Stochastic Cumulative DNN Inference with RL-Aided Adaptive IoT Device-Edge Collaboration

Kaige Qu, *Member, IEEE*, Weihua Zhuang, *Fellow, IEEE*, Wen Wu, *Senior Member, IEEE*, Mushu Li, *Member, IEEE*, Xuemin (Sherman) Shen, *Fellow, IEEE*, Xu Li, and Weisen Shi

***Abstract*—The advances in artificial intelligence (AI) and edge computing enable edge intelligence to support pervasive intelligent Internet of Things (IoT) applications in the future wireless networks. We focus on deep neural network (DNN) based classification tasks, and investigate how to improve the confidence level and delay performance of DNN inference via device-edge collaboration. We first develop a stochastic cumulative DNN inference scheme that aggregates multiple random DNN inference results and generates a cumulative DNN inference result with improved confidence level. Then, based on a computation-efficient DNN model deployment strategy with shared computation between a locally deployed fast DNN model and a full DNN model partitioned between the device and edge, a closed-loop adaptive device-edge collaboration scheme is developed to support cumulative DNN inference for multiple devices. We adaptively determine how to offload DNN inference computation to the edge and how to allocate transmission and edge computing resources among multiple devices, for quality-of-service (QoS) satisfaction in terms of both confidence level and inference delay with resource and energy efficiency. A reinforcement learning (RL) approach is used for adaptive offloading decision, which relies on a resource allocation solution for reward calculation. Simulation results demonstrate the effectiveness of the adaptive device-edge collaboration scheme for cumulative DNN inference, in terms of confidence level improvement, delay violation minimization, network resource efficiency, and device energy efficiency.**

***Index Terms*—Internet of Things (IoT), edge computing, edge intelligence, DNN inference, partial offloading, device edge collaboration, reinforcement learning (RL).**

I. Introduction

In the future wireless networks, artificial intelligence (AI) models such as deep neural networks (DNNs) are pervasively deployed to support diverse intelligent Internet of Things (IoT) applications such as intelligent surveillance, autonomous driving, and factory automation [1]–[6]. Many intelligent IoT applications rely on DNN models for classification. For example, in autonomous driving, the nearby objects should be detected and classified to build an environment model for an autonomous vehicle [7]. For a general classification task, a pre-trained DNN model processes an input data sample such as raw sensing data and generates a classification result as output, which is referred to as DNN inference. The DNN model output, also referred to as a DNN inference result, has a random confidence level due to the random amount of information provided by a single data sample. For example, for an object moving under the surveillance of a smart camera, one video frame can be sampled and processed by a DNN model for object classification. However, due to the random viewing angle to the object and the random sensing data quality across different video frames, a DNN inference result based on a random data sample may not provide satisfactory confidence level and accuracy for object classification.

As different data samples corresponding to the same object (e.g., consecutive video frames containing an object) usually capture different spatial/temporal features, and different DNN models provide random DNN inference results with different confidence levels by processing the same data sample, a potential approach to improving the confidence level of a classification task is to consider multiple DNN inference results based on different data samples and DNN models to exploit the *data diversity* and *model diversity*. A straightforward method is to select a DNN inference result with the maximum confidence level and ignore other results. Then, if a confidence level requirement is not satisfied, more data samples should be collected and more computation should be triggered to obtain a more accurate DNN inference result. This approach may lead to high latency if the required confidence level is high, which may violate a task completion time requirement. Actually, it is inefficient to completely ignore the DNN inference results with lower confidence levels, especially for those whose confidence levels are close to the required threshold. This observation motivates our investigation on a *cumulative DNN inference* scheme for a classification task, that progressively incorporates the different contributions from multiple random DNN inference results and improves a *cumulative confidence level* for the classification task.

To obtain the multiple DNN inference results that support cumulative DNN inference with computing resource efficiency, the trade-off between confidence level and computing demand of different DNN models should be considered. Typically, more complex DNN models with deeper layers provide a higher confidence level on average at the cost of high computing demand. If DNN inference is executed locally, it is intractable for the resource-limited IoT devices to satisfy a high confidence level requirement with low latency. An edge-only solution, which fully offloads the raw data samples with

This work was financially supported by research grants from Huawei Technologies Canada and from the Natural Sciences and Engineering Research Council (NSERC) of Canada.

Kaige Qu, Weihua Zhuang, Mushu Li, and Xuemin (Sherman) Shen are with the Department of Electrical and Computer Engineering, University of Waterloo, Waterloo, ON, Canada, N2L 3G1 (emails: {k2qu, wzhuang, m475li, sshen}@uwaterloo.ca).

Wen Wu is with the Frontier Research Center, Peng Cheng Laboratory, Shenzhen, Guangdong, China, 518055 (email: wuw02@pcl.ac.cn). This work was done when Wen Wu was with the Department of Electrical and Computer Engineering, University of Waterloo, Waterloo, ON, Canada, N2L 3G1.

Xu Li and Weisen Shi are with Huawei Technologies Canada Inc., Ottawa, ON, Canada, K2K 3J1 (emails: {Xu.LiCA, weisen.shi1}@huawei.com).

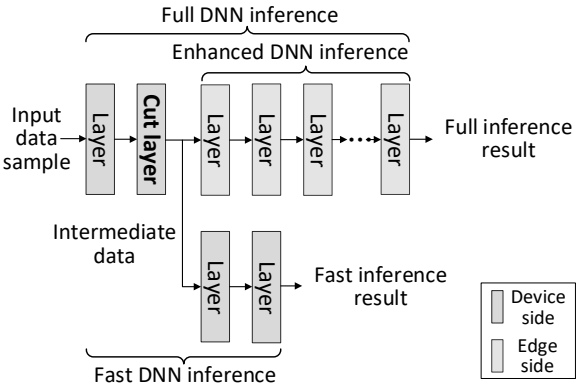


Fig. 1: An illustration of computation-efficient DNN model deployment.

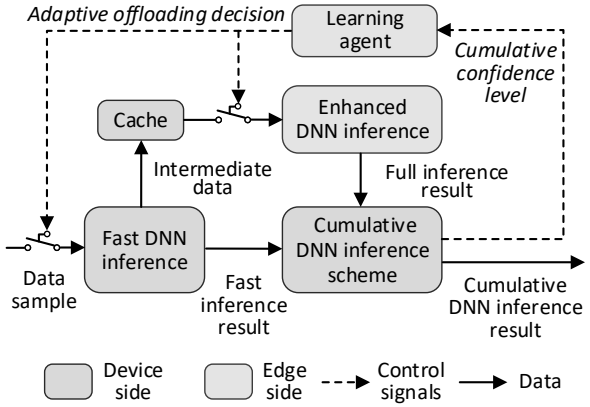


Fig. 2: A learning-based control loop supporting cumulative DNN inference with device-edge collaboration for one device.

a large data size to a resource-rich edge server, suffers from a long transmission delay. Recently, collaborative inference over a device-edge-cloud computing hierarchy has been investigated to improve the delay performance, by partially offloading the DNN model computation to nearby devices, edge server or cloud [8]–[11]. For example, in a horizontal DNN partitioning scheme, the inference before and at a selected cut layer is computed locally at the device, and the remaining inference after the cut layer (referred to as *enhanced DNN inference*) is computed at the edge server based on the *intermediate data* generated at the cut layer output, which reduces the total DNN inference delay [12]–[16]. Benefiting from the delay improvement via device-edge collaboration, a complex DNN model providing a high confidence level can be used for real-time IoT applications. We refer to this complex DNN model partitioned across device and edge as a *full DNN model*. To improve the computing efficiency, we also deploy a compact DNN model of less layers with a lower confidence level at the device side, referred to as a *fast DNN model*, which shares the local computation with the full DNN model. Such a computation-efficient DNN model deployment strategy with device-edge collaboration is illustrated in Fig. 1. If both the fast and full DNN models are executed for a data sample, two DNN inference results (referred to as fast and full DNN inference results respectively) are generated with a limited computation increase as compared with the original DNN partitioning scheme.

Consider an edge-assisted multi-device IoT scenario, where each IoT device has a classification task with quality-of-service (QoS) requirements in terms of the confidence level and task delay, and the computation-efficient DNN model deployment strategy with device-edge collaboration is applied to each device. For each device, a new input data sample is first processed locally by a fast DNN model, generating a fast inference result. The intermediate data at the cut layer is temporally stored in a local cache at the device, which is obtained as a by-product during fast DNN inference. A full inference result is generated by offloading a cached intermediate data sample to the edge server for enhanced DNN inference, which consumes network resources for transmission and edge computing. The fast or full DNN inference results are sequentially aggregated to obtain a continually updated cumulative DNN inference result, until the cumulative confidence level reaches a predefined threshold. If the confidence level requirement is not satisfied within a predefined deadline in time, a delay violation penalty is applied to the device.

To improve the cumulative confidence level within a given time and reduce the delay violation probability, it is preferable to execute more full DNN inference than fast DNN inference, i.e., offloading is preferred to local computing, as a full inference result has a higher confidence level on average. However, more offloading means consuming more transmission and edge computing resources, both of which are shared among multiple devices. Moreover, the local energy consumption should be considered, as the IoT devices are usually battery powered and thus the energy efficiency is a concern. As the intermediate data size is relatively small, the local transmission energy for offloading one intermediate data sample to obtain one full inference result is usually smaller than the local computing energy for fast DNN inference. A choice should be made between using less local energy but more network resources to offload a cached intermediate data sample for a full inference result with a higher confidence level, and using more local energy but no network resources to locally process a new data sample for a fast inference result with a lower confidence level.

In this paper, a resource and energy efficient adaptive device-edge collaboration scheme is presented, which supports cumulative DNN inference among multiple devices in an edge-assisted intelligent IoT scenario, for confidence level satisfaction with a minimum delay violation penalty. The main contributions are summarized as follows:

- A data-driven stochastic cumulative DNN inference scheme is proposed for classification tasks, to combine the contributions of multiple fast and full inference results, based on non-parametric probability density estimation of both the fast and full DNN model outputs;
- A computation-efficient and device-edge collaborative DNN model deployment strategy is proposed with shared local computation between fast and full DNN models, based on which full inference results can be generated by reusing the intermediate data of fast inference;
- A learning-based control loop is designed to support the cumulative DNN inference with adaptive device-edge

collaboration, as illustrated in Fig. 2. Based on a continually updated cumulative confidence level, an adaptive offloading decision sequence is learned to dynamically trigger the local fast DNN inference or the remote full DNN inference for each device, until the cumulative confidence level reaches a predefined threshold;
- A joint offloading and resource allocation problem is studied, to minimize 1) the total cost accounting for both network resources and local energy consumption, and 2) the total delay violation penalty. The problem is formulated as a Markov decision process (MDP), and a deep Q-learning solution is proposed with a per-episode updated extra experience replay.

The rest of this paper is organized as follows. Related works are reviewed in Section II. The system model is described in Section III, and a joint offloading and resource allocation problem is formulated in Section IV. A reinforcement learning based solution is discussed in Section V. Simulation results are presented in Section VI, and conclusions are drawn in Section VII. Table I summarizes the main mathematical symbols.

II. Related Works

Collaborative DNN inference has been extensively studied in recent years. We give a brief overview of the *state-of-the-art* collaborative DNN inference schemes that are based on partial offloading for DNN inference acceleration.

