IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received July 16, 2020, accepted July 28, 2020, date of publication August 7, 2020, date of current version August 20, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3014922

Knowledge Transfer for On-Device Deep Reinforcement Learning in Resource Constrained Edge Computing Systems

INGOOK JANG, HYUNSEOK KIM, DONGHUN LEE, YOUNG-SUNG SON, AND SEONGHYUN KIM

Autonomous IoT Research Section, Electronics and Telecommunications Research Institute (ETRI), Daejeon 34129, South Korea

Corresponding author: Seonghyun Kim (kim-sh@etri.re.kr)

This work was supported by the Electronics and Telecommunications Research Institute (ETRI) funded by the Korean Government (Core Technologies of Distributed Intelligence Things for Solving Industry and Society Problems) under Grant 20ZR1100.

ABSTRACT Deep reinforcement learning (DRL) is a promising approach for developing control policies by learning how to perform tasks. Edge devices are required to control their actions by exploiting DRL to solve tasks autonomously in various applications such as smart manufacturing and autonomous driving. However, the resource limitations of edge devices make it unfeasible for them to train their policies from scratch. It is also impractical for such an edge device to use the policy with a large number of layers and parameters, which is pre-trained by a centralized cloud infrastructure with high computational power. In this paper, we propose a method, on-device DRL with distillation (OD3), to efficiently transfer distilled knowledge of how to behave for on-device DRL in resource-constrained edge computing systems. Our proposed method makes it possible to simultaneously perform knowledge transfer and policy model compression in a single training process on edge devices with considering their limited resource budgets. The novelty of our method is to apply a knowledge distillation approach to DRL based edge device control in integrated edge cloud environments. We analyze the performance of the proposed method by implementing it on a commercial embedded system-on-module equipped with limited hardware resources. The experimental results show that 1) edge policy training with the proposed method achieves near-cloud-performance in terms of average rewards, although the size of the edge policy network is significantly smaller compared to that of the cloud policy network and 2) the training time elapsed for edge policy training with our method is reduced significantly compared to edge policy training from scratch.

INDEX TERMS Deep reinforcement learning, edge computing, edge AI, knowledge transfer, policy model compression, on-device training.

I. INTRODUCTION

Today, the Internet of Things (IoT) is used for a wide range of industrial applications, including smart cities [1], autonomous transportation [2], urban surveillance [3], and intelligent manufacturing [4]. Although a cloud computing approach has been adopted to enhance the applicability of IoT in a centralized manner, it has been faced with challenges such as scalability, bandwidth efficiency, and privacy protection [5]. To alleviate these limitations of cloud-based systems, edge computing is emerging as a complementary strategy, which places data processing at the edge of the network [6].

Various edge devices in complex applications are required to make optimal control actions with minimal human intervention. To meet such demands, the state-of-the-art edge computing systems employ artificial intelligence (AI) at the edge (edge AI) or on devices (on-device AI) to handle not only data processing but decision making for control. Sophisticated AI techniques (for not only inferencing or predicting but even training) need to be applied to provide intelligent edge services although most edge devices suffer from hardware resource constraints. The edge and on-device AI play a crucial role to bring various benefits such as immediate response, privacy preservation, enhanced availability (even without network connection), efficient use of network bandwidth, and low cost.

The associate editor coordinating the review of this manuscript and approving it for publication was Tai-Hoon Kim.

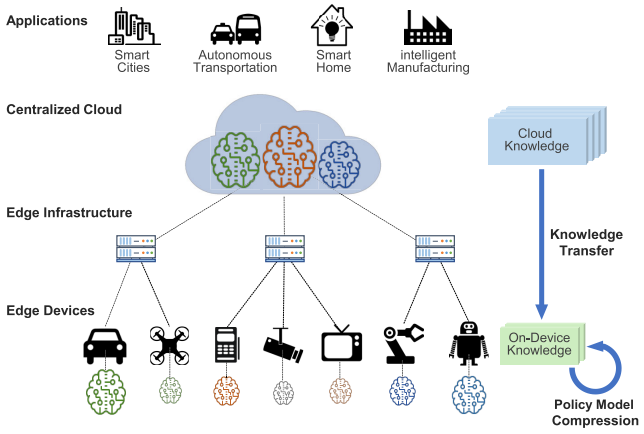

FIGURE 1. Illustration of an edge computing architecture. To efficiently train control policies on resource-constrained edge devices, the distilled knowledge is required to be transferred from the cloud to edge devices and should be compressed depending on their limited resource budgets.

