IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received February 15, 2021, accepted March 15, 2021, date of publication April 2, 2021, date of current version April 14, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3070627

RL-PDNN: Reinforcement Learning for Privacy-Aware Distributed Neural Networks in IoT Systems

EMNA BACCOUR[1], AIMAN ERBAD[1], (Senior Member, IEEE), AMR MOHAMED[2], (Senior Member, IEEE), MOUNIR HAMDI[1], (Fellow, IEEE), AND MOHSEN GUIZANI[2], (Fellow, IEEE)

[1]Division of Information and Computing Technology, College of Science and Engineering, Hamad Bin Khalifa University, Qatar Foundation, Doha, Qatar
[2]CSE Department, College of Engineering, Qatar University, Doha, Qatar

Corresponding author: Emna Baccour (ebaccourepbesaid@hbku.edu.qa)

ABSTRACT Due to their high computational and memory demand, deep learning applications are mainly restricted to high-performance units, e.g., cloud and edge servers. Particularly, in Internet of Things (IoT) systems, the data acquired by pervasive devices is sent to the computing servers for classification. However, this approach might not be always possible because of the limited bandwidth and the privacy issues. Furthermore, it presents uncertainty in terms of latency because of the unstable remote connectivity. To support resource and delay requirements of such paradigm, joint and real-time deep co-inference framework with IoT synergy was introduced. However, scheduling the distributed, dynamic and real-time Deep Neural Network (DNN) inference requests among resource-constrained devices has not been well explored in the literature. Additionally, the distribution of DNN has drawn the attention to the privacy protection of sensitive data. In this context, various threats have been presented, including white-box attacks, where malicious devices can accurately recover received inputs if the DNN model is fully exposed to participants. In this paper, we introduce a methodology aiming at distributing the DNN tasks onto the resource-constrained devices of the IoT system, while avoiding to reveal the model to participants. We formulate such an approach as an optimization problem, where we establish a trade-off between the latency of co-inference, the privacy of the data, and the limited resources of devices. Next, due to the NP-hardness of the problem, we shape our approach as a reinforcement learning design adequate for real-time applications and highly dynamic systems, namely RL-PDNN. Our system proved its ability to outperform existing static approaches and achieve close results compared to the optimal solution.

INDEX TERMS IoT devices, distributed DNN, privacy, white-box, resource constraints, DQN.

I. INTRODUCTION

The deep neural networks represent the core technique for a wide spectrum of applications, including computer vision [2] and natural language processing [1]. The prevalence and achievements of DNN learning are related to the deep structure of the adopted neural network. A typical DNN network may contain tens of layers and hundreds of neurons, and these parameters can even scale to reach millions of nodes. A prominent state-of-the-art DNN is VGG network [3], constructed of convolutional layers and deep architecture, which results in an outstanding performance for many applications, particularly image processing. Such DNN-based applications are typically characterized by a high memory occupation and computational load, which restricts their implementation in low cost and limited energy devices and their deployments in powerful machines. In pervasive systems, relying on resource-constrained IoT devices, the deep learning tasks (e.g., image processing) require data gathering and transmission from sensors (e.g,. cameras and microphones) to remote servers for processing. However, this approach might not be possible if the data gathering-to-classification latency does not match the real-time requirements of the applications, specifically those that need prompt interventions. Moreover, the system stability highly depends on the availability of remote connectivity between IoT units and

The associate editor coordinating the review of this manuscript and approving it for publication was Junggab Son.

computing servers, resulting in a fluctuating latency. Additionally, in surveillance systems, as IoT devices send high-resolution images to remote servers at small intervals of time despite the rare occurrence of events of interest, the large data volume transmitted by source units becomes problematic and may result in frequent congestion and higher network cost.

Because of tremendous data shared with remote servers and the volatile latency to obtain the classification results, AI processing should be moved as close as possible to data-generating devices. However, the embedded resource-constrained IoT systems fail to deploy large-scale computation with reasonable latency and energy consumption. To support DNN processing in IoT devices, the design of deep learning solutions has been completely rethought, while considering hardware and physical constraints. More specifically, in order to fit the requirements of DNN into the resource-constrained devices, collaborative deep learning strategies have been recently proposed in the literature. The basic idea is to divide the DNN model into segments (e.g., layers and multiplication tasks), and each segment is allocated within a participant. Each participant shares the output to the next participant until generating the final prediction. Efforts dealing with DNN partitioning can be classified into three categories: The first approach is to offload a fraction of the DNN network to remote servers. In such partitioning approaches, shallow layers are computed in constrained devices, while deep layers leverage the cloud [19]. In this way, intermediate transmitted data is minimized. A second partition strategy is the hierarchical architecture, where the offloading is accomplished across cloud, edge servers, and mobile devices [21]. These strategies consider mainly the delegation of DNN fractions from resource-constrained devices to more powerful servers. Still, bottleneck scenarios can occur in case of high classification demands. Recently, researchers have investigated the feasibility of leveraging the resources of IoT devices to jointly allocate different segments of deep neural networks and execute the classification at the vicinity of the source device [5]–[7].

These aforementioned works mainly studied the optimal partition strategy that reduces the transmitted data and the dependency between inference participants. Additionally, the proposed approaches are based on static distribution of the model, without taking into account the dynamics of the system in terms of online classification requests. Rather, offloading network segments needs to be conducted in real-time to consider the distribution of the requests arriving to the network, the dynamics of participants (participants joining and leaving IoTs, etc.) and their capacity to handle multiple parallel inference tasks. To summarize, scheduling the IoT devices to perform real-time DNN tasks, while respecting their available memory and computation capabilities, was not explored by existing efforts. Furthermore, the distribution of deep learning technologies has underlined the privacy issues of sensitive data. In fact, when splitting the trained model and distributing its segments among different participants, a malicious device can recover the input received from the previous participant. Authors in [26] proposed different attacks against collaborative inference, which include white-box attacks. In white-box settings, the untrusted participant has complete knowledge about the model (e.g., weights, biases, and neurons) and attempts to map the received data to the initial input in order to obtain the original data. Thus, exposing only few segments to each participant should be considered in the distribution process to enhance the security of the system against white-box attacks.

In this paper, we study the distribution of Convolutional Neural Networks (CNNs), as they have demonstrated unprecedented efficiency in image and video classification. To match resource-consuming DNN solutions with the constraints of IoT devices, we exploit the hierarchical design characterizing deep learning models in order to suitably place layers of dynamic incoming classification requests. Particularly, we propose an approach that receives as an input the set of CNN classification requests (whose model was previously trained) and the technological characteristics of the IoT participants, and provides as an output the optimal placement of inferences' layers among the participants, while having as an objective to minimize the classification latency and cover the maximum of the model parameters from the untrusted devices. The contributions of our paper are presented as follows:

- We formulate our privacy-aware collaborative inference as an optimization problem that aims to minimize the classification latency, while respecting the required white-box privacy by covering the structure of the model from untrusted participants.
- To relax the optimization problem, we propose a novel approach based on Reinforcement Learning (RL) for privacy-aware distributed CNN networks, namely RL-PDNN. This approach learns the allocation policy and takes real-time actions based on the available resources, the dynamics of requests, and the required privacy level. In this context, we define the set of states, actions, and reward function, and we use an efficient RL-approach: Deep Q-Learning (DQN).
- We conduct extensive simulations to evaluate the performance of the RL-PDNN predictive model. Additionally, we demonstrate that our RL-approach presents close results to the optimal solution, and better results compared to existing approaches.

Our paper is organized as follows: Section II explores some related works. Section III presents our proposed framework, the problem formulation and the RL-PDNN approach. The evaluation of our approach is conducted in section IV, using three state-of-the-art CNN networks. Finally, we present the conclusion and future works in section V.

II. RELATED WORKS

Deep Neural Networks have become widely ubiquitous owing to their ability to revolutionize a large variety of applications in many research fields, including image recognition. Such high performance of DNN systems is related to the

number of layers and the complex structure constituting the network. A typical DNN network is composed of a series of connected layers, where each layer is designed to execute a defined task on the input data and to generate a specific output. More precisely, some DNNs comprise tens of layers associated with thousands of neurons, incurring a computation and a memory footprint of a terabyte of floating point operations per second (flops) [27], [28]. As an example, VGG is composed of more than 130 million parameters, resulting in high performance in a variety of applications [3]. Another example of CNNs applied mainly to image recognition is AlexNet [9]. This model comprises 60 million parameters, making it able to revolutionize visual recognition challenges.

Because of their intensive computational and memory demands, deep learning tasks have mainly been assigned to highly powerful machines requiring long transmission time and stable connectivity, e.g., cloud/edge servers. In this context, several efforts have been conducted to fit deep neural networks to resource-constrained devices from computation, memory, or hardware perspective. Particularly, a promising approach that has been proposed in the literature, is compression, where the main idea is to reduce the number of parameters of the deep network in order to decrease the memory demand and the execution time. Among the most popular compression techniques designed for resource-constrained IoT devices is pruning, which consists of eliminating parameters (e.g. weights and channels) with minor importance [10]–[12]. Binarization [15] is another compression method that uses binary values to design weights and activation parameters. Another research direction aiming at accelerating the learning process is the design of custom integrated circuits dedicated for deep learning tasks, including the tensor processing unit (TPU) of Google [16]. Finally, many libraries and software tools are commercialized to parallelize the DNN computation (e.g. matrix multiplications), such as RSTensorFlow [17] and DeepX [18]. Even though the aforementioned techniques presented high-performance solutions that enable to integrate deep learning networks in pervasive IoTs, many of them cannot be supported by all types of devices neither all types of data. Additionally, most of these approaches compromise the classification accuracy in order to compact the deep models, which makes them inadequate for applications requiring precision and reliability of the inference tasks (e.g., critical areas monitoring).

The wide prevalence and connection between IoT devices paved the path for the research community to propose collaborative deep learning strategies in order to reduce bandwidth bottlenecks, incur less latency, and preserve the accuracy of the model [23], [24]. More specifically, collaborative deep learning approaches consist of partitioning the DNN network into segments and assigning different tasks to the IoT participants intending to collaborate in the distributed system. Efforts exploring DNN partitioning can be classified into three categories: The first approach proposes to offload a fraction of the DNN network to be computed in edge or cloud servers. As the network goes deeper, the intermediate outputs become relatively small, which makes the offloading over the network less bandwidth consuming. Therefore, it is more efficient to execute shallow layers on resource-constrained devices and delegate the deep layers that require high computation to powerful servers. Based on this motivation, the objective is to determine the optimal split point that minimizes the size of intermediate data and the latency to transmit it. In this context, the authors of Neurosurgeon [19] examined the impact of splitting the DNN network at different layers on the latency and energy consumption and selected the best partition point, accordingly. Similarly, authors in [20] studied the computation time of different DNNs' layers and the transmission delays of their intermediate outputs, when using different IoT technologies. Then, they defined accordingly the optimal partitioning configuration that meets the minimum classification latency. The second partition strategy is the hierarchical splitting where the offloading is accomplished across the cloud, edge servers, and mobile devices. HDDNN [22] is one of the hierarchical partitioning approaches, where authors tested the performance of the distributed system with heterogeneous nodes' capacities, heterogeneous DNN networks, and heterogeneous tasks.

