when the maximum tolerated delay

is set too small, the penalty term becomes large leading smaller reward. Fig. 12(b) shows that the proposed multi-task learning based A3C approach has a less edge inference delay. Specifically, the proposed approach can achieve up to 8.63%, 7.22%, and 7.28% performance improvements compared with the A3C strategy, 9.59%, 10.85%, and 10.74% performance improvements compared with the distributed DQN strategy at the maximum tolerated delay set as 1s, 3s and 7s, respectively. Meanwhile, from Fig.12(c), the proposed multi-task learning based A3C approach can achieve up to 7.85%, 7.57%, and 7.20% performance improvements compared with the greedy strategy at the maximum tolerated delay set as 1s, 3s and 5s, respectively.

Fig. 13 shows the impacts of the proposed multi-task learning based A3C approach on reducing the total inference delay under different link capacity from IoT devices to edge servers in the considered MEC networks. We conduct the experiment with four different DNN models (i.e., NiN, AlexNet, VGG16 and ResNet32). We can clearly find that the total inference delay decreases gradually as the link capacity increases. To explain it, this is because the increase of link capacity that results in a decrease of transmission delay between the ESs and IoT devices. In addition, we can find that the ResNet32 model has a longer total inference delay than other DNN models, and the total inference delay of Alexnet and VGG16 is larger than that of NiN model. This is because the model of Resnet32 is more complex, and a large number of convolutional layers leads to more floating point operations of total inference delay in the considered MEC networks.

6 Conclusion

In this paper, we have investigated the problem of distributed DNN inference with fine-grained model partitioning under the collaboration of ESs and IoT devices in MEC networks. Our target is to minimize the total inference delay by optimizing the fine-grained model partitioning policy with specific delay constraints. The problem is formulated as a non-convex optimization problem, which is NP-hard. To tackle this problem, we have proposed a multi-task learning based A3C approach to find a competitive fine-grained DNN model partitioning policy. Specifically, we have combined shared layers via soft parameter sharing, and expand the output layer into multiple branches to determine the model partitioning policy for each DNN block individually. The experimental results have demonstrated that our approach achieves superior performance in reducing the total inference delay, edge inference delay and local inference delay in the considered MEC networks. In future, we plan to validate our proposed approach on a practical testbed and further optimize the algorithm performance.

References

[1] K. B. Letaief, Y. Shi, J. Lu, and J. Lu, "Edge artificial intelligence for 6g: Vision, enabling technologies, and applications," *IEEE J. Sel. Areas Commun.*, vol. 40, no. 1, pp. 5–36, Nov. 2021.

[2] J. Du, F. R. Yu, G. Lu, J. Wang, J. Jiang, and X. Chu, "Mec-assisted immersive vr video streaming over terahertz wireless networks: A deep reinforcement learning approach," *IEEE Internet of Things J.*, vol. 7, no. 10, pp. 9517–9529, Jun. 2020.

[3] S. Duan, D. Wang, J. Ren, F. Lyu, Y. Zhang, H. Wu, and X. Shen, "Distributed artificial intelligence empowered by end-edge-cloud computing: A survey," *IEEE Commun. Surveys Tuts*, vol. 25, no. 1, pp. 591–624, Nov. 2022.

[4] Z. Liu, Z. Zhao, X. Wang, M. Dong, C. Qiu, and C. Zhang, "Toward mobility-aware edge inference via model partition and service migration," in *Proc. IEEE ICC*, May 2023, pp. 3258–3263.

[5] J. Song, Z. Liu, X. Wang, C. Qiu, and X. Chen, "Adaptive and collaborative edge inference in task stream with latency constraint," in *Proc. IEEE ICC*, Jun. 2021, pp. 1–6.

[6] T. Qiu, J. Chi, X. Zhou, Z. Ning, M. Atiquzzaman, and D. O. Wu, "Edge computing in industrial internet of things: Architecture, advances and challenges," *IEEE Commun. Surveys Tuts*, vol. 22, no. 4, pp. 2462–2488, Jul. 2020.

[7] Q. Luo, S. Hu, C. Li, G. Li, and W. Shi, "Resource scheduling in edge computing: A survey," *IEEE Commun. Surveys Tuts*, vol. 23, no. 4, pp. 2131–2165, Aug. 2021.