As an emerging AI technique, DRL provides a deep learning-based approach for an agent (i.e., device) to learn how to perform a task well and control its actions by trial-and-error interactions with an environment. The basic idea of DRL is to train an effective control policy (i.e., model) by utilizing the powerful approximation capabilities of deep neural networks (DNN). A number of works using DRL have studied the challenges of cloud computing, such as resource allocation [7]. Recently, DRL can be considered as a key technology to enable edge devices to solve complex tasks intelligently in real-world applications such as smart manufacturing [8] and autonomous vehicle [9].

For on-device AI, it should be considered not only to transfer knowledge of the pre-trained control policy from the cloud to edge devices but also to compress the policies of the edge devices depending on their hardware resources, as shown in Fig. 1. Training a DRL model (i.e., policy) requires high computational power and sufficient hardware resources such as processors (e.g., CPUs and GPUs), memory, storage, and power supply. Due to the limited resource capacity of edge devices, it is hard for them to train their control policies from scratch by leveraging DRL methods [10]. Moreover, it is impractical for resource-limited edge devices to use control policies trained by centralized cloud infrastructure with high computational power and sufficient hardware resources. The available hardware resources of the edge devices may not be sufficient to execute inference tasks (i.e., action prediction). This is why it is hard for edge devices to handle the pre-trained policies transferred from the cloud systems, which tend to be large and deep with dozens of hidden layers and millions of neurons. Typically, a DNN with many layers and parameters requires considerable computation for testing (e.g., inference) as well as training. To overcome resource limitations at the edge devices when training a DRL model, it is a key challenge to transfer only essential action knowledge from the cloud to them. Policy model compression at edge devices utilizing the transferred knowledge is also challenging to meet their resource constraints significantly depending on the device type and manufacturers.

This paper investigates a method to efficiently transfer distilled knowledge of how to behave for edge device control using Deep Reinforcement Learning (DRL) in resource-constrained edge computing systems, where a huge number of various edge devices are connected through cloud infrastructure. Our proposed method, on-device DRL with distillation (OD3), makes it possible to simultaneously perform knowledge transfer and policy model compression in a single training process on edge devices with considering their limited resource budgets.

Our main contributions in this paper are as follows:

- By applying a policy distillation technique [11] to integrated edge cloud computing systems, we propose a method to transfer distilled knowledge from a pre-trained policy of the cloud to resource-constrained edge devices, referred to as *on-device DRL with distillation (OD3)*. Our proposed method aims to simultaneously conduct knowledge transfer and policy model compression in a single training process on edge devices by leveraging the policy distillation.
- To the best of our knowledge, there has not been any study on knowledge transfer for control of real-world resource-constrained edge devices for DRL in integrated edge cloud computing systems. We perform the first comprehensive and most recent study on this problem and demonstrate the feasibility by applying OD3 to a commercial hardware platform.
- We demonstrate the advantages of the OD3 for resource-constrained edge computing systems. The performance of the OD3 is validated and analyzed from various perspectives. In particular, the experimental results show that the performance of edge policy training with OD3 reaches very close to that of cloud policy training in terms of average rewards, although the size of the edge policy network is significantly smaller compared to that of the cloud policy network. The results also show that the training time elapsed for edge policy training with OD3 is reduced significantly compared to edge policy training from scratch.

This paper discusses the related work in Section II and the essential preliminaries of DRL and knowledge distillation in Section III. The proposed OD3 is described in Section IV and the experimental results are shown in Section V. Section VI presents the conclusion and future work.