Even though offloading only a fraction of the DNN network to remote servers can be a solution to avoid network bottlenecks, the transmission latency can prevent the implementation of such approaches, specifically for applications with low-delays requirements and with high classification demands. Recently, researchers have investigated a third approach that consists of involving only the pervasive IoT devices that offer their limited computational resources and collaborate to execute different segments of the DNN. In this context, two partitioning techniques have been introduced: either to adopt per-layer distribution or to apply per-input partitioning (e.g., rows or columns of feature maps). The work in [25] presented an offline per-layer-CNN partitioning strategy, where each layer computation is statically assigned to one of the resource-constrained devices, without considering the dynamic of the online requests. On the other hand, two contributions marked the input-wise partitioning, namely MoDNN [4] and DeepThings [5]. These approaches examined the optimal segmentation of DNN models that minimizes the amount of data transmitted between participants. However, scheduling the correlation between IoT devices to perform online and dynamic inference tasks, while respecting their computational and memory constraints, has not been considered by previous works. Following the same input-wise partitioning strategy, we proposed in our work in [6] to distribute the feature-maps transformation tasks into different IoT devices, such as each untrusted participant receives only a part of the shared data and cannot apply black-box attacks to revert it and generate the original input. Such a fine-grained partition results in a high dependency between devices and a large bandwidth utilization to transmit intermediate data. In addition, a pervasive system that includes tens and even hundreds of IoT participants is required to allocate different segments of the DNN network.

Our proposed method is distinguished from the aforementioned approaches by several points: (1) The works that proposed re-designing the model to fit the DNN networks on IoT devices, such as compression, showed a high potential to reduce the size of CNN networks with lower accuracy performance [13], [14] or without loss of accuracy [11], [12]. However, squeezed deep networks may still be not suitable for IoT devices, particularly those resulting from very large DNNs or when the end-devices are small. In contrast, our approach presents a solution, where no modification of the DNN characteristics is introduced during the distribution. Instead, we distribute a high-performance trained model that proved its efficiency on classification tasks. Furthermore, decomposing the DNN into small segments allows its deploying on constrained devices. (2) The input-wise distribution is designed for highly resource-limited devices or for sensor networks that are not even able to execute layers of deep DNN models. Furthermore, these methods are implemented on large pervasive systems that include hundreds of devices and support large data transmission between participants. In our work, we opt for per-layer distribution in order to design a system with a reduced number of participants and a lower dependency between them, while guaranteeing the pervasive deep learning and the security of the model. (3) Some previously-described efforts proposed a static distribution of DNN models, where each subset of devices is responsible for processing a specific portion of the DNN model. This does not fit our online system, where a dynamic incoming load of classifications is received randomly and the connected IoT devices must cooperate to parallelize all requests and optimally leverage the limited resources, without being constrained by a static planned partitioning. (4) To the best of our knowledge, we are the first to investigate real-time CNN inference distribution over an IoT network, while considering the security of the system and covering the structure of the network from untrusted participants against white-box attacks, using reinforcement learning technique.

III. PRIVACY-AWARE DISTRIBUTED CNN FOR IoT DEVICES

In this section, we present our distributed CNN approach for privacy-aware and low decision-latency applications, the system model, followed by our strategy formulated as an optimization problem. As the optimal solution is NP-hard, we shape our problem as a RL approach adequate for highly dynamic systems and online classification.

A. SYSTEM MODEL

Our pervasive system is composed of a set $\mathcal{S} = \{s_1, \ldots, s_t\}$ of data-generating IoT devices (e.g., microphones, cameras and medical sensors), collecting data to be classified by a trained CNN. We define $\mathcal{I} = \{I_1, \ldots, I_D\}$ as the set of D heterogeneous IoT participants intending to cooperate in computing different layers of the CNN network. In order to save the resources (e.g., bandwidth, memory and energy) for data collection, processing, and offloading, the source devices

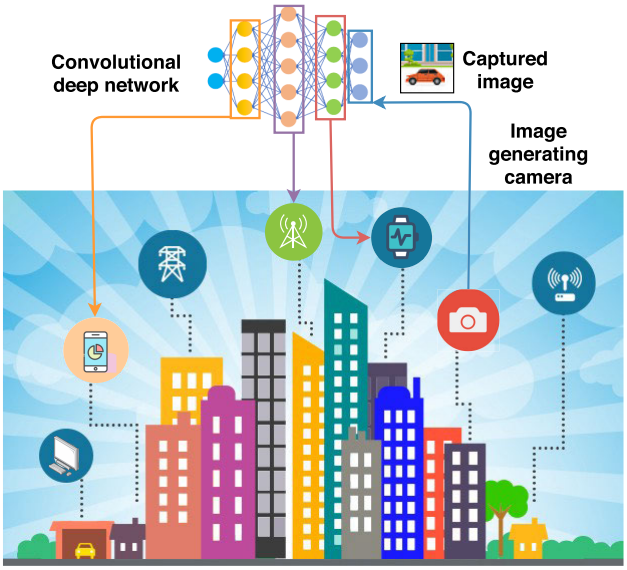

FIGURE 1. Illustration of the privacy-aware distributed CNN.

do not participate in the computation of intermediate layers of the neural network and they are only used to execute the first and last layers to ensure the privacy of the input data and the resultant prediction. Besides, each IoT participant I_i is characterized by limited resources, including the computation capacity $\bar{c}_i$, the memory usage $\bar{m}_i$ and the bandwidth availability $\bar{bd}_i$. In this paper, we assume that all devices have the same transmission technology with different transmission rates ρ_i. Furthermore, the study of the noise and the impact of the channel (e.g., loss or distortion) on the shared data between devices is beyond the scope of this work. Finally, without loss of generality, we assume that all IoT participants are connected and can reach each other.

We emphasize that, in this work, the proposed distribution encompasses only typical CNN models organized into a pipeline of processing layers (e.g., convolutional and ReLU), and without any residual block [29]. Thus, we define L_{n_j} as the number of layers composing the CNN network $n_j \in \{1 \ldots .N\}$, where N is the number of trained CNNs leading the applications in our IoT system. Each layer $l^k_{n_j} \in \{1, \ldots, L_{n_j}\}$ is characterized by a computation requirement c^k_j and a memory demand m^k_j. More specifically, the computational requirement of each layer is calculated as the number of multiplications needed to accomplish the layer's goal as done in [27]. Hence, the computational requirement of a convolution layer $l^k_{n_j}$ is measured as follows:

$$c^k_j = n_{k-1}.S_k.n_k.o_k, \tag{1}$$

where n_{k-1} is the number of input channels of the convolutional layer $l^k_{n_j}$, which is equal to the number of feature maps resulting from the layer $l^{k-1}_{n_j}$. S_k presents the spatial size of the filter defined by the layer $l^k_{n_j}$, n_k denotes the number of filters of $l^k_{n_j}$ and o_k presents the spatial size of the output map.

The computational requirement of a fully-connected layer is expressed as:

$$c_j^k = n_{k-1}^* . n_k^*, \tag{2}$$

where n_k^* and n_{k-1}^* represent the number of neurons of the layers $l_{n_j}^k$ and $l_{n_j}^{k-1}$, respectively. The computational loads introduced by pooling and ReLU layers can be typically neglected [27]. Furthermore, the memory requirement of the layer $l_{n_j}^k$ can be measured as the number of weights W_j^k composing this layer multiplied by the number of bits b allocated to store each weight [28]:

$$m_j^k = W_j^k . b. \tag{3}$$

Let O_j^k denote the memory occupation of the output data generated by the layer $l_{n_j}^k$ of the CNN n_j. These data are communicated between any two devices I_{i1} computing the layer $l_{n_j}^k$ and I_{i2} handling the subsequent layer $l_{n_j}^{k+1}$. The memory size of the output is calculated according to Eq. (3).

B. PROBLEM FORMULATION

In this section, we present the optimal placement of different layers within the IoT computing devices available in the pervasive network, while considering the available resources and the privacy level against white-box attacks. The proposed methodology relies on the decision variable $A_{r^*,k}^i$ that is equal to 1, if the IoT device I_i computes the layer $l_{n_j}^k$ of the request r; 0 otherwise. We note that $r \in RQ$ presents a request for data classification. In practice, r represents the index of the source device that requests the computation of an inference. Furthermore, for simplicity purposes, we suppose that each source device requests only one type of inference, namely r^*. We also define $\mathrm{M} = \max\{L_{n_1} \ldots L_{n_N}\}$, which is the maximum number of layers among the considered N CNNs. The distribution of IoT devices contributing to the collaborative system is supposed to be fixed during the optimization to maintain a static state of data rates. Finally, to cover the variation of the network (e.g., new requests, new joining devices, and removal of participants), the optimization should be executed periodically.