[8] S. Duan, F. Lyu, H. Wu, W. Chen, H. Lu, Z. Dong, and X. Shen, "Moto: Mobility-aware online task offloading with adaptive load balancing in small-cell mec," *IEEE Trans. Mobile Comput.*, Nov. 2022, doi:10.1109/TMC.2022.3220720.

[9] Z. Liu, J. Song, C. Qiu, X. Wang, X. Chen, Q. He, and H. Sheng, "Hastening stream offloading of inference via multi-exit dnns in mobile edge computing," *IEEE Trans. Mobi. Comp.*, Nov. 2022, doi: 10.1109/TMC.2022.3218724.

[10] X. Wang, Y. Han, V. C. Leung, D. Niyato, X. Yan, and X. Chen, "Convergence of edge computing and deep learning: A comprehensive survey," *IEEE Commun. Surveys Tuts*, vol. 22, no. 2, pp. 869–904, Jan. 2020.

[11] J. Na, H. Zhang, J. Lian, and B. Zhang, "Partitioning DNNs for optimizing distributed inference performance on cooperative edge devices: A genetic algorithm approach," *Applied Sciences*, vol. 12, no. 20, p. 10619, Oct. 2022.

[12] Z. Hao, G. Xu, Y. Luo, H. Hu, J. An, and S. Mao, "Multi-agent collaborative inference via dnn decoupling: Intermediate feature compression and edge learning," *IEEE Transactions on Mobile Computing*, Jun. 2022, doi: 10.1109/TMC.2022.3183098.

[13] Z. Liu, Q. Lan, and K. Huang, "Resource allocation for multiuser edge inference with batching and early exiting," *IEEE J. Sel. Areas Commun.*, vol. 41, no. 4, pp. 1186–1200, Feb. 2023.

[14] T. Zhu, T. Shi, J. Li, Z. Cai, and X. Zhou, "Task scheduling in deadline-aware mobile edge computing systems," *IEEE Internet of Things J.*, vol. 6, no. 3, pp. 4854–4866, Oct. 2018.

[15] J. Li, W. Liang, Y. Li, Z. Xu, X. Jia, and S. Guo, "Throughput maximization of delay-aware dnn inference in edge computing by exploring dnn model partitioning and inference parallelism," *IEEE Trans. Mobile Comput.*, 2021, doi: 10.1109/TMC.2021.3125949.

[16] Y. Su, W. Fan, L. Gao, L. Qiao, Y. Liu, and F. Wu, "Joint dnn partition and resource allocation optimization for energy-constrained hierarchical edge-cloud systems," *IEEE Trans. Vehi. Tech.*, Nov. 2022, doi: 10.1109/TVT.2022.3219058.

[17] J. Meng, H. Tan, C. Xu, W. Cao, L. Liu, and B. Li, "Dedas: Online task dispatching and scheduling with bandwidth constraint in edge computing," in *Proc. IEEE INFOCOM*, Jun. 2019, pp. 2287–2295.

[18] T. Zheng, J. Wan, J. Zhang, and C. Jiang, "Deep reinforcement learning-based workload scheduling for edge computing," *Journal of Cloud Computing*, vol. 11, no. 1, pp. 1–13, Jan. 2022.

[19] M. Gao, R. Shen, L. Shi, W. Qi, J. Li, and Y. Li, "Task partitioning and offloading in dnn-task enabled mobile edge computing networks," *IEEE Trans. Mobi. Comp.*, Sep. 2021, doi: 10.1109/TMC.2021.3114193.

[20] J.-A. Lim and Y. Kim, "Real-time dnn model partitioning for qoe enhancement in mobile vision applications," in *Proc. IEEE SECON*. IEEE, Sep. 2022, pp. 407–415.

[21] H. Zhou, M. Li, N. Wang, G. Min, and J. Wu, "Accelerating deep learning inference via model parallelism and partial computation offloading," *IEEE Trans. Para. Dist. Sys.*, vol. 34, no. 2, pp. 475–488, Nov. 2022.

[22] Z. Xu, L. Zhao, W. Liang, O. F. Rana, P. Zhou, Q. Xia, W. Xu, and G. Wu, "Energy-aware inference offloading for dnn-driven applications in mobile edge clouds," *IEEE Trans. Para. and Dist. Sys.*, vol. 32, no. 4, pp. 799–814, Apr. 2020.