II. RELATED WORK

A. DEEP REINFORCEMENT LEARNING

Reinforcement learning (RL) is creating a new paradigm shift in machine learning, which allows agents to decide on their sequential actions for autonomous control. It enables agents to learn their control policies in a way to maximize the expected reward through a large number of trial-and-error interactions with an environment, as shown in Fig. 2. DRL, as RL combined with a DNN, has recently

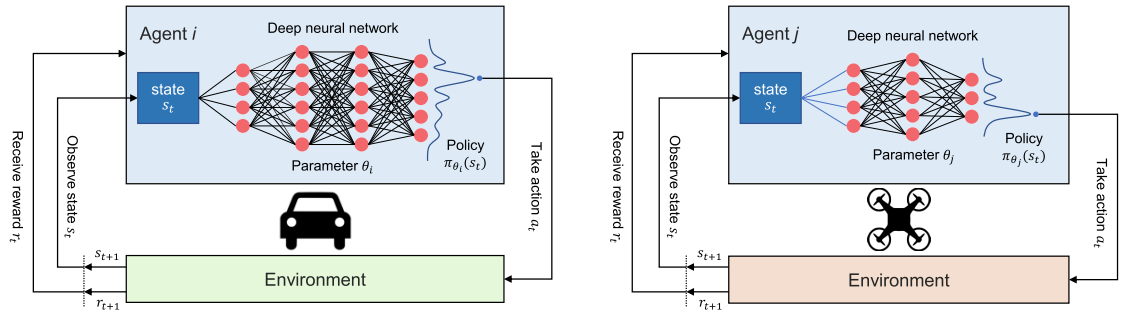

FIGURE 2. Agent-environment interactions via deep reinforcement learning. Each edge device takes its limited resource budget into account and generates a different policy network architecture.

achieved great success in the fields of playing Atari games [12] or Go [13], controlling continuous actions in robotics [14], and autonomous driving (including flight) [9], [15]. Deep Q-Networks (DQN) [12] is one of the representative algorithms of DRL and enhances the performance of the Q-learning algorithm by adopting a DNN and a replay-buffer memory. However, DRL is known as a simple and easy implementation in a simulation environment consisting of sufficient resources but as hard to be realized in a real-world environment, such as edge devices equipped with limited hardware resources [16].

B. DRL IN THE EDGE

There are many kinds of research employing DRL approaches to solve crucial challenges in edge computing systems, such as computation offloading [17], [18], resource orchestration [19], and mobile edge caching [20]. These studies have focused on the network performance improvement of edge architectures rather than on the realization of intelligent edge devices capable of deciding their actions for solving tasks.

For the realization of edge and on-device AI, inference on devices is the important first step. However, on-device training also must be explored to overcome the crucial challenges such as privacy, scalability, and latency. So far, only a few studies have been conducted for on-device training based on supervised and unsupervised learning algorithms. Fang *et al.* [21] proposed an approach to train a convolutional neural network (CNN) by considering the dynamics of runtime resources for mobile vision systems. Xu *et al.* [22] introduced DeepType using on-device deep learning to personalize user input for preserving privacy.

In the case of RL, there has not been an on-device DRL training method for resource-constrained edge devices. For a complex task, it is often impractical to train a DRL policy from scratch under a limited resource budget. Moreover, different hardware resource budgets among edge devices should be considered to support DRL training for heterogeneous edge devices, as shown in Fig. 2. To the best of our knowledge, our research is the first meaningful study on this problem, which utilizes both knowledge transfer and policy model compression.

C. TRANSFER LEARNING

Transfer learning is a simple technique of exploiting the parameters of a pre-trained neural network for one task and adapting them to a new neural network for the other task. This approach can be useful when there is insufficient training data set for the new task and not much time to execute training processes. In integrated edge cloud computing environments, transfer learning can be a key promising technique to realize on-device training for resource-constrained edge devices [23]. Li *et al.* [24] introduced on-device training based on a transfer learning technique for visual recognition in edge-based IoT environments. In RL tasks, the parameters of a DRL policy network (referred to as knowledge) obtained by cloud policy training can be transferred to a new edge policy network to enable DRL training even though there is only an insufficient hardware resource budget and not enough time for training. However, it is impractical to apply transfer learning for DRL into edge devices with different resource constraints.