The objective function of the optimization models the latency of computing all the inferences presented by the set of requests RQ. This latency defines the delay to transmit the output of layers to the subsequent devices, as well as the time to execute different CNN tasks. Hence, the objective function to be minimized is expressed in equation (4). We remind that we adopted the per-layer distribution, as it is more adequate for systems with limited number of devices, and compatible as well with heaving IoT environments. Furthermore, per-layer partitioning results in less data transmission, consequently less energy utilization and in reduced dependency between devices.

$$\min_{A_{r^*,l}^i} \sum_{r\in RQ} \sum_{I_i\in\mathcal{I}\cup\mathcal{S}} \sum_{I_j\in\mathcal{I}\cup\mathcal{S}} \sum_{l=1}^{L_{n_{r^*}}-1} A_{r^*,l}^i * A_{r^*,l+1}^j * \frac{O_{r^*}^l}{\rho_i} * 1_{I_i\neq I_j} + \sum_{I_i\in\mathcal{I}\cup\mathcal{S}} t_c^i \tag{4}$$

subject to:

$$(4.a): \sum_{r\in RQ} \sum_{l=1}^{L_{c_{r^*}}} A_{r^*,l}^i * m_{r^*}^l \leq \bar{m}_i \quad \forall I_i \in \mathcal{I}\cup\{s_1..s_n\},$$

$$(4.b): \sum_{r\in RQ} \sum_{l=1}^{L_{n_{r^*}}} A_{r^*,l}^i * c_{r^*}^l \leq \bar{c}_i \quad \forall I_i \in \mathcal{I}\cup\{s_1..s_n\},$$

$$(4.c): \sum_{r\in RQ} \sum_{l=1}^{L_{n_{r^*}}} \sum_{I_j\in\mathcal{I}} A_{r^*,l}^i * A_{r^*,l+1}^j * O_{r^*}^l * 1_{I_i\neq I_j} \leq \bar{bd}_i \quad \forall I_i \in \mathcal{I}\cup\{s_1..s_n\},$$

$$(4.d): \sum_{i\in\mathcal{I}\cup\mathcal{S}} A_{r^*,l}^i = \begin{cases} 1 & \text{if } l \leq L_{n_{r^*}} \\ 0 & \text{Otherwise} \end{cases} \quad \forall r \in RQ,$$

$$(4.e): A_{r^*,1}^r = 1 \quad \forall r \in RQ,$$

$$(4.f): A_{r^*,L_{n_{r^*}}}^r = 1 \quad \forall r \in RQ, L_{n_{r^*}} \leq M,$$

$$(4.g): H_{r^*,l}^i = \Pi_{k=2}^{l-1} max(A_{r_j^*,k}^i, \forall r^* = r_1^* = r_2^*..) \; \forall I_i \in \mathcal{I},$$

$$(4.h): 1 - H_{r^*,(l-1)}^i \geq A_{r^*,l}^i \quad \forall I_i \in \mathcal{I}, r \in RQ, l \in \{2 \ldots M-1\},$$

$$(4.i): A_{r^*,l}^i \in \{0,1\}.$$

where:

$$t_c^i = \sum_{r\in RQ} \sum_{l=1}^{L_{n_{r^*}}} A_{r^*,l}^i * \frac{c_{r^*}^l}{e(i)}, \qquad \forall I_i \in \mathcal{I}.$$

The objective function in Eq. (4) consists of two latencies:

- The total latency to transmit intermediate outputs between subsequent IoT devices, executing the CNN layers: In practice, the transmission latency of the output generated by the l-th layer of the r^*-th CNN request is expressed as follows:

$$\frac{O_{r^*}^l}{\rho_i} * 1_{I_i\neq I_j}, \tag{5}$$

  where ρ_i is the transmission data rate of the device I_i. We note that the transmitted output is equal to 0, if $I_i = I_j$, including the scenario where the source device generates the input data and processes the first layer.
- The processing latency of different layers on the IoT participants: The computation time of the layer l belonging to the CNN r^* by the I_i-th IoT unit is approximated as the ratio between the computational demand $c_{r^*}^l$ of the layer l and $e(i)$ presenting the number of multiplications performed by the device I_i in one second [25]. In practice, $e(i)$ indirectly depicts the number of cores available within the participant I_i and the presence of GPU or CPU to parallelize operations.

The equations (4.a), (4.b) and (4.c) ensure that the computational, memory, and bandwidth constraints are respected for different participants, including source devices. The constraint in Eq. (4.d) guarantees that each layer l, of each inference request is assigned to one node.

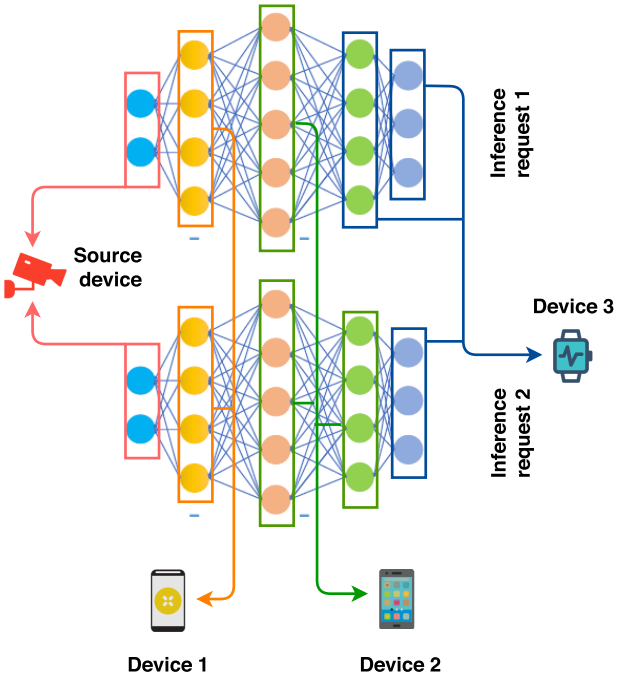

FIGURE 2. Illustration of the privacy strategy.

Equations (4.e) and (4.f) ensure that the first and the last layers of each inference request should be computed on the source device. The transmission and computation requirements to execute these two layers justify further our design related to source devices. More specifically, data-generating devices do not participate in the inference process to save their endowed resources for data collection and execution of the first and last layers. The constraint in (4.g) introduces the variable $H^i_{r^*,l}$, which is the multiplication of all allocation decisions of layers from 1 to $l-1$, in all requests of the same CNN ($r^* = r^*_1 = r^*_2 = \ldots$). It means $H^i_{r^*,l}$ is equal to 1, if all sequential layers of the trained CNN r^* are assigned/exposed to the same device, 0 otherwise. We remind that $\{r, r_1, r_2, \ldots\}$ present the source devices and $\{r^*, r^*_1, r^*_2, \ldots\}$ denote their requested CNNs. The constraint in Eq. (4.h) guarantees that $H^i_{r^*,l}$ can never be 1, meaning at least one layer is not exposed to the untrusted device. Note that, we can replace the equation (4.g) and the constraint (4.h) by:

$$H^i_{r^*,l} = \sum_{k=2}^{l-1} max(A^i_{r^*_j,k}, \forall r^* = r^*_1 = r^*_2 = ..) \quad \forall I_i \in \mathcal{I}$$
$$H^i_{r^*,l} \leq l * Sl \tag{6}$$

This equation adds a constraint of the security level Sl, which means the proportion of the model to be exposed to each participant. Figure 2 shows a scenario of distributing two inference requests, where Sl is equal to 50%, meaning, half of the previous layers should be hidden from the participant for all the executed classifications related to the same model.

The problem in (4) is NP-hard, which means finding the optimal solution is highly difficult in terms of time and cannot be used as an online solution dealing with real-time classifications. Furthermore, the optimization relies on assuming a full overview about the set of incoming requests and the dynamics of participants during the considered period of time. However, our system is covering an extremely dynamic system, where different IoT devices can participate or leave the collaborative framework randomly. Also, the load of requests is highly dynamic and the available resources are volatile. Therefore, assuming a static environment to get the optimal solution is not practical. Recently, RL approaches are becoming increasingly popular, particularly for applications having large and complex problem spaces. These approaches are able to react under unforeseen environments, by a continuous process of gaining rewards and incurring penalties for each taken action. Moreover, the RL system learns an optimal decision making policy from historical knowledge about the environment states, which is similar to the optimization process. Another asset of reinforcement learning approaches is their continuous online learning that enables them to adapt to any system fluctuation. To summarize, in order to reduce the complexity of the optimization and implement an online and realistic layers' distribution while respecting the required privacy, we propose an RL-based approach to approximate an optimal allocation, namely RL-PDNN.

C. RL-PDNN: REINFORCEMENT LEARNING FOR PRIVACY-AWARE DISTRIBUTED NEURAL NETWORKS

In this section, we shape our privacy-aware distribution of online CNN classifications as a reinforcement learning process. Specifically, we introduce the environment design, present the set of states and actions, design the appropriate reward function as well as the RL-agent, and finally choose the most effective RL technique.

The reinforcement learning concept is based on learning how to map situations and environment states to actions in order to maximize the reward signal. The RL-agent is not apprised which action to choose; instead, it discovers the actions that achieve the highest reward by trying different combinations and receiving immediate gains and penalties. In some challenging scenarios, the chosen action does not impact only the direct reward, but also all subsequent situations and related rewards. These two features, trial and error, and search and delayed reward assignment, are the key characteristics of the RL enabling it to learn by interacting with its environment and then adapting to it. In our context, at each time step of the distribution process, the RL-agent selects the IoT device that can handle each layer computation. This decision is taken depending on the CNN network trained for the classification, the current layer, the number of layers that can be exposed to each device at each step, and the available resources offered by the participants, with an objective to enable the inference on small devices while achieving the minimum latency and respecting the required privacy. During this learning process, the RL-PDNN system gains rewards and experiences penalties for each action it takes until converging to the optimal policy. Accordingly, we abstract

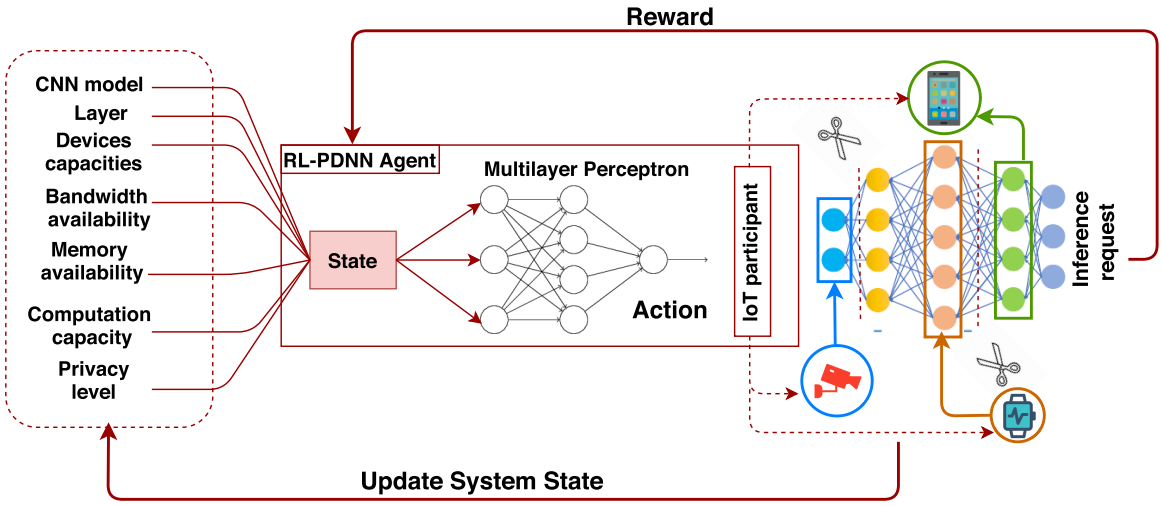

FIGURE 3. The environment design of the RL-PDNN system for participants selection.

our problem as a Markov Decision Process (MDP) framework depicted by (S, A, P, R, γ). S defines the set of states, A presents the possible actions, P is the state transition probability, R defines the direct reward of the RL-step, and γ presents the discount factor. The introduced elements are discussed further in the following sections.