In a typical horizontal DNN partitioning scheme, the sequential layers of a linear-topology DNN model (e.g., AlexNet) are grouped into two subsets which are partitioned at a selected cut layer. The inference before and at the cut layer is computed locally at one network device, and the remaining inference after the cut layer is computed at another network device. The combination of the two network devices can be an IoT device and an edge server, two IoT devices, or an edge server and a cloud server [12]–[14]. For a DNN model with branches (e.g., GoogleNet), which can be modeled as a directed acyclic graph, the set of layers can be partitioned into two or more disjoint subsets at multiple parallel cut layers and deployed at multiple network devices in the device-edge-cloud environment [15]–[18]. How to select the cut layer(s) for an efficient DNN model partitioning between network devices has been extensively studied to reduce the total inference delay and (or) the total energy consumption for both communication and computing, by using delay/energy performance regression techniques, optimization models, and reinforcement learning approaches [12]–[19]. Besides the horizontal DNN partitioning schemes in which the DNN inference is divided into two serial parts at selected cut layer(s), vertical DNN partitioning schemes have been extensively investigated, which exploit the model parallelism and improve the inference delay by distributing the fine-grained parallel inference workload to multiple network devices [20]–[23]. In [23], the principles for input/output splitting of fully-connected (`FC`) layers and channel/spatial/filter splitting of convolution (`CONV`) layers are introduced in detail.

Typically, a DNN model with deeper layers produces DNN inference results of a higher accuracy at the cost of a longer inference delay and a higher inference energy. To reduce the average inference delay and energy without significantly sacrificing the accuracy, a DNN model early exit architecture has been proposed, which adds one or more early exit outputs to the original DNN model, and allows a DNN inference to finish at an early exit output as long as the confidence level is satisfied [24]–[27]. To further enhance the delay performance for DNN inference, a new collaborative DNN inference scheme has been proposed by combining the DNN partitioning and the model early exit schemes, which dynamically adjusts the cut layer locations and the early exit locations under a dynamic network environment, while addressing a trade-off between accuracy and delay [28].

In summary, most existing works on collaborative DNN inference focus on the performance enhancement for DNN inference based on single data sample in reducing the total delay and (or) energy consumption without degrading the accuracy. In comparison, our adaptive device-edge collaboration scheme is to support the stochastic cumulative DNN inference based on multiple DNN inference results over time, to exploit the data and model diversity and further improve the accuracy beyond the accuracy limit of a single DNN inference result. We focus on how to satisfy a cumulative confidence level requirement within a required time limit, while using a minimum total network resource consumption and total device energy consumption across multiple time slots. The underlying DNN model partitioning and early exit design for a minimum inference delay and energy is not the focus of this work. We can use the existing approaches to select a proper cut layer and an early exit location for our DNN model, which can boost the overall delay and energy performance in the long run.

III. System Model

A. Edge-Assisted Intelligent IoT Scenario

Consider an IoT scenario with one access point (AP) and multiple stationary IoT devices, in a time-slotted system with time slot duration τ. Let integer k ($k \geq 1$) be the time slot index. The AP is co-located with an edge server. For each device i in a device set $\mathcal{I}$, we consider both a locally deployed fast DNN model, and a full DNN model partitioned between the device and the edge server, with shared layers from an input layer to a pre-determined cut layer. The cut layer output is referred to as an intermediate data sample. Let $\mathcal{L}^A$ and $\mathcal{L}^U$ denote the layer sets of the fast and full DNN models respectively, with subset $\mathcal{L}_0 = \mathcal{L}^A \cap \mathcal{L}^U$ including all the shared layers. Let $\hat{l} \in \mathcal{L}_0$ denote the cut layer. For the full DNN model, all the layers in subset $\mathcal{L}_0$ are executed locally, while all the remaining layers in subset $\mathcal{L}^U \backslash \mathcal{L}_0$, corresponding to enhanced DNN inference, are executed at the edge server.

Each device i initiates a classification task at the beginning of time slot $k = 1$, with a task completion time requirement, denoted by T_i in second and $K_i = \lfloor \frac{T_i}{\tau} \rfloor$ in number of time slots. For the task, there are multiple input data samples generated by an embedded data source in the device. For example, a smart camera can generate consecutive video frames for the classification of a moving object. Based on these input data samples, multiple DNN inference results can be generated for

TABLE I: List of important notations

Parameters	
$c(k)$	Cost during time slot k
d_i^A	Computing delay for fast DNN inference at device i
$d_i^E(k)$	Computing delay for enhanced DNN inference of device i during time slot k
$d_i^T(k)$	Transmission delay for offloading one intermediate data sample from device i to the AP during time slot k
e_i^A	Local computing energy at device i
$e_i^T(k)$	Transmission energy for offloading one intermediate data sample by device i during time slot k
$e_i(k)$	Local energy consumption at device i during time slot k
f_i (f_0)	CPU frequency (in cycle/s) of device i (edge server)
K_i	Task completion time requirement for device i
$P_i(k)$	Delay violation penalty of device i at time slot k
$q_i(k)$	Number of intermediate data samples in local cache of device i at the beginning of time slot k
w	Intermediate data size (output data size of cut layer $\hat{l}$) in bit
η_T	Confidence level requirement
$\eta_i(k)$	Cumulative confidence level of device i at time slot k
μ_l	Computing demand in CPU cycles for layer $l \in \mathcal{L}^A \cup \mathcal{L}^U$
τ	Time slot duration in second
Decision variables	
$a_i(k)$	Number of intermediate data samples offloaded by device i during time slot k
$\beta_i^T(k)$	Fraction of bandwidth allocated to device i at time slot k
$\beta_i^E(k)$	Fraction of edge computing resources allocated to device i during time slot k
$\rho(k)$	Network resource consumption ratio during time slot k

the task at device i over time by the fast and full DNN models. A local cache is used at each device to temporarily store the intermediate data samples during each execution of fast DNN inference. Let $\boldsymbol{q}_k \in \mathbb{R}^{|\mathcal{I}|}$ denote the caching state at the beginning of time slot k, where the i-th element, $q_i(k)$, represents the number of intermediate data samples stored in the local cache of device i, with initial state $q_i(1) = 0$. Let $\boldsymbol{a}_k \in \mathbb{R}^{|\mathcal{I}|}$ denote an integer offloading decision vector during time slot k, where element $a_i(k)$ represents the offloading decision variable for device i. Specifically, if $a_i(k) = 0$, no offloading takes place but one new data sample is locally processed by fast DNN inference at device i, generating one new fast inference result, and one new intermediate data sample is added to the local cache; otherwise, a number of $a_i(k)$ intermediate data samples are offloaded from the local cache of device i to the edge server for enhanced DNN inference, generating one or more full inference results. Accordingly, the caching state $\boldsymbol{q}_{k+1} = \{q_i(k+1), \forall i \in \mathcal{I}\}$ at the beginning of time slot $k+1$, is updated as

$$q_i(k+1) = \begin{cases} q_i(k) + 1, & \forall i \in \mathcal{I} \text{ if } a_i(k) = 0 \\ [q_i(k) - a_i(k)]^+, & \forall i \in \mathcal{I} \text{ if } a_i(k) > 0. \end{cases} \quad (1)$$

At most $q_i(k)$ intermediate data samples can be offloaded from device i during time slot k, i.e.,

$$0 \le a_i(k) \le q_i(k), \quad \forall i \in \mathcal{I}. \quad (2)$$

B. DNN Model Layer Parameters

A DNN model for classification usually begins with a feature extraction module composed mainly of `CONV` layers and pooling (e.g., `MaxPool`) layers, followed by a classifier composed mainly of `FC` layers. A pooling layer down-samples the output of a `CONV` layer and reduces the data dimension. An activation layer is usually applied to each `CONV` and `FC` layer output for nonlinearity. The last `FC` layer is usually activated with `Softmax` function, to generate a nonnegative probability vector adding up to one.

For `CONV` layer $l \in \mathcal{L}^A \cup \mathcal{L}^U$, let $\left(H_l^I, W_l^I, D_l^I\right)$ and $\left(H_l^O, W_l^O, D_l^O\right)$ denote the dimensions in height, width, and number of channels, for the input and output feature maps respectively. To generate the output feature map at layer l, a number of D_l^O filters, each with a dimension of $\left(h_l, h_l, D_l^I\right)$, are applied to the input feature map, by sweeping each filter over it and calculating the dot products [23], [29]. Each filter creates a channel in the output feature map. Filter n ($1 \le n \le D_l^O$) at layer l performs $\left(h_l\right)^2 D_l^I$ multiplications and $\left(h_l\right)^2 D_l^I - 1$ additions, in calculating each of the $H_l^O W_l^O$ data elements in the j-th channel of the output feature map. Thus, a total number of $2\left(h_l\right)^2 D_l^I - 1$ floating-point operations, including both multiplications and additions, are required for computing each of the $H_l^O W_l^O D_l^O$ output data elements at `CONV` layer l. For `FC` layer $l \in \mathcal{L}^A \cup \mathcal{L}^U$, let X_l^I and X_l^O denote the input and output dimensions, respectively. To compute each of the X_l^O output data elements, the X_l^I-dimension inputs are multiplied with the corresponding weights, and the weighted inputs are then summed with a bias. Hence, X_l^I multiplications and X_l^I additions are required for each output data element at `FC` layer l. Accordingly, a total number of $2X_l^I$ floating-point operations are required for computing each of the X_l^O output data element at `FC` layer l.

Let v_l denote the number of output data elements at layer $l \in \mathcal{L}^A \cup \mathcal{L}^U$, given by

$$v_l = \begin{cases} H_l^O W_l^O D_l^O, & \text{if layer } l \text{ is a } \texttt{CONV} \text{ layer} \\ X_l^O, & \text{if layer } l \text{ is an } \texttt{FC} \text{ layer.} \end{cases} \quad (3)$$

Let π_l denote the number of floating-point operations, including both multiplications and additions, for computing one output data element at layer $l \in \mathcal{L}^A \cup \mathcal{L}^U$, given by

$$\pi_l = \begin{cases} 2\left(h_l\right)^2 D_l^I - 1, & \text{if layer } l \text{ is a } \texttt{CONV} \text{ layer} \\ 2X_l^I, & \text{if layer } l \text{ is an } \texttt{FC} \text{ layer.} \end{cases} \quad (4)$$

Note that π_l is set to 0 by default, as other DNN layers such as the pooling and activation layers have negligible computing demand in comparison with the `CONV` and `FC` layers.

C. Computing Model

Let $\mu_l = \varphi v_l \pi_l$ denote the computing demand in number of CPU cycles for layer $l \in \mathcal{L}^A \cup \mathcal{L}^U$, where φ is the number of CPU cycles for one floating-point operation. Let f_i and f_0 denote the CPU frequencies (in cycle/s) of device i and the edge server respectively.

Let d_i^A denote the computing delay for one fast DNN inference at device i, given by

$$d_i^A = \frac{\sum_{l \in \mathcal{L}^A} \mu_l}{f_i}, \quad \forall i \in \mathcal{I}. \quad (5)$$

We should have $\tau \ge \max_{i \in \mathcal{I}} d_i^A$, to ensure that the fast DNN inference for one data sample at any device can finish in one

time slot. The local computing energy consumption for fast DNN inference at device i, denoted by e_i^A, is given by

$$e_i^A = \kappa_i \left(f_i\right)^3 d_i^A = \kappa_i \left(f_i\right)^2 \sum_{l \in \mathcal{L}^A} \mu_l, \quad \forall i \in \mathcal{I} \tag{6}$$

where κ_i is the energy efficiency coefficient for device i [30].

For enhanced DNN inference, each device is allocated with a virtual CPU at the edge server. Let $\boldsymbol{\beta}^E(k) \in \mathbb{R}^{|\mathcal{I}|}$ denote an edge computing resource allocation decision vector at time slot k, with element $\beta_i^E(k)$ representing the fraction of edge computing resources allocated to the virtual CPU for device i. We have $\beta_i^E(k) = 0$ for device i if $a_i(k) = 0$, with

$$0 \le \beta_i^E(k) \le a_i(k)\mathbb{M}, \quad \forall i \in \mathcal{I} \tag{7}$$

where $\mathbb{M} \gg 1$ is a very large constant. The computing delay for enhanced DNN inference of device i during time slot k, denoted by $d_i^E(k)$, is given by

$$d_i^E(k) = \frac{\sum_{l \in \mathcal{L}^U \setminus \mathcal{L}_0} \mu_l}{\left[\beta_i^E(k) + \varepsilon\right] f_0}, \quad \forall i \in \mathcal{I} \tag{8}$$

where ε is a very small constant parameter with $0 < \varepsilon \ll 1$. Adding such a small value, e.g., 10^{-6}, to $\beta_i^E(k)$ in the denominator of (8) avoids $d_i^E(k)$ being undetermined when no edge computing resources are allocated to device i at time slot k, i.e., $\beta_i^E(k) = 0$, while making nearly no difference on the optimal value of $\beta_i^E(k)$. Let constant $C_i^{(1)} = \frac{\sum_{l \in \mathcal{L}^U \setminus \mathcal{L}_0} \mu_l}{f_0}$ be the value of $d_i^E(k)$ for f_0 total edge computing resources.