[23] C. Deng, X. Fang, and X. Wang, "Uav-enabled mobile edge computing for ai applications: Joint model decision, resource allocation and trajectory optimization," *IEEE Internet of Things Journal*, Feb. 2022, doi: 10.1109/JIOT.2022.3151619.

[24] P. Ren, X. Qiao, Y. Huang, L. Liu, C. Pu, and S. Dustdar, "Fine-grained elastic partitioning for distributed dnn towards mobile web ar services in the 5g era," *IEEE Trans. Services Computing*, Jul. 2021, doi: 10.1109/TSC.2021.3098816.

[25] Z. Zhang, Q. Li, L. Lu, D. Guo, and Y. Zhang, "Joint optimization of the partition and scheduling of dnn tasks in computing and network convergence," *IEEE Networking Letters*, Mar. 2023, doi: 10.1109/TSC.2021.3098816.

[26] Y. Huang, X. Qiao, S. Dustdar, J. Zhang, and J. Li, "Toward decentralized and collaborative deep learning inference for intelligent iot devices," *IEEE Network*, vol. 36, no. 1, pp. 59–68, Feb. 2022.

[27] Y. Wu, L. Zhang, Z. Gu, H. Lu, and S. Wan, "Edge-ai-driven framework with efficient mobile network design for facial expression recognition," *ACM Trans. Embedded Computing Systems*, vol. 22, no. 3, pp. 1–17, Apr. 2023.

[28] L. Chen, M. Yao, Y. Wu, and J. Wu, "Eecdn: Energy-efficient cooperative dnn edge inference in wireless sensor networks," *ACM Trans. Internet Technology*, vol. 22, no. 4, pp. 1–30, Nov. 2022.

[29] F. Dong, H. Wang, D. Shen, Z. Huang, Q. He, J. Zhang, L. Wen, and T. Zhang, "Multi-exit dnn inference acceleration based on multi-dimensional optimization for edge intelligence," *IEEE Trans. Mobile Comp.*, 2022, doi: 10.1109/TMC.2022.3172402.

[30] T. Kim, H. Park, Y. Jin, S.-s. Lee, and S. Lee, "Partition placement and resource allocation for multiple dnn-based applications in heterogeneous iot environments," *IEEE Internet of Things J.*, 2023, doi: 10.1109/JIOT.2023.3235993.

[31] Y. Qian, R. Wang, J. Wu, B. Tan, and H. Ren, "Reinforcement learning-based optimal computing and caching in mobile edge network," *IEEE J. Selected Areas in Commun.*, vol. 38, no. 10, pp. 2343–2355, Jun. 2020.

[32] L. Ale, N. Zhang, X. Fang, X. Chen, S. Wu, and L. Li, "Delay-aware and energy-efficient computation offloading in mobile-edge computing using deep reinforcement learning," *IEEE Trans. Cogn. Commun. Netw.*, vol. 7, no. 3, pp. 881–892, Mar. 2021.

[33] T. Dong, Z. Zhuang, Q. Qi, J. Wang, H. Sun, F. R. Yu, T. Sun, C. Zhou, and J. Liao, "Intelligent joint network slicing and routing via gcn-powered multi-task deep reinforcement learning," *IEEE Trans. Cogn. Commun. Netw.*, vol. 8, no. 2, pp. 1269–1286, Dec. 2021.

[34] X. Chen, R. Proietti, C.-Y. Liu, and S. B. Yoo, "A multi-task-learning-based transfer deep reinforcement learning design for autonomic optical networks," *IEEE J. Sel. Areas Commun.*, vol. 39, no. 9, pp. 2878–2889, Mar. 2021.

[35] Y. Zhang and Q. Yang, "A survey on multi-task learning," *IEEE Trans. Knowledge and Data Engineering*, vol. 34, no. 12, pp. 5586–5609, Mar. 2021.

[36] S. Vandenhende, S. Georgoulis, W. Van Gansbeke, M. Proesmans, D. Dai, and L. Van Gool, "Multi-task learning for dense prediction tasks: A survey," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 44, no. 7, pp. 3614–3633, Jan. 2021.

[37] M. Xue, H. Wu, R. Li, M. Xu, and P. Jiao, "Eosdnn: An efficient offloading scheme for dnn inference acceleration in local-edge-cloud collaborative environments," *IEEE Tran. Green Commun. and Netw.*, vol. 6, no. 1, pp. 248–264, Sep. 2021.