1) ENVIRONMENT DESIGN

In this work, the MDP environment presents a pervasive system involving a set of IoT devices desiring to participate in the collaborative inference. This environment is designed to interact with a centralized agent that generates episodes of experiences by: selecting an action A_t at each time-step t, generating the next set of states S_{t+1}, and receiving the step reward R_t. By generating episodes of experiences, the agent is able to learn from past actions and the related rewards. The episode is defined as a sequence of time-steps. A time-step illustrates, in our system, the allocation of one CNN layer belonging to one of the incoming classification requests, whereas an episode defines the distribution of one of the requested inferences among the IoT participants. Different episodes are independent and the total reward is initialized to 0 at the beginning of each episode. We emphasize that the RL-agent does not have visibility regarding the environment design. Instead, the optimal policy is learned by interacting with the environment, selecting advantageous actions, and receiving rewards/penalties. Meaning, the optimal policy Π^* is a mapping between the states S_t and the corresponding actions A_t, i.e., $\Pi^* : S \rightarrow A$. The environment design of the RL-PDNN system is presented in Figure 3.

2) STATES AND ACTIONS

The set of states S consists of all possible circumstances that can impact the assignment of the layer computation to one of the IoT devices. Thus, at each time step t, the observations received by the agent comprises the following components:

- n_i: the type of the CNN request (e.g. VGG and LeNet),
- $l^k_{n_i}$: the current layer to be allocated,
- $\{\bar{m_1}, \ldots, \bar{m_D}\}$: the set of memory capacities within different IoT participants,
- $\{\bar{b_1}, \ldots, \bar{b_D}\}$: the set of available bandwidth resources,
- $\{\bar{c_1}, \ldots, \bar{c_D}\}$: the set of available computation resources,
- $\{\rho_1, \ldots, \rho_D\}$: the set of data rates,
- $\{\bar{L_1}, \ldots, \bar{L_D}\}$: the number of layers' structures, preceding the current layer and exposed to each participant. We underline that a layer structure is exposed to a participant if it is computed by this device in a previous request.

The system state S is therefore a vector defined as: $\{n_i, l^k_{n_i}, \{\bar{m_1}, \ldots, \bar{m_D}\}, \{\bar{b_1}, \ldots, \bar{b_D}\}, \{\bar{c_1}, \ldots, \bar{c_D}\}, \{\rho_1, \ldots, \rho_D\}, \{\bar{L_1}, \ldots, \bar{L_D}\}\}$. We remind that D denotes the number of IoT devices that envisage to participate in the collaborative system. Following the described design, the set of states becomes highly sizeable (equal to $k * D + 2$) and affected by the number of participants D. Note that k denotes the number of device parameters (memory, bandwidth, etc.), equal to 5 in our case. Moreover, the scales of different states (e.g. number of layers $\bar{L_i} \sim 2 - 100$ and bandwidth capacity $\bar{b_i} \sim 10^6$) are highly unbalanced and need to be normalized. Therefore, we propose to convert the set of privacy levels and resource availabilities to binary, where 1 implies that the related device has sufficient resources and can compute the current layer while still respecting the privacy constraint; 0 otherwise. In this way, S will include the product of different binary states and devices' capacities. Regarding the potentially selected actions, we define a set A containing D binaries, where each value indicates whether the current layer computation is assigned to the related IoT participant. For example, $A(4) = 1$ means that the fourth device is chosen to perform the correspondent task. Finally, at the end of

each time step, the memory, bandwidth and computational resources are updated according to the predicted action and available devices are updated following the dynamics of participants. Additionally, the number of layers' structures exposed to each participant is also updated. The updated resources and allocated layers are the input to the next time-step.

3) REWARD FUNCTION

To match our high-level objective aiming at minimizing the inference latency, while exploiting the distributed resources and meeting the privacy requirements of the deep learning system, the reward function R should be well designed. More specifically, when the RL agent executes an action A_t based on the state S_t, a reward is given. This reward is attributed depending on the ability of the RL-system to respect the following constraints:

$$\begin{cases} \text{C1: if } 1 < l^t_{n_i} < L_{n_i} \rightarrow \sum A_t = 1, & \text{constraint (4.d)} \\ \text{C2: if } A_t(j) = 1 \rightarrow m^t_i \leq \bar{m}_j, c^t_i \leq \bar{c}_j, & \text{constraint} \\ O^{t-1}_i * 1_{j \neq j'} \leq \bar{bd}_j & \text{(4.a,b,c)} \\ \text{C3: } A_t(j) = 1 \rightarrow \sum^t_{l=1} A_l <= Sl * t, & \text{constraint (4.h)} \end{cases} \tag{7}$$

C_1 indicates that the sum of the action set should be equal to 1, which matches the constraint (4.d). The first and last layers are assigned directly to the source devices to secure the original data and the classification results, as stated in constraints (4.e) and (4.f). C_2 indicates that the available resources within the selected device should be sufficient to achieve the layer goals, which is equivalent to the constraints (4.a), (4.b), and (4.c). C_3 shows that each participant should respect the privacy level required by the collaborative system. Particularly, only a proportion Sl of previous layers should be exposed to any untrusted device, which is equivalent to the constraint (4.h) or the constraint (6). Next, we define the immediate reward as follows:

$$R_t = C_1 * C_2 * C_3 \tag{8}$$

Not respecting the first constraint ($C_1 = 0$) means either the task is rejected and could not be handled by any participant or it is assigned to multiple devices simultaneously. Hence, we set the reward to be 0 to avoid these invalid situations. Furthermore, if the chosen participant does not have enough resources to compute the current layer, or it is not able to meet the privacy requirements, we attribute 0 as a reward. The maximum immediate reward, equal to 1, is only received, when all constraints are met. In addition to the rewards given for respecting different constraints, the RL-PDDN system charges the agent for selecting low-capacity devices and incurring higher latencies to achieve the classification tasks. Therefore, these charges are counted as penalties experienced by the system and added to the reward function. In this way, the RL-agent tries to maximize the cumulative rewards by minimizing the penalties and learning the optimal layers allocation. More specifically, when $t = 1$, the reward R_t is initialized to $c^1_{r*}/e(r)$, which is the delay of computing the first layer at the source device as indicated by the constraint (4f). Next, at each time-step t, the penalty equal to $(\frac{O^t_{r*}}{\rho_i} * 1_{I_i \neq I_j} + c^t_{r*}/e(i))$ is added to the reward function.

4) AGENT DESIGN

The RL-agent has an objective to learn an optimal policy Π^*, defined as the strategy giving the maximum reward throughout the experienced episodes. In order to design the optimal strategy, the agent starts by building an approximation of the optimal action-value function $Q(S_t, A_t)$, which represents the expected future reward and assesses the performance of taking a specific action A_t for a given state S_t. The action-value function $Q(S_t, A_t)$ is expressed as follows:

$$Q(s, a) = E[\sum_{k=1}^{N} \gamma^k R_{t+k} | S_t = s, A_t = a], \tag{9}$$

We underline that $\gamma \in [0, 1]$ defines the discount factor, which presents the importance of the immediate reward compared to long-term reward accumulated at the end of each episode. To evaluate $Q(s, a)$, a temporal difference method is used:

$$Q(s, r) \leftarrow Q(s, r) + \alpha(R_t + \gamma \max_{a'} Q(s', a') - Q(s, a)), \tag{10}$$

where $\alpha \in (0, 1]$ presents the learning rate.

5) DEEP Q-LEARNING ALGORITHM

Our distributed environment is highly variable due to the dynamics of IoT participants joining and leaving the pervasive system and the varying load of requests. Additionally, the action space is highly dimensional and depends on the number of existing IoT participants. Hence, it is challenging to apply traditional reinforcement learning methods for resource allocation problems, where Q-tables are used to store dimensional historical Q-values. Due to these challenges, we chose to use Deep Q-Network (DQN) [30], which is an algorithm that is able to learn the parametric representation of the action-value function $Q(s, a)$ by interacting with the environment based only on the current states. The DQN is trained to minimize the loss function illustrated as follows:

$$L(\theta) = E[(Rt + \gamma \max_{a'} Q(s', a', \theta') - Q(s, a, \theta))^2]. \tag{11}$$

where $Q(s, r)$ is approximated to $Q(s, r, \theta)$ and θ denotes the weights of the deep networks. To stabilize the training of the DQN network and guarantee the convergence of the learning, multiple techniques should be followed, which are detailed in algorithm 1. In this algorithm, we start by initializing two copies of the Q-Network with the same NN weights (lines 2-3). Next, the training process generates a number of sequential episodes repeatedly (lines 5-20) and stores the experience tuples in a buffer D (lines 21-23). To ensure that the agent explores the quality of different possible actions, we follow an $\epsilon-$ greedy algorithm (line 9), where decisions are taken randomly with probability ϵ or by

Algorithm 1 RL-PDNN

1: **Initialization:**
2: Initialize randomly the parameters θ of the first Q-network.
3: Initialize parameters of the second target network: $\theta' \leftarrow \theta$.
4: **DQN Learning:**
5: **for** each episode $r \in RQ$ **do**
6: **for** each time-step $t = 1..L_{r^*}$ **do**
7: $S(t) = \{n_i, l^t_{n_i}, \{\bar{m_1}, \ldots, \bar{m_D}\}, \{\bar{b_1}, \ldots, \bar{b_D}\},$
8: $\{\rho_1, \ldots, \rho_D\}, \{\bar{c_1}, \ldots, \bar{c_D}\}, \{\bar{L_1}, \ldots, \bar{L_D}\}\}$
9: Select A_t based on $\epsilon-$greedy policy
10: **if** $\sum A_t = 1$ **then**
11: **if** $t = 1$ **then**
12: $R_t = R_t - c^t_{r^*}/e(r)$
13: $i = r$
14: **else**
15: **for** $j = 1 : D$ **do**
16: **if** $A_t(j) = 1$ **then**
17: $R_t = R_t - (\frac{O^t_{r^*}}{\rho_j} * 1_{I_i \neq I_j} + c^t_{r^*}/e(j))$
18: $i = j$
19: **if** $C_2 * C_3 = 1$ **then**
20: $R_t = R_t + 1$
21: Observe R_t and the next state S_{t+1}.
22: Save (S_t, A_t, R_t, S_{t+1}) in the experience
23: memory D.
24: Sample a mini-batch of $(S(j), A(j), R(j),$
25: $S(j+1))$ from the memory D.
26: Find target Q-value $Q_{target}(j)$ from target
27: Q-network: $Q_{target}(j) = R(j)+$
28: $\gamma max_{a'} Q(s', a', \theta')$.
29: Update the target Q-network with loss function:
30: $L(\theta) = [Q_{target}(j) - Q(s, a, \theta)]^2$ every G steps.