D. Communication Model

Let $w = \phi v_{\hat{l}}$ denote the intermediate data size in bit, where ϕ is the number of bits to represent a floating-point number, and $v_{\hat{l}}$ is given by (3) if cut layer $\hat{l}$ is a `CONV` or `FC` layer. In practice, the cut layer is usually selected as a pooling layer following a `CONV` layer. In this case, $v_{\hat{l}}$ is the number of output data elements after data down-sampling by the pooling layer.

Consider orthogonal frequency division multiple access (OFDMA) for the uplink wireless transmission between IoT devices and the AP with total radio spectrum bandwidth B. Let $\boldsymbol{\beta}^T(k) \in \mathbb{R}^{|\mathcal{I}|}$ denote the bandwidth allocation decision vector during time slot k, with element $\beta_i^T(k)$ representing the fraction of bandwidth allocated to device i. We have

$$0 \le \beta_i^T(k) \le a_i(k)\mathbb{M}, \quad \forall i \in \mathcal{I}. \tag{9}$$

The transmission delay for offloading one intermediate data sample from the local cache of device i to the edge server via the AP during time slot k, denoted by $d_i^T(k)$, is given by

$$d_i^T(k) = \frac{w}{\left[\beta_i^T(k) + \varepsilon\right] B \log_2\left(1 + \frac{p_i g_i}{\sigma^2}\right)}, \quad \forall i \in \mathcal{I} \tag{10}$$

where p_i denotes the transmit power of device i, g_i denotes the constant uplink transmission power gain between device i and AP, and σ^2 represents the noise power. Let constant $C_i^{(2)} = \frac{w}{B \log_2\left(1 + \frac{p_i g_i}{\sigma^2}\right)}$ be the value of $d_i^T(k)$ when using whole bandwidth B. We ignore the downlink delay for transmitting the full inference results back to the devices due to the small data size. The transmission energy consumption for offloading one intermediate data sample from device i to the AP during time slot k, denoted by $e_i^T(k)$, is given by

$$e_i^T(k) = p_i d_i^T(k), \quad \forall i \in \mathcal{I}. \tag{11}$$

E. Cumulative DNN Inference Scheme

Consider an M-class classification task, where class label Y is a random variable with M possible integer outcomes in $\{1, \ldots, M\}$. An input data sample, x, to the fast or full DNN model is a random sample following an unknown probability distribution. For example, a grayscale image with resolution 200×200 is a random sample from a 200^2-dimension unknown joint probability distribution of pixel intensity values. A fast or full DNN model output, i.e., a fast or full inference result, is represented by an M-dimension estimated class probability vector, $\boldsymbol{z} = \{z_m\}$, with $z_m = \Pr\left(Y = m | x\right)$ denoting the estimated conditional probability of class $m \in \{1, \cdots, M\}$ given data sample x. The entropy of Y conditioned on x, calculated as $-\sum_{m=1}^{M} z_m \log z_m$, measures the uncertainty of the DNN inference result $\boldsymbol{z}$. We use one minus normalized entropy to represent the confidence level of $\boldsymbol{z}$, given by

$$\eta\left(\boldsymbol{z}\right) = 1 + \sum_{m=1}^{M} \frac{z_m \log z_m}{\log M} \tag{12}$$

which has a value between 0 and 1 [24], [25]. Typically, a full inference result has a higher confidence level on average than a fast inference result.

Consider multiple DNN inference results for the M-class classification task. Let $\boldsymbol{z}_j = \{z_{j,m}\}$ denote the j-th DNN inference result. Let $\chi_j \in \{0, 1\}$ indicate whether $\boldsymbol{z}_j$ is generated by the fast or full DNN inference, with $\chi_j = 1$ indicating full DNN inference, and $\chi_j = 0$ otherwise. Let $Z_j = \{\boldsymbol{z}_1, \ldots, \boldsymbol{z}_j\}$ denote the set of DNN inference results up to $\boldsymbol{z}_j$. Define the cumulative DNN inference result given Z_j as an M-dimension estimated class probability vector, denoted by $\boldsymbol{o}_j = \{o_{j,m}\}$, with $o_{j,m} = \Pr\left(Y = m | Z_j\right)$ representing the estimated conditional probability of class m given Z_j. For true class $\hat{m}$, $z_{j,\hat{m}}$ is referred to as the estimated true class probability by the j-th DNN inference result, and $o_{j,\hat{m}}$ is referred to as the cumulative estimated true class probability given Z_j. Based on Bayes' theorem and under the assumption of conditional independence among different DNN inference results given the same true class label, $o_{j,m}$ is written as

$$o_{j,m} = \frac{\Pr\left(Y = m\right) \prod_{j'=1}^{j} \Pr\left(\boldsymbol{z}_{j'} \middle| Y = m\right)}{\Pr\left(\boldsymbol{z}_1, \ldots, \boldsymbol{z}_j\right)} \tag{13}$$

where $\Pr\left(Y = m\right)$ represents the prior class distribution, and $\Pr\left(\boldsymbol{z}_{j'} \middle| Y = m\right)$ represents the joint probability density of the j'-th DNN inference result, i.e., vector $\boldsymbol{z}_{j'}$, given true class label $Y = m$. Let $\mathsf{f}_m^A\left(\boldsymbol{z}\right)$ and $\mathsf{f}_m^U\left(\boldsymbol{z}\right)$ denote the joint probability density functions of the fast and full DNN inference results given $Y = m$, respectively. Then, we have

$$\Pr\left(\boldsymbol{z}_{j'} \middle| Y = m\right) = \left(1 - \chi_{j'}\right) \mathsf{f}_m^A\left(\boldsymbol{z}_{j'}\right) + \chi_{j'} \mathsf{f}_m^U\left(\boldsymbol{z}_{j'}\right). \tag{14}$$

We use a data-driven non-parametric probability density estimation method, kernel density estimation (KDE), to profile functions $\mathsf{f}_m^A\left(\boldsymbol{z}\right)$ and $\mathsf{f}_m^U\left(\boldsymbol{z}\right)$ for each class m. Specifically, we

first partition a labeled training dataset into M class-specific subsets according to the known class labels. With each class-specific data subset, we collect a subset of fast inference results and a subset of full inference results by running the fast and full DNN models. Then, $\mathsf{f}_m^A(\boldsymbol{z})$ and $\mathsf{f}_m^U(\boldsymbol{z})$ can be profiled with the corresponding subset of DNN inference results.

As denominator $\Pr(\boldsymbol{z}_1, \ldots, \boldsymbol{z}_j)$ in (13) is unknown, we can calculate $o_{j,m}$ through normalization based on the property that $\sum_{m=1}^{M} o_{j,m} = 1$. Hence, (13) can be rewritten as

$$o_{j,m} = \frac{\Pr(Y=m) \prod_{j'=1}^{j} \Pr\left(\boldsymbol{z}_{j'} \middle| Y=m\right)}{\sum_{m=1}^{M} \left[\Pr(Y=m) \prod_{j'=1}^{j} \Pr\left(\boldsymbol{z}_{j'} \middle| Y=m\right)\right]}. \quad (15)$$

The confidence level of cumulative DNN inference result $\boldsymbol{o}_j$, denoted by $\eta(\boldsymbol{o}_j)$, is referred to as the cumulative confidence level given Z_j, and is calculated based on (12).

Note that (15) provides a general model to calculate a cumulative DNN inference result given an arbitrary number (e.g., j) of fast or full inference results for an M-class classification task. In the considered scenario, each device i obtains either one new fast inference result or at least one full inference result during time slot k for its task, depending on offloading decision $a_i(k)$. Let $\boldsymbol{\eta}_k \in \mathbb{R}^{|\mathcal{I}|}$ denote the cumulative confidence levels for all devices at the beginning of time slot k, where element $\eta_i(k)$ represents the cumulative confidence level for device i with $\eta_i(1) = 0$. A cumulative DNN inference result is updated at the end of time slot k for each device i based on (15), by combining all the old inference results through time slots 1 to $k-1$ and the new inference results obtained during time slot k. Accordingly, the new cumulative confidence levels, i.e., $\boldsymbol{\eta}_{k+1}$, can be calculated. For each device i, the update is performed iteratively across multiple time slots, until the classification task is completed when the cumulative confidence level reaches a predefined threshold, η_T. If the threshold, η_T, is reached before or at the required task completion time limit, K_i, the QoS requirement of device i is satisfied; otherwise, a delay violation penalty is applied to the device. Let $P_i(k)$ denote the delay violation penalty of device i at the end of time slot k. The penalty is zero for $1 \leq k < K_i$. For $k \geq K_i$, if the required confidence level is not satisfied, i.e., $\eta_i(k) < \eta_T$, the penalty increases linearly with the number of time slots behind deadline. Let P be a constant denoting the unit penalty for each time slot with delay violation. Accordingly, we have

$$P_i(k) = \begin{cases} (k - K_i + 1)^+ P, & \text{if } \eta_i(k) < \eta_T \\ 0, & \text{otherwise.} \end{cases} \quad (16)$$

Computational Complexity: At a data pre-processing stage before the online execution of the cumulative DNN inference scheme, the joint probability density functions should be profiled using the KDE method. Consider an M-class training dataset including X training data samples for each class. Each of the fast and full DNN models should be executed by MX times to generate the subsets of fast and full inference results, which incurs an $O(MX)$ time complexity. For each class m, one KDE is required to profile function $\mathsf{f}_m^A(\boldsymbol{z})$, and one KDE is required to profile function $\mathsf{f}_m^U(\boldsymbol{z})$. Each KDE has a time complexity of $O(X^2)$ for kernel function evaluation and optimal kernel bandwidth selection, all depending on the number of data points, X [31]. Thus, the total time complexity for the KDE of both the fast and full inference results of all classes is $O(MX^2)$. For a large dataset, the computational complexity of KDE can be prohibitive, but there are efficient algorithms, such as the Fast Fourier Transform method, that can be used to speed up the computation [31]. When the cumulative DNN inference scheme is used online, an M-dimension cumulative DNN inference result is calculated in each time slot, where each element is calculated based on (15). Accordingly, the time complexity of the online cumulative DNN inference scheme in each time slot is $O(M)$.

IV. Joint Offloading and Resource Allocation for Multi-Device Cumulative DNN Inference

A full inference result has a higher average confidence level than a fast inference result, and more full inference results generally improve more on the cumulative confidence level once they are combined. Hence, device i tends to achieve a higher gain in the cumulative confidence level (i.e., confidence level gain) during time slot k for a larger value of $a_i(k)$, i.e., if more intermediate data samples are offloaded to the edge server and processed by enhanced DNN inference. For each device to reduce the delay violation penalty at task completion, offloading more intermediate data samples for edge computing is preferable in each time slot.