D. MODEL COMPRESSION

Another way for efficient on-device inference and training is to compress a well-trained existing model. DNNs for RL tasks typically have dozens of layers and millions of parameters. Moreover, the architecture of DNNs becomes more complicated as the tasks become more complex. Recently, there are various approaches to compress such pre-trained models [25]–[28]. These model compression approaches promise considerable savings on the model complexity, power efficiency, and inference time. Knowledge distillation is one of the most popular approaches to compress a DNN, which distills knowledge from a pre-trained model and transfers it to a new model with a smaller number of layers and parameters [29], [30]. Recently, Hinton *et al.* [31] proposed an approach to distill knowledge from an ensemble of large models (i.e., a teacher model) into a small single model (i.e., a student model). This approach can compress a student model by utilizing softened softmax outputs of the teacher model, which have more knowledge than traditional softmax outputs. The student model tries to mimic the teacher model by minimizing the loss (e.g., cross-entropy) between the outputs of the teacher and student. Rusu *et al.* [11] proposed policy distillation, which applied knowledge distillation to RL policy training. This study uses a teacher-student policy training for transferring action knowledge from one or more policies to an untrained model.

In this work, we do not simply focus on enabling RL training on devices. From a fully different perspective, we try to

view and address DRL training among resource-constrained edge devices in edge cloud computing systems. The main similarities and differences between the proposed method and the surveyed studies are summarized as follows:

E. SIMILARITIES

- A deep learning-based approach is adopted for solving real-world problems in edge computing systems.
- Transferred knowledge can be useful to build a model efficiently, though there is insufficient training data or a limited resource budget.
- The saving on model complexity is achieved by using a model compression method.

F. DIFFERENCES

- This study focuses on how to realize on-device DRL for resource-constrained edge computing systems, which cooperate with cloud infrastructure.
- The proposed method addresses the diversity of resource constraints of individual edge devices.
- It also demonstrates the effectiveness of applying a distillation approach to a real-world edge cloud system.

III. BACKGROUND

A. REINFORCEMENT LEARNING

An RL task is defined by $M = (S, A, P, r)$, where M is a Markov decision process (MDP) described as follows:

1) A set of states $S = \{s_1, s_2, \ldots, s_n\}$, where $s_t \in S$ at every timestep t.
2) An action space $A = \{a_1, a_2, \ldots, a_m\}$ available to the RL agent at each state, where $a_t \in A(s_t)$ is an action choice based on its policy π: $a_t = \pi(s_t)$.
3) A distribution of state transition $P = p(s_{t+1}|s_t, a_t)$, which is a mapping from state-action pairs s_t, a_t to a probability distribution over the next states.
4) A distribution of initial states $P_0 = p(s_0)$, where every episode starts with sampling an initial state independently.
5) A reward function $r_t = r(s_t, a_t)$.
6) A discount factor $\gamma \in [0, 1]$.

The expected discounted return at time t is given by $R_t = \sum_{t'=t}^{\infty} \gamma^{t'-t} r_{t'}$ and the goal of the RL agent is to maximize its expected return $E_{s_0}[R_0|s_0]$. The action-value function (e.g., Q function) is defined as $Q^{\pi}(s, a) = \mathbb{E}[R_t|s_t = s, a_t = a, \pi]$, which represents the expected discounted return after observing the state s and taking the action a depending on the policy π. The *optimal Q function* Q^* satisfies the following Bellman equation:

$$Q^*(s, a) = \mathbb{E}_{s' \sim p(\cdot|s,a)}\Big[r(s, a) + \gamma \max_{a' \in A} Q^*(s', a')\Big] \quad (1)$$