choosing the highest Q-value experienced until the current time-step: A_t =arg max$Q(S_t, A_t)$. Additionally, we use the decaying scheduling of ϵ. More specifically, ϵ is fixed to 1 at the beginning of the training; thus the agent can discover the environment and then it decays over time to allow the system to exploit the learned knowledge and select optimal actions without losing the ability to behave randomly and detect any occurring changes. Once the action is taken, a reward is given depending on the accuracy of the decision. Meaning, if the resource and privacy constrains are respected, a reward is added to the agent (lines 19-20). Besides, if a valid participant is chosen (line 10), a penalty is added according to the incurred latency to compute the current layer (lines 11-18). Next, a new state S_{t+1} is generated for the next layer allocation. To improve the Q-values and the accuracy of the decisions, the agent samples a random minibatch from the replay buffer D (line 24) at each time step, and target values for each tuple in the sample-batch are calculated using the target Q-network (line 26). These values are used for the network update. The described mechanism, known as experience replay, helps to further learn from past experience and leads to the stabilization and convergence of the learning process. While improving the main Q-network, the target network is held fixed, to stabilize the learning. Then, at each G steps, the target network is updated toward the main one, in order to always reflect the most recent knowledge (line 30) [30].

IV. PERFORMANCE EVALUATION

In this section, our RL-PDNN distributed system is evaluated under different networking configurations. Particularly, the impact of the privacy level, the number of participants, the type of devices, and the type of requests on the total latency, accuracy of decisions, and shared data is examined. Next, to prove the performance of our RL approach, we compared it to the static distribution approach [25] and the optimal framework.

A. FRAMEWORK SETTINGS

Our RL-PDNN approach has been validated on image classification scenarios. Deep learning applications, involving image classification, can serve for example in a surveillance system monitoring a critical area (e.g., highway accidents and borders). Three standard CNN benchmark datasets are adopted in our simulation: CIFAR10, MNIST, and Stanford CARs. More specifically, CIFAR10 is trained with a medium-sized CNN (6 convolutional and 2 fully connected layers), MNIST is trained with a small CNN, namely LeNet (2 convolutional and 3 fully connected layers), and VGG16 (13 convolutional and 3 fully connected layers) is trained on Stanford CARs dataset. Following the data characteristics, we suppose that the source devices (e.g., cameras) capture 28×28 gray images (MNIST), 36×36 RGB images (CIFAR), and 128×128 RGB images (Stanford CARs).

Furthermore, our system is composed of three types of technological devices, which are LG Nexus 5, Raspberry Pi B+, and STM32H7. The first type is considered as a powerful unit endowed with a 2.28 GHz processor and a 2 GB RAM. The second one is less powerful and it is equipped with 1.4 GHz 64-bit quad-core processor and 1 GB RAM. The last type corresponds to a very resource-limited device (e.g., smart watch), which is supplied with a 400 MHz-cortex and 1 MB RAM. The number of multiplications per second e [25], is equal to 800 for LG Nexus, 560 for RPi3 and 40 for STM32H7. Besides, the powerful IoT devices (RPi3 and LG Nexus) are equipped with the bandwidth standard IEEE 802.11n having an average ρ equal to 72.2 Mb/s. Meanwhile, the small devices (STM32H7) own a low bandwidth technology, namely IEEE 802.11ah, with an average data rate ρ equal to 7.2 Mb/s. We note that all cameras are deployed with RPi3 system. Furthermore, we suppose that 51 IoT units participate in the cooperative classification, where each 17 devices are related to a technological family. In our simulation, the classification requests are generated using a Poisson process with a data rate λ equal to 3/s. Moreover, the set of participants is changing every 60 s. It means the set of available resources is changing

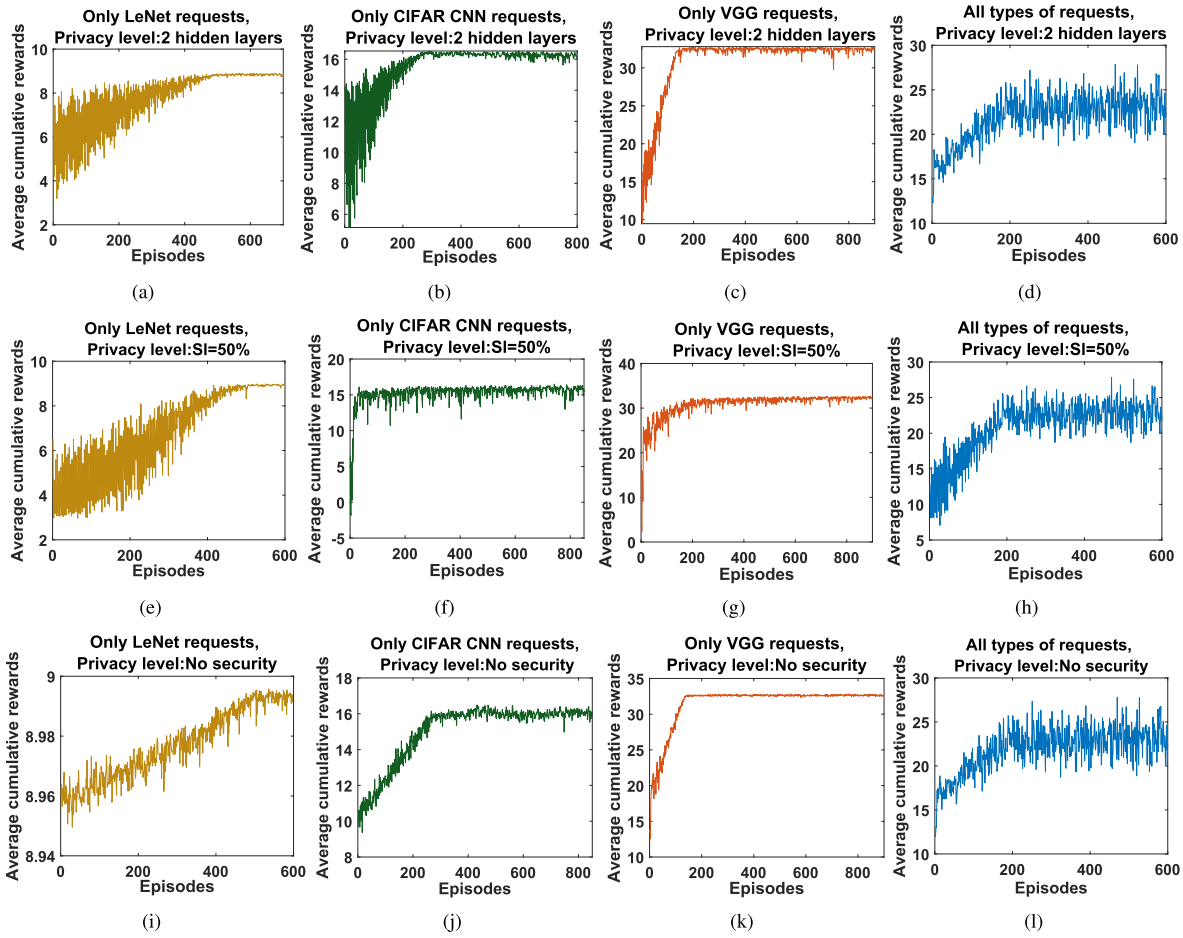

FIGURE 4. Average cumulative rewards vs. training episodes: LeNet, CIFAR CNN, VGG, and heterogeneous requests distribution.

and the number of exposed layers to the new participants is equal to 0.

Our approach is first evaluated through measuring the inference latency, presented by the delay between acquiring the data and generating the prediction. Second, we evaluate the amount of data transmitted between all IoT devices to execute all CNN tasks. Third, the accuracy of the system is assessed, which is defined as the capacity to respect different constraints (resources and privacy requirements) after the convergence. The privacy robustness of the system is examined on 3 security levels (all layers are exposed/no security, 2 hidden layers prior to the currently executed, half of the previous layers are covered) and 4 types of networks (only Mnist-LeNet requests, only CIFAR-CNN requests, only VGG requests, and heterogeneous requests.).

The RL-PDNN algorithm is validated according to the parameters shown in Table 1. These parameters are empirically chosen, and we expect that similar architectures perform identically.

TABLE 1. Parameters of the RL simulation.

Parameter	Description	Value
γ	discount factor	0.95
ϵ	exploration rate	1
bz	buffer size	100000
G	update frequency	1000
α	learning rate	0.0005, except VGG for all levels and CIFAR CNN, Sl=50% :0.0001
ϵ_{decay}	decay	0.99, except VGG for all levels and CIFAR CNN, Sl=50% :0.995
$Policy$	DNN policy	MLP, 2 layers, 64 neurons
M	batch size	64

B. SIMULATION RESULTS

1) PERFORMANCE OF THE RL CONVERGENCE

Figure 4 depicts the variation of the average cumulative rewards among training episodes. Particularly, the obtained cumulative rewards are smoothed over a window of 50 episodes. In the first 1000 episodes, ϵ is set to 1 to act randomly for all taken decisions in order to obtain an initial estimate of the random policy. Next, a decay parameter, namely ϵ_{decay}, is applied after each episode to pass quickly and smoothly to an enhanced policy. Based on this configuration, We can see, in Figure 4, that most of the constraints (resources and privacy) are not respected at the

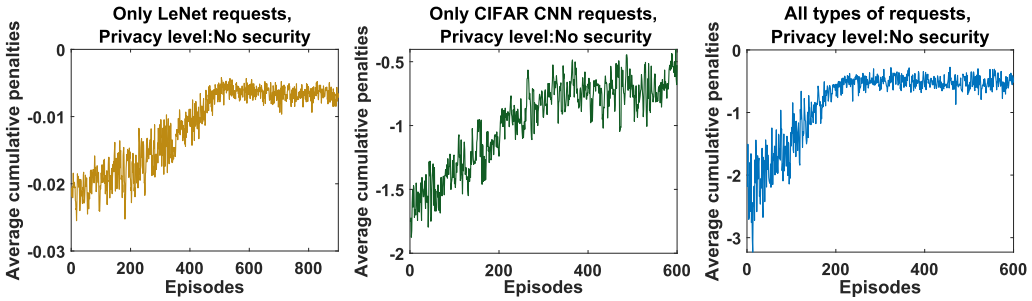

FIGURE 5. Average cumulative penalties vs. training episodes: LeNet, CIFAR CNN, and heterogeneous requests distribution.

beginning of the learning process. However, as the number of episodes increases, the number of respected constraints increases until reaching stability where all requirements are met, which confirms the convergence performance of our RL system for different types of networks. Moreover, we can notice that the convergence of LeNet requests is slower than the convergence of other types of networks, which can be explained by the fact that the number of layers of LeNet network is small and a larger number of episodes is needed to learn how to respect the privacy requirements. Accordingly, VGG network converges faster, as it has a higher number of layers compared to the latter one. Also, when all requests are related to LeNet classifications and no privacy is required, Figure 4(i) shows that all constraints on different layers are respected from the beginning of the learning process, as the cumulative rewards are equal to 9 (number of layers in LeNet) from the first episodes summed with the penalties related to the latency. In fact, the LeNet network does not require high resource demand, and the available IoT devices are able to handle the load of requests. Furthermore, the learning process consists only in determining the optimal policy to minimize the classification latency. Finally, when the network is heterogeneous, we can see that the convergence is not smooth, which is related to the dynamics of the incoming requests, their number, and their types.

Figure 5 depicts the convergence of the penalty added to the reward to learn the latency minimization strategy. We can see that in addition to respecting the constraints (privacy and resource constraints), the RL system is able to find the fine-grained policy that schedules an efficient layers' allocation with reduced latency. Furthermore, the figure shows that minimizing the latency is slowly learned and the penalties start to converge after 15000 episodes depending on the network type (penalties are smoothed over a window of 50 episodes). This can be explained by the fact that the exploration space is large and the system needs to discover optimal actions related to each received set of states, then near-optimal decisions will be taken on the fly.

Figure 6 depicts the average accuracy of the RL decisions for different types of networks, under different security levels. We can see that the accuracy of our RL model is very high,

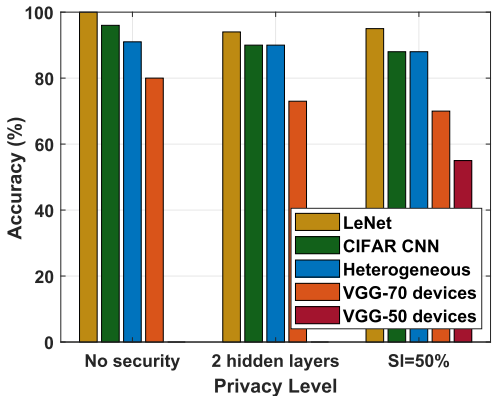

FIGURE 6. Accuracy of the system in terms of meeting constraints.

reaching 80% and more for all types of requests, when no privacy is required. It means more than 80% of episodes respect privacy and resources constraints. Moreover, we can notice that the LeNet system respects the resource constraints for all requests when privacy is not required, meaning the accuracy is equal to 100%. This can be attributed to the fact that processing a LeNet request is not resource-consuming and does not require any distribution. Similarly, even though the distribution is required for privacy purposes, the accuracy of LeNet network is still high due to its lightweight computational demand. The Figure also shows that the studied networks exhibit different performances. More specifically, around 90% of Mnist-LeNet, CIFAR-CNN, and heterogeneous requests are classified by the pervasive IoT devices, while respecting different privacy requirements. If all inferences are classified by a VGG model, only 70 % of them meet the privacy requirements. This accuracy is related to the fact that the neighboring IoT participants are not sufficient to distribute different layers of the resource-consuming VGG requests, while respecting the privacy distribution requirements. Hence, the system either chooses to leverage the existing resources without considering privacy or assigns tasks to occupied devices and respects the security requirements. For example, when the privacy level is high ($SI = 50\%$), 51 devices are not able to compute all incoming requests with

challenging distribution requirements and accordingly, less than 60% of classifications are accomplished. Involving more devices could enhance the accuracy of the system by more than 10%, as illustrated in Figure 6.

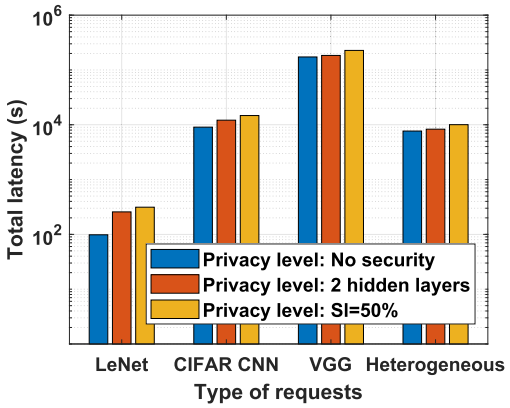

FIGURE 7. Total latency of 15000 inferences including different types of requests.

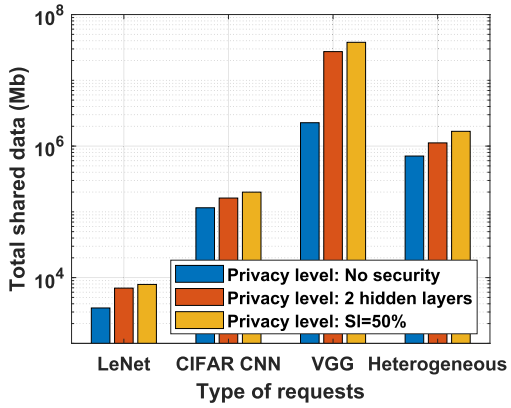

FIGURE 8. Total shared data to execute 15000 inferences for different types of requests.

2) IMPACT OF THE NETWORK CONFIGURATION ON THE RL-PDNN PERFORMANCE

Figures 7 and 8 depict the total latency and the total shared data of 15000 incoming requests comprising only one type of CNN or heterogeneous CNN inferences. When all requests are classified by LeNet and no security is imposed, the latency of the system (0.006 s per inference) and transmitted data (0.2 Mb per inference) are very low as the network structure is only composed of 9 layers (Conv, ReLu and Fc) and deals with small 28 × 28 images, which implies that only one participant is able to compute one incoming request without being forced to share intermediate outputs. When the privacy of data is required and more participants are involved, the latency (0.019 s per inference on average among both privacy levels) and shared data (0.5 Mb per inference on average) are still low, due to the lightweight of LeNet layers. For a larger network (CIFAR CNN), the latency (0.8 s per inference on average among all levels) is still low compared to the time needed to process VGG requests (11 s per inference on average among all levels), which is justified by the limited number of layers (17 layers), and the small size of images (32 × 32). VGG 16 is well known for its performance in computer vision and image classification owing to its deep network. However, the memory occupation and computation requirements restrain the VGG model from being processed on resource-limited devices. By distributing such complex models, high computations and large memory occupation could be shared between edge participants, which avoids transmissions to remote servers. In fact, in case of remote cloud inference, the transmission of the intermediate data of the first layers takes around 15 s according to the reference paper Neurosurgeon [19]. To minimize the transmission overhead, only the last layers should be executed in the cloud. In this case, end-devices with GPUs have to be used to fit the high computation requirements of the shallow layers. In our approach, we use collaborative resource-constrained devices, existing everywhere, to execute VGG with an acceptable delay and without resorting to the cloud. Finally, We can notice, in Figures 7 and 8, that imposing privacy requirements and hiding layers structure from participants contribute to involving more IoT devices, which results in higher latency and data sharing.

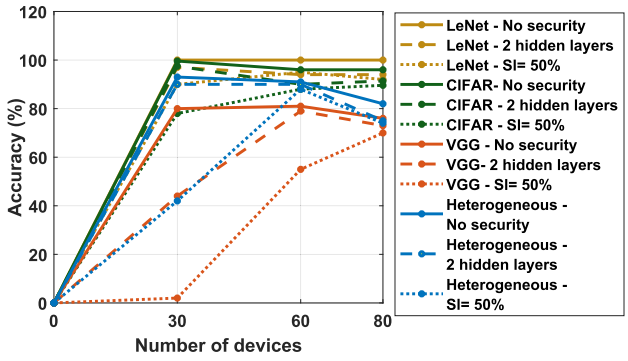

FIGURE 9. Performance of RL-PDNN when varying the number of devices.

Figure 9 illustrates the performance of RL-PDNN, when varying the number of participants. We can see that for LeNet or CIFAR load of requests, 30 devices are enough to accomplish a high accuracy and meet different requirements of the system (e.g., privacy and resources.). When the load of requests is heterogeneous or involves only VGG classifications, a small number of devices cannot achieve the distribution while respecting the privacy constraints. We can also notice that 60 devices are enough for most of request types to achieve good performance, when following a similar network configuration. Still, when classifying the data using only resource-consuming VGG and while respecting a high privacy level, 60 IoT devices are not able to accomplish a high performance and more participants should be involved in the distribution. Finally, we can notice that increasing the number of IoT devices participating in the CNN partitioning is not enhancing the system performance in most of the cases.

This can be explained by the fact that the set of action size is equal to the number of IoT participants and increasing the number of actions can decrease the accuracy of decisions slightly.

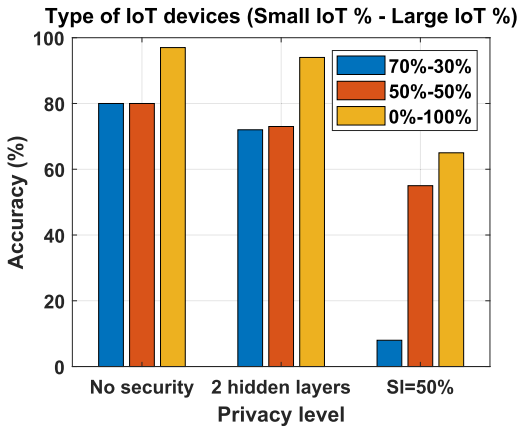

FIGURE 10. Performance of RL-PDNN, applied on VGG requests, when varying the capacity of 51 IoT participants.

Figure 10 shows the accuracy of the system receiving VGG requests, when varying the capacity of 51 IoT participants. The system presents similar performance when half or more IoT devices are small, for low privacy constraints. When the privacy requirements are high, performant devices are indispensable for a better accuracy, in addition to higher number of devices as depicted in previous simulations. We can also note that involving only high-performance devices in the distribution enhances the accuracy for all privacy levels.

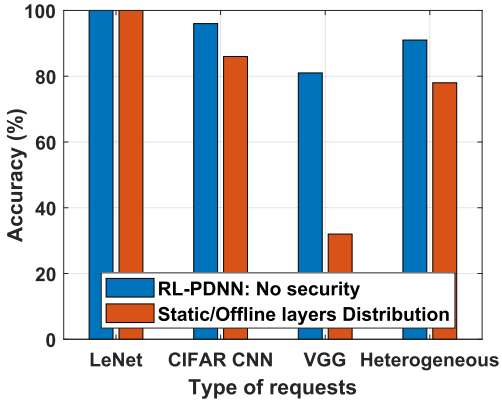

FIGURE 11. Performance of RL-PDNN VS offline and static distribution.

3) COMPARISON TO STATE-OF-THE-ART APPROACHES AND OPTIMAL SOLUTION

Figure 11 depicts the performance of our RL-PDNN system compared to the static and offline layers distribution proposed in [25]. In the latter approach, the authors assign the computation of different layers to the most performant devices, without considering the load of the online incoming requests. Furthermore, this strategy needs to be re-run each time the network configuration changes, e.g., new participants join the collaborative system or others leave it. This increases the complexity of the system that requires real-time decisions. We emphasize that the approach in [25] does not present any privacy requirement. Hence, for a fair comparison, we compared it to our RL-PDNN with no security constraints. Figure 11 shows that our system is able to adapt to the system dynamics, while respecting the available resources, in real time. Meanwhile, the offline distribution presents a low accuracy and incapacity to handle all online CNN tasks using the static partitioning. More specifically, when only LeNet or CIFAR requests are presented to the system, the static distribution is able to handle the lightweight CNNs. However, when the load corresponds to deep networks, assigning statically specific tasks to fixed devices exhibits low performance, particularly when the load of requests is high and exceeds the capacity of the chosen participants. This is not the case of our approach that takes into consideration all available resources to distribute different layers.

Most of the privacy-preserving algorithms focus on securing the training data, however, fewer defense strategies to mitigate the inference privacy attacks are proposed in the literature. One of the proposed approaches is the deep split [26]. More specifically, the fully-connected layers generate data that is difficult to recover owing to their ability to mix the input and hide the sample features. Hence, the idea is to divide the model after the first fully-connected layer and compute the first part in the source device while offloading the rest to a second participant. However, this approach requires the computation of all convolutional layers in one device, which is a heavy task and cannot be deployed in all IoT devices. As a solution, the work in [26] proved that the reverted data become blurrier as the CNN network goes deeper. Therefore, a smaller segment can be deployed in the IoT device, while sacrificing in terms of privacy level. Meanwhile, the second large segment should be allocated in high-performing machine such as remote servers, which increases the latency overhead. This is not the case of our approach that distributes the computational tasks among small devices existing everywhere, while hiding the structure of the model.

Another solution to secure the inference samples is to use the differential privacy, adopted in [31]. This approach consists of adding noise to the intermediate shared data in order to obfuscate it. Obviously, a trade-off between the accuracy loss and the privacy should be established as a higher level of noise causes a lower classification performance. Encryption, used in [32], [33], is another robust privacy-preserving method that allows to compute the inference on encoded data, so that the private information is not leaked. However, the main drawbacks of such approach are the computation overhead to perform the encryption, the potential accuracy loss, and incompatibility with some DNN operations [26]. We remind that our approach is applicable to all DNN tasks and it is designed to distribute trained models without affecting the accuracy. Table 2 summarizes the characteristics of different privacy-preserving techniques.

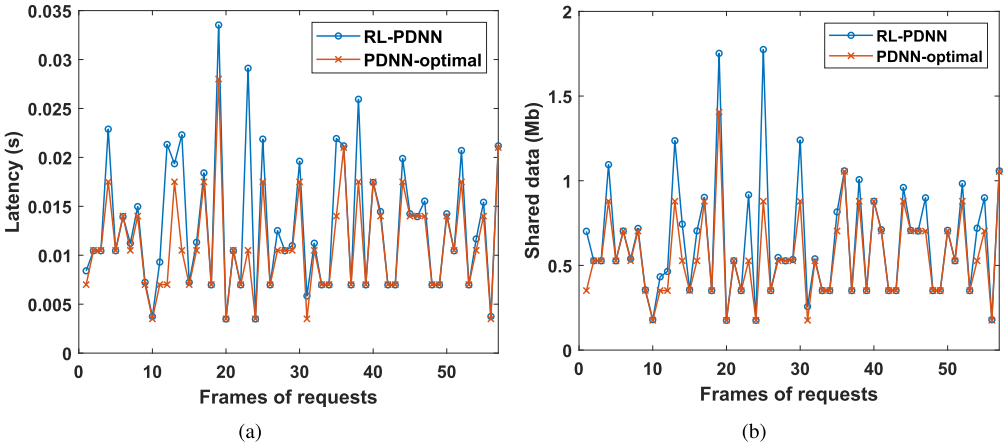

FIGURE 12. RL-PDNN vs PDNN-optimal for LeNet requests with no security requirements: (a) shows the latency, and (b) shows the shared data.

TABLE 2. RL-PDNN Vs privacy-preserving techniques.

Privacy Technique	Latency overhead	accuracy preserving	Compatibility with IoT devices and DNNs
RL-PDNN	✓	✓	✓
Cryptography	✓	✗	✗
Differential privacy	✓	✗	✗
Deep split	✓	✓	✗

Figure 12 illustrates the performance of our RL-PDNN approach compared to the optimal solution. Because of the high complexity of the problem defined in equation (4), the optimal simulation is conducted on a network with small parameters' values. More specifically, the network is receiving one type of requests namely LeNet and 30 IoT participants, without any privacy requirements. We note that we run our optimization on a 'No security' network as the same configuration in the RL is giving 100% accuracy (see Figure 6), which guarantees a fair comparison and lower complexity. Furthermore, the optimization is conducted on 60 frames of requests' arrival, where for each frame we represent the total latency (Figure 12(a)) and total shared data (Figure 12(b)). Distributing the classification requests optimally with reduced parameters and without privacy constraints contributes to relax the runtime of the simulation. Following this configuration, the optimization takes around 5 hours and if we add security requirements, it reaches more than 20 hours. The time complexity of the optimization justifies the design of an online solution adequate for real-time allocation. It is worth mentioning that our RL approach uses an MLP model with 2 layers and 64 neurons, which is a lightweight network with a small complexity. We note that we run our simulation using CVX tool[1] on a computer, having the following characteristics: core i7 and 16 GB RAM. Figure 12 shows that the RL-PDNN presents close results compared to the optimal results, exceeding 90% in some frames, which proves the high performance of our RL approach that is able to adapt to the system dynamics (devices capacities', load of requests, etc.).

To summarize:

- In this paper, we are studying an extremely dynamic system, where different IoT devices can participate or leave the collaborative framework. Moreover, the load of requests is highly dynamic and the available resources are volatile. Hence, we chose to adopt reinforcement learning that learns an optimal decision making policy from knowledge about the environment states and adapts to any network changes owing to the past learning.
- Our designed RL-PDNN is validated according to the described parameters in Table 1. Based on this configuration, we confirmed the convergence of our system for different types of networks and we proved that the accuracy is high for all privacy levels.
- Since the memory occupation and computation requirements of complex models restrain them from being processed on resource-limited devices and knowing that remote inference is costly in terms of latency, pervasive distribution of DNN tasks is an efficient solution to minimize the delay caused by the transmission overhead.
- By including more devices in the collaborative system or choosing performant participants, the capacity of the network becomes higher and sufficient particularly for inferences related to extremely complex networks such as VGG.
- Compared to existing approaches that propose static assignment of CNN computation, our RL approach is able to leverage the available resources and schedule the CNN partition by taking into consideration the dynamics of participants and online load of requests.
- Compared to the privacy-preserving techniques, our approach maintains the high accuracy of the model and supports heterogeneous IoT devices as well as

[1] http://cvxr.com/cvx/doc/

multiple DNNs and datasets, making the pervasive system a general-purpose platform for privacy-aware and low decision-latency applications.

V. CONCLUSION

In this paper, we re-thought the execution of deep learning inference, requiring high memory and computation demands, in order to fit it to the limited-resources characterizing the IoT units, while considering the white-box risks and real-time requirements of the classification requests. Our distribution approach is formulated as an NP-hard optimization problem, where the classification latency is minimized. Then, due to the complexity of the problem, we proposed an online RL-based solution, namely RL-PDNN that is adequate for real-time scenarios and dynamic systems. Our simulation revealed different parameters that should be present to achieve the distribution and privacy of online classifications, including the number of devices, their capacities, and the deployed networks. As a future work, we can apply compression strategies, e.g., pruning and quantization, in conjunction with our approach. The motivation behind this is to reduce the requirements of layers in terms of memory and computation, allocate more sequential segments in one device, and consequently minimize the latency and data transmission. We intend also to study the distribution of CNN networks into smaller segments (e.g., feature maps and multiplication tasks.) and exploit this distribution to protect the pervasive system from white and black-box attacks.

ACKNOWLEDGMENT

Open Access funding provided by the Qatar National Library.

REFERENCES

[1] M.-T. Luong, H. Pham, and C. D. Manning, "Effective approaches to attention-based neural machine translation," *CoRR*, vol. abs/1508.04025, pp. 1–11, Aug. 2015. [Online]. Available: http://arxiv.org/abs/1508.04025

[2] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *CoRR*, vol. abs/1512.03385, pp. 1–5, Dec. 2015. [Online]. Available: http://arxiv.org/abs/1512.03385

[3] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *Proc. Int. Conf. Learn. Represent.*, 2015, pp. 1–6.

[4] J. Mao, X. Chen, K. W. Nixon, C. Krieger, and Y. Chen, "MoDNN: Local distributed mobile computing system for deep neural network," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Lausanne, Switzerland, Mar. 2017, pp. 1396–1401.

[5] 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, Nov. 2018.

[6] E. Baccour, A. Erbad, A. Mohamed, M. Hamdi, and M. Guizani, "DistPrivacy: Privacy-aware distributed deep neural networks in IoT surveillance systems," 2020, *arXiv:2010.13234*. [Online]. Available: http://arxiv.org/abs/2010.13234

[7] 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, Jun. 2020.

[8] K. Ganju, Q. Wang, W. Yang, C. A. Gunter, and N. Borisov, "Property inference a acks on fully connected neural networks using permutation invariant representations," in *Proc. ACM Conf. Comput. Commun. Secur.*, 2018, pp. 1–3.

[9] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," *Commun. ACM*, vol. 60, no. 6, pp. 84–90, May 2017.

[10] H.-J. Kang, "Accelerator-aware pruning for convolutional neural networks," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 30, no. 7, pp. 2093–2103, Jul. 2020.