However, more offloading requires more resources for both transmission and edge computing. In this work, we focus on the fraction rather than the absolute value of total resource usage for both transmission and edge computing. The factions for both have the same range between 0 and 1. We do not put more emphasis on minimizing the fraction of either the transmission resource usage or the edge computing resource usage, but treat them equally. Define a network resource consumption ratio during time slot k, denoted by $\rho(k)$, which is a continuous decision variable in $[0, 1]$ representing an upper limit for the fractions of total transmission and edge computing resource usage during time slot k. Accordingly, we have

$$\sum_{i \in \mathcal{I}} \beta_i^T(k) \leq \rho(k) \text{ and } \sum_{i \in \mathcal{I}} \beta_i^E(k) \leq \rho(k). \quad (17)$$

By minimizing such an upper limit, both the fractions of total transmission and edge computing resource usage can be minimized. As the network resources are shared among multiple devices, only feasible offloading decisions under the network resource availability can be selected to guarantee the delay performance in each time slot. Specifically, for device i, in order to offload $a_i(k)$ intermediate data samples to the edge server and obtain $a_i(k)$ full inference results during time slot k, the total transmission and edge computing delay should not exceed time slot length τ, given by

$$a_i(k) \left[d_i^T(k) + d_i^E(k)\right] \leq \tau, \quad \forall i \in \mathcal{I}. \quad (18)$$

More offloading also requires more local transmission energy at the devices for transmitting more intermediate data samples to the edge server, while no offloading incurs local computing energy consumption for executing fast DNN inference. Let $\boldsymbol{\xi}(k) = \{\xi_i(k), \forall i \in \mathcal{I}\}$ be an auxiliary binary

decision variable set in time slot k, with $\xi_i(k) = 1$ indicating that at least one intermediate data sample is offloaded from device i, and $\xi_i(k) = 0$ otherwise. We have

$$\frac{a_i(k)}{\mathbb{M}} \leq \xi_i(k) \leq a_i(k), \quad \forall i \in \mathcal{I}. \tag{19}$$

Let $e_i(k)$ denote the local energy consumption at device i during time slot k, given by

$$e_i(k) = \xi_i(k) a_i(k) e_i^T(k) + [1 - \xi_i(k)]\, e_i^A, \quad \forall i \in \mathcal{I}. \tag{20}$$

Consider the total cost during time slot k, denoted by $c(k)$, as the linearly weighted summation of the total local energy consumption among all devices and the network resource consumption ratio, with weight $\omega_1 \in (0, 1)$, given by

$$\begin{aligned} c(k) &= \omega_1 \sum_{i \in \mathcal{I}} e_i(k) + (1 - \omega_1)\, \rho(k) \\ &= \omega_1 \sum_{i \in \mathcal{I}} \left\{ \xi_i(k) a_i(k) \frac{p_i C_i^{(2)}}{\beta_i^T(k) + \varepsilon} + [1 - \xi_i(k)]\, e_i^A \right\} \\ &\quad + (1 - \omega_1)\, \rho(k). \end{aligned} \tag{21}$$

To support the device-edge collaborative cumulative DNN inference for multiple devices with QoS satisfaction and cost efficiency, a joint offloading and resource allocation is necessary, to minimize the long-run total cost and the total delay violation penalty until the task confidence level requirements at all devices are satisfied, by jointly determining the integer offloading decision variables, $\boldsymbol{a}_k$, the binary auxiliary decision variables, $\boldsymbol{\xi}(k)$, and the continuous resource allocation decision variables, $\rho(k)$, $\boldsymbol{\beta}^T(k)$ and $\boldsymbol{\beta}^E(k)$, in each time slot k. The long-run joint offloading and resource allocation problem can be decoupled into a long-run adaptive offloading problem, and multiple instantaneous resource allocation subproblems in each time slot. Given an offloading decision vector, $\boldsymbol{a}_k$, for time slot k, a resource allocation subproblem in Subsection IV-A is solved to first check the feasibility and then find the optimal resource allocation solutions, $\rho^*(k)$, $\boldsymbol{\beta}^{T^*}(k)$, and $\boldsymbol{\beta}^{E^*}(k)$, with minimal cost $c^*(k)$. For the long-run adaptive offloading problem, a sequence of feasible offloading decisions are adaptively determined based on the evolving network status, by solving a Markov decision process (MDP) in Subsection IV-B with a reward function incorporating the minimal cost $c^*(k)$ by optimal resource allocation.

A. Resource Allocation Subproblem

Given an offloading decision vector, $\boldsymbol{a}_k$, for time slot k, $\boldsymbol{\xi}(k)$ is a known vector determined by (19). Then, a resource allocation optimization problem is formulated as

$$\begin{aligned} \min_{\boldsymbol{\beta}^T(k), \boldsymbol{\beta}^E(k), \rho(k)} \quad & c(k) \\ \text{s.t.} \quad & \beta_i^T(k), \beta_i^E(k), \rho(k) \in [0, 1], \quad \forall i \in \mathcal{I} && \text{(22a)} \\ & (7), (9), (17) && \text{(22b)} \\ & \frac{a_i(k) C_i^{(2)}}{\beta_i^T(k) + \varepsilon} + \frac{a_i(k) C_i^{(1)}}{\beta_i^E(k) + \varepsilon} \leq \tau, \ \forall i \in \mathcal{I}. && \text{(22c)} \end{aligned}$$

Constraint (22a) specifies the lower and upper limits for the decision variables. The delay requirement in (18) is rewritten as constraint (22c) with explicit resource allocation decision variables. Problem (22) is a convex optimization problem due to the convex objective function and all the convex inequality constraints. However, both the objective function and constraint (22c) involve the division by decision variables, which is not supported by an optimization solver such as Gurobi [32]. To solve the problem using a Gurobi optimization solver, we transform the problem to a second-order cone programming (SOCP) problem with zero optimality gap, by introducing two auxiliary continuous decision variable sets, $\boldsymbol{\psi}(k) = \{\psi_i(k), \forall i \in \mathcal{I}\}$ and $\boldsymbol{\delta}(k) = \{\delta_i(k), \forall i \in \mathcal{I}\}$, and one auxiliary continuous decision variable, ζ. An SOCP problem has a linear objective function and all the constraints are either linear or (rotated) second-order cone constraints, which is a convex problem that can be solved in polynomial time using interior point methods [33]. Let $C_i^{(3)} = \xi_i(k) a_i(k) p_i C_i^{(2)}$ and $C_i^{(4)} = [1 - \xi_i(k)]\, e_i^A$ be two constant values for device i given $\boldsymbol{a}_k$. Then, the SOCP problem is given by

$$\begin{aligned} \min_{\substack{\boldsymbol{\beta}^T(k), \boldsymbol{\beta}^E(k), \zeta \\ \boldsymbol{\psi}(k), \boldsymbol{\delta}(k), \rho(k)}} \quad & \omega_1 \sum_{i \in \mathcal{I}} \left[C_i^{(3)} \psi_i(k) + C_i^{(4)} \right] + (1 - \omega_1)\, \rho(k) \\ \text{s.t.} \quad & (22a), (22b) \\ & a_i(k) \left[C_i^{(2)} \psi_i(k) + C_i^{(1)} \delta_i(k) \right] \leq \tau, \ \forall i \in \mathcal{I} && \text{(23a)} \\ & \left[\beta_i^T(k) + \varepsilon \right] \psi_i(k) \geq \zeta^2, \quad \forall i \in \mathcal{I} && \text{(23b)} \\ & \left[\beta_i^E(k) + \varepsilon \right] \delta_i(k) \geq \zeta^2, \quad \forall i \in \mathcal{I} && \text{(23c)} \\ & \zeta = 1. && \text{(23d)} \end{aligned}$$

An optimum of problem (22) is either a unique optimum or one of multiple optimal solutions to SOCP problem (23). Specifically, second-order cone constraint (23b) must be active (i.e., achieving equality) in an SOCP optimum, and an SOCP optimum with inactive second-order cone constraint (23c) can always be mapped to another SOCP optimum with active constraint (23c), without affecting other constraints and the objective value. Relevant proofs involving the active/inactive status of second-order cone constraints can be found in [34].

The feasibility of each candidate offloading decision, $\boldsymbol{a}_k$, is checked by checking the feasibility of the SOCP problem. Let $\mathcal{A}$ denote the set of feasible offloading decisions. The minimal cost, $c^*(k)$, under each feasible offloading decision, $\boldsymbol{a}_k$, is precalculated by solving the SOCP problem.

B. Adaptive Offloading Problem

We formulate the adaptive offloading problem as an MDP, where the offloading decisions are adaptively determined for each device based on the evolving network status, to minimize the total cost in the long run while reducing the delay violation penalty. An MDP is represented as a tuple, $(\mathcal{S}, \mathcal{A}, P, R, \gamma)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, P is the state transition probability matrix, with $P(\boldsymbol{s}'|\boldsymbol{s}, \boldsymbol{a})$ denoting the probability distribution of next state $\boldsymbol{s}' \in \mathcal{S}$ given the current state $\boldsymbol{s} \in \mathcal{S}$ and action $\boldsymbol{a} \in \mathcal{A}$, $R : \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}$ is a reward function, and γ is a discount factor in $[0, 1)$. The goal of an MDP is to maximize an expected total discounted reward in the long run, by adaptively taking action $\boldsymbol{a} \in \mathcal{A}$

based on dynamic network state $\boldsymbol{s} \in \mathcal{S}$. The state, action, and reward of the MDP are given as follows.

- *State*: The cumulative confidence level, $\eta_i(k)$, for each device i shows an increasing trend with random fluctuations over time, as more fast or full inference results are combined. Hence, the dynamics in the cumulative confidence levels of all devices depend on the sequence of offloading decisions. An offloading decision at time slot k, which is to offload more intermediate data samples from a device whose current cumulative confidence level, $\eta_i(k)$, is low and the remaining time to deadline, $K_i - k$, is short, potentially brings more benefit in terms of reducing the total delay violation penalty. Therefore, the current cumulative confidence levels at each device, $\boldsymbol{\eta}_k$, and the current time slot index, k, should be considered in making adaptive offloading decisions over time. Moreover, the caching state at each device, $\boldsymbol{q}_k$, should be considered, as the number of offloaded intermediate data samples from a device should not exceed the number of intermediate data samples currently stored in the local cache. Accordingly, the state at time slot k, denoted by $\boldsymbol{s}_k$ is given by
$$\boldsymbol{s}_k = [\boldsymbol{q}_k, \boldsymbol{\eta}_k, k]; \tag{24}$$
- *Action*: The action at time slot k is the integer offloading decision vector during time slot k, i.e., $\boldsymbol{a}_k \in \mathcal{A}$, with the i-th element $a_i(k) \in \boldsymbol{a}_k$ representing the offloading decision for device i. Let $\mathcal{A}(\boldsymbol{s}) \subset \mathcal{A}$ denote a state-dependent feasible action subspace, which includes all actions satisfying constraint (2) at state $\boldsymbol{s}$. The action at time slot k should satisfy $\boldsymbol{a}_k \in \mathcal{A}(\boldsymbol{s}_k)$;
- *Reward*: To minimize the total delay violation penalty for all devices while minimizing the total cost, we consider a reward, r_k, for each time slot k as
$$r_k = -\exp\left(\omega_2 c^*(k)\right) - \sum_{i \in \mathcal{I}} P_i(k) \tag{25}$$
where ω_2 is a positive weight, $c^*(k)$ is the minimal cost obtained by solving the resource allocation subproblem, and $P_i(k)$ is the delay violation penalty for device i at the end of time slot k. We use an exponential function of $c^*(k)$ to increase the cost gaps among different offloading decisions and make r_k more sensitive to the action.

V. Deep Reinforcement Learning Solution

We propose a deep Q-learning algorithm presented in Algorithm 1 to solve the MDP using a reinforcement learning (RL) approach. Deep Q-learning adopts two deep Q networks (DQNs) with the same neural network structure, i.e., an evaluation DQN (Q) with weights $\boldsymbol{\theta}$ and a target DQN ($\hat{Q}$) with slowly updated weights $\hat{\boldsymbol{\theta}}$, as approximators for a state-action value function $Q(\boldsymbol{s}_k, \boldsymbol{a}_k) = \mathbb{E}\left[\sum_{k'=k}^{K-1} \gamma^{k'-k} r_{k'} | \boldsymbol{s}_k, \boldsymbol{a}_k\right]$, where K is the maximum number of learning steps in an episode. Both $\boldsymbol{\theta}$ and $\hat{\boldsymbol{\theta}}$ are randomly initialized before training (line 1) and then continually updated. Over every $K_{\boldsymbol{\theta}}$ learning steps, $\hat{\boldsymbol{\theta}}$ is replaced by $\boldsymbol{\theta}$ (line 15).

An RL agent running Algorithm 1 interacts with the intelligent IoT environment in a sequence of episodes. Each episode contains a finite and variable number of learning steps, one learning step for one time slot. An episode starts when the devices initiate a new group of classification tasks, and ends when the last device finishes its task with confidence level satisfaction. At the beginning of each episode, the state is initialized as $\boldsymbol{s}_1 = [\boldsymbol{q}_1, \boldsymbol{\eta}_1, 1] = [\boldsymbol{0}, \boldsymbol{0}, 1]$ (line 3). The total number of time slots in an episode can be smaller than $\max_{i \in \mathcal{I}} K_i$ if all tasks are finished before the required deadlines, in which case there is no delay violation penalty. It can also be larger than $\max_{i \in \mathcal{I}} K_i$ when there is delay violation penalty. Let u_k be a binary flag indicating if time slot k is the last time slot in the corresponding episode. If $u_k = 1$, the episode terminates at time slot k, and a *done* signal is generated by the environment.

At the beginning of time slot k within each episode, the RL agent observes environment state $\boldsymbol{s}_k$ and takes action $\boldsymbol{a}_k$ using an ϵ-greedy policy (line 5), i.e., $\boldsymbol{a}_k$ is given by either $\operatorname{argmax}_{\boldsymbol{a} \in \mathcal{A}(\boldsymbol{s}_k)} Q(\boldsymbol{s}_k, \boldsymbol{a})$ with probability $1 - \epsilon$ based on the evaluation DQN (Q), or a random action in $\mathcal{A}(\boldsymbol{s}_k)$ with probability ϵ. We use a gradually decreasing exploration probability, ϵ, from 1 to a minimum value ϵ_0, with decaying factor $\Delta_\epsilon \in (0, 1)$, to transit smoothly from exploration to exploitation (line 14). Then, the RL agent informs each device of the new offloading decision based on the action. At the end of time slot k, the agent calculates reward r_k for executing action $\boldsymbol{a}_k$, and observes new state $\boldsymbol{s}_{k+1}$. Specifically, the RL agent collects the new cumulative confidence levels, $\boldsymbol{\eta}_{k+1}$,

Algorithm 1: The proposed deep Q-learning Algorithm with extra replay memory

1 **Initialize:** $\boldsymbol{\theta}$ and $\hat{\boldsymbol{\theta}}$ for evaluation and target DQNs.
2 **for** *each episode* **do**
3 Initialize the state as $\boldsymbol{s}_1$.
4 **for** *each learning step* **do**
5 Observe current state $\boldsymbol{s}_k$, and select action $\boldsymbol{a}_k$ according to an ϵ-greedy policy.
6 Execute action $\boldsymbol{a}_k$, collect reward r_k and next state $\boldsymbol{s}_{k+1}$, and determine u_k (*done* signal).
7 Store transition $(\boldsymbol{s}_k, \boldsymbol{a}_k, r_k, \boldsymbol{s}_{k+1}, u_k)$ into the ordinary replay memory and temporary memory.
8 **if** *done* **then**
9 **if** *total penalty is zero* **then**
10 Pop out all transitions in the temporary memory to the extra replay memory.
11 Empty out the temporary memory.
12 Sample random mini-batches of transitions $(\boldsymbol{s}_n, \boldsymbol{a}_n, r_n, \boldsymbol{s}_{n+1}, u_n)$ from both the ordinary and extra replay memory.
13 Perform a gradient descent step on $\boldsymbol{\theta}$ based on $2N$ transitions according to (26).
14 Update ϵ as $\epsilon \times \Delta_\epsilon$, if $\epsilon > \epsilon_0$.
15 Every $K_{\boldsymbol{\theta}}$ steps, set $\hat{\boldsymbol{\theta}} = \boldsymbol{\theta}$.
16 **Output:** Trained evaluation and target DQNs.

which are updated based on the cumulative DNN inference scheme at each device, as illustrated in Fig. 2. The new caching state, $\boldsymbol{q}_{k+1}$, can be updated inside the RL agent based on (1). The $done$ signal, u_k, is checked at the end of time slot k to determine the episode termination (line 6). The signaling overhead between the RL agent and the devices is negligible due to a small signaling data size in comparison with the task data size. Then, the new transition $(\boldsymbol{s}_k, \boldsymbol{a}_k, r_k, \boldsymbol{s}_{k+1}, u_k)$ is added to both an ordinary replay memory which is adopted in traditional deep Q-learning algorithms and a temporary memory which is a new component in our algorithm (line 7). The temporary memory gradually gathers the transitions in each episode and is emptied out per episode (line 11). Only if the total delay violation penalty for all transitions in an episode is zero, the corresponding transitions in the temporary memory are popped out and stored in another new memory component, an extra replay memory, before the temporary memory is emptied out at the end of each episode (lines 9 and 10).

The per-episode updated extra replay memory, along with the temporary memory, is dedicated for handling the delay violation penalty which depends on all transitions from the beginning of an episode. The extra replay memory stores only transitions in the zero-penalty episodes, which are rare transitions with QoS satisfaction especially at the early learning stage. Note that a traditional prioritized experience replay which prioritizes the transitions with no instantaneous penalty cannot provide the desired property of zero episode-level penalty as the proposed extra replay memory [35], [36]. In our problem, all the transitions before task deadline in all episodes have no instantaneous delay violation penalty regardless of their reward, but the transitions after the task deadline have delay violation penalty. Most transitions with no instantaneous penalty should not be prioritized especially in the early learning stage, because they may lead to eventual delay violation at the end of the corresponding episodes. Only the transitions in an episode whose total penalty is zero should be prioritized, as in the proposed extra experience replay.

Then, for training the evaluation DQN via a gradient descent step on $\boldsymbol{\theta}$, a mini-batch of N experiences are sampled from the ordinary replay memory and a mini-batch of N experiences are sampled from the extra replay memory (lines 12 and 13), which improves the sampling frequency for the rare no-penalty transitions as compared with the traditional deep Q-learning algorithms. The gradient decent step is given by

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \mathbb{E}\left[\left(y_k - Q(\boldsymbol{s}_k, \boldsymbol{a}_k; \boldsymbol{\theta})\right) \nabla_{\boldsymbol{\theta}} Q(\boldsymbol{s}_k, \boldsymbol{a}_k)\right] \tag{26}$$

where α is the learning rate and y_k is a target value estimated by target DQN, given by

$$y_k = r_k + \gamma \max_{\boldsymbol{a} \in \mathcal{A}(\boldsymbol{s}_{k+1})} \hat{Q}(\boldsymbol{s}_{k+1}, a; \hat{\boldsymbol{\theta}}). \tag{27}$$

If an episode terminates at time slot k, y_k is set as r_k.

Computational Complexity: In the offline training stage, the time complexity depends on several factors, including the state and action spaces, the neural network size for the DQNs, and the number of iterations until convergence. The larger the state and action spaces, the larger the neural network size, and the more iterations required for convergence, the computational complexity for training the deep Q-learning algorithm increases. In the online execution stage, we use the trained DQN to select an optimal action with the largest Q value based on the current state. The time complexity for calculating the Q values depends on the state and action spaces, and the neural network size. Given the trained neural network, the computation through the layers of the DQN in each time slot incurs a pre-determined time overhead, which is much smaller than that for implementing the DNN inference of the classification tasks, due to a much smaller neural network size. Moreover, the reward calculation for each learning step in both the offline training and online execution stages depends on solving an SOCP problem, which has a polynomial time complexity [33].

TABLE II: System parameters in simulation

Parameters	Value
Bandwidth (B)	15 MHz
Noise power (σ^2)	-104 dBm
Transmit power (p_i)	20 dBm
Channel gain (g_i)	4×10^{-13}
Local CPU frequency (f_i)	0.45 GHz
Edge server CPU frequency (f_0)	20 GHz
Energy efficiency coefficient (κ_i)	10^{-28}
CPU cycles for a floating-point operation (φ)	4
Bits for a floating-point number (ϕ)	32

VI. Simulation Results

A. Simulation Setup

We set up a simulation model with three IoT devices, each being a smart camera, under the coverage of one AP. A classification task for each device is to classify a moving object under its surveillance by using multiple video frames (data samples). The devices have differentiated task completion time requirements, which are $[9, 11, 13]$ respectively in number of time slots. The devices have the same confidence level requirement, η_T, for their classification tasks. Other system parameters are given in Table II [30], [37], [38]. We use a typical video dataset `UCF101` integrated in Tensorflow, which contains videos capturing moving objects belonging to 101 classes. For simplicity, we select 600 videos belonging to 5 classes, referred to as `UCF5` video dataset. For each video, we extract the video frames at a frame sampling rate of 5 frames per second, and randomly select $J = 50$ video frames as data samples. All the extracted data samples from the `UCF5` video dataset constitute a 5-class image dataset. By randomly reordering the J data samples from each video by 100 times, we create 100 different sequences of data samples, each being a data trace based on which a classification task is performed. As such, 60000 classification tasks with different data traces are simulated. Note that the video frames are not disordered in a real IoT scenario. Here, we disorder the video frames to create more synthetic data traces for simulation.

Without loss of generality, we consider the same DNN models for all devices. The full and fast DNN model architecture and layer parameters (including layer input/output dimensions

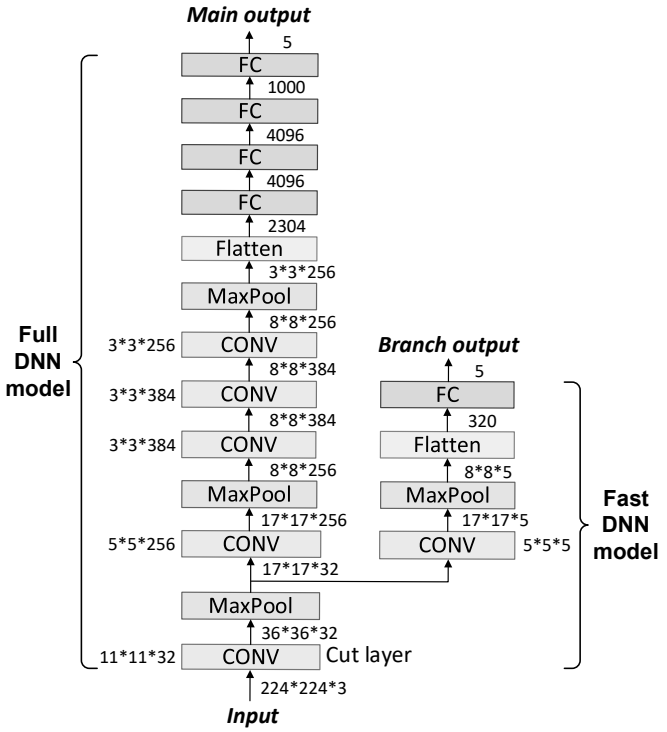


Fig. 3: Full and fast DNN model architecture and layer parameters.

and the filter size for each CONV layer) are shown in Fig. 3. The structure of the full DNN model is based on the well-known AlexNet model with slight difference customized to our use case [39]. The full DNN model consists of five CONV layers, three of which are followed by a MaxPool layer for data dimension reduction. The $3D$ output feature map of the last MaxPool layer is flatten to a $1D$ input to a sequence of FC layers. Compared with the original AlexNet model with default layer parameters, our full DNN model has two customized modifications. First, instead of applying 96 filters with size 11×11 to the input at the first CONV layer, we use 32 filters with size 11×11, to fit the IoT device with a limited computing capacity; second, we add an extra FC layer with 5 neurons at the model output to customize the AlexNet model to our 5-class image classification problem. The fast DNN model is composed of two CONV layers, two MaxPool layers, and one FC layer, where the first group of CONV and MaxPool layers are shared with the full DNN model. Due to the layer sharing property, the fast and full DNN models can be seen as a combined DNN model with one branch, as illustrated in Fig. 3, which can be trained as a whole by minimizing the combined loss of both model outputs based on the 5-class image dataset [24]–[26], [28]. The last FC layers at each output are activated with Softmax function to generate an estimated class probability vector, and all other CONV and FC layers are activated with Relu function. With the trained DNN models, we generate a set of fast inference results and a set of full inference results corresponding to the image dataset, based on which the joint probability density functions, $\mathsf{f}_m^A(\boldsymbol{z})$ and $\mathsf{f}_m^U(\boldsymbol{z})$ for each class m, are profiled by the KDE in Matlab.