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

[39] W. Zhang, D. Yang, H. Peng, W. Wu, W. Quan, H. Zhang, and X. Shen, "Deep reinforcement learning based resource management for dnn inference in industrial iot," *IEEE Trans. Vehi. Tech.*, vol. 70, no. 8, pp. 7605–7618, Aug. 2021.

[40] W. Fan, Z. Chen, Z. Hao, Y. Su, F. Wu, B. Tang, and Y. Liu, "Dnn deployment, task offloading, and resource allocation for joint task inference in iiot," *IEEE Trans. Indust. Inform.*, vol. 19, no. 2, pp. 1634–1646, Apr. 2022.

[41] T. Niu, Y. Teng, Z. Han, and P. Zou, "An adaptive device-edge co-inference framework based on soft actor-critic," in *Proc. IEEE WCNC*, Apr. 2022, pp. 2571–2576.

[42] W. Wu, P. Yang, W. Zhang, C. Zhou, and X. Shen, "Accuracy-guaranteed collaborative dnn inference in industrial iot via deep reinforcement learning," *IEEE Trans. Indust. Inform.*, vol. 17, no. 7, pp. 4988–4998, Aug. 2020.

[43] X. He, X. Wang, Z. Zhou, J. Wu, Z. Yang, and L. Thiele, "On-device deep multi-task inference via multi-task zipping," *IEEE Trans. Mobile Comput.*, 2021, doi: 10.1109/TMC.2021.3124306.

[44] J. Zhang, W. Zhang, and J. Xu, "Bandwidth-efficient multi-task ai inference with dynamic task importance for the internet of things in edge computing," *Comp. Netw.*, vol. 216, p. 109262, Oct. 2022.

[45] J. Mills, J. Hu, and G. Min, "Multi-task federated learning for personalised deep neural networks in edge computing," *IEEE Trans. Para. and Dist. Sys.*, vol. 33, no. 3, pp. 630–641, Jul. 2021.

[46] C. Sun, H. Li, X. Li, J. Wen, Q. Xiong, X. Wang, and V. C. Leung, "Task offloading for end-edge-cloud orchestrated computing in mobile networks," in *Proc. IEEE WCNC*, Jun. 2020, pp. 1–6.

[47] J.-W. Chang, K.-W. Kang, and S.-J. Kang, "An energy-efficient fpga-based deconvolutional neural networks accelerator for single image super-resolution," *IEEE Trans. Circuits and Systems for Video Technology*, vol. 30, no. 1, pp. 281–295, Dec. 2018.

[48] T. Mohammed, C. Joe-Wong, R. Babbar, and M. Di Francesco, "Distributed inference acceleration with adaptive dnn partitioning and offloading," in *Proc. IEEE INFOCOM*. IEEE, Aug. 2020, pp. 854–863.

[49] B. Gu, M. Alazab, Z. Lin, X. Zhang, and J. Huang, "Ai-enabled task offloading for improving quality of computational experience in ultra dense networks," *ACM Trans. Internet Technology*, vol. 22, no. 3, pp. 1–17, Mar. 2022.

[50] J. Jiang, J. Guo, M. Khan, Y. Cui, and W. Lin, "Energy-saving service offloading for the internet of medical things using deep reinforcement learning," *ACM Trans. Sensor Networks*, Aug. 2022.

[51] Z. Wang, M. Li, L. Zhao, H. Zhou, and N. Wang, "A3c-based computation offloading and service caching in cloud-edge computing networks," in *Proc. IEEE INFOCOM WKSHPS*, May 2022, pp. 1–2.

[52] J. Zou, T. Hao, C. Yu, and H. Jin, "A3c-do: A regional resource scheduling framework based on deep reinforcement learning in edge scenario," *IEEE Trans. Comp.*, vol. 70, no. 2, pp. 228–239, Apr. 2020.

[53] C.-C. Lin, D.-J. Deng, Y.-L. Chih, and H.-T. Chiu, "Smart manufacturing scheduling with edge computing using multiclass deep q network," *IEEE Trans. Industrial Informatics*, vol. 15, no. 7, pp. 4276–4284, Mar. 2019.

[54] H. Gauttam, K. Pattanaik, S. Bhadauria, G. Nain, and P. B. Prakash, "An efficient dnn splitting scheme for edge-ai enabled smart manufacturing," *Journal of Industrial Information Integration*, p. 100481, Aug. 2023, doi: https://doi.org/10.1016/j.jii.2023.100481.

Hui Li (S'20) received the B.S. degree from Northeast Agricultural University, Harbin, China, in 2019. He is currently a Ph.D. student with the School of Big Data & Software Engineering, Chongqing University, Chongqing, China. His current research interests include edge computing and caching, and edge intelligence.

Xiuhua Li (S'12, M'19) received his both B.S. degree and M.S. degree from Harbin Institute of Technology, China, in 2011 and 2013, respectively, and his Ph.D. degree from the Department of Electrical and Computer Engineering, University of British Columbia, Vancouver, Canada, in 2018. He is currently a tenure-track assistant professor with the School of Big Data & Software Engineering, Chongqing University, Chongqing, China, and also a member of the Haihe Laboratory of ITAI. He is the Head of the Institute of Intelligent Software and Services Computing associated with the Key Laboratory of Dependable Service Computing in Cyber Physical Society, Chongqing University, Education Ministry, China. Focusing on the research of edge computing and caching, and edge intelligence, he has published more than 90 technical papers in IEEE JSAC, TCC, TWC, IoTJ, TNSE, TNSM, ICC, GLOBECOM and so on.

This article has been accepted for publication in IEEE Transactions on Mobile Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TMC.2024.3357874

Qilin Fan (M'19) is currently an Associate Professor in the School of Big Data and Software Engineering, Chongqing University, Chongqing, China. She received the B.E. degree in the College of Software Engineering, Sichuan University, Chengdu, China, in 2011, and the Ph.D. degree from the Department of Computer Science and Technology, Tsinghua University, Beijing, China, in 2017. Her research interests include network optimization, mobile edge computing and caching, network virtualization and machine learning.

Qiang He received his first PhD degree from Swinburne University of Technology, Australia, in 2009 and his second PhD degree from Huazhong University of Science and Technology, China, in 2010. He is currently a full professor. His research interests include edge computing, cloud computing, software engineering, and service computing. More details about his research can be found at https://sites.google.com/site/heqiang/.

Xiaofei Wang (S'06, M'13, SM'18) received the B.S. degree from Huazhong University of Science and Technology, China, and received M.S. and Ph.D. degrees from Seoul National University, Seoul, South Korea. He was a Postdoctoral Fellow with The University of British Columbia, Vancouver, Canada, from 2014 to 2016. He is currently a Professor with the College of Intelligence and Computing, Tianjin University, Tianjin, China. Focusing on the research of edge computing, edge intelligence, and edge systems, he has published more than 150 technical papers in IEEE JSAC, TCC, ToN, TWC, IoTJ, COMST, TMM, INFOCOM, ICDCS and so on. In 2017, he was the recipient of the "IEEE ComSoc Fred W. Ellersick Prize", and in 2022, he received the "IEEE ComSoc Asia-Pacific Outstanding Paper Award".

Victor C. M. Leung (S'75, M'89, SM'97, F'03, LF'21) is a Distinguished Professor of Computer Science and Software Engineering at Shenzhen University, China. He is also an Emeritus Professor of Electrical and Computer Engineering and Director of the Laboratory for Wireless Networks and Mobile Systems at the University of British Columbia (UBC), Canada. His research is in the broad areas of wireless networks and mobile systems, and he has published widely in these areas. Dr. Leung is serving on the editorial boards of the IEEE Transactions on Green Communications and Networking, IEEE Transactions on Cloud Computing, IEEE Transactions on Computational Social Systems, IEEE Access, IEEE Network, and several other journals. He received the 1977 APEBC Gold Medal, 1977-1981 NSERC Postgraduate Scholarships, IEEE Vancouver Section Centennial Award, 2011 UBC Killam Research Prize, 2017 Canadian Award for Telecommunications Research, 2018 IEEE TCGCC Distinguished Technical Achievement Recognition Award, and 2018 ACM MSWiM Reginald Fessenden Award. He co-authored papers that won the 2017 IEEE ComSoc Fred W. Ellersick Prize, 2017 IEEE Systems Journal Best Paper Award, 2018 IEEE CSIM Best Journal Paper Award, and 2019 IEEE TCGCC Best Journal Paper Award. He is a Life Fellow of IEEE, and a Fellow of the Royal Society of Canada (Academy of Science), Canadian Academy of Engineering, and Engineering Institute of Canada. He is named in the current Clarivate Analytics list of "Highly Cited Researchers".