[11] S. Han, J. Pool, J. Tran, and W. J. Dally, "Learning both weights and connections for efficient neural networks," in *Proc. 28th Int. Conf. Neural Inf. Process. Syst.*, vol. 1, 2015, pp. 1–9.

[12] A. Marchisio, M. A. Hanif, M. Martina, and M. Shafique, "PruNet: Class-blind pruning method for deep neural networks," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, Jul. 2018, pp. 1–8.

[13] P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz, "Pruning convolutional neural networks for resource efficient inference," 2016, *arXiv:1611.06440*. [Online]. Available: http://arxiv.org/abs/1611.06440

[14] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding," 2015, *arXiv:1510.00149*. [Online]. Available: http://arxiv.org/abs/1510.00149

[15] H. Qin, R. Gong, X. Liu, X. Bai, J. Song, and N. Sebe, "Binary neural networks: A survey," *Pattern Recognit.*, vol. 105, Sep. 2020, Art. no. 107281.

[16] (Dec. 2020). *Edge TPU*. [Online]. Available: https://cloud.google.com/edge-tpu/

[17] M. Alzantot, Y. Wang, Z. Ren, and M. B. Srivastava, "RSTensorFlow: GPU enabled tensorflow for deep learning on commodity Android devices," in *Proc. 1st Int. Workshop Deep Learn. Mobile Syst. Appl. (EMDL)*, 2017, pp. 7–12.

[18] N. D. Lane, S. Bhattacharya, P. Georgiev, C. Forlivesi, L. Jiao, L. Qendro, and F. Kawsar, "Deepx: A software accelerator for low-power deep learning inference on mobile devices," in *Proc. 15thACM/IEEE Int. Conf. Inf. Process. Sensor Netw. (IPSN)*, Oct. 2016, pp. 1–12.

[19] 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 SIGOPS Oper. Syst. Rev.*, vol. 51, no. 2, pp. 615–629, Apr. 2017.

[20] A. Erfan Eshratifar, M. Saeed Abrishami, and M. Pedram, "JointDNN: An efficient training and inference engine for intelligent mobile cloud computing services," 2018, *arXiv:1801.08618*. [Online]. Available: http://arxiv.org/abs/1801.08618

[21] S. Teerapittayanon, B. McDanel, and H. T. Kung, "Distributed deep neural networks over the cloud, the edge and end devices," in *Proc. IEEE 37th Int. Conf. Distrib. Comput. Syst. (ICDCS)*, Jun. 2017, pp. 328–339.

[22] Z. Zhang, T. Song, L. Lin, Y. Hua, X. He, Z. Xue, R. Ma, and H. Guan, "Towards ubiquitous intelligent computing: Heterogeneous distributed deep neural networks," *IEEE Trans. Big Data*, early access, Nov. 26, 2018, doi: 10.1109/TBDATA.2018.2880978.

[23] 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. Surveys Tuts.*, vol. 22, no. 2, pp. 869–904, 2nd Quart., 2020.

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

[25] S. Disabato, M. Roveri, and C. Alippi, "Distributed deep convolutional neural networks for the Internet-of-Things," 2019, *arXiv:1908.01656*. [Online]. Available: http://arxiv.org/abs/1908.01656

[26] Z. He, T. Zhang, and R. B. Lee, "Model inversion attacks against collaborative inference," in *Proc. 35th Annu. Comput. Secur. Appl. Conf.*, New York, NY, USA, Dec. 2019, pp. 148–162.

[27] C. Alippi, S. Disabato, and M. Roveri, "Moving convolutional neural networks to embedded systems: The AlexNet and VGG-16 case," in *Proc. 17th ACM/IEEE Int. Conf. Inf. Process. Sensor Netw. (IPSN)*, Apr. 2018, pp. 212–223.

[28] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan, "Deep learning with limited numerical precision," in *Proc. 32nd Int. Conf. Int. Conf. Mach. Learn. (ICML)*, vol. 37. JMLR.org, 2015, pp. 1737–1746.

[29] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," 2015, *arXiv:1512.03385*. [Online]. Available: http://arxiv.org/abs/1512.03385

[30] K. V. M. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjel, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 7540, pp. 529–533, Feb. 2015.

[31] L. Lyu, J. C. Bezdek, J. Jin, and Y. Yang, "FORESEEN: Towards differentially private deep inference for intelligent Internet of Things," *IEEE J. Sel. Areas Commun.*, vol. 38, no. 10, pp. 2418–2429, Oct. 2020.
[32] Z. Ghodsi, A. Veldanda, B. Reagen, and S. Garg, "CryptoNAS: Private inference on a ReLU budget," 2020, *arXiv:2006.08733*. [Online]. Available: http://arxiv.org/abs/2006.08733
[33] Q. Lou and L. Jiang, "SHE: A fast and accurate deep neural network for encrypted data," 2019, *arXiv:1906.00148*. [Online]. Available: http://arxiv.org/abs/1906.00148

EMNA BACCOUR received the Ph.D. degree in computer science from the University of Burgundy, France, in 2017. She was a Postdoctoral Fellow with Qatar University on a project covering the interconnection networks for massive data centers and then on a project covering video caching and processing in mobile edge computing networks. She currently holds a postdoctoral position at Hamad Ben Khalifa University. Her research interests include data center networks, cloud computing, green computing, software defined networks, distributed systems, edge networks, and mobile edge caching and computing.

AIMAN ERBAD (Senior Member, IEEE) received the Master of Computer Science in embedded systems and robotics from the University of Essex, U.K., and the Ph.D. degree in computer science from The University of British Columbia, Canada. He is currently an Associate Professor with the College of Science and Engineering, Hamad Bin Khalifa University (HBKU). His research interests include cloud computing, edge computing, the IoT, private and secure networks, and multimedia systems. He received the Platinum Award from the H. H. The Emir Sheikh Tamim Bin Hamad Al Thani at the Education Excellence Day 2013 (Ph.D. category), the 2020 Best Research Paper Award from the *Computer Communications* journal, the IWCMC 2019 Best Paper Award, and the IEEE CCWC 2017 Best Paper Award. He is also an Editor of *KSII Transactions on Internet and Information Systems* and a Guest Editor of *IEEE Networks*.

AMR MOHAMED (Senior Member, IEEE) received the M.S. and Ph.D. degrees in electrical and computer engineering from The University of British Columbia, Vancouver, Canada, in 2001, and 2006 respectively. He has worked as an Advisory IT Specialist with the IBM Innovation Centre, Vancouver, from 1998 to 2007, taking a leadership role in systems development for vertical industries. He is currently a Professor with the College of Engineering, Qatar University, and the Director of the Cisco Regional Academy. He has over 25 years of experience in wireless networking research and industrial systems development. He has authored or coauthored over 160 refereed journal and conference papers, textbook, and book chapters in reputable international journals, and conferences. His research interests include wireless networking, and edge computing for IoT applications. He holds three awards from IBM Canada for his achievements and leadership, and four Best Paper Awards from IEEE conferences. He has served as the Technical Program Committee (TPC) Co-Chair for workshops in IEEE WCNC'16. He has also served as the Co-Chair for technical symposia of international conferences, including Globecom'16, Crowncom'15, AICCSA'14, IEEE WLN'11, and IEEE ICT'10. He has served on the organization committee for many other international conferences as a TPC member, including the IEEE ICC, GLOBECOM, WCNC, LCN, and PIMRC, and a Technical Reviewer for many international IEEE, ACM, Elsevier, Springer, and Wiley journals. He is also serving as a Technical Editor for the *Journal of Internet Technology* and the *International Journal of Sensor Networks*.

MOUNIR HAMDI (Fellow, IEEE) received the B.S. degree (Hons.) in electrical engineering (computer engineering) from the University of Louisiana, in 1985, and the M.S. and Ph.D. degrees in electrical engineering from the University of Pittsburgh, in 1987 and 1991, respectively. He was the Chair Professor and a Founding Member of The Hong Kong University of Science and Technology (HKUST), where he was the Head of the Department of Computer Science and Engineering. From 1999 to 2000, he held a Visiting Professor positions with Stanford University and the Swiss Federal Institute of Technology. He is currently the Founding Dean of the College of Science and Engineering, Hamad Bin Khalifa University (HBKU). He has published more than 360 publications, graduated more 50 M.S./Ph.D. students, and awarded numerous research grants. His research interest includes high-speed wired/wireless networking. In addition, he has frequently consulted for companies and governmental organizations in the USA, Europe, and Asia. He is also a Fellow of the IEEE for his contributions to design and analysis of high-speed packet switching, which is the highest research distinction bestowed by IEEE. He is also a Frequent Keynote Speaker in international conferences and forums. He is/was on the editorial board of more than ten prestigious journals and magazines. He has chaired more than 20 international conferences and workshops. In addition to his commitment to research and academic/professional service, he is also a Dedicated Teacher and a Quality Assurance Educator. He received the Best ten Lecturer Award and the Distinguished Engineering Teaching Appreciation Award from HKUST. He is frequently involved in higher education quality assurance activities and engineering programs accreditation all over the world.

MOHSEN GUIZANI (Fellow, IEEE) received the B.S. (Hons.) and M.S. degrees in electrical engineering, and the M.S. and Ph.D. degrees in computer engineering from Syracuse University, Syracuse, NY, USA, in 1984, 1986, 1987, and 1990, respectively. He served in different academic and administrative positions at the University of Idaho, Western Michigan University, the University of West Florida, the University of Missouri–Kansas City, the University of Colorado–Boulder, and Syracuse University. He is currently a Professor with the Computer Science and Engineering Department, Qatar University, Qatar. He is the author of nine books and more than 500 publications in refereed journals and conferences. His research interests include wireless communications and mobile computing, computer networks, mobile cloud computing, security, and smart grid. He is also a Senior Member of ACM. He also served as a member, the Chair, and the General Chair for a number of international conferences. Throughout his career, he received three teaching awards and four research awards. He also received the 2017 IEEE Communications Society WTC Recognition Award and the 2018 AdHoc Technical Committee Recognition Award for his contribution to outstanding research in wireless communications and Ad-Hoc Sensor networks. He was the Chair of the IEEE Communications Society Wireless Technical Committee and the of the TAOS Technical Committee. He is also the Editor-in-Chief of the *IEEE Network Magazine*, serves on the editorial boards of several international technical journals and the Founder and Editor-in-Chief of *Wireless Communications and Mobile Computing* (Wiley) journal. He guest edited a number of special issues in IEEE journals and magazines. He served as the IEEE Computer Society Distinguished Speaker. He is also the IEEE ComSoc Distinguished Lecturer.