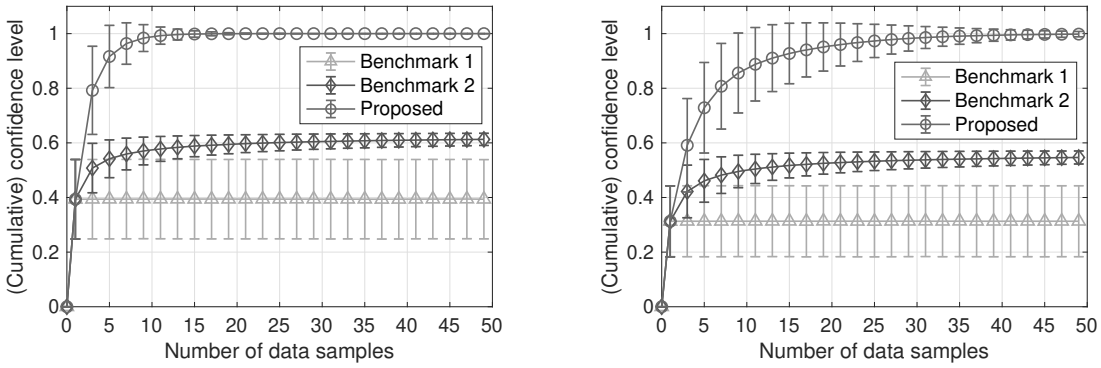
TABLE III: Parameters in deep Q-learning algorithm

Learning parameters	Value
Learning rate (α)	10^{-4}
Discount factor (γ)	0.85
Minimum exploration probability (ϵ_0)	0.01
Decaying factor for exploration probability (Δ_ϵ)	0.9995
Number of steps to replace $\hat{\boldsymbol{\theta}}$ by $\boldsymbol{\theta}$ ($K_{\boldsymbol{\theta}}$)	200
Memory size	2000
Batch size (N)	32

Under the simulation setup, we have constants $C_i^{(1)} = 0.02$ and $C_i^{(2)} = 0.115$, and set the time slot length as $\tau = d_i^A = 0.288s$. At most two intermediate data samples can be offloaded and finish the enhanced DNN inference during one time slot. Accordingly, the action space, $\mathcal{A}$, for the deep Q-learning algorithm includes 10 discrete offloading actions, i.e., $(0,0,0)$, $(0,0,1)$, $(0,0,2)$, $(0,1,0)$, $(0,1,1)$, $(0,2,0)$, $(1,0,0)$, $(1,0,1)$, $(1,1,0)$, $(2,0,0)$. Both the evaluation and target D$\mathcal{Q}$Ns have three hidden layers with $(128, 64, 32)$ neurons and Relu activation functions. We set weight $\omega_2 = 30$ and unit penalty $P = 400$ in the reward function. Other learning parameters are summarized in Table III. We evaluate the performance of the deep Q-learning algorithm for three different values of η_T among $\{0.93, 0.95, 0.97\}$. To evaluate the trade-off between local energy consumption and network resource consumption, we vary the value of weighting factor ω_1 in (21) among $\{0.90, 0.95, 0.99\}$. The three weights are all close to 1, but they place different priorities on the two costs. Both $\omega_1 = 0.90$ and $\omega_1 = 0.95$ place more priority on minimizing the network resource consumption, while $\omega_1 = 0.99$ places more priority on minimizing the local energy consumption. We use $\omega_1 = 0.90$ by default.

B. Performance Evaluation

We first evaluate performance of the proposed stochastic cumulative DNN inference scheme and compare with two benchmark DNN inference schemes. For each classification task, a DNN inference is performed for each data sample in the associated data trace containing J data samples, by using either the full or fast DNN model, which generates J different DNN inference results for the classification task. In the proposed scheme, stochastic cumulative DNN inference is performed by sequentially incorporating all the J DNN inference results based on (15). At the j-th ($1 \leq j \leq J$) data sample, the cumulative confidence level based on the first j data samples in the data trace is recorded. In the first benchmark scheme, no "cumulation" is considered, and the confidence level of the DNN inference result based on each single data sample in the data trace is reordered. In the second benchmark scheme, a straightforward approach is used to consider multiple data samples, which selects the DNN inference result with the maximum confidence level among the first j data samples and records the maximum confidence level until the j-th data sample as the cumulative confidence level at the j-th data sample. Fig. 4 shows the average (cumulative) confidence level for 60000 simulated classification tasks as



(a) Full DNN Inference (b) Fast DNN Inference

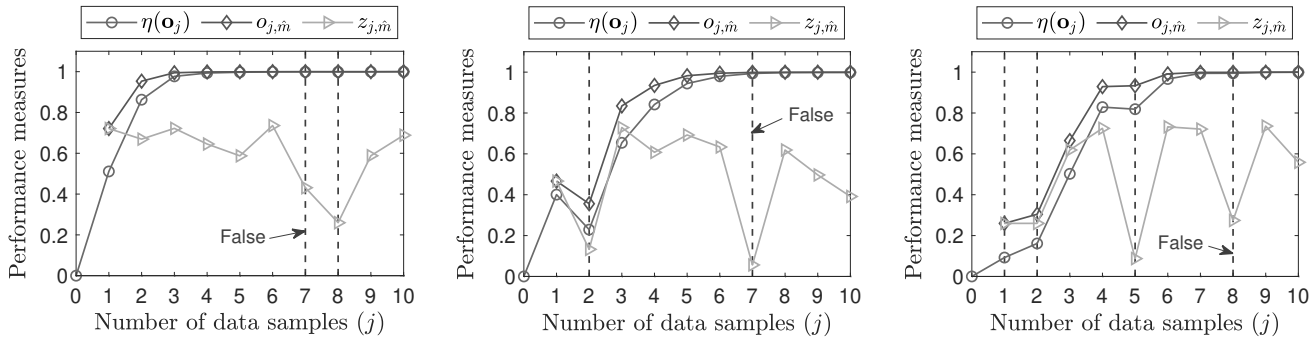
Fig. 4: Performance comparison between the proposed cumulative DNN inference scheme and benchmarks for both full and fast DNN inference.

the number of data samples increases, when all data samples are processed by either the full or fast DNN model, for both the proposed and benchmark DNN inference schemes. The standard deviations of the results are also plotted for reference.

Due to the lack of data diversity in each single data sample, the confidence level achieved by the first benchmark scheme without cumulation is low for both full and fast DNN inference, showing a constant average confidence level with a constant standard deviation across different data samples. Both the proposed scheme and the second benchmark scheme demonstrate the benefit of data diversity in increasing the cumulative confidence level for a classification task, as the average cumulative confidence level shows an increasing trend and the standard deviation shows a decreasing trend as more data samples are considered. However, as the number of data samples increases, in comparison with a gradual increase of the average cumulative confidence level to 1 by the proposed scheme, the average cumulative confidence level by the second benchmark scheme gradually converges to around 0.61 and 0.55 for the full and fast DNN inference respectively. In the proposed scheme, every newly added data sample contributes to the gradual increase of the cumulative confidence level in a stochastic manner, thus leading to an almost certain estimation when there is a sufficient number of data samples. However, although the second benchmark scheme takes the maximum confidence level among multiple data samples into consideration, the performance is still limited by the "best" confidence level of a single DNN inference result. As the number of data samples increases, the DNN inference results with the best confidence levels are gradually generated, and the maximal achievable average cumulative confidence level converges to the best confidence level.

Next, we focus on the performance of the proposed stochastic cumulative DNN inference scheme. We observe that the increasing speed of the average cumulative confidence level with full DNN inference is higher than that with fast DNN inference, demonstrating that the average confidence level gain with one more full inference result is larger than that with one more fast inference result. The average confidence level gain per inference shows a decreasing trend and gradually approaches zero. The standard deviation is large especially at a small number of data samples, which gradually decreases and approaches zero with more data samples, with a higher decreasing speed for full DNN inference. A large standard deviation captures the uncertainty in cumulative DNN inference, which is due to randomness in the DNN inference results.

As the cumulative DNN inference scheme sequentially incorporates different random DNN inference results corresponding to each data sample in a data trace, the relationship between the cumulative confidence level and the number of data samples changes for different data traces. Fig. 5 shows three different performance metrics for the proposed cumulative DNN inference scheme with full DNN inference as the number of data samples, j, increases in three different data traces. The performance metrics include 1) the cumulative confidence level until data sample j, i.e., $\eta\left(\boldsymbol{o}_j\right)$, 2) the estimated true class probability with single data sample j, i.e., $z_{j,\hat{m}}$, and 3) the cumulative estimated true class probability until data sample j, i.e., $o_{j,\hat{m}}$. Only the results for $j \leq 10$ are shown, as we observe in Fig. 4(a) that the average cumulative confidence level with full DNN inference is very high with small standard deviation at $j = 10$. The vertical dashed lines in Fig. 5 indicate the data sample positions with false full inference results. For a false inference result, $\boldsymbol{z}_j = \{z_{j,m}\}$, the class label with the maximum estimated probability is not the true class label $\hat{m}$, i.e., $\hat{m} \neq \arg\max_{m=1}^{M} z_{j,m}$, in which case $z_{j,\hat{m}}$ is usually small, as demonstrated in the result. Next, we look into each sub-figure corresponding to different data traces. In Fig. 5(a), $\eta\left(\boldsymbol{o}_j\right)$ keeps increasing from 0 until it approaches 1 with only four data samples and then remains stable, while $o_{j,\hat{m}}$ follows a similar trend as $\eta\left(\boldsymbol{o}_j\right)$. We also observe that $z_{j,\hat{m}}$ is around or above 0.6 for $j \leq 6$, indicating that each inference result for $j \leq 6$ provides correct inference with a good confidence level, which results in the initial fast increasing speed for $\eta\left(\boldsymbol{o}_j\right)$ and $o_{j,\hat{m}}$. Although there are two consecutive false inference results at $j = 7$ and $j = 8$, $\eta\left(\boldsymbol{o}_j\right)$ remains high, as there have been sufficient inference results providing correct inference with a good confidence level before $j = 7$. In Fig. 5(b), $\eta\left(\boldsymbol{o}_j\right)$ and $o_{j,\hat{m}}$ also show a roughly gradual increasing trend with one big drop at $j = 2$. The drop is due to a false inference result at $j = 2$. As there is only one correct inference result with good confidence level before $j = 2$, the false inference result

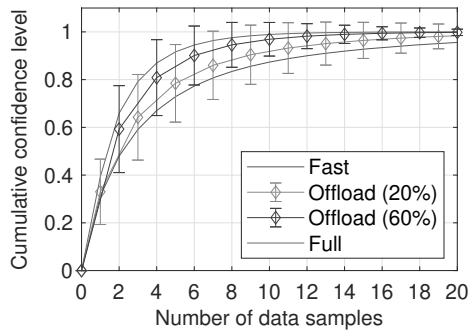


(a) Using Data Trace 1 (b) Using Data Trace 2 (c) Using Data Trace 3

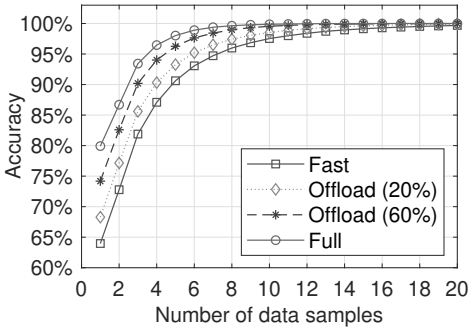
Fig. 5: Performance of cumulative DNN inference scheme with full DNN inference for three data traces.

at $j = 2$ brings a significant negative impact. To counteract the negative impact brought by false inference and to obtain a high cumulative confidence level, it requires more data samples to be aggregated. For data trace 2, four more data samples with correct inference and good confidence levels after $j = 2$ are aggregated to have $\eta(\boldsymbol{o}_6)$ approaching 1. An interesting observation is that the cumulative confidence level remains high although there is a subsequent false inference at $j = 7$, indicating that the cumulative DNN inference scheme is more robust to false inference if there have been a sufficient number of data samples and if the current cumulative confidence level is sufficiently high. In Fig. 5(c), the initial values of $\eta(\boldsymbol{o}_j)$ and $o_{j,\hat{m}}$ are small due to two consecutive false inferences at the beginning. After that, we see a gradual increasing trend for both $\eta(\boldsymbol{o}_j)$ and $o_{j,\hat{m}}$. The false inference at $j = 5$ brings a small and almost negligible drop in $\eta(\boldsymbol{o}_j)$, and the false inference result at $j = 8$ does not degrade the performance, demonstrating the robustness of the cumulative DNN inference scheme with a sufficient number of "good" data samples.

For a classification task, define the offloading ratio as the fraction of full inference results among all DNN inference results that are combined using the cumulative DNN inference scheme. Fig. 6 shows two performance metrics, i.e., the average cumulative confidence level and the accuracy, for 60000 simulated classification tasks as the number of data samples increases, under a different offloading ratio at 0% (i.e., pure fast DNN inference), 20%, 60%, and 100% (i.e., pure full DNN inference). Here, the accuracy is defined as the average ratio of correct inference among all the simulated classification tasks. The standard deviations of the cumulative confidence levels for offloading ratios at 20% and 60% are also plotted for reference. With more data samples, both performance metrics gradually increase to one (100%), with a larger increasing speed for a larger offloading ratio, demonstrating the benefit of offloading in terms of both confidence level and accuracy improvement. As the confidence level represents uncertainty in a DNN inference result rather than its accuracy, a single DNN inference result with high confidence level is possible to be false, if the estimated probability for a wrong class is high. However, if the cumulative confidence level is high, it is highly likely that the cumulative DNN inference result is accurate, as the estimated true class probability is improved



(a) Cumulative Confidence Level



(b) Accuracy

Fig. 6: Performance of the cumulative DNN inference scheme versus number of data samples for different offloading ratios.

by aggregating more data samples. Hence, the cumulative DNN inference reduces the uncertainty in prediction results and increases the robustness to randomness and even false detections in the DNN inference results.

Next, we evaluate the performance of the proposed deep Q-learning algorithm for adaptive offloading decisions. To determine the state transitions for the cumulative confidence levels, we use the average cumulative confidence level traces obtained from the cumulative DNN inference scheme. Fig. 7 shows the convergence performance of the proposed deep Q-learning algorithm in terms of the training loss versus the number of learning steps at $\omega_1 = 0.90$, for different confidence level requirements (η_T). As mentioned, weight $\omega_1 = 0.90$ places more emphasis on minimizing the network resource

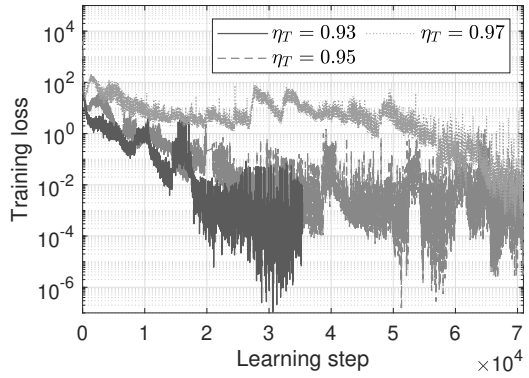


Fig. 7: Training loss for different confidence level requirements at $\omega_1 = 0.90$.

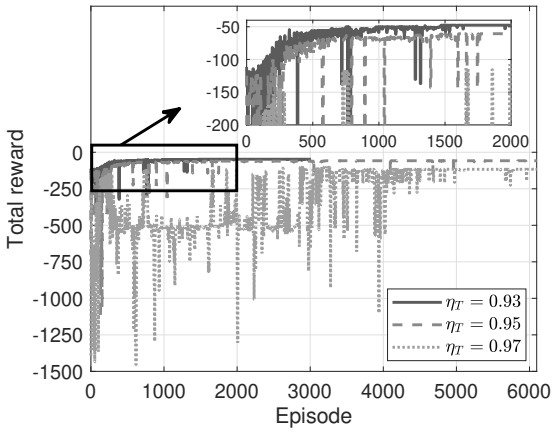


Fig. 8: Episodic total reward versus training episode for different confidence level requirements at $\omega_1 = 0.90$.

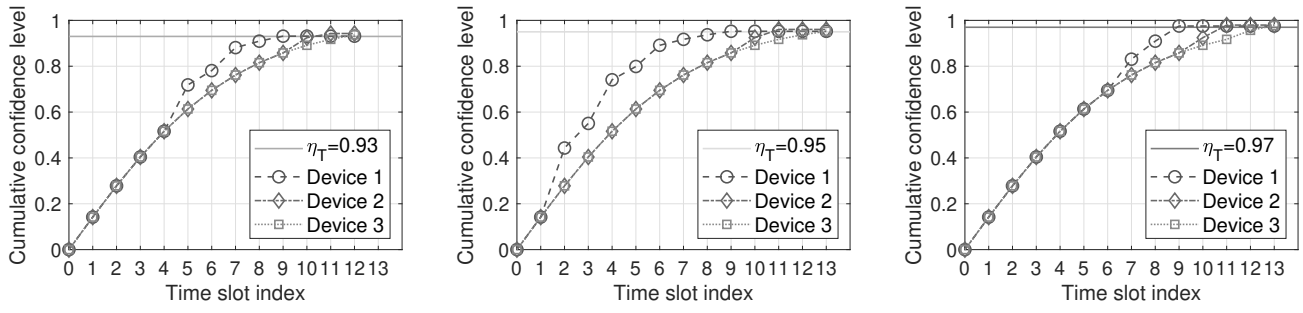
consumption than local energy consumption. In this case, it is preferable to execute more fast DNN inference locally to minimize the total cost. As each fast DNN inference tends to provide less confidence level gain and at most one fast inference result can be generated at each device in one time slot, it requires more time slots to satisfy the confidence level requirement. Hence, if η_T is large, it is more difficult for the RL agent to learn the no delay violation penalty behavior corresponding to confidence level satisfaction before deadline. Accordingly, with a larger value of η_T, it is more difficult for the learning algorithm to converge and it takes a longer time to reduce the training loss to below 10^{-2}. For example, the training loss for $\eta_T = 0.93$ is quickly reduced to below 10^{-5} at around 30000 learning steps, while the training loss for $\eta_T = 0.97$ is slowly reduced to below 10^{-2} in a more than doubled training time. For different confidence level requirements, we can see that, although the training loss is very small after convergence, there is still randomness in it due to the random experience sampling in different learning steps. Specifically, the training loss in each learning step is the average loss after a gradient descent step for a randomly sampled mini-batch of experiences from the replay memory.

Fig. 8 shows the episodic total reward during the training process at $\omega_1 = 0.90$, for different values of η_T. It is observed that the total reward for $\eta_T = 0.93$ increases most quickly and converges at around 1700 episodes without delay violation penalty (indicated by negative glitches in the episodic total reward). In comparison, the total reward for $\eta_T = 0.95$ increases in a slightly slower speed and converges after 2000 episodes. The total reward for $\eta_T = 0.97$ has the worst convergence, with huge delay violation penalty before 2000 episodes and significant fluctuations between episodes 2000 and 5000. It finally converges after around 5000 episodes. After convergence, the delay violation penalty is suppressed, and a larger total reward (or a lower total cost) is obtained for a lower confidence level requirement.

Fig. 9 shows the cumulative confidence levels over time for the three devices with the trained RL agents at different values of η_T for $\omega_1 = 0.90$. We observe that the confidence level requirements for all the devices are satisfied at (or very close to) the required deadlines, i.e., in $[9, 11, 13]$ time slots. As local computing incurs less cost than offloading but has lower confidence level gain for $\omega_1 = 0.90$, the RL agent learns an offloading decision sequence with minimal offloading that satisfies the confidence level requirements without delay violation and with minimum cost. An intuitive solution which always prioritizes offloading cannot provide such intelligence in terms of cost minimization. The trained RL agent also learns how to prioritize the offloading opportunities among the three devices with different delay requirements. For device 1 with the most stringent delay requirement, the cumulative confidence level increases faster than the other two devices due to earlier offloading opportunities.

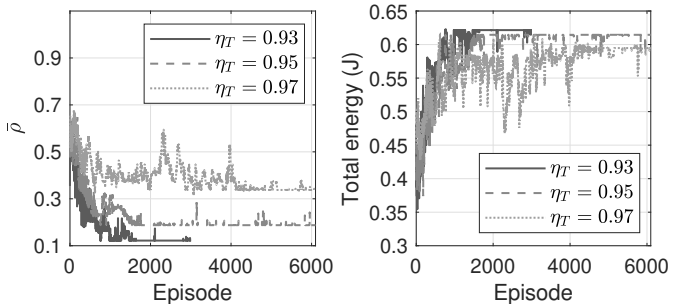
Due to the priority on minimizing the network resource consumption at $\omega_1 = 0.90$, we see in Fig. 10 that the episodic average network resource consumption ratio ($\bar{\rho}$) decreases with more learning episodes, while the episodic total local energy shows an increasing trend, demonstrating a trade-off between the two metrics. For a smaller value of η_T, the average resource consumption is lower as less offloading is triggered to satisfy the QoS requirement, but the total energy is higher due to a larger local computing energy for one fast DNN inference than the transmission energy for offloading one intermediate data sample under the simulation setup. Fig. 11 further demonstrates the trade-off between energy and resource consumption with different weight ω_1 at a constant confidence level requirement $\eta_T = 0.95$. For both $\omega_1 = 0.90$ and $\omega_1 = 0.95$, we see a decreasing trend in average resource consumption and an increasing trend in total local energy over the training process. However, we see an opposite trend for a large weight $\omega_1 = 0.99$, as energy becomes dominant in the cost. For a smaller ω_1 value, the average resource consumption is lower and the total energy is higher after convergence.

To evaluate the benefit of the extra experience replay in the proposed deep Q-learning algorithm, we compare the episodic (smoothed) total reward during the training process of the proposed algorithm and a benchmark Q-learning algorithm without extra experience replay at $\omega_0 = 0.90$ and $\eta_T = 0.97$, as shown in Fig. 12. A mini-batch of $2N$ experiences are sampled from the ordinary replay memory for training at each learning step in the benchmark. We observe that the total reward in the proposed algorithm converges after around 5000



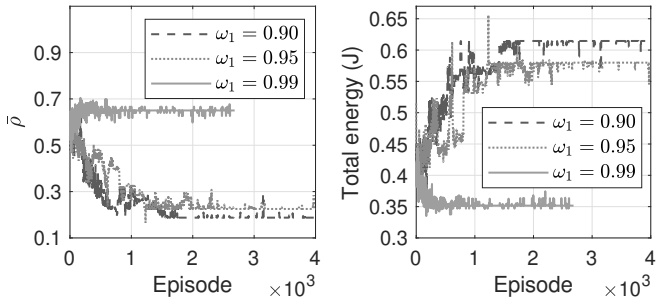
(a) $\eta_T = 0.93$ (b) $\eta_T = 0.95$ (c) $\eta_T = 0.97$

Fig. 9: The increase of cumulative confidence levels over time at different confidence level requirements.