B. DEEP Q-NETWORK (DQN)

DQN algorithm is a model-free approach for RL using DNNs in environments with discrete action spaces, which optimizes neural networks to approximate the optimal Q function Q^*. In DQN, the expected discounted future return of each possible action is predicted at timestep t and the RL agent take the action with the highest predicted return: $\pi_Q(s_t) = \arg\max_{a \in A} Q(s_t, a)$. During training the RL agent collects the tuples (s, a, r, s') from its experience and stores them in an experience replay memory, which is a key technique to improve training performance in the DQN algorithm. The purpose of the replay memory is to remove correlations between samples experienced by the agent. The neural network to approximate $Q^*(s, a)$ is trained using a mini-batch gradient descent approach and minimizes the following loss by using samples (s, a, r, s') from the replay memory: $L = \mathbb{E}_{s,a,r,s'}[(Q(s, a) - y)]^2$, where $y = r + \gamma \max_{a' \in A} Q(s', a')$. In DQN, the RL agent uses a separate target Q-network, which has the same architecture as the original Q-network but with frozen parameters. The purpose of the target network is to temporarily fix the Q value targets because non-stationary targets make a training process unstable and reduce performance. The parameters of the target Q-network θ^- are updated with that of the original Q-network θ every fixed number of iterations. For the use of the target Q-network, the loss function can be reformulated as follows:

$$L(\theta) = \mathbb{E}_{s,a,r,s'}\Big\{\overbrace{Q(s, a; \theta)}^{\text{prediction}} - \underbrace{[r + \gamma \max_{a' \in A} Q(s', a'; \theta^-)]}_{\text{target}}\Big\}^2 \quad (2)$$

Since DQN uses two Q-networks with identical neural network architecture, training may require more computational resources such as memory.

C. KNOWLEDGE DISTILLATION

Knowledge distillation is a transfer learning approach to distill useful knowledge from a teacher model and transfer it to a student model. In a distillation process, the distilled knowledge can be transferred from a teacher network to a student network by utilizing *softened targets* parameterized by a temperature τ [31]. For a higher temperature $\tau > 1$, teacher's outputs (e.g., logit vector) z_i can be softened and converted into a probability distribution by passing them through a softmax function:

$$F_{sm}(z_i/\tau) = \frac{exp(z_i/\tau)}{\sum_j exp(z_j/\tau)} \quad (3)$$

In the classification problem, raising the temperature enables more of the knowledge to be transferred to the student network since the teacher's output typically tends to be very peaked. However, in the RL setting, the softmax function is used to make the distribution sharper by lowering the temperature $\tau < 1$ because the teacher's output is a set of the expected discounted return values for its action space [11]. If the temperature goes to 0, a softmax function becomes greedy. Otherwise, it is computed as a softmax function with a Boltzmann distribution shown in Equation (3). The *sharpened distribution* not only provides more unambiguous information for action selection but also serves as a regression target for the student training. These characteristics enable an RL training process using distillation to be accelerated.

Algorithm 1: On-Device DRL With Distillation

Input: Pre-trained cloud policy network parameters (teacher) θ_T; Resource budget R; Training step number T; Temperature τ

Output: Edge policy network parameters (student) θ_S

Initialize the architecture of the edge policy network using $PolicyNetGenerator(R)$
Randomly initialize θ_S; Initialize $t \leftarrow 1$
while $t \leq T$ **do**
  Compute the sharpened target of the cloud policy network $F_{sm}(\zeta_t^T/\tau; \theta_T)$ (Equation (3) with a temperature τ)
  Compute the prediction of the edge policy network $F_{sm}(\zeta_t^S; \theta_S)$ (Equation (3) with a temperature 1)
  Compute the loss function $\sum_t l_{kl}(\zeta_t^T, \zeta_t^S, \tau)$ (Equation (4))
  Update the edge policy network parameters θ_S through a SGD algorithm
end

IV. ON-DEVICE DRL WITH DISTILLATION

The proposed OD3 allows an edge device to train its control policy on its hardware platform by using the distillation technique. The proposed approach has two advantages over existing methods. The first is to accelerate a policy training process of an edge device compared to that from scratch since the OD3 adopts a sharpened target distribution to transfer much more unambiguous knowledge of how good it would be to take a specific action in a given state. The second is to enable an edge device to compress its control policy based on its resource budgets by utilizing the transferred knowledge from the cloud.

The proposed OD3 will be described based on the DQN algorithm mentioned in Section III-B. As one of the most popular RL algorithms, DQN efficiently works in the simple control problems (e.g., Atari games) with discrete action spaces. Therefore, DQN serves as a good baseline for performance comparisons as well as a base algorithm for teacher training performed in the cloud. Although several advanced algorithms that outperform the DQN have been proposed, we choose the vanilla DQN to focus on exploring the feasibility of OD3.

A. LOSS FUNCTION

The OD3 uses a softmax function with a lower temperature $\tau < 1$ to sharpen the teacher's output. We assume that a control policy pre-trained by the cloud and that to be trained by an edge device play roles of a teacher T and a student S, respectively, in an integrated edge cloud environment. To distill the knowledge from the cloud policy network, the difference between two distributions of the cloud policy network and edge policy network is computed, which are given by $F_{sm}(\zeta^T/\tau)$ and $F_{sm}(\zeta^S)$, respectively, where ζ^T and ζ^S are the logit vectors from the output layers of the policy networks of the teacher T and the student S, respectively.

The OD3 allows each edge device to train its control policy by optimizing the Kullback-Leibler (KL) divergence, which measures the difference between two given distributions. In [11], the KL divergence showed superior performance compared to the mean squared error (MSE) and negative log likelihood (NLL), as viewing the outputs of the teacher and student as distributions over the actions to be taken. The loss function to be minimized is formulated as follows:

$$L_{kl}(D, \theta_S, \tau) = \sum_t l_{kl}(\zeta_t^T, \zeta_t^S, \tau), \tag{4}$$

where

$$l_{kl}(\zeta_t^T, \zeta_t^S, \tau) = F_{sm}(\zeta_t^T/\tau) \log \frac{F_{sm}(\zeta_t^T/\tau)}{F_{sm}(\zeta_t^S)}. \tag{5}$$

Here θ_S denotes a set of parameters of the student network to be trained, $D = \{(s_t, \zeta_t^T)\}_{t=0}^N$ is a dataset sampled by the teacher, and ζ_t^T and ζ_t^S represent the logit vectors generated from a cloud policy network and edge policy network at timestep t, respectively.

B. ALGORITHM DETAILS

The OD3 details are summarized in Algorithm 1. $PolicyNetGenerator$ with the input R initializes the architecture of a policy network to be trained by using its resource budget, which includes information about the shape of a neural network architecture used for edge policy training with considering the resource budget. For instance, such information contains the numbers of layers (e.g., convolutional or fully connected layers) and output filters when a CNN is optimized. In the following experiment, the architectures of neural networks used for cloud and edge policy training are heuristically chosen. To improve the efficiency of on-device AI performance, constructing a ''better'' neural network architecture for a resource-constrained edge device can be realized by exploiting emerging network architecture search (NAS) techniques [32]–[34].

The edge device performs the RL training with knowledge distilled from the pre-trained cloud policy network after neural network generation. The edge device computes the loss function described in Equation (4) by using its prediction and the sharpened target of the cloud policy network. Then, the edge device updates its policy network parameters by carrying out stochastic gradient descent (SGD) within each timestep. The distilled action knowledge can be transferred from the cloud to the edge device in a single training process.

Figure 3 depicts the proposed OD3 based on the DQN algorithm. To approximate the Q function, CNNs are

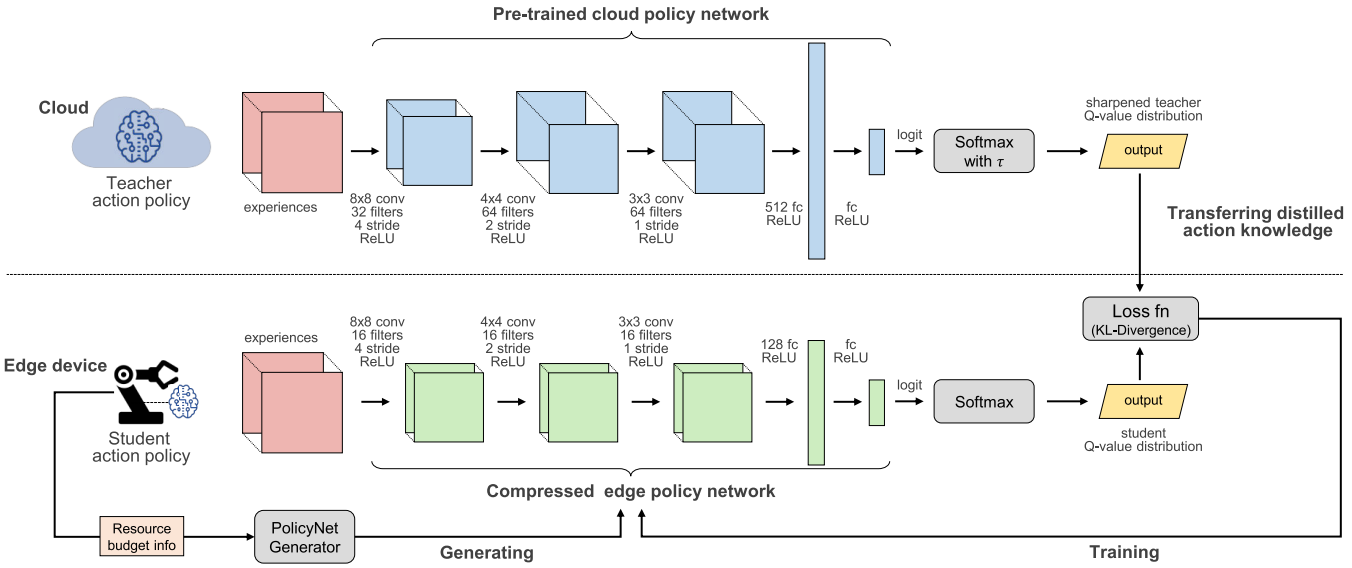

FIGURE 3. Our approach based on the DQN algorithm. This method enables an edge device to simultaneously conduct knowledge transfer and policy model compression considering its limited resource budgets in a single training process on the device. The edge device utilizes the distilled action knowledge from the pre-trained cloud policy to train its own optimized policy.

employed, which passes observation images as inputs. Depending on the resource budget, the edge device generates the smaller CNN that mimics the behavior produced from the pre-trained cloud CNN. The smaller CNN of the edge device approximates the function learned by the cloud policy network. Our approach enables the edge device to complete cloud-level RL training with a significantly reduced number of timesteps compared to edge policy training from scratch. Moreover, the performance of the newly trained edge policy reaches close to that of the pre-trained cloud policy while using only the significantly compressed policy network compared to the cloud policy.

V. EVALUATION

This section investigates the performance of the proposed OD3. We use Tensorflow [35] to train DNNs both in a cloud server and an edge device. We first introduce the details of the experiment setup and then present the corresponding results. The experimental results demonstrate the advantages of our method over plain RL training approaches in integrated edge cloud settings.

A. EXPERIMENTAL SETUP

1) HARDWARE

We run our teacher policy training on a GPU server that can be treated as an entity of the cloud infrastructure. The GPU server is composed of two CPUs (Intel Xeon processors with 2.3GHz each), a single GPU (NVIDIA Tesla V100 with 5120 CUDA cores), and 128GB RAM. For student policy training performed by an edge device, a popular embedded system-on-module (NVIDIA Jetson TX2) is used, which consists of two CPUs (a dual-core NVIDIA Denver 2 and a quad-core ARM A57), a single GPU (NVIDIA Pascal GPU with 256 CUDA cores), and 8GB RAM shared between the CPUs and GPU. This system-on-module has been commonly used for various edge applications such as autonomous cars, drones, and robotics. Table 1 summarizes the specifications of both the cloud server and the edge device used in our experiments.

TABLE 1. Specifications of the cloud server and the edge device used in the experiments.

H/W	Cloud (GPU server)	Edge device (System-on-module)
CPU	2 Intel Xeon 5118 2.3GHz	Quad-core ARM A57 Dual-core NVIDIA Denver 2
GPU	NVIDIA Tesla V100 (5120 CUDA cores)	NVIDIA Pascal GPU (256 CUDA cores)
Memory	128GB (DDR4 32GB×4)	8GB (LPDDR4 8GB×1)
Storage	2TB Solid State Disk	32GB eMMC 5.1
Power	1600W - 2200W	7.5W - 15W

2) BENCHMARK AND BASELINES

To evaluate the impact of the proposed OD3, we focus on the Atari platform (especially a *Pong* game), which is a representative benchmark problem for examining the performance of RL agent control. Since Pong has a discrete action space (i.e., 6 actions), the output layers before the softmax functions both in the cloud policy network and the edge policy network have 6 units each. The discounted future reward is defined as $R_t = \sum_{t'=t}^{T} \gamma^{t'-t} r_{t'}$, where T is the training timestep.

We compare the performance between edge policy training with the proposed OD3 (represented as *EDGE-OD3*), edge policy training without the OD3 (represented as *EDGE*), and cloud policy training (represented as *CLOUD*). Comparing *EDGE-OD3* with *EDGE* and *CLOUD* provides a good baseline on how much improvement our method achieves in terms of average rewards, time elapsed for each training, and policy network size.

TABLE 2. Major hyper-parameters for the training procedures on the cloud server and the edge device.

Hyper-parameter	Cloud (GPU server)	Edge device (System-on-module)
ϵ-greedy exploration	1 to 0.1	1 to 0.1
Learning rate	$2.5e^{-4}$ to $5e^{-5}$	$2.5e^{-4}$ to $5e^{-5}$
Training timesteps	$3e^{6}$	$3e^{6}$
Replay memory size	$1e^{6}$	$1e^{6}$
Discount factor γ	0.99	0.99
Mini-batch size	32	**32 to 4**
Update frequency	4	**4 to 32**

3) HYPER-PARAMETERS

Policy training processes both in the cloud and the edge device are performed with the architectures illustrated in Fig. 3. Each policy is trained with three convolutional layers and two fully connected layers. The edge device scales down the numbers of each layer's filters (i.e., the dimensionality of the output space of each layer) depending on its limited resource budget. In our experiments, a smaller neural network (i.e., 16, 16, 16, 128, and 6 filters) is used for training on the edge device while a larger neural network (i.e., 32, 64, 64, 512, and 6 filters) is learned by the cloud server. We choose the architectures of these two neural networks heuristically and refer them to as a *smallnet* and *largenet* in experimental results, respectively. Rectifier linear units (i.e., ReLU) are used as activation functions between every two successive layers. The temperature τ for teacher policy is set to 0.01 empirically selected for the best OD3 performance. The KL divergence loss function defined in Equation (4) is used.

Table 2 summarizes the major hyper-parameter settings for the training procedures on the cloud server and the edge device used in our experiments. Most of the hyper-parameters used in our experiments are identical to the parameters used in [12]. The *Adam* optimizer [36] is employed with the leaning rate decayed from $2.5e^{-4}$ to $5e^{-5}$, which can be used for the update of the neural network parameters by conducting mini-batch gradient descent. In the evaluation procedures, the trained agent acts with a value of $\epsilon = 0.05$.

We investigate the performance of the proposed method from two perspectives: 1) an experiment using the exactly identical hyper-parameters between the cloud and the edge device (*Case 1*) and 2) an experiment with varying hyper-parameters closely related to the limited hardware resource of the edge device (*Case 2*). In *Case 2*, the same hyper-parameter setting is used for training both in the cloud and the edge device, except for the mini-batch size and the update frequency. The use of small batches improves memory efficiency by exