Adaptive Device-Edge Collaboration on DNN Inference in AIoT: A Digital-Twin-Assisted Approach

Shisheng Hu, *Graduate Student Member, IEEE*, Mushu Li, *Member, IEEE*, Jie Gao, *Senior Member, IEEE*, Conghao Zhou, *Member, IEEE*, and Xuemin Shen, *Fellow, IEEE*

***Abstract*—Device-edge collaboration on deep neural network (DNN) inference is a promising approach to efficiently utilizing network resources for supporting Artificial Intelligence of Things (AIoT) applications. In this article, we propose a novel digital twin (DT)-assisted approach to device-edge collaboration on DNN inference that determines whether and when to stop local inference at a device and upload the intermediate results to complete the inference on an edge server. Instead of determining the collaboration for each DNN inference task only upon its generation, multi-step decision making is performed during the on-device inference to adapt to the dynamic computing workload status at the device and the edge server. To enhance the adaptivity, a DT is constructed to evaluate all potential offloading decisions for each DNN inference task, which provides augmented training data for a machine learning-assisted decision-making algorithm. Then, another DT is constructed to estimate the inference status at the device to avoid frequently fetching the status information from the device, thus reducing the signaling overhead. We also derive necessary conditions for optimal offloading decisions to reduce the offloading decision space. Simulation results demonstrate the outstanding performance of our DT-assisted approach in terms of balancing the tradeoff among inference accuracy, delay, and energy consumption.**

***Index Terms*—Artificial Intelligence of Things (AIoT), device-edge collaborative inference, digital twin (DT), networking for AI.**

I. Introduction

ARTIFICIAL Intelligence of Things (AIoT) is receiving increasingly attention due to the remarkable capability of artificial intelligence (AI) on data analysis for Internet of Things (IoT) applications, such as smart cities and smart manufacturing [2], [3]. Deep neural network (DNN) has been widely adopted in AIoT due to its ability of automatic feature extraction for pattern recognition or decision making [2], [3]. DNNs for AIoT applications are commonly deployed on cloud servers and edge servers since AIoT devices, e.g., smart cameras, have limited computing capabilities. Cloud servers, which possess extensive computing capabilities and data, can be leveraged for DNN training. In addition, edge servers, deployed in close proximity of AIoT devices, can process offloaded task data and generate processing results by DNN inference with low latency [4]. With the growing popularity of AIoT applications, the demand for DNN inference tasks (referred to as DNN tasks) can be substantial, e.g., tens of frames are generated per second by a smart camera for real-time intersection monitoring [5]. In this regard, optimizing the performance of DNN inference via efficient utilization of network resources becomes a significant issue.

DNN partitioning is a promising solution to achieve the collaboration on DNN inference between AIoT devices and edge servers for enhanced performance of DNN inference. In DNN partitioning, the inference of a multi-layer DNN can be partitioned into two parts, i.e., on-device inference and edge inference [6]. Specifically, an AIoT device executes the layers before a partition layer, i.e., on-device inference, and uploads the intermediate result, as the input to the partition layer, to an edge server. The edge server then executes the remaining layers of the DNN, i.e., edge inference, to obtain the final inference result. The partition layer for a DNN task determines computing load of on-device inference and edge inference, respectively, and the size of data to be offloaded. Therefore, the partition layer should be properly chosen to minimize the overall delay and energy consumption for task processing [7]. When the edge server is heavily loaded, the overall delay to process a task can be high, even if the optimal partition layer is chosen for the task. In this case, the DNN inference can be early exited by AIoT devices for some tasks instead of offloading them to the edge server for completing the inference [8]. In this way, the overall delay of task processing can be reduced at the cost of reduced inference accuracy [9]. By determining whether to offload a DNN task to the edge server or not, such a tradeoff can be made as needed.

With device-edge collaboration, DNN task offloading decisions are made on whether and when an AIoT device should stop on-device inference and upload intermediate results to an edge server to complete the inference. Such decision making should adapt to the computing workload status at

Manuscript received 24 June 2023; revised 9 November 2023; accepted 20 November 2023. Date of publication 28 November 2023; date of current version 26 March 2024. This work was supported by the Natural Sciences and Engineering Research Council (NSERC) of Canada. This article was presented in part at IEEE Global Communications Conference, Rio de Janeiro, Brazil, 2022 [DOI: 10.1109/GLOBECOM48099.2022.10001005]. *(Corresponding author: Mushu Li.)*

Shisheng Hu, Conghao Zhou, and Xuemin Shen are with the Department of Electrical and Computer Engineering, University of Waterloo, Waterloo, ON N2L 3G1, Canada (e-mail: shisheng.hu@uwaterloo.ca; c89zhou@uwaterloo.ca; sshen@uwaterloo.ca).

Mushu Li is with the Department of Electrical, Computer and Biomedical Engineering, Toronto Metropolitan University, Toronto, ON M5B 2K3, Canada (e-mail: mushu.li@ieee.org).

Jie Gao is with the School of Information Technology, Carleton University, Ottawa, ON K1S 5B6, Canada (e-mail: jie.gao6@carleton.ca).

Digital Object Identifier 10.1109/JIOT.2023.3336600

the device and the edge server, which is challenging since the workload status can change dynamically due to stochastic task generation at each AIoT device covered by the edge server [10]. In particular, conventional approaches that make the offloading decision for each DNN task only upon its generation is unlikely to be optimal [11]. This is because on-device inference delay for a task can be as long as hundreds of milliseconds for executing one convolutional layer [9], which can result in 1) a drastic change of the instantaneous workload status between the generation of the task and the completion of the on-device inference for the task, and 2) difficulty in predicting such a change especially when the dynamics of task arrival is unavailable [9], [12].

The digital twin (DT) paradigm, proposed as a promising next stage of network virtualization [2], can be leveraged to facilitate adaptive device-edge collaboration on DNN inference [1]. Through data synchronization, DTs can be constructed as digital representations of physical objects or processes to manage network data [13], estimate network status (e.g., the running states of IoT devices [14]), and predict network performance (e.g., the end-to-end delay of network slices [15]). In addition, DTs can empower network emulations for efficient network management [2], e.g., constructing a simulation environment for pretraining a machine learning-assisted task offloading algorithm [16]. For the case of device-edge collaborative DNN inference, DTs can empower machine learning-assisted offloading decision making by providing augmented training data. For example, after a DNN task is offloaded to an edge server, DTs can be used to evaluate alternative offloading decisions of the task in hypothetical scenarios, i.e., offloading the task later or locally completing the task.

In this article, we leverage DTs to facilitate adaptive device-edge collaborative DNN inference. We consider an AIoT scenario, where a full-size DNN is deployed at an edge server for high-accuracy inference and a shallow DNN, which consists of the first several layers of the full-size DNN concatenated by an exit branch, is deployed at an AIoT device for low-latency on-device inference. In addition, the on-device and edge server computing workloads are dynamic due to stochastic task arrival. The objective is to optimize the performance of DNN inference in terms of balancing inference accuracy, delay, and energy consumption by determining the device-edge collaboration for the DNN tasks generated by the AIoT device. To achieve this objective, we propose a DT-assisted approach to adaptive device-edge collaboration on DNN inference, which determines whether to offload a DNN task each time when the on-device inference status changes, i.e., when a layer of the shallow DNN is about to be locally executed for the task. An offloading decision-making algorithm based on the optimal stopping theorem and assisted by machine learning is developed. Moreover, DTs are constructed in the proposed approach to play the following two roles.

1) Evaluating all potential offloading decisions of each DNN task to provide augmented training data for the learning-assisted offloading decision-making algorithm.
2) Estimating the inference status at the device to avoid frequently fetching the status information from the device, thus reducing the signaling overhead.

The main contributions of this article are summarized as follows.

1) We propose an approach to device-edge collaboration on DNN inference that adapts to the dynamic on-device and edge server workloads in offloading decision making to optimize the DNN inference performance.
2) We propose a learning-assisted algorithm, which generates effective offloading decisions with unknown task arrival statistics. Empowered by DTs, the algorithm can promptly attain an effective offloading solution with sufficient training data.
3) We derive necessary conditions for optimal offloading decisions and accordingly use them for decision space reduction, which reduces the complexity for adaptive device-edge collaboration on DNN inference.

The remainder of this article is organized as follows. In Section II, we review the related works. In Section III, we introduce the system model. In Section IV, a DT-assisted approach to adaptive device-edge collaboration on DNN inference is proposed. The problem is formulated and transformed in Section V, and the DT and learning-assisted offloading decision-making algorithm is proposed in Section VI. In Section VII, we investigate offloading decision space reduction. Simulation results are provided in Section VIII. Section IX concludes this article.

II. Related Works

Collaboration among end devices and computing servers on task processing has attracted increasing research attention for efficient utilization of network computing resources. The collaboration among various computing servers within an IoT network was investigated in [17] and [18]. For the collaboration of AIoT devices and computing servers on DNN inference, some existing works optimized the number of DNN tasks offloaded from each AIoT device to the computing servers to minimize the overall delay in task completion [19], [20], [21]. Due to the limited computing capabilities, a lightweight DNN was executed by AIoT devices for task processing to reduce the delay of on-device DNN inference at the cost of inference accuracy [20], [21], [22]. In such a case, the offloading decision was made to optimize the tradeoff between DNN inference costs (in terms of delay, energy consumption, etc.) and inference accuracy.

Leveraging DNN task partitioning, some works investigated DNN task offloading by selecting the partition layer in light of server workload [6], [23], [24]. In [6], DNN partitioning-based device-edge collaboration was proposed, and the partition layer to minimize the delay or energy consumption of a DNN task was dynamically selected based on the real-time workload at a computing server. In a multidevice scenario, partition layer selection was jointly optimized with task scheduling and computing resource allocation at an edge server [23], [24]. Other works evaluated the on-device computing workload for DNN task partitioning [12], [25].

Specifically, considering periodical task generation at the device, Liang et al. [25] maximized the throughput of the DNN inference while preventing the DNN tasks from experiencing on-device queuing delay. For the case when the DNN task generation at the device is nonperiodical, Song et al. [12] proposed a deep reinforcement learning (DRL)-based algorithm to dynamically select the partition layer. The aforementioned works adapted to the network dynamics, e.g., dynamic channel condition [26], on-device workload [12], and server workload [6], by making offloading decision for each DNN task upon its generation. In order to make the device-edge collaboration more adaptive, the offloading decision of each DNN task was adjusted during the on-device DNN inference based on the real-time estimation and the statistics of channel conditions [11].

Different from the existing works, this work proposes a DT-assisted approach to device-edge collaboration on DNN inference that can adapt to the dynamic on-device and edge server computing workloads with unknown task arrival statistics. In addition, we investigate how to reduce the signaling overhead for and the complexity of adaptive device-edge collaboration on DNN inference, by establishing a DT and by analyzing properties of the workload evolution, respectively. Moreover, instead of simply employing a learning algorithm to identify unknown network dynamics in offloading decision making, we investigate how a DT can be used to augment the training data for empowering such a learning algorithm.

III. System Model

In this section, we first introduce the models of DNN task generation, computing and queuing. Then, we define the utility of a DNN task, which incorporates the delay, inference accuracy, and energy consumption for processing a DNN task.

A. Task Generation Model

Consider an AIoT device that connects to an edge computing server through an access point (AP). The AIoT device collects sensing data, such as images, and generates computing tasks, such as object recognition, for processing the collected data. The time horizon is divided into slots with equal duration denoted by ΔT, and the index of a time slot is denoted by $t \in \{1, 2, \ldots\}$. At the beginning of each time slot, a task is probabilistically generated by the device with an unknown probability [12]. Let ΔT_n represent the interval between the time instants when the $(n+1)$th and the nth tasks are generated, where $n \in \{1, 2, \ldots, N\}$, and let ΔT_0 represent the time instant when the first task is generated.

B. Task Computing Model

For processing the tasks, one full-size DNN with L consecutive layers (e.g., $L = 7$ in Fig. 1) is deployed at the edge server, and one shallow DNN with fewer layers is deployed at the AIoT device. For DNNs that contain parallel layers or residual blocks, the layers in each residual block or the layers with parallel execution can be abstracted as one logical layer [24], [27]. In the design and training of the full-size and shallow DNNs, the BranchyNet architecture [28] is used such

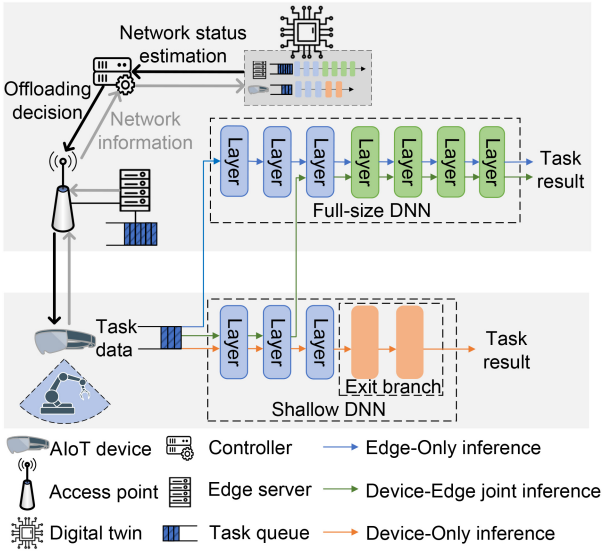


Fig. 1. System model.

that the first l_e layers of the two DNNs are identical (e.g., the three blue-colored layers in Fig. 1). The remaining part of the shallow DNN is referred to as an exit branch, and the layers therein (e.g., the two orange-colored layers in Fig. 1) are abstracted as one logical layer, i.e., the $(l_e + 1)$th layer of the shallow DNN.

Each time before the AIoT device executes the next layer in the shallow DNN, a network controller decides whether the AIoT device should stop locally processing the task at this point and offload the task to the edge server instead. For the nth task, when the task is offloaded to the edge server or locally completed, the number of executed layers in the shallow DNN for the task is determined, which is denoted by $x_n \in \{0, 1, \ldots, l_e + 1\}$. We define x_n as the offloading decision for the nth task, which can result in three DNN inference scenarios as illustrated in Fig. 1.

1) *Edge-Only Inference:* If $x_n = 0$, the nth DNN task is offloaded to the edge server without being processed by any layer of the shallow DNN by the device. In other words, the task is completely processed by the full-size DNN at the edge server.
2) *Device-Edge Joint Inference:* If $1 \leq x_n \leq l_e$, the nth DNN task is locally processed by the first x_n layers of the shallow DNN (e.g., $x_n = 2$ for the device-edge joint inference scenario in Fig. 1). Then, the output of the x_nth layer of the shallow DNN, as the intermediate result, is uploaded to the edge server and processed by the remaining layers of the full-size DNN.
3) *Device-Only Inference:* If $x_n = l_e + 1$, the nth DNN task is not offloaded to the edge server but completely processed by the shallow DNN at the device.

To assist in making offloading decisions of DNN tasks, as shown in Fig. 1, DTs are constructed by the network controller. Specifically, the network controller collects the network information, i.e., the per-layer on-device DNN inference delay and the information on task arrivals at the AIoT device and the edge server. The collected information is processed by DT

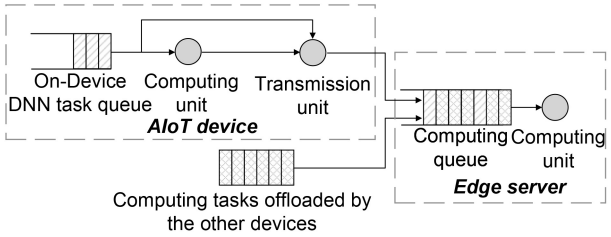


Fig. 2. Task queuing model.

models to estimate the status of the AIoT device and the edge server for assisting in making offloading decisions of DNN tasks. Detailed introduction of the DTs will be presented in Section IV.

C. Task Queuing Model

As shown in Fig. 2, the task queuing model includes queuing at the device and queuing at the edge server.

1) On-Device Task Queue: The device has one computing unit and one transmission unit, which can, respectively, process and transmit only one task at any time instant. The tasks that have been generated but not yet processed or offloaded are stored in an on-device task queue following the first-come-first-serve (FCFS) rule. When the computing unit becomes idle, the first task in the on-device task queue leaves the queue. If the transmission unit is also idle, edge-only inference, i.e., the transmission unit offloads the task to the edge server without any on-device processing, can be chosen for the task. Otherwise, the task will be processed by the computing unit until the transmission unit is idle and a decision to offload the task is made. At the beginning of the $(t+1)$th time slot, the on-device computing workload in terms of the number of tasks in the on-device queue, denoted by $Q^D(t+1)$, is updated as

$$Q^D(t+1) = Q^D(t) + I(t+1) - O(t+1) \tag{1}$$

where $I(t)$ equals to 1 if one DNN task is generated in the $(t-1)$th time slot and 0 otherwise; $O(t)$ equals to 1 if a DNN task leaves the on-device task queue at the beginning of the tth time slot and 0 otherwise.

2) Edge Server Queue: The edge server has one computing unit, which can process only one task at any time instant, and maintains a computing queue for tasks yet to be processed. The computing tasks arriving at the edge server in each time slot will either start to be processed by the computing unit or enter the computing queue at the beginning of the next time slot.[1] The edge server workload is the sum of CPU cycles required to complete the processing of the task in the computing unit as well as all the tasks in the computing queue. At the beginning of the $(t+1)$th time slot, the edge server workload, denoted by $Q^E(t+1)$, is updated as

$$Q^E(t+1) = \max\left\{Q^E(t) - f^E\Delta T, 0\right\} + D(t) + W(t) \tag{2}$$

[1]The processing order of the computing tasks arriving at the edge server in a time slot is determined by a task scheduling algorithm. For simplicity, we assume that the task offloaded by the considered device will be first processed.

where $D(t)$ and $W(t)$ represent the workload of the tasks from the considered device and other devices connected to the edge server, respectively, in the tth time slot; f^E represents the computation frequency of the edge server in the unit of CPU cycles per second.

D. Task Utility

In this section, we first derive the delay, inference accuracy, and energy consumption of processing a DNN task. Then, we define the utility of a DNN task.

1) Delay: Depending on the task offloading decision, the processing of a DNN task can incur 1) on-device queuing delay; 2) on-device inference delay; 3) transmission delay from uploading the intermediate result; 4) queuing delay at the edge server; and 5) edge inference delay. The delay from delivering the task result to the device is neglected due to the typically small size of the result.

a) On-device inference delay: The on-device inference delay of the nth task is the sum of the execution delay of the first x_n layers in the shallow DNN, given by

$$T_n^{\mathrm{lc}}(x_n) = \begin{cases} \sum_{l=1}^{x_n} d_l^D, & x_n \geq 1 \\ 0, & x_n = 0 \end{cases} \tag{3}$$

where d_l^D represents the time to execute the lth layer in the shallow DNN by the device $\forall l \in \{1, 2, \ldots, l_e+1\}$. The value of d_l^D is rounded to an integer multiple of the time slot duration ΔT.

b) On-device queuing delay: The queuing delay of the nth DNN task at the device is determined by the offloading decisions for the DNN tasks generated earlier, which are denoted as $\mathbf{x}_{n-1} = \{x_1, x_2, \ldots, x_{n-1}\}$. Let $\mathbf{x}_0 = \emptyset$ for consistency. The on-device queuing delay for the nth task, denoted by $T_n^{\mathrm{lq}}(\mathbf{x}_{n-1})$, can be calculated by

$$T_n^{\mathrm{lq}}(\mathbf{x}_{n-1}) = \begin{cases} 0, & n = 1 \\ \max\{T_{n-1}^{\mathrm{lq}}(\mathbf{x}_{n-2}) + T_{n-1}^{\mathrm{lc}}(x_{n-1}) - \Delta T_{n-1}, 0\}, & n \geq 2. \end{cases} \tag{4}$$

c) Data uploading delay: If edge-only inference or device-edge joint inference is adopted for the nth DNN task, i.e., $0 \leq x_n \leq l_e$, the task processing will incur an uploading delay denoted by T_n^{up}. The data to be uploaded is the input to the (x_n+1)th layer of the shallow DNN, and the uploading delay can be calculated by

$$T_n^{\mathrm{up}}(x_n) = \begin{cases} s_{x_n}/R_0, & 0 \leq x_n \leq l_e \\ 0, & x_n = l_e + 1 \end{cases} \tag{5}$$

where s_l represents the size of the data to the $(l+1)$th layer of the shallow DNN $\forall l \in \{0, 1, \ldots, l_e\}$; R_0 is the uplink transmission rate between the device and the AP.

d) Edge queuing delay: If the nth DNN task is offloaded to the edge server, i.e., $x_n \leq l_e$, the task processing can incur a queuing delay at the edge server, given by

$$T_n^{\mathrm{eq}}(\mathbf{x}_n) = \begin{cases} Q^E\left(t_{n,x_n}\right)/f_s^E, & 0 \leq x_n \leq l_e \\ 0, & x_n = l_e + 1 \end{cases} \tag{6}$$

where t_{n,x_n} is the index of the time slot when the nth DNN task arrives at the edge server.[2]

e) Edge inference delay: If $x_n \le l_e$, the task processing will incur an inference delay at the edge server from executing the remaining layers of the full-size DNN, given by

$$T_n^{\mathrm{ec}}(x_n) = \begin{cases} \sum_{l=x_n+1}^{L} d_l^E, & 0 \le x_n \le l_e \\ 0, & x_n = l_e + 1 \end{cases} \tag{7}$$

where d_l^E represents the execution time in the lth layer of the full-size DNN by the edge server.

Summarizing the delay in (3) to (7), the overall delay of the nth DNN task, denoted by $T_n(\mathbf{x}_n)$, can be calculated by

$$\begin{aligned} T_n(\mathbf{x}_n) = {} & T_n^{\mathrm{lq}}(\mathbf{x}_{n-1}) + T_n^{\mathrm{lc}}(x_n) \\ & + T_n^{\mathrm{up}}(x_n) + T_n^{\mathrm{eq}}(\mathbf{x}_n) + T_n^{\mathrm{ec}}(x_n). \end{aligned} \tag{8}$$

2) Inference Accuracy: The accuracy of the task result is denoted by $A_n(x_n)$, which equals η^E if the task is offloaded to the edge server for processing by the full-size DNN, i.e., $1 \le x_n \le l_e$, and η^D if the result is derived by the shallow DNN, i.e., $x_n = l_e + 1$. The accuracy of the task result derived by the full-size DNN is higher than that derived by the shallow DNN, i.e., $\eta^E > \eta^D$.

3) Energy Consumption: The energy consumption for processing the nth DNN task, denoted by $E_n(x_n)$, includes two parts, i.e., the energy consumption for DNN inference and that for task data uploading. The overall energy consumption can be calculated by

$$E_n(x_n) = \kappa^D \left(f_s^D\right)^3 T_n^{\mathrm{lc}} + \kappa^E \left(f_s^E\right)^3 T_n^{\mathrm{ec}} + p^{\mathrm{up}} T_n^{\mathrm{up}} \tag{9}$$

where κ^D and κ^E are the energy efficiency coefficients of the device and the edge server, respectively [19]; p^{up} is the transmit power of the device.

4) Task Utility: We define the utility of the nth DNN task, denoted by $U_n(\mathbf{x}_n)$, as a weighted sum of the overall delay, inference accuracy, and energy consumption of the task

$$U_n(\mathbf{x}_n) = -T_n(\mathbf{x}_n) + \alpha A_n(x_n) - \beta E_n(x_n) \tag{10}$$

where α and β are positive weights for the inference accuracy and energy consumption, respectively.

IV. DT-Assisted Adaptive Device-Edge Collaboration on DNN Inference

In this section, we first introduce the data required to establish the DTs of two processes, i.e., a DT of the on-device DNN inference and a DT of the computing workload evolution. Then, we introduce how the DT of on-device inference can be established to estimate the on-device inference status, and how the DT of computing workload evolution can be established to emulate the on-device workload and edge server workload, respectively. Finally, assisted by the two DTs, an approach to adaptive device-edge collaboration on DNN inference is proposed.

[2]Assuming a high-data transmission rate, the data uploading for a DNN task can be completed within one time slot.

A. Data for Establishing DTs

The following information for establishing the DTs is estimated or collected by the network controller.

1) *Per-Layer On-Device Inference Delay:* The delay for executing each layer of the shallow DNN by the device, i.e., $\{d_l^D, l = 1, 2, \ldots, l_e + 1\}$, is estimated by the network controller. Specifically, the estimation can be based on a) the number of FLOPs required for each layer and the computing capability of the device [29] or b) regression models given the configuration of each layer, such as the input and output data sizes [23].
2) *Task Generation and Arrival:* At the beginning of each time slot, the controller collects information on a) whether a DNN task is generated by the AIoT device at the beginning of each time slot, i.e., $\{I(t), t = 1, 2, \ldots\}$ and b) the workload of computing tasks arriving at the edge server not from the considered device in the time slot, i.e., $\{W(t), t = 1, 2, \ldots\}$.

B. DT of On-Device DNN Inference

For each DNN task, the network controller determines whether to continue the on-device inference before the device executes each layer in the shallow DNN. As a result, the change of on-device DNN inference status, i.e., a layer is about to be locally executed, needs to be known by the network controller. Instead of acquiring the on-device inference status from the device in real-time, which can result in large signaling overhead, the network controller establishes a DT to emulate the on-device inference and estimate the on-device inference status. Denote the index of the time slot right before the on-device execution of the $(l+1)$th layer for the nth DNN task by $t_{n,l} \quad \forall l \in \{0, 1, \ldots, l_e\}$. Then, $t_{n,l}$ can be estimated based on the time instant when the nth DNN task is generated, the on-device queuing delay of the nth DNN task, and the on-device inference delay before executing the $(l+1)$th layer

$$t_{n,l}(\mathbf{x}_{n-1}) = \frac{1}{\Delta T}\left(\left(\sum_{i=0}^{n-1} \Delta T_i\right) + T_n^{\mathrm{lq}}(\mathbf{x}_{n-1}) + \sum_{i=0}^{l} d_i^D\right). \tag{11}$$

In addition, $t_{n,l_e+1}(\mathbf{x}_{n-1})$ is also calculated, which represents the index of the next time slot if the task were completed by device-only inference.

C. DT of Computing Workload Evolution

The evolution of the computing workload on the AIoT device and the edge server affects the average utility of the DNN tasks. To capture the evolution with unknown statistics, a learning algorithm can be applied when determining whether to continue the on-device inference for a DNN task. Such an algorithm can be trained based on the utility of the tasks and the decisions made for the tasks. However, if a decision to offload a task is made, options of continuing the on-device inference for the task cannot be evaluated for training the algorithm. This is because offloading the task would affect the workload evolution, which makes the workload evolution if the on-device inference continued unobservable.

To estimate the on-device and edge server computing workload status under each candidate offloading decision for

a DNN task, the DT of computing workload evolution is established. Specifically, for each task, the DT emulates the workload evolution in a hypothetical case, i.e., if the task processing were completed by the shallow DNN at the device. The time slots to execute the shallow DNN for the nth task are denoted by $t \in \{t_{n,0}, t_{n,0}+1, \ldots, t_{n,l_e+1}-1\}$, and the emulated on-device workload and edge server workload, denoted by $\tilde{Q}^D(t)$ and $\tilde{Q}^E(t)$, respectively, are calculated by

$$\tilde{Q}^D(t) = \begin{cases} Q^D(t), & t = t_{n,0} \\ \tilde{Q}^D(t-1) + I(t), & \text{otherwise} \end{cases} \tag{12a}$$

$$\tilde{Q}^E(t) = \begin{cases} Q^E(t), & t = t_{n,0} \\ \max\{\tilde{Q}^E(t-1) - f^E \Delta T, 0\} + W(t), & \text{otherwise.} \end{cases} \tag{12b}$$

The difference between the actual computing workloads in (1) and (2) and the emulated computing workloads in (12) is that the former are calculated based on the actual offloading decision of a DNN task while the latter are calculated in the hypothetical case that the task processing were locally completed. In such a hypothetical case, the decrease of the computation workload at the device due to the offloading of the task, i.e., $O(t)$ in (1), and the increase of the computation workload at the edge server due to the offloading of the task, i.e., $D(t)$ in (2), both equal zero. Correspondingly, (12) for calculating the emulated workloads in the hypothetical case omits $O(t)$ and $D(t)$, which is the major difference with (1) and (2).

D. Analysis of DTs' Overheads

The data collection and processing for DTs can result in a delay and resource consumption. Nevertheless, such a delay and resource consumption are relatively low. This is because, first, the data collected for DTs as described in Section IV-A have a small size. Second, the processing of the collected data, following (11) and (12), only involves several addition operations. Third, only the recently collected data, i.e., the information on tasks that arrive at the AIoT device and the edge server during the on-device processing of the current DNN task, are required for data processing. As a result, the outdated data, such as the information on tasks, that arrive during the on-device processing of the preceding DNN tasks, can be discarded to minimize the cost of storing the DTs.

E. DT-Assisted Approach to Adaptive Device-Edge Collaboration

As shown in Fig. 3, with the DT of on-device inference and the DT of computing workload evolution, we propose an approach to adaptive device-edge collaboration on DNN inference, which involves the following four steps.

1) *Step 1 (Task Information Gathering):* In the beginning of each time slot, the device sends the DNN task generation indicator $I(t)$ to the controller, which then calculates the gaps of task generations $\{\Delta T_n, n = 0, 1, \ldots, (\sum_t I(t)) - 1\}$. After the controller determines the offloading decision for the $(n-1)$th DNN task, i.e., x_{n-1}, the on-device queuing delay of the nth DNN task, i.e., $T_n^{\text{lq}}(\mathbf{x}_{n-1})$, is calculated. Then, the controller utilizes the DT of on-device inference and (11) to estimate the indices of the time slots when the device would execute each layer of the shallow DNN for the nth task, i.e., $\{t_{n,l}(\mathbf{x}_n)\}_{l=0}^{l_e}$, assuming the task continues to be processed locally.
2) *Step 2 (Learning-Assisted Offloading Decision Making):* After the transmission unit is idle, i.e., $t \geq t_{n,\widehat{x}_n}$, at the beginning of any time slot $t \in \{t_{n,l}(\mathbf{x}_{n-1})\}_{l=\widehat{x}_n}^{l_e}$, leveraging a learning-assisted algorithm, the controller determines whether the device should continue the on-device execution of the $(l+1)$th layer of the shallow DNN or upload the intermediate result to the edge server.
3) *Step 3 (Signaling of Task Offloading):* If the controller determines to stop the on-device inference for the nth DNN task in the beginning of the tth time slot, it sends a signal to the device, corresponding to setting $O(t) = 1$, to the device. The device then uploads the intermediate result to the edge server for executing the remaining layers of the full-size DNN. Without receiving this signal, the device will continue executing the next layer of the shallow DNN.
4) *Step 4 (Training of Learning-Assisted Offloading Algorithm):* After the time slot $t = t_{n,l_e+1}(\mathbf{x}_{n-1})$, the workload evolution if the nth DNN task were locally completed by the shallow DNN, is generated using the DT of computing workload evolution and (12). Then, a learning-assisted offloading algorithm will be trained with the data augmented by the emulated workload evolution.

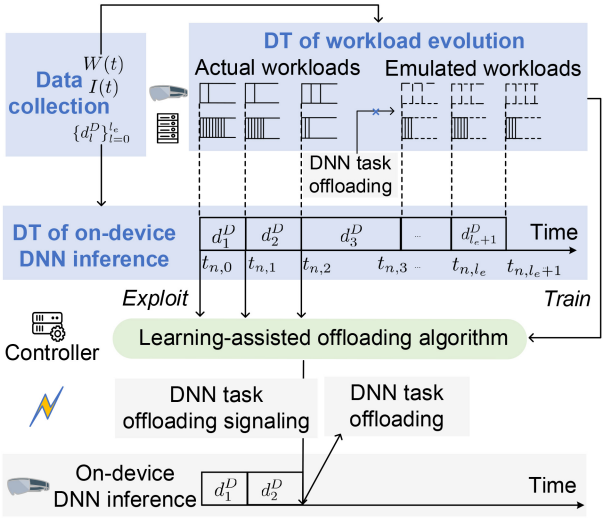


Fig. 3. DT-Assisted approach to adaptive device-edge collaboration on DNN inference.

V. Problem Formulation and Transformation

In this section, we formulate the problem of maximizing the average utility of DNN tasks generated by the considered AIoT device. Due to the stochastic DNN task generation, we transform the problem into multiple subproblems, which are solved sequentially to make the offloading decision for each task given the decisions for its preceding tasks.

A. Problem Formulation

We aim to maximize the average utility of DNN tasks generated at the considered device by optimizing the offloading decisions for the tasks. The problem is given by

$$\text{(P1)}\ \mathbf{max}_{\mathbf{x}_N} \lim_{N\to\infty} \frac{1}{N}\sum_{n=1}^{N} U_n(\mathbf{x}_n) \tag{13a}$$

$$\mathbf{s.t.}\quad 0 \le x_n \le l_e + 1 \quad \forall n \in \{0, 1, \ldots, N\} \tag{13b}$$

$$\left(\sum_{i=m}^{n-1} \Delta T_i\right) + T_n^{\mathrm{lq}}(\mathbf{x}_{n-1}) + T_n^{\mathrm{lc}}(x_n) \ge T_m^{\mathrm{lq}}(\mathbf{x}_{m-1}) + T_m^{\mathrm{lc}}(x_m) + T_m^{\mathrm{up}}(x_m) \quad \forall m \in \{1, \ldots, n-1\}, n \in \{1, 2, \ldots, N\} \tag{13c}$$

where (13c) ensures that the nth DNN task can be offloaded to the edge server only when the offloading of preceding DNN tasks, if any, has been completed and the transmission unit is idle. Note that $T_n^{\mathrm{lc}}(x_n)$ monotonically increases with x_n. As a result, given any $\mathbf{x}_{n-1}$, we can find for the nth task a minimum number of layers in the shallow DNN, denoted by $\widehat{x}_n(\mathbf{x}_{n-1})$, which should be locally executed to satisfy (13c). An equivalent form of (13b) and (13c) is

$$\widehat{x}_n(\mathbf{x}_{n-1}) \le x_n \le l_e + 1 \quad \forall n \in \{0, 1, 2, \ldots, N\}. \tag{14}$$

B. Problem Transformation

Solving (P1) requires the joint optimization of offloading decisions for all the DNN tasks, which is challenging due to the stochastic task generation at the device with unknown statistics [12]. In this section, we first define the long-term utility of one task, which incorporates the main impact of the offloading decision of the task on the other tasks, i.e., the on-device queuing delay of the other tasks resulting from the on-device inference of the task.[3] Then, we transform (P1) into the online optimization of the offloading decision for each DNN task given the offloading decisions of the preceding DNN tasks, with the objective to maximize the expected long-term utility of each DNN task.

1) Long-Term Utility of DNN Task: The on-device queuing delay of a DNN task results from the on-device inference for its preceding tasks. As shown by the red lines in Fig. 4, we first decompose the on-device queuing delay of a task as the summation of the queuing delay resulting from each of its preceding tasks. Then, we define the long-term on-device queuing delay for processing each task, which is the sum on-device queuing delay of subsequent tasks resulting from the on-device processing of the task. For brevity, we omit the parentheses and the offloading decision variables in notations.

First, define $D_{m\to n}^{\mathrm{lq}}$ as the on-device queuing delay of the nth DNN task directly resulting from the on-device inference

[3]Although the edge queuing delay of a DNN task from the considered device can be affected by the offloading decisions of its preceding DNN tasks, such effect is relatively small as the edge queuing delay is affected by the computing tasks offloaded to the edge server from the other devices.

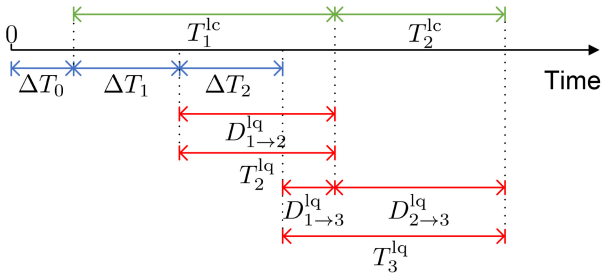


Fig. 4. Decomposition of the on-device queuing delay T_n^{lq}.

of the mth DNN task, calculated by

$$D_{m\to n}^{\mathrm{lq}} = \begin{cases} 0, & m \ge n \\ \max\{\min\{T_m^{\mathrm{lq}} - \sum_{i=m}^{n-1}\Delta T_i, 0\} + T_m^{\mathrm{lc}}, 0\}, & m < n. \end{cases} \tag{15}$$

When $m \ge n$, $D_{m\to n}^{\mathrm{lq}}$ equals zero since tasks leave the on-device task queue following the FCFS rule; when $m < n$, the maximum $D_{m\to n}^{\mathrm{lq}}$ is the on-device inference delay of the mth task, i.e., T_m^{lc} (e.g., $D_{2\to 3}^{\mathrm{lq}} = T_2^{\mathrm{lc}}$ in Fig. 4). Given the offloading decision of the mth DNN task, $D_{m\to n}^{\mathrm{lq}}$ decreases with the increase of the gap between the generation instants of the mth and the nth tasks (e.g., $D_{1\to 3}^{\mathrm{lq}}$ decreases with the increase of ΔT_1 or ΔT_2 in Fig. 4).

Proposition 1: The on-device queuing delay of the nth DNN task is the sum of the on-device queuing delay resulting from all the N DNN tasks generated by the device

$$T_n^{\mathrm{lq}} = \sum_{m=1}^{N} D_{m\to n}^{\mathrm{lq}}. \tag{16}$$

Proof: See Appendix A. ■

Proposition 2: Define the queuing delay of all N DNN tasks due to the on-device inference for the nth DNN task, i.e., $D_n^{\mathrm{lq}} = \sum_{m=1}^{N} D_{n\to m}^{\mathrm{lq}}$, as the long-term on-device queuing delay of the nth DNN task, which can be calculated as follows:

$$D_n^{\mathrm{lq}} = \begin{cases} 0, & T_n^{\mathrm{lc}} = 0 \\ \sum_{t=t_{n,0}}^{t_{n,0}+(T_n^{\mathrm{lc}}/\Delta T)-1} Q^D(t)\Delta T, & \text{otherwise.} \end{cases} \tag{17}$$

Proof: See Appendix B. ■

Equation (17) can be interpreted as follows. First, if a DNN task is directly offloaded to the edge server without any on-device inference, i.e., $T_n^{\mathrm{lc}} = 0$, the on-device queuing delay of the other tasks would not be increased, i.e., $D_n^{\mathrm{lq}} = 0$. Otherwise, if the device is processing the task in the tth time slot, the on-device queuing delay of each task in the on-device task queue will be increased by the time duration of a time slot, i.e., ΔT. As a result, the sum of the increased on-device queuing delay for the tasks in the on-device queue in the tth time slot is $Q^D(t)\Delta T$. The increased queuing delay of the subsequent tasks due to the on-device inference for the nth task is thus the summation term in (17).

By substituting $T_n^{\mathrm{lq}}(x_n)$ in the overall delay (8) with $D_n^{\mathrm{lq}}(x_n)$, we define the time cost of the nth task as

$$C_n(\mathbf{x}_n) = D_n^{\mathrm{lq}}(\mathbf{x}_n) + T_n^{\mathrm{lc}}(x_n) + T_n^{\mathrm{up}}(x_n) + T_n^{\mathrm{eq}}(\mathbf{x}_n) + T^{\mathrm{ec}}(x_n) \tag{18}$$

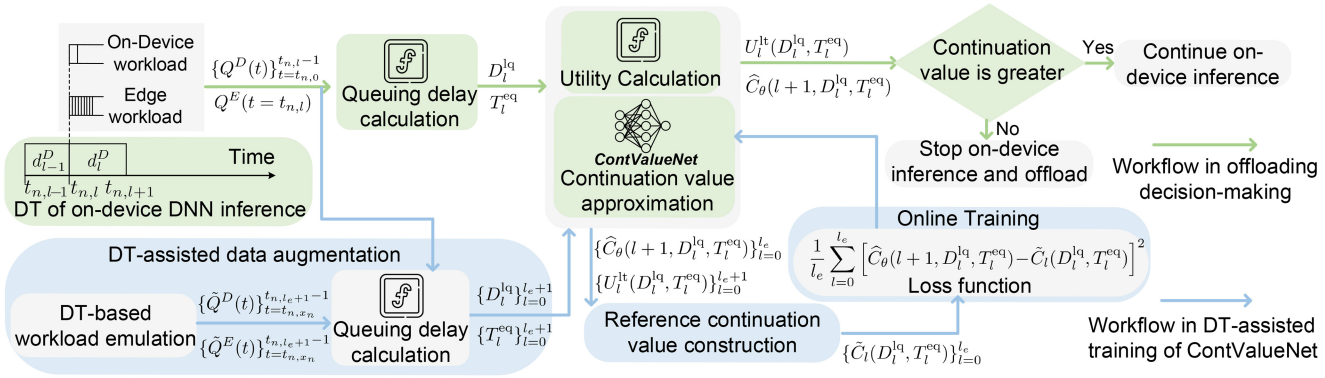


Fig. 5. DT and learning-assisted algorithm for DNN task offloading decision making.

where D_n^{lq} is a function of $\mathbf{x}_n$ since $t_{n,0}$ and T_n^{lc} in (17) are functions of $\mathbf{x}_{n-1}$ and x_n, respectively.

By substituting $T_n(\mathbf{x}_n)$ in (10) with $C_n(\mathbf{x}_n)$, we define the long-term utility of the nth DNN task by

$$U_n^{\mathrm{lt}}(\mathbf{x}_n) = -C_n(\mathbf{x}_n) + \alpha A_n(x_n) - \beta E_n(x_n). \tag{19}$$

Compared with the original utility of a DNN task in (10), which incorporates the on-device queuing delay of the task, the long-term utility in (19) incorporates the on-device queuing delay of subsequent tasks due to the on-device inference for the task. As a result, when sequentially and individually maximizing the long-term utility of each task, the on-device queuing delay of subsequent tasks can be minimized accordingly.

Based on Proposition 1, the sum of the on-device queuing delay of all the N DNN tasks can be calculated by

$$\begin{aligned}\sum_{n=1}^{N} D_n^{\mathrm{lq}} &= \sum_{n=1}^{N}\left(\sum_{m=1}^{N} D_{n\to m}^{\mathrm{lq}}\right)\\ &= \sum_{n=1}^{N}\left(\sum_{m=1}^{N} D_{m\to n}^{\mathrm{lq}}\right)\\ &= \sum_{n=1}^{N} T_n^{\mathrm{lq}}.\end{aligned} \tag{20}$$

Thus, we have

$$\sum_{n=1}^{N} U_n(\mathbf{x}_n) = \sum_{n=1}^{N} U_n^{\mathrm{lt}}(\mathbf{x}_n) \tag{21}$$

which indicates that the average utility of the N DNN tasks, as the objective in (P1), equals to the average long-term utility of the N DNN tasks.

2) Problem Transformation: Due to the stochastic DNN task generation at the considered device and the task arrival at the edge server, we transform (P1) into the sequential and individual maximization of the *expected* long-term utility of a task given the offloading decisions of all preceding tasks

$$\text{(P2)}\ \max_{\hat{x}_n(\mathbf{x}_{n-1})\le x_n\le l_e+1} \mathbb{E}\left[U_n^{\mathrm{lt}}(\mathbf{x}_n)\right]. \tag{22}$$

VI. DT and Learning-Assisted Algorithm for Offloading Decision Making

In this section, to solve the transformed problem (P2), we propose an algorithm for DNN task offloading decision making, assisted by DTs and machine learning techniques as shown in Fig. 5. In this algorithm, we use the optimal stopping theorem and a neural network to make offloading decisions, and propose DT-assisted training for the neural network.

For brevity, we denote $D_n^{\mathrm{lq}}(x_n = l|\mathbf{x}_{n-1})$, $T_n^{\mathrm{eq}}(x_n = l|\mathbf{x}_{n-1})$ and $U_n^{\mathrm{lt}}(x_n = l|\mathbf{x}_{n-1})$ by D_l^{lq}, T_l^{eq} and U_l^{lt}, which, respectively, represent the long-term on-device queuing delay, the edge queuing delay and the long-term task utility if the offloading decision of the nth DNN task is l, i.e., $x_n = l$, given that the offloading decisions of the first $n-1$ DNN tasks are $\mathbf{x}_{n-1}$. To emphasize the impact of the long-term on-device queuing delay and the edge queuing delay on the long-term utility of a DNN task, we represent U_l^{lt} as $U_l^{\mathrm{lt}}(D_l^{\mathrm{lq}}, T_l^{\mathrm{eq}})$. In addition, we define $\mathbf{D}_l^{\mathrm{lq}} = \{D_0^{\mathrm{lq}}, D_1^{\mathrm{lq}}, \ldots, D_l^{\mathrm{lq}}\}$ and $\mathbf{T}_l^{\mathrm{eq}} = \{T_0^{\mathrm{eq}}, T_1^{\mathrm{eq}}, \ldots, T_l^{\mathrm{eq}}\}$.

A. DT and Learning-Assisted Offloading Decision Making

1) Continuation Value-Based Offloading Decision Making: Define $V_l(\mathbf{D}_l^{\mathrm{lq}}, \mathbf{T}_l^{\mathrm{eq}})$ as the *maximum* expected long-term utility of a DNN task when l layers of the shallow DNN have been executed for the task, where $l \in \{0, 1, \ldots, l_e+1\}$. Such a value satisfies a recursion rule

$$V_l\left(\mathbf{D}_l^{\mathrm{lq}}, \mathbf{T}_l^{\mathrm{eq}}\right) = \begin{cases} \max\left\{U_l^{\mathrm{lt}}\left(D_l^{\mathrm{lq}}, T_l^{\mathrm{eq}}\right), C_l\left(\mathbf{D}_l^{\mathrm{lq}}, \mathbf{T}_l^{\mathrm{eq}}\right)\right\}, & l \le l_e \\ U_{l_e+1}^{\mathrm{lt}}\left(D_{l_e+1}^{\mathrm{lq}}, T_{l_e+1}^{\mathrm{eq}}\right), & l = l_e+1 \end{cases} \tag{23}$$

where $\forall l \in \{0, 1, \ldots, l_e\}$, we have

$$C_l\left(\mathbf{D}_l^{\mathrm{lq}}, \mathbf{T}_l^{\mathrm{eq}}\right) = \mathbb{E}\left[V_{l+1}\left(\mathbf{D}_{l+1}^{\mathrm{lq}}, \mathbf{T}_{l+1}^{\mathrm{eq}}\right)|\mathbf{D}_l^{\mathrm{lq}}, \mathbf{T}_l^{\mathrm{eq}}\right] \tag{24}$$

referred to as the *continuation value* [30].

Proposition 3: Based on optimal stopping [31], the optimal offloading decision for maximizing the expected long-term utility of the nth DNN task is

$$x_n = \begin{cases} \min\{\Psi\}, & \text{if } \Psi \neq \emptyset \\ l_e+1, & \text{otherwise} \end{cases} \tag{25a}$$

where

$$\Psi = \left\{ l \mid \widehat{x}_n \le l \le l_e, U_l^{\mathrm{lt}}\left(D_l^{\mathrm{lq}}, T_l^{\mathrm{eq}}\right) \ge C_l\left(\mathbf{D}_l^{\mathrm{lq}}, \mathbf{T}_l^{\mathrm{eq}}\right) \right\}. \quad (25b)$$

The continuation value-based offloading decision making given in (25) is explained as follows. For the nth DNN task, after the device executes l layers of the shallow DNN, where $l \in \{\widehat{x}_n, \widehat{x}_n + 1, \ldots, l_e\}$, if the long-term utility of offloading the DNN task now, i.e., $U_l^{\mathrm{lt}}(D_l^{\mathrm{lq}}, T_l^{\mathrm{eq}})$, is no less than the continuation value, i.e., $C_l(\mathbf{D}_l^{\mathrm{lq}}, \mathbf{T}_l^{\mathrm{eq}})$, the on-device inference for the task should be stopped for offloading the task to the edge server. Otherwise, the device should continue executing the $(l+1)$th layer of the shallow DNN.

2) DT and Learning-Assisted Offloading Decision Making: The continuation value for each layer of the shallow DNN can be obtained using backward induction [31], which however requires the prior statistics of the workload evolution and introduces computing overhead [32]. Inspired by [30], our approach is to construct a neural network referred to as ContValueNet to generate an approximated continuation value $\widehat{C}_\theta(l+1, D_l^{\mathrm{lq}}, T_l^{\mathrm{eq}})$ given the input $\{l+1, D_l^{\mathrm{lq}}, T_l^{\mathrm{eq}}\}$, where θ represents the parameters in ContValueNet. By substituting the continuation value in (25b) with the approximated continuation value, a learning-assisted offloading algorithm is obtained. Specifically, as shown in the green blocks in Fig. 5, through the DT of on-device DNN inference given in (11), the controller is informed when a layer of the shallow DNN is about to be executed at the device. Before the on-device execution of the $(l+1)$th layer by the device, the controller calculates 1) the queuing delay D_l^{lq} and T_l^{eq}; 2) calculates the long-term utility of the task $U_l^{\mathrm{lt}}(D_l^{\mathrm{lq}}, T_l^{\mathrm{eq}})$; and 3) approximates the continuation value by the ContValueNet, i.e., $\widehat{C}_\theta(l+1, D_l^{\mathrm{lq}}, T_l^{\mathrm{eq}})$. If the continuation value is greater than the utility, the controller lets the device execute the next layer for the task. Otherwise, the network controller lets the device upload the input data of the $(l+1)$th layer in the full-size DNN to the edge server.

B. DT-Assisted Training of ContValueNet

As shown in the blue blocks in Fig. 5, DTs are leveraged to assist in training ContValueNet, which involves three steps, i.e., DT-assisted data augmentation, reference continuation value construction, and online ContValueNet training.

1) DT-Assisted Data Augmentation: If the nth DNN task is determined to offload to the edge server, i.e., $x_n \le l_e$, the controller emulates the long-term on-device queuing delay and the edge queuing delay given the remaining potential offloading decisions for the task, i.e., $\{D_l^{\mathrm{lq}}\}_{l=x_n+1}^{l_e+1}$ and $\{T_l^{\mathrm{eq}}\}_{l=x_n+1}^{l_e+1}$, using the DT of workload evolution and (12). Specifically, $\{D_l^{\mathrm{lq}}\}_{l=x_n+1}^{l_e}$ are calculated by substituting $Q^D(t)$ in (17) with $\tilde{Q}^D(t)$ in (12a), and $\{T_l^{\mathrm{eq}}\}_{l=x_n+1}^{l_e+1}$ are calculated by substituting the on-device workload in (6) with the emulated on-device workload in (12b). Then, the controller calculates the approximated continuation values $\{\widehat{C}_\theta(l+1, D_l^{\mathrm{lq}}, T_l^{\mathrm{eq}})\}_{l=x_n+1}^{l_e+1}$ using ContValueNet. Combining the above values with $\{D_l^{\mathrm{lq}}\}_{l=0}^{x_n}$, $\{T_l^{\mathrm{eq}}\}_{l=0}^{x_n}$ and $\{\widehat{C}_\theta(l+1, D_l^{\mathrm{lq}}, T_l^{\mathrm{eq}})\}_{l=0}^{x_n}$ which are collected during the decision making for the nth DNN task, the controller obtains $\{D_l^{\mathrm{lq}}\}_{l=0}^{l_e+1}$, $\{T_l^{\mathrm{eq}}\}_{l=0}^{l_e+1}$ and $\{\widehat{C}_\theta(l+1, D_l^{\mathrm{lq}}, T_l^{\mathrm{eq}})\}_{l=0}^{l_e+1}$, which will be used to construct the reference continuation values.

2) Reference Continuation Value Construction: To optimize the parameters θ of ContValueNet, reference continuation values of evaluating the approximation error of ContValueNet should be constructed [30].

First, for any $l \in \{0, 1, 2, \ldots, l_e\}$, we calculate $\widehat{V}_l(D_l^{\mathrm{lq}}, T_l^{\mathrm{eq}})$ by substituting the continuation value in (23) with the approximated continuation value

$$\widehat{V}_l\left(D_l^{\mathrm{lq}}, T_l^{\mathrm{eq}}\right) = \max\left\{U_l^{\mathrm{lt}}\left(D_l^{\mathrm{lq}}, T_l^{\mathrm{eq}}\right), \widehat{C}_\theta\left(l+1, D_l^{\mathrm{lq}}, T_l^{\mathrm{eq}}\right)\right\}. \quad (26)$$

Then, the reference continuation value can be obtained according to (24)

$$\tilde{C}_l\left(D_l^{\mathrm{lq}}, T_l^{\mathrm{eq}}\right) = \mathbb{E}\left[\widehat{V}_{l+1}\left(D_{l+1}^{\mathrm{lq}}, T_{l+1}^{\mathrm{eq}}\right) | \mathbf{D}_l^{\mathrm{lq}}, \mathbf{T}_l^{\mathrm{eq}}\right]. \quad (27)$$

Due to the unknown statistics, we use the single-sample estimation method [33] to approximate the expectation term in (27)

$$\mathbb{E}\left[\widehat{V}_{l+1}\left(D_{l+1}^{\mathrm{lq}}, T_{l+1}^{\mathrm{eq}}\right) | \mathbf{D}_l^{\mathrm{lq}}, \mathbf{T}_l^{\mathrm{eq}}\right] \approx \widehat{V}_{l+1}\left(D_{l+1}^{\mathrm{lq}}, T_{l+1}^{\mathrm{eq}}\right). \quad (28)$$

Finally, by combining (27) and (28), and calculating $\widehat{V}_{l+1}(D_{l+1}^{\mathrm{lq}}, T_{l+1}^{\mathrm{eq}})$ in (28) according to (26), the reference continuation value is obtained by

$$\tilde{C}_l\left(D_l^{\mathrm{lq}}, T_l^{\mathrm{eq}}\right) = \max\left\{U_{l+1}^{\mathrm{lt}}\left(D_{l+1}^{\mathrm{lq}}, T_{l+1}^{\mathrm{eq}}\right), \widehat{C}_\theta\left(l+2, D_{l+1}^{\mathrm{lq}}, T_{l+1}^{\mathrm{eq}}\right)\right\}. \quad (29)$$

Remark 1: To calculate the reference continuation value $\tilde{C}_l(D_l^{\mathrm{lq}}, T_l^{\mathrm{eq}})$ in (29), D_{l+1}^{lq} and T_{l+1}^{eq} are required. As a result, with $\{D_l^{\mathrm{lq}}\}_{l=0}^{x_n}$ and $\{T_l^{\mathrm{eq}}\}_{l=0}^{x_n}$ collected in the offloading decision making for the nth task, reference continuation values before executing each of the first x_n layers of the shallow DNN can be obtained, where $x_n \le l_e + 1$. In contrast, with DT-assisted data augmentation, which generates $\{D_l^{\mathrm{lq}}\}_{l=x_n+1}^{l_e+1}$ and $\{T_l^{\mathrm{eq}}\}_{l=x_n+1}^{l_e+1}$, reference continuation values before executing each of the l_e+1 layers in the shallow DNN can be obtained. In this way, the data for training ContValueNet is augmented.

3) Online ContValueNet Training: To mitigate the impact of the one-sample estimation method in (28), ContValueNet is trained in an online manner with a carefully designed loss function. Specifically, the loss function for training ContValueNet after processing the first $\widehat{n}$ DNN tasks, where $1 \le \widehat{n} \le M$, is denoted by $L^{\mathrm{NN}}(\theta, \widehat{n})$ and defined by

$$L^{\mathrm{NN}}(\theta, \widehat{n}) = \frac{1}{\widehat{n}(l_e+1)} \sum_{n=0}^{\widehat{n}} \sum_{l=0}^{l_e} \left[\widehat{C}_\theta\left(l+1, D_l^{\mathrm{lq}}, T_l^{\mathrm{eq}}\right) - \tilde{C}_l\left(D_l^{\mathrm{lq}}, T_l^{\mathrm{eq}}\right)\right]^2 \quad (30)$$

which is the mean squared error of the continuation value approximation. Then, the gradient descent method is applied to optimize the parameters θ in ContValueNet to minimize the loss function. Specifically, the update rule of θ is

$$\theta' = \theta - \gamma \nabla_\theta L^{\mathrm{NN}}(\theta, \widehat{n}) \quad (31)$$

where γ is the learning rate; $\nabla_\theta L^{\mathrm{NN}}(\theta, \widehat{n})$ is the first-order derivative of the loss function with respect to θ.

VII. Offloading Decision Space Reduction

The complexity of the approach to adaptive device-edge collaboration on DNN inference in Section VI-A can be high due to the potentially large number of layers in the DNN. In this section, from two aspects, we will derive necessary conditions for an offloading decision to be optimal in maximizing the expected long-term utility. Correspondingly, we investigate decision space reduction to reduce the complexity.

A. DNN Layers With Negligible Execution Time

The execution time of some layers in a DNN, e.g., pooling layers in a convolutional neural network (CNN), is negligible due to relatively simple operations [34]. As a result, such a layer has a negligible impact on the long-term on-device queuing delay and the edge queuing delay of a DNN task. In contrast, the data size before and after the execution of such layers can be drastically changed. For example, a max-pooling layer downsamples input data to learn spatially robust features for image recognition tasks, while a max-unpooling layer upsamples the input data to restore the original data size for semantic segmentation tasks.

Remark 2: If a layer in a DNN has negligible execution time and outputs data with reduced (increased) size, offloading the DNN task before (after) the on-device execution of the layer cannot be optimal for maximizing the expected long-term utility of the DNN task. This is because offloading a DNN task after the data size is increased or before the data size is reduced would result in a higher data uploading delay. Based on this necessary condition, we can treat each max-pooling layer and its preceding layer as one logical layer and each max-unpooling layer and its succeeding layer as one logical layer in the offloading decision making for a DNN task.

B. Properties of Workload Evolution

Based on the properties in the evolution of the on-device workload and the edge server workload over time, properties in the evolution of long-term on-device queuing delay and edge queuing delay (as the on-device inference continues) can be derived, respectively.

Property 1: During the on-device inference for a DNN task, the on-device workload $Q^D(t)$ is nondecreasing with t. In addition, the on-device inference delay T_n^{lc} increases with x_n. As a result, when x_n increases, the long-term on-device queuing delay $D_n^{\mathrm{lq}}(x_n)$, calculated by (17), is nondecreasing with x_n. The increase of $D_n^{\mathrm{lq}}(x_n)$ is minimized if no DNN tasks are generated after the controller starts to make the offloading decision for the nth DNN task, i.e., $Q^D(t) = Q^D(t_{n,\widehat{x}_n}) \quad \forall t \geq t_{n,\widehat{x}_n}$. The minimum increase is equal to the product of $Q^D(t_{n,\widehat{x}_n})$ and the increase of on-device inference delay $T_n^{\mathrm{lc}}(x_n)$.

Property 2: The edge server workload can decrease or increase during the extended on-device inference for the nth task due to the increase of x_n. The decrease of the workload is maximized if no tasks arrive at the edge server during the extended on-device inference, and the maximum decrease is equal to the workload that the edge server can process during the extended on-device inference. As a result, given the increase of x_n, the maximum decrease of the edge queuing delay $T_n^{\mathrm{eq}}(x_n)$, calculated by dividing the edge server workload by the edge server processing capability, equals to the increase of the on-device inference delay $T_n^{\mathrm{lc}}(x_n)$.

Using the above properties, a necessary condition for an offloading decision $x_n \in \{\widehat{x}_n, \ldots, l_e\}$ to be optimal can be derived.

Lemma 1: Define a deterministic part in the long-term utility of the nth DNN task as $U_n^{\mathrm{pt}}(x_n) \overset{def}{=} -T_n^{\mathrm{up}}(x_n) - T_n^{\mathrm{ec}}(x_n) - \beta E_n(x_n)$. If $x_n^* \leq l_e$ maximizes $\mathbb{E}[U_n^{\mathrm{lt}}(x_n)]$, then for any $x_n \in \{\widehat{x}_n, \ldots, x_n^*\}$, it holds

$$U_n^{\mathrm{pt}}(x_n^*) \geq U_n^{\mathrm{pt}}(x_n) + Q^D\left(t = t_{n,\widehat{x}_n}\right)\left(T_n^{\mathrm{lc}}(x_n^*) - T_n^{\mathrm{lc}}(x_n)\right) \quad (32)$$

where $t_{n,\widehat{x}_n}$ is the index of the time slot from which the nth DNN task can be offloaded to the edge server.

Proof: Note that $U_n^{\mathrm{lt}}(x_n) = U_n^{\mathrm{pt}}(x_n) - T_n^{\mathrm{lc}}(x_n) - D_n^{\mathrm{lq}}(x_n) - T_n^{\mathrm{eq}}(x_n) + \alpha A_n(x_n)$. If $x_n^* \leq l_e$ maximizes $\mathbb{E}[U_n^{\mathrm{lt}}(x_n)]$, then for any $x_n \in \{\widehat{x}_n, \ldots, x_n^*\}$, we have

$$\begin{aligned} &U_n^{\mathrm{pt}}(x_n^*) - T_n^{\mathrm{lc}}(x_n^*) - \mathbb{E}\left[D_n^{\mathrm{lq}}(x_n^*)\right] - \mathbb{E}\left[T_n^{\mathrm{eq}}(x_n^*)\right] + \alpha A_n(x_n^*) \\ &\geq U_n^{\mathrm{pt}}(x_n) - T_n^{\mathrm{lc}}(x_n) - \mathbb{E}\left[D_n^{\mathrm{lq}}(x_n)\right] - \mathbb{E}\left[T_n^{\mathrm{eq}}(x_n)\right] + \alpha A_n(x_n). \end{aligned} \quad (33)$$

When $x_n \leq l_e$ (including the case when $x_n^* \leq l_e$), the nth DNN task is offloaded to the edge server for processing by the full-size DNN. The inference accuracy is thus the same under the two cases

$$A_n(x_n^*) = A_n(x_n). \quad (34)$$

According to Property 1, we have

$$D_n^{\mathrm{lq}}(x_n^*) \geq D_n^{\mathrm{lq}}(x_n) + Q^D\left(t = t_{n,\widehat{x}_n}\right)\left(T_n^{\mathrm{lc}}(x_n^*) - T_n^{\mathrm{lc}}(x_n)\right). \quad (35)$$

According to Property 2, we have

$$T_n^{\mathrm{lc}}(x_n^*) - T_n^{\mathrm{lc}}(x_n) \geq T_n^{\mathrm{eq}}(x_n) - T_n^{\mathrm{eq}}(x_n^*). \quad (36)$$

Combining (33)–(35), and (36), (32) is obtained. ■

In addition, we derive a necessary condition of the device-only inference, i.e., $x_n = l_e + 1$, to be optimal in maximizing the *long-term utility* $U_n^{\mathrm{lt}}(x_n)$.[4]

Lemma 2: If $x_n = l_e + 1$ maximizes the long-term task utility $U_n^{\mathrm{lt}}(x_n)$, then

$$\begin{aligned} U_n(l_e + 1) \geq\ & U_n(\widehat{x}_n) \\ & + Q^D\left(t = t_{n,\widehat{x}_n}\right)\left(T_n^{\mathrm{lc}}(l_e + 1) - T_n^{\mathrm{lc}}(\widehat{x}_n)\right). \end{aligned} \quad (37)$$

Proof: Since $x_n = l_e + 1$ maximizes the long-term task utility $U_n^{\mathrm{lt}}(x_n)$, we have

$$U_n^{\mathrm{lt}}(l_e + 1) \geq U_n^{\mathrm{lt}}(\widehat{x}_n). \quad (38)$$

[4]The long-term utility of a task by device-only inference would not be affected by the dynamic workload at the edge server, while the utility by the other inference scenarios would be affected. Due to the unknown statistics of the dynamic workload, we cannot derive any necessary condition for device-only inference to be optimal in maximizing the *expected* long-term utility.

Algorithm 1: Decision Space Reduction

Input: $\widehat{x}_n$;
Output: $\mathcal{L}_n$;
Initialization: $\mathcal{L}_n = \{\widehat{x}_n, \widehat{x}_n + 1, \ldots, l_e + 1\}$;
for $l \in \{\widehat{x}_n, \widehat{x}_n + 1, \ldots, l_e\}$ **do**
  Given $x_n^* = l$, for any $x_n \in \{\widehat{x}_n, \ldots, x_n^*\}$, if (32) is violated, delete l in the set $\mathcal{L}_n$;
end
if $\mathcal{L}_n = \{\widehat{x}_n, l_e + 1\}$ **then**
  if (37) is violated **then**
    delete $l_e + 1$ in the set $\mathcal{L}_n$;
  end
end
return $\mathcal{L}_n$

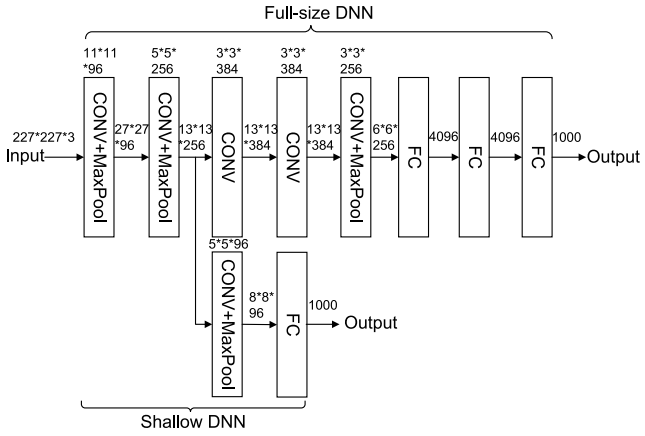


Fig. 6. Full-size DNN and shallow DNN.

According to Property 1, we have

$$D_n^{\mathrm{lq}}(l_e+1) \geq D_n^{\mathrm{lq}}(\widehat{x}_n) + Q^D\big(t = t_{n,\widehat{x}_n}\big)\Big(T_n^{\mathrm{lc}}(l_e+1) - T_n^{\mathrm{lc}}(\widehat{x}_n)\Big). \quad (39)$$

Note that

$$U_n^{\mathrm{lt}}(l_e+1) + D_n^{\mathrm{lq}}(l_e+1) = U_n(l_e+1) + T_n^{\mathrm{lq}}(\mathbf{x}_{n-1}) \quad (40a)$$

$$U_n^{\mathrm{lt}}(\widehat{x}_n) + D_n^{\mathrm{lq}}(\widehat{x}_n) = U_n(\widehat{x}_n) + T_n^{\mathrm{lq}}(\mathbf{x}_{n-1}). \quad (40b)$$

Combining (38) and (39), (37) is obtained. ■

Based on Lemmas 1 and 2, an algorithm for reducing the offloading decision space is proposed in Algorithm 1. Specifically, when the nth DNN task can be offloaded to the edge server, the potential offloading decisions $x_n \in \{\widehat{x}_n, \widehat{x}_n + 1, \ldots, l_e\}$ will be checked, and only the decisions satisfying the condition given in Lemma 1 will be considered in the learning-assisted offloading decision making. In addition, if all the possible offloading decisions $x_n \in \{\widehat{x}_n + 1, \ldots, l_e\}$ violate the necessary condition given in Lemma 1, the condition in Lemma 2 will be checked to determine whether device-only inference, i.e., $x_n = l_e + 1$, can be optimal for maximizing the long-term task utility. If not, $x_n = l_e + 1$ will not be considered. Finally, only the remaining decisions, i.e, $\mathcal{L}_n$, will be considered.

VIII. Simulation Results

In this section, through simulations, we evaluate the performance of the DT-assisted approach to adaptive device-edge collaboration on DNN inference.

A. Simulation Settings and Benchmarks

The main parameters used in the simulations are given in Table I, and the detailed configurations of the full-size and shallow DNNs are shown in Fig. 6, where Alexnet is chosen as the full-size DNN. Based on Remark 2, we consider each pooling layer and its preceding layer as one logical layer. The execution time for the layers of the shallow DNN by the device, i.e., $\{d_l^D\}_{l=1}^{l_e+1}$, and that for the layers of the full-size DNN by the edge server, i.e., $\{d_l^E\}_{l=1}^{L}$, are estimated based on the number of FLOPs for the layers and the computation frequency of the AIoT device and the edge server [29].

TABLE I
Simulation Parameters

Parameters	Symbol	Value
Time duration of a time slot	ΔT	10 ms
Exit layer index	l_e	2
Computation frequency of the edge server	f^E	50 GHz
Computation frequency of the AIoT device	f^D	1 GHz
Accuracy of full-size DNN	η^E	0.9
Accuracy of shallow DNN	η^D	0.6
Uplink transmission rate	R_0	126 Mbps
Transmit power of the AIoT device	p^{up}	20 dBm
Energy coefficient	κ^E, κ^D	10^{-30}
Weight for accuracy	α	1.0
Weight for energy consumption	β	0.2

The neural network for approximating the continuation values, i.e., ContValueNet, consists of three fully connected layers with 200, 100, and 20 neurons, respectively. For optimizing the parameters therein, the learning rate γ is set to be 1×10^{-3} and the Adam optimizer is chosen. In the processing of the first 2000 DNN tasks, ContValueNet for continuation value approximation is trained, i.e., $M = 2000$. The average DNN task utility, inference delay, accuracy and energy consumption are derived by applying the trained ContValueNet to assist making offloading decisions of 8000 DNN tasks. The DNN task generation at the considered device follows a Bernoulli distribution with probability p and task arrival from the other devices to the edge server follows a Poisson distribution with arrival rate λ. The CPU cycles required for processing a computing task follows uniform distribution $U(0, U^{\max})$, with the value of $U^{\max}$ set as 8×10^9. The DNN task generation rate in the unit of tasks per second can be calculated by p/Δ. In addition, the edge processing load can be calculated by $\lambda U^{\max}/2f^E$. As a unitless ratio, the edge processing load is not tied to a specific system parameter, such as computing capability, task arrival rate, and task computation cost. As a result, using this metric allows our simulation results to offer findings that are applicable across various AIoT network settings.

In the simulations, the following three benchmarks are used.

1) *One-Time Ideal Case:* The offloading decision of a DNN task is made only once (upon task generation) to maximize the long-term utility of the task, assuming

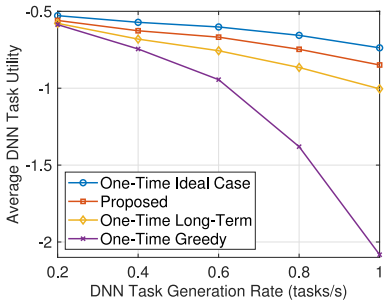


Fig. 7. Average DNN task utility versus task generation rate.

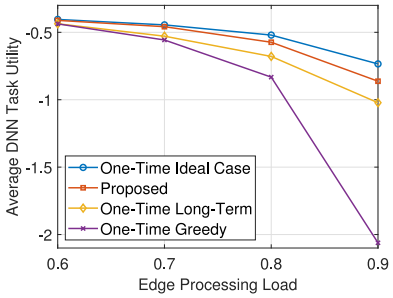


Fig. 8. Average DNN task utility versus edge processing load.

perfect knowledge of the future on-device workload and edge server workload.

2) *One-Time Long-term:* The offloading decision of a DNN task is made only once (upon task generation) to maximize the long-term utility of the DNN task based on the current on-device workload and edge server workload.
3) *One-Time Greedy:* Similar to [6], the offloading decision of a DNN task is made only once (upon task generation) to maximize the utility of the DNN task based on the current on-device workload and edge server workload.

In addition, we will compare the performance of the learning-assisted task offloading algorithm 1) with and without the DT-assisted training data augmentation and 2) with and without the offloading decision space reduction, respectively.

B. Adaptiveness to Dynamic Computing Workload

In Fig. 7, the average utility of DNN tasks under varying DNN task generation rate is shown, where the edge processing load is 0.9. It can be observed that the proposed approach outperforms the one-time greedy benchmark. This is because the proposed approach makes the offloading decision of a DNN task while considering the on-device queuing delay of subsequent DNN tasks. In addition, the proposed approach outperforms the one-time long-term benchmark, since instead of making task offloading decision upon the task generation, our approach continuously monitors the on-device workload during the on-device inference for each task and adaptively makes the offloading decision.

In Fig. 7, it can be observed that the performance gain compared to the one-time long-term benchmark increases with DNN task generation rate. This is because the increase of the task generation rate results from the increase of the task generation probability, which in turn increases the variance of the Bernoulli distribution and the dynamics of the on-device workload. Since the performance gain comes from better adaptiveness to the dynamics, it increases with the dynamics. In Fig. 8, the average DNN task utility under varying edge processing load is shown, where the DNN task generation rate is 1.0. Similarly, the proposed approach outperforms the one-time long-term and one-time greedy benchmarks and the performance gain increases with the edge processing load.

In Fig. 9, the average inference delay, accuracy and energy consumption under varying DNN task generation rate is

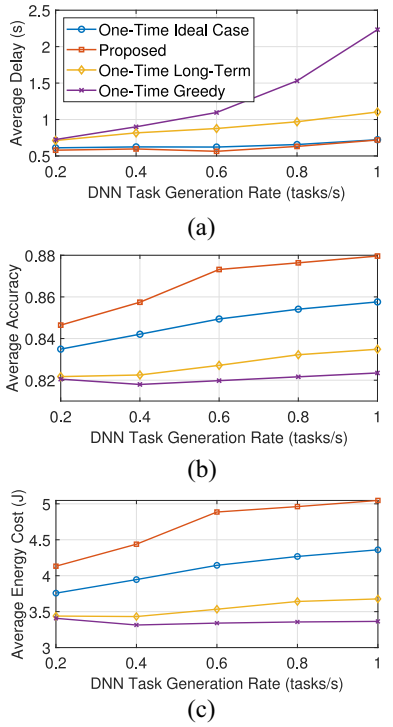


Fig. 9. (a) Average delay. (b) Average accuracy. (c) Average energy consumption versus DNN task generation rate.

shown, where the edge processing load is 0.9. Compared with the benchmarks, the proposed approach achieves lower delay and higher inference accuracy at the cost of higher energy consumption. This results from the weights for the delay, inference accuracy and the energy consumption in the task utility, which are set as 1.0, 1.0, and 0.002, respectively. Combining the ranges of the three metrics shown in Fig. 9, it can be seen that the delay and the accuracy contribute significantly more than the energy consumption in determining the task utility. Since the proposed approach does not assume perfect knowledge of the workload evolution, it can conservatively offload a task to the edge server for reducing the on-device queuing delay of the tasks in the on-device queue and increasing the inference accuracy.

C. Effectiveness of DT-Assisted Training Data Augmentation

In Fig. 10, the number of training samples collected during the training of ContValueNet is shown, where the edge

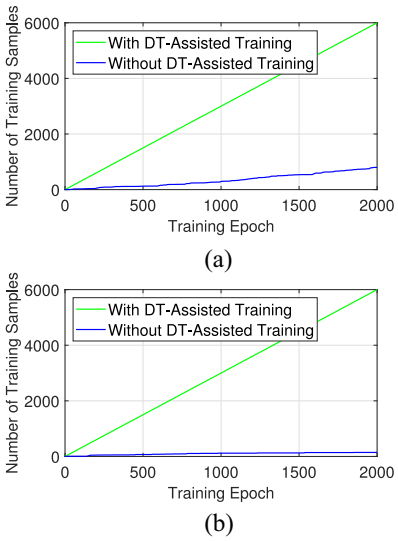


Fig. 10. Number of collected training samples with and without DT-assisted training data augmentation, where the DNN task generation rates are (a) 0.4 and (b) 0.8.

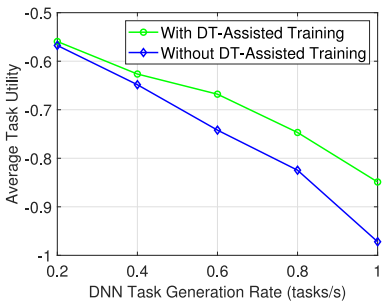


Fig. 11. Average DNN task utility with and without DT-assisted training data augmentation.

processing load is 0.9, and the DNN task generation rate is 0.4 and 0.8 in Fig. 10(a) and (b), respectively. It can be observed that with DT-assisted training data augmentation, the number of training samples linearly grows as the training continues. This is because the DT of workload evolution is used by the network controller to emulate the on-device and edge server workload under every possible offloading scenario so that, for each DNN task, we can obtain l_e+1 training samples, where l_e is 2 based on the configuration of the shallow DNN in Fig. 6. In contrast, without DT-assisted training data augmentation, very limited training samples can be obtained since only the workload evolution before the offloading of a DNN task can be used to construct training samples.

In Fig. 11, the average DNN task utility versus the task generation rate, with and without DT-assisted training data augmentation, is shown, where the edge processing load is 0.9. With DT-assisted training data augmentation, the average task utility increases. In addition, as the task generation rate increases, the performance gain increases. This is because the dynamics of the on-device workload increase with the task generation rate, which in turn requires a larger amount of and more diverse training data. The performance gain due to the DT-assisted data augmentation can also be

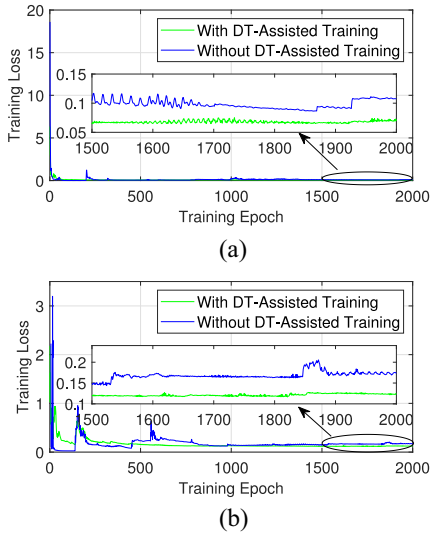


Fig. 12. Training loss with and without DT-assisted training data augmentation, where the DNN task generation rates are (a) 0.4 and (b) 0.8.

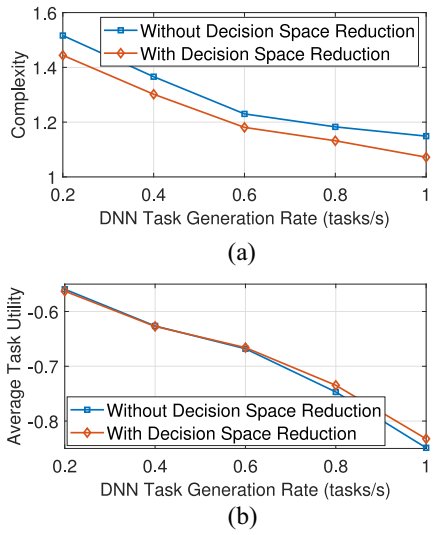


Fig. 13. (a) Complexity and (b) average DNN task utility with and without decision space reduction.

explained by analyzing the training loss during the training of ContValueNet. In Fig. 12, with the same simulation settings as those used to generate Fig. 10, we can observe that without DT-assisted data augmentation, the training loss fluctuates more due to overfitting caused by insufficient training data. In contrast, with DT-assisted data augmentation, the training loss decreases more steadily to a lower level at the end of the training.

D. Complexity Reduction by Decision Space Reduction

In Fig. 13, the performance of the proposed device-edge collaborative DNN inference with and without decision space reduction in Algorithm 1 is shown, where the edge processing load is 0.9. As shown in Fig. 13(a), with the decision space reduction, the proposed approach can be implemented with reduced complexity in terms of the average number of times

$$
\begin{aligned}
D_n^{\mathrm{lq}} &= \sum_{m=1}^{N} D_{n\to m}^{\mathrm{lq}} \\
&= q_0 T_n^{\mathrm{lc}} + \sum_{m=n+q_0+1}^{n+q_1} \left(T_n^{\mathrm{lq}} + T_n^{\mathrm{lc}} - \sum_{i=n}^{m-1} \Delta T_i \right) \\
&= (q_1 - q_0) T_n^{\mathrm{lq}} + q_1 T_n^{\mathrm{lc}} - (q_1 - q_0) \sum_{i=n}^{n+q_0} \Delta T_i - \sum_{i=q_0+n+1}^{q_1+n-1} (q_1 - i + n) \Delta T_i
\end{aligned} \tag{44}
$$

$$
\begin{aligned}
\sum_{t=t_{n,0}}^{t_{n,0}+\left(T_n^{\mathrm{lc}}/\Delta T\right)-1} Q^D(t)\Delta T &= q_0 \left(-t_{n,0}\Delta T + \sum_{i=1}^{n+q_0} \Delta T_i \right) + \sum_{m=q_0+n+1}^{q_1+n-1} (m-n)\Delta T_m + q_1 \left(t_{n,0}\Delta T + T_n^{\mathrm{lc}} - \sum_{i=1}^{n+q_1-1} \Delta T_i \right) \\
&= q_0 \left(-T_n^{\mathrm{lq}} + \sum_{i=n}^{n+q_0} \Delta T_i \right) + \sum_{m=q_0+n+1}^{q_1+n-1} (m-n)\Delta T_m + q_1 \left(T_n^{\mathrm{lq}} + T_n^{\mathrm{lc}} - \sum_{i=n}^{n+q_1-1} \Delta T_i \right) \\
&= (q_1 - q_0) T_n^{\mathrm{lq}} + q_1 T_n^{\mathrm{lc}} - (q_1 - q_0) \sum_{i=n}^{n+q_0} \Delta T_i + \left(\sum_{m=q_0+n+1}^{q_1+n-1} (m-n)\Delta T_m - q_1 \sum_{i=q_0+n+1}^{q_1+n-1} \Delta T_i \right) \\
&= (q_1 - q_0) T_n^{\mathrm{lq}} + q_1 T_n^{\mathrm{lc}} - (q_1 - q_0) \sum_{i=n}^{n+q_0} \Delta T_i - \sum_{i=q_0+n+1}^{q_1+n-1} (q_1 - i + n) \Delta T_i
\end{aligned} \tag{45}
$$

when the controller determines whether to continue on-device inference for a DNN task based on (25). This is because with the decision space reduction, the on-device DNN inference continues if the corresponding offloading decision violates the necessary condition given in Lemma 1. In addition, in Fig. 13(b), it can be observed that with the decision space reduction, the average task utility is nearly unaffected and even improved in the high regime of DNN task generation rate. This is because the necessary conditions sometimes prevent the controller from choosing nonoptimal offloading decisions that could have been chosen due to the imperfect continuation value approximation by ContValueNet.

IX. Conclusion

In this article, we have proposed a DT-assisted approach to device-edge collaboration on DNN inference, which can adapt to dynamic computing workloads at AIoT devices and edge servers with low-signaling overhead. The proposed approach can be applied to support AIoT scenarios where densely deployed AIoT devices dynamically generate AI model inference tasks. For the future work, we will further explore the DTs to capture the time-varying data distribution for joint AI model training and AI model inference in AIoT.

Appendix A
Proof of Proposition 1

Case 1: If $T_{n-1}^{\mathrm{lq}} + T_{n-1}^{\mathrm{lc}} \le \Delta T_{n-1}$, the on-device queuing delay of the nth DNN task T_n^{lq} equals to zero. Based on (15), $D_{n-1\to n}^{\mathrm{lq}} = \max\{\min\{T_{n-1}^{\mathrm{lq}} - \Delta T_{n-1}, 0\} + T_{n-1}^{\mathrm{lc}}, 0\} = 0$. Similarly, we have $D_{m\to n}^{\mathrm{lq}} = 0 \ \ \forall m < n$, we have $D_{m\to n}^{\mathrm{lq}} = 0 \ \ \forall m \ge n$. Therefore, $T_n^{\mathrm{lq}} = \sum_{m=1}^{N} D_{m\to n}^{\mathrm{lq}}$.

Case 2: If $T_{n-1}^{\mathrm{lq}} + T_{n-1}^{\mathrm{lc}} > \Delta T_{n-1}$, $T_n^{\mathrm{lq}} = T_{n-1}^{\mathrm{lq}} + T_{n-1}^{\mathrm{lc}} - \Delta T_{n-1} > 0$, which indicates that one preceding DNN task (its index is denoted by m_0) is in the computing unit upon the generation of the nth task. As a result, we have

$$
T_{m_0}^{\mathrm{lq}} \le \sum_{i=m_0}^{n-1} \Delta T_i \le T_{m_0}^{\mathrm{lq}} + T_{m_0}^{\mathrm{lc}} \tag{41}
$$

and for any mth task, where $m_0 + 1 \le m \le n-1$, $T_m^{\mathrm{lq}} \ge \sum_{i=m}^{n-1} \Delta T_i$.

Based on (15), $D_{m\to n}^{\mathrm{lq}}$ can be calculated by

$$
D_{m\to n}^{\mathrm{lq}} = \begin{cases} T_m^{\mathrm{lq}} + T_m^{\mathrm{lc}} - \sum_{i=m}^{n-1} \Delta T_i, & m = m_0 \\ T_m^{\mathrm{lc}}, & m_0 + 1 \le m \le n-1 \\ 0, & \text{otherwise.} \end{cases} \tag{42}
$$

Then, $\sum_{n=1}^{N} D_{m\to n}^{\mathrm{lq}}$ can be calculated as

$$
\begin{aligned}
\sum_{n=1}^{N} D_{m\to n}^{\mathrm{lq}} &= T_{m_0}^{\mathrm{lq}} + T_{m_0}^{\mathrm{lc}} - \sum_{i=m_0}^{n-1} \Delta T_i + \sum_{m=m_0+1}^{n-1} T_m^{\mathrm{lc}} \\
&= \left(T_{m_0}^{\mathrm{lq}} + T_{m_0}^{\mathrm{lc}} - \Delta T_{m_0} \right) - \sum_{i=m_0+1}^{n-1} \Delta T_i + \sum_{m=m_0+1}^{n-1} T_m^{\mathrm{lc}} \\
&= T_{m_0+1}^{\mathrm{lq}} - \sum_{i=m_0+1}^{n-1} \Delta T_i + \sum_{m=m_0+1}^{n-1} T_m^{\mathrm{lc}} \\
&= T_{m_0+1}^{\mathrm{lq}} + T_{m_0+1}^{\mathrm{lc}} - \sum_{i=m_0+1}^{n-1} \Delta T_i + \sum_{m=m_0+2}^{n-1} T_m^{\mathrm{lc}}, \ldots \\
&= T_{n-1}^{\mathrm{lq}} + T_{n-1}^{\mathrm{lc}} - \Delta T_{n-1} = T_n^{\mathrm{lq}}.
\end{aligned} \tag{43}
$$

APPENDIX B
PROOF OF PROPOSITION 2

When $T_n^{\mathrm{lc}} = 0$, $D_{n\to m}^{\mathrm{lq}} = 0 \ \ \forall 1 \le m \le N$. As a result, $D_n^{\mathrm{lq}} = \sum_{m=1}^{N} D_{n\to m}^{\mathrm{lq}} = 0$. When $T_n^{\mathrm{lc}} \ge \Delta T$, for brevity, we denote $Q^D(t_{n,0})$ and $Q^D(t_{n,0} + (T_n^{\mathrm{lc}}/\Delta T) - 1)$ by q_0 and q_1, which represent the number of tasks in the on-device task queue in the time slots when the on-device inference for the nth DNN task starts and completes, respectively. During the on-device inference for the nth task, the mth task is in the on-device task queue, where $n < m \le n + q_1$. We can derive the queuing delay of the mth task that results from the on-device inference of the nth task as

$$D_{n\to m}^{\mathrm{lq}} = \begin{cases} T_n^{\mathrm{lc}}, & n < m \le n + q_0 \\ T_n^{\mathrm{lq}} + T_n^{\mathrm{lc}} - \sum_{i=n}^{m-1} \Delta T_i, & n + q_0 + 1 \le m \le n + q_1 \end{cases} \tag{46}$$

and calculate on-device workload during the on-device inference of the nth task by

$$Q^D(t) = \begin{cases} q_0, & t_{n,0} \le t < \frac{1}{\Delta T} \sum_{i=1}^{n+q_0} \Delta T_i \\ m - n, & \frac{1}{\Delta T} \sum_{i=1}^{n+q_0} \Delta T_i \le t < \frac{1}{\Delta T} \sum_{i=1}^{m-1} \Delta T_i \\ q_1, & \frac{1}{\Delta T} \sum_{i=1}^{n+q_1-1} \Delta T_i \le t < t_{n,0} + \frac{1}{\Delta T} T_n^{\mathrm{lc}} \end{cases} \tag{47}$$

where $q_0 + n \le m \le q_1 + n$. Based on (42) and (47), we can calculate $\sum_{m=1}^{N} D_{n\to m}^{\mathrm{lq}}$ and $\sum_{t=t_{n,0}}^{t_{n,0}+(T_n^{\mathrm{lc}}/\Delta T)-1} Q^D(t)\Delta T$ using (44) and (45), shown at the top of the previous page, respectively, and it can be observed from (44) and (45) that they are equal.

REFERENCES

[1] S. Hu, M. Li, J. Gao, C. Zhou, and X. Shen, "Digital twin-assisted adaptive DNN inference in Industrial Internet of Things," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, 2022, pp. 1025–1030.

[2] X. Shen, J. Gao, W. Wu, M. Li, C. Zhou, and W. Zhuang, "Holistic network virtualization and pervasive network intelligence for 6G," *IEEE Commun. Surveys Tuts.*, vol. 24, no. 1, pp. 1–30, 1st Quart., 2022.

[3] J. Zhang and D. Tao, "Empowering things with intelligence: A survey of the progress, challenges, and opportunities in artificial intelligence of things," *IEEE Internet Things J.*, vol. 8, no. 10, pp. 7789–7817, May 2021.

[4] Z. Zhou, X. Chen, E. Li, L. Zeng, K. Luo, and J. Zhang, "Edge intelligence: Paving the last mile of artificial intelligence with edge computing," *Proc. IEEE*, vol. 107, no. 8, pp. 1738–1762, Aug. 2019.

[5] X. Huang, P. He, A. Rangarajan, and S. Ranka, "Intelligent intersection: Two-stream convolutional networks for real-time near-accident detection in traffic video," *ACM Trans. Spatial Algorithms Syst.*, vol. 6, no. 2, pp. 1–28, Jun. 2020.

[6] Y. Kang et al., "Neurosurgeon: Collaborative intelligence between the cloud and mobile edge," *ACM SIGARCH Comput. Archit. News*, vol. 45, no. 1, pp. 615–629, Mar. 2017.

[7] X. Chen, J. Zhang, B. Lin, Z. Chen, K. Wolter, and G. Min, "Energy-efficient offloading for DNN-based smart IoT systems in cloud-edge environments," *IEEE Trans. Parallel Distrib. Syst.*, vol. 33, no. 3, pp. 683–697, Mar. 2021.

[8] Y. Huang et al., "An integrated cloud-edge-device adaptive deep learning service for cross-platform Web," *IEEE Trans. Mobile Comput.*, vol. 22, no. 4, pp. 1950–1967, Oct. 2023.

[9] L. Zeng, E. Li, Z. Zhou, and X. Chen, "Boomerang: On-demand cooperative deep neural network inference for edge intelligence on the Industrial Internet of Things," *IEEE Netw.*, vol. 33, no. 5, pp. 96–103, Sep./Oct. 2019.

[10] B. Han, V. Sciancalepore, Y. Xu, D. Feng, and H. D. Schotten, "Impatient queuing for intelligent task offloading in multiaccess edge computing," *IEEE Trans. Wireless Commun.*, vol. 22, no. 1, pp. 59–72, Jan. 2023.

[11] J. Yan, S. Bi, and Y.-J. A. Zhang, "Optimal model placement and online model splitting for device-edge co-inference," *IEEE Trans. Wireless Commun.*, vol. 21, no. 10, pp. 8354–8367, Oct. 2022.

[12] 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 Int. Conf. Commun. (ICC)*, 2021, pp. 1–6.

[13] L. U. Khan, W. Saad, D. Niyato, Z. Han, and C. S. Hong, "Digital-twin-enabled 6G: Vision, architectural trends, and future directions," *IEEE Commun. Mag.*, vol. 60, no. 1, pp. 74–80, Jan. 2022.

[14] Y. Lu, S. Maharjan, and Y. Zhang, "Adaptive edge association for wireless digital twin networks in 6G," *IEEE Internet Things J.*, vol. 8, no. 22, pp. 16219–16230, Nov. 2021.

[15] H. Wang, Y. Wu, G. Min, and W. Miao, "A graph neural network-based digital twin for network slicing management," *IEEE Trans. Ind. Informat.*, vol. 18, no. 2, pp. 1367–1376, Feb. 2022.

[16] W. Sun, H. Zhang, R. Wang, and Y. Zhang, "Reducing offloading latency for digital twin edge networks in 6G," *IEEE Trans. Veh. Technol.*, vol. 69, no. 10, pp. 12240–12251, Oct. 2020.

[17] M. Li, J. Gao, L. Zhao, and X. Shen, "Deep reinforcement learning for collaborative edge computing in vehicular networks," *IEEE Trans. Cogn. Commun. Netw.*, vol. 6, no. 4, pp. 1122–1135, Dec. 2020.

[18] C. Zhou et al., "Deep reinforcement learning for delay-oriented IoT task scheduling in SAGIN," *IEEE Trans. Wireless Commun.*, vol. 20, no. 2, pp. 911–925, Feb. 2021.

[19] Z. Lin, S. Bi, and Y.-J. A. Zhang, "Optimizing AI service placement and resource allocation in mobile edge intelligence systems," *IEEE Trans. Wireless Commun.*, vol. 20, no. 11, pp. 7257–7271, Nov. 2021.

[20] B. Yang, X. Cao, X. Li, Q. Zhang, and L. Qian, "Mobile-edge-computing-based hierarchical machine learning tasks distribution for IIoT," *IEEE Internet Things J.*, vol. 7, no. 3, pp. 2169–2180, Mar. 2020.

[21] W. Fan, Z. Chen, Y. Su, F. Wu, B. Tang, and Y. Liu, "Accuracy-based task offloading and resource allocation for edge intelligence in IoT," *IEEE Wireless Commun. Lett.*, vol. 11, no. 2, pp. 371–375, Feb. 2022.

[22] 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. Ind. Informat.*, vol. 17, no. 7, pp. 4988–4998, Jul. 2020.

[23] 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. Mobile Comput.*, vol. 22, no. 4, pp. 2435–2445, Apr. 2023.

[24] X. Tang, X. Chen, L. Zeng, S. Yu, and L. Chen, "Joint multiuser DNN partitioning and computational resource allocation for collaborative edge intelligence," *IEEE Internet Things J.*, vol. 8, no. 12, pp. 9511–9522, Jun. 2021.

[25] H. Liang et al., "DNN surgery: Accelerating DNN inference on the edge through layer partitioning," *IEEE Trans. Cloud Comput.*, vol. 11, no. 3, pp. 3111–3125, Jul./Sep. 2023, doi: 10.1109/TCC.2023.3258982.

[26] W. Fan et al., "DNN deployment, task offloading, and resource allocation for joint task inference in IIoT," *IEEE Trans. Ind. Informat.*, vol. 19, no. 2, pp. 1634–1646, Feb. 2023.

[27] W. Shi, S. Zhou, Z. Niu, M. Jiang, and L. Geng, "Multiuser co-inference with batch processing capable edge server," *IEEE Trans. Wireless Commun.*, vol. 22, no. 1, pp. 286–300, Jan. 2023.

[28] S. Teerapittayanon, B. McDanel, and H.-T. Kung, "BranchyNet: Fast inference via early exiting from deep neural networks," in *Proc. Int. Conf. Pattern Recognit. (ICPR)*, 2016, pp. 2464–2469.

[29] X. Deng et al., "Low-latency federated learning with DNN partition in distributed industrial IoT networks," *IEEE J. Sel. Areas Commun.*, vol. 41, no. 3, pp. 755–775, Dec. 2023.

[30] C. Herrera, F. Krach, P. Ruyssen, and J. Teichmann, "Optimal stopping via randomized neural networks," 2021, *arXiv:2104.13669*.

[31] T. S. Ferguson. "Optimal stopping and applications." 2019. [Online]. Available: https://www.math.ucla.edu/ tom/Stopping/Contents.html

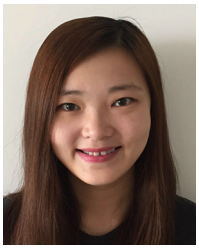
[32] S. Becker, P. Cheridito, and A. Jentzen, "Deep optimal stopping," *J. Mach. Learn. Res.*, vol. 20, pp. 1–25, Apr. 2019.

[33] J. Tsitsiklis and B. Van Roy, "Regression methods for pricing complex American-style options," *IEEE Trans. Neural Netw.*, vol. 12, no. 4, pp. 694–703, Jul. 2001.

[34] Z. Wu, J. Wen, Y. Xu, J. Yang, X. Li, and D. Zhang, "Enhanced spatial feature learning for weakly supervised object detection," *IEEE Trans. Neural Netw. Learn. Syst.*, early access, Jun. 8, 2022, doi: 10.1109/TNNLS.2022.3178180.

Shisheng Hu (Graduate Student Member, IEEE) received the B.Eng. and the M.A.Sc. degrees from the University of Electronic Science and Technology of China, Chengdu, China, in 2018 and 2021, respectively. He is currently pursuing the Ph.D. degree with the Department of Electrical and Computer Engineering, University of Waterloo, Waterloo, ON, Canada.

His research interests include AI for wireless networks and networking for AI.



Mushu Li (Member, IEEE) received the Ph.D. degree in electrical and computer engineering from the University of Waterloo, ON, Canada, in 2021.

She is currently a Postdoctoral Fellow with Toronto Metropolitan University, Toronto, ON, Canada. She was a Postdoctoral Fellow with the University of Waterloo from 2021 to 2022. Her research interests include mobile edge computing, the system optimization in wireless networks, and machine learning-assisted network management.

Dr. Li was the recipient of the Natural Science and Engineering Research Council of Canada (NSERC) Postdoctoral Fellowship in 2022, the NSERC Canada Graduate Scholarship in 2018, and the Ontario Graduate Scholarship in 2015 and 2016.

Jie Gao (Senior Member, IEEE) received the M.Sc. and Ph.D. degrees in electrical engineering from the University of Alberta, Edmonton, AB, Canada, in 2009 and 2014, respectively.

He was a Postdoctoral Fellow with Toronto Metropolitan (formerly Ryerson) University, Toronto, ON, Canada, from 2017 to 2019 and a Research Associate with the University of Waterloo, Waterloo, ON, Canada, from 2019 to 2020. He was an Assistant Professor with the Department of Electrical and Computer Engineering, Marquette University, Milwaukee, WI, USA, from 2020 to 2022, and is currently an Assistant Professor with the School of Information Technology, Carleton University, Ottawa, ON, Canada. His research interests include machine learning for communications and networking, cloud and multiaccess edge computing, Internet of Things and Industrial IoT solutions, and B5G/6G networks in general.

Conghao Zhou (Member, IEEE) received the B.Eng. degree from Northeastern University, Shenyang, China, in 2017, the M.Sc. degree from the University of Illinois at Chicago, Chicago, IL, USA, in 2018, and the Ph.D. degree in electrical and computer engineering from the University of Waterloo, Waterloo, ON, Canada, in 2022.

He is currently a Postdoctoral Fellow with the University of Waterloo. His research interests include space–air–ground integrated networks, network slicing, and machine learning for wireless networks.

Xuemin (Sherman) Shen (Fellow, IEEE) received the Ph.D. degree in electrical engineering from Rutgers University, New Brunswick, NJ, USA, in 1990.

He is a University Professor with the Department of Electrical and Computer Engineering, University of Waterloo, Waterloo, ON, Canada. His research focuses on network resource management, wireless network security, Internet of Things, 5G and beyond, and vehicular networks.

Dr. Shen received the "West Lake Friendship Award" from Zhejiang Province in 2023, the President's Excellence in Research from the University of Waterloo in 2022, the Canadian Award for Telecommunications Research from the Canadian Society of Information Theory in 2021, the R.A. Fessenden Award in 2019 from IEEE, Canada, the Award of Merit from the Federation of Chinese Canadian Professionals (Ontario) in 2019, the James Evans Avant Garde Award in 2018 from the IEEE Vehicular Technology Society, the Joseph LoCicero Award in 2015 and the Education Award in 2017 from the IEEE Communications Society (ComSoc), and the Technical Recognition Award from Wireless Communications Technical Committee (2019) and AHSN Technical Committee (2013). He has also received the Excellent Graduate Supervision Award in 2006 from the University of Waterloo and the Premier's Research Excellence Award in 2003 from the Province of Ontario, Canada. He serves/served as the General Chair for the 6G Global Conference'23 and the ACM Mobihoc'15, the Technical Program Committee Chair/Co-Chair for IEEE Globecom'24, 16, and 07, IEEE Infocom'14, IEEE VTC'10 Fall, and the Chair for the IEEE ComSoc Technical Committee on Wireless Communications. He is the President of the IEEE ComSoc. He was the Vice President for Technical and Educational Activities, the Vice President for Publications, the Member-at-Large on the Board of Governors, the Chair of the Distinguished Lecturer Selection Committee, and the member of IEEE Fellow Selection Committee of the ComSoc. He served as the Editor-in-Chief for the IEEE IoT JOURNAL, IEEE NETWORK, and *IET Communications*. He is a Registered Professional Engineer of Ontario, Canada, an Engineering Institute of Canada Fellow, a Canadian Academy of Engineering Fellow, a Royal Society of Canada Fellow, a Chinese Academy of Engineering Foreign Member, and a Distinguished Lecturer of the IEEE Vehicular Technology Society and Communications Society.