(a) Average Resource Consumption (b) Total Local Energy

Fig. 10: Cost comparison with different values of η_T for $\omega_1 = 0.90$.



(a) Average Resource Consumption (b) Total Local Energy

Fig. 11: Cost comparison with different weight ω_1 for $\eta_T = 0.95$.

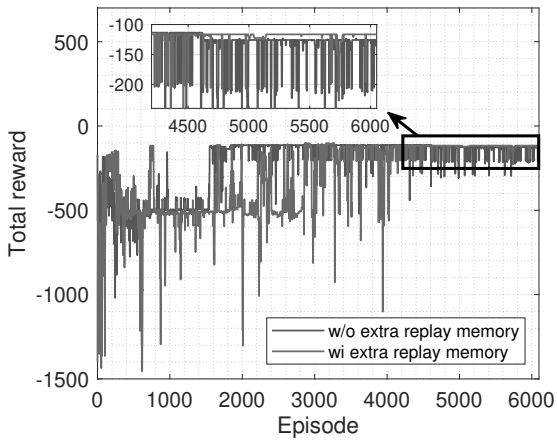


Fig. 12: Comparison of episodic total reward versus training episode with and without extra experience replay.

episodes with no penalty at most time instants, while the penalty in the benchmark is still large after convergence. We also observe that the total reward in the benchmark increases earlier due to more training with diverse training experiences in the early training stage. In the proposed algorithm, the sampled experiences from the extra replay memory lack diversity in the early training stage, as the episodes with no penalty are rare and the number of samples in the extra replay memory increases slowly. As a result, the training with sampled experiences from the extra replay memory does not fully explore the state action space at the early training stage. However, after the extra replay memory stores sufficient good samples, the total reward finally converges to a larger value with negligible delay violation penalty. For the benchmark, the earlier convergence to a worse solution is because it cannot put priority on remembering and learning from the special good samples in the no-penalty episodes. In this case, all the samples have equal priority, and are gradually replaced by new samples once the memory is full. To summarize, we can see that the advantages of the extra replay memory in improving the sampling efficiency towards the almost no-penalty convergence results outweigh its disadvantages in the slower reward increase at the early training stage.

VII. Conclusion

In this paper, we develop a device-edge collaborative inference framework in an edge-assisted multi-device intelligent IoT scenario. A data-driven stochastic DNN inference scheme is used at each device to improve the confidence level for DNN-based classification tasks, while an RL-aided adaptive device-edge collaboration scheme supports the cumulative DNN inference among multiple devices with QoS satisfaction and cost efficiency. Simulation results demonstrate the effectiveness of the proposed inference framework including both the cumulative DNN inference scheme and the adaptive device-edge collaboration scheme. The former provides a theoretical foundation for improving the confidence level of general DNN-based classification tasks by combining multiple DNN inference results under the performance limit of DNN models, which is potentially useful in AI applications with an extremely high reliability requirement such as those in remote

surgery and autonomous driving. The latter demonstrates the benefit of hierarchical computing (e.g., across the device and edge hierarchy) in terms of cost efficiency and delay satisfaction, which potentially can be explored for ultra-reliable low-latency applications in the future wireless networks.

In the future work, we will extend the cumulative DNN inference scheme from the time domain to the space domain. For example, when a group of spatially distributed sensors provide multiple data sources for one classification task, a spatial cumulative DNN inference scheme is required, and the sensor access control to AP and edge server is important in the sensing data collection. We will also consider more general application scenarios, extending from the device-edge collaboration to the device-edge-cloud and edge-edge collaboration for network-wide efficient resource utilization.

REFERENCES

[1] X. Shen, J. Gao, W. Wu, M. Li, C. Zhou, and W. Zhuang, "Holistic network virtualization and pervasive network intelligence for 6G," *IEEE Commun. Surv. Tutor.*, vol. 24, no. 1, pp. 1–30, 2021.

[2] G. Ananthanarayanan, P. Bahl, P. Bodík, K. Chintalapudi, M. Philipose, L. Ravindranath, and S. Sinha, "Real-time video analytics: The killer app for edge computing," *Computer*, vol. 50, no. 10, pp. 58–67, 2017.

[3] A. H. Jiang, D. L.-K. Wong, C. Canel, L. Tang, I. Misra, M. Kaminsky, M. A. Kozuch, P. Pillai, D. G. Andersen, and G. R. Ganger, "Mainstream: Dynamic stem-sharing for multi-tenant video processing," in *2018 USENIX Annual Technical Conf. (USENIX ATC '18)*, 2018, pp. 29–42.

[4] S. Wang, S. Yang, and C. Zhao, "Surveiledge: Real-time video query based on collaborative cloud-edge deep learning," in *Proc. IEEE INFOCOM'20*, 2020, pp. 2519–2528.

[5] 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. Veh. Tech.*, vol. 70, no. 8, pp. 7605–7618, 2021.

[6] 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. Surv. Tutor.*, vol. 25, no. 1, pp. 591–624, 2023.

[7] E. Arnold, M. Dianati, R. de Temple, and S. Fallah, "Cooperative perception for 3D object detection in driving scenarios using infrastructure sensors," *IEEE Trans. Intell. Transp. Syst.*, vol. 23, no. 3, pp. 1852–1864, 2022.

[8] J. Chen and X. Ran, "Deep learning with edge computing: A review." *Proc. IEEE*, vol. 107, no. 8, pp. 1655–1674, 2019.

[9] Y. Shi, K. Yang, T. Jiang, J. Zhang, and K. B. Letaief, "Communication-efficient edge AI: Algorithms and systems," *IEEE Commun. Surv. Tutor.*, vol. 22, no. 4, pp. 2167–2191, 2020.

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

[11] 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, 2019.

[12] Y. Kang, J. Hauswald, C. Gao, A. Rovinski, T. Mudge, J. Mars, and L. Tang, "Neurosurgeon: Collaborative intelligence between the cloud and mobile edge," *ACM SIGARCH Comput. Architect. News*, vol. 45, no. 1, pp. 615–629, 2017.

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

[14] Y. Xiao, L. Xiao, K. Wan, H. Yang, Y. Zhang, Y. Wu, and Y. Zhang, "Reinforcement learning based energy-efficient collaborative inference for mobile edge computing," *IEEE Trans. Commun.*, vol. 71, no. 2, pp. 864 – 876, 2023.

[15] M. Xu, F. Qian, M. Zhu, F. Huang, S. Pushp, and X. Liu, "DeepWear: Adaptive local offloading for on-wearable deep learning," *IEEE Trans. Mob. Comput.*, vol. 19, no. 2, pp. 314–330, 2020.

[16] C. Hu, W. Bao, D. Wang, and F. Liu, "Dynamic adaptive DNN surgery for inference acceleration on the edge," in *Proc. IEEE INFOCOM'19*, 2019, pp. 1423–1431.

[17] M. Xue, H. Wu, G. Peng, and K. Wolter, "DDPQN: An efficient DNN offloading strategy in local-edge-cloud collaborative environments," *IEEE Trans. Serv. Comput.*, vol. 15, no. 2, pp. 640–655, 2021.

[18] 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 Trans. Green Commun.*, vol. 6, no. 1, pp. 248–264, 2021.

[19] W. Wu, M. Li, K. Qu, C. Zhou, X. Shen, W. Zhuang, X. Li, and W. Shi, "Split learning over wireless networks: Parallel design and resource management," *IEEE J. Sel. Areas Commun.*, vol. 41, no. 4, pp. 1051–1066, 2023.

[20] 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. Netw.*, vol. 29, no. 2, pp. 595–608, 2021.

[21] Z. Zhao, K. M. Barijough, and A. Gerstlauer, "Deepthings: Distributed adaptive deep learning inference on resource-constrained IoT edge clusters," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 37, no. 11, pp. 2348–2359, 2018.

[22] S. Zhang, S. Zhang, Z. Qian, J. Wu, Y. Jin, and S. Lu, "Deepslicing: collaborative and adaptive CNN inference with low latency," *IEEE Trans. Parallel. Distrib. Syst.*, vol. 32, no. 9, pp. 2175–2187, 2021.

[23] R. Hadidi, J. Cao, M. S. Ryoo, and H. Kim, "Toward collaborative inferencing of deep neural networks on internet-of-things devices," *IEEE Internet Things J.*, vol. 7, no. 6, pp. 4950–4960, 2020.

[24] S. Teerapittayanon, B. McDanel, and H.-T. Kung, "Branchynet: Fast inference via early exiting from deep neural networks," in *Proc. IEEE ICPR'16*, 2016, pp. 2464–2469.

[25] ——, "Distributed deep neural networks over the cloud, the edge and end devices," in *Proc. IEEE ICDCS'17*, 2017, pp. 328–339.

[26] Y. Matsubara, M. Levorato, and F. Restuccia, "Split computing and early exiting for deep learning applications: Survey and research challenges," *ACM Comput. Surv.*, vol. 55, no. 5, pp. 1–30, 2022.

[27] B. Yang, X. Cao, C. Yuen, and L. Qian, "Offloading optimization in edge computing for deep-learning-enabled target tracking by internet of UAVs," *IEEE Internet Things J.*, vol. 8, no. 12, pp. 9878–9893, 2020.

[28] E. Li, L. Zeng, Z. Zhou, and X. Chen, "Edge AI: On-demand accelerating deep neural network inference via edge computing," *IEEE Trans. Wirel. Commun.*, vol. 19, no. 1, pp. 447–457, 2020.

[29] T. Mohammed, C. Joe-Wong, R. Babbar, and M. Di Francesco, "Distributed inference acceleration with adaptive DNN partitioning and offloading," in *Proc. IEEE INFOCOM'20*, 2020, pp. 854–863.

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

[31] V. C. Raykar and R. Duraiswami, "Fast optimal bandwidth selection for kernel density estimation," in *Proc. 2006 SIAM International Conf. on Data Mining*. SIAM, 2006, pp. 524–528.

[32] "Gurobi Optimizer Reference Manual," http://www.gurobi.com, 2022, [Online; accessed 26-March-2023].

[33] E. D. Andersen, C. Roos, and T. Terlaky, "On implementing a primal-dual interior-point method for conic quadratic optimization," *Mathematical Programming*, vol. 95, pp. 249–277, 2003.

[34] K. Qu, W. Zhuang, Q. Ye, X. Shen, X. Li, and J. Rao, "Dynamic flow migration for embedded services in SDN/NFV-enabled 5G core networks," *IEEE Trans. Commun.*, vol. 68, no. 4, pp. 2394–2408, 2020.

[35] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," in *Proc. ICLR'16*, May 2016, pp. 1–7.

[36] K. Qu, W. Zhuang, X. Shen, X. Li, and J. Rao, "Dynamic resource scaling for VNF over nonstationary traffic: A learning approach," *IEEE Trans. Cogn. Commun. Netw.*, vol. 7, no. 2, pp. 648–662, 2020.

[37] 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. Industr. Inform.*, vol. 17, no. 7, pp. 4988–4998, 2020.

[38] Q. Ye, W. Shi, K. Qu, H. He, W. Zhuang, and X. Shen, "Joint RAN slicing and computation offloading for autonomous vehicular networks: a learning-assisted hierarchical approach," *IEEE Open J. Veh. Technol.*, vol. 2, pp. 272–288, 2021.

[39] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems (NIPS'12)*, vol. 25, pp. 1097–1105, 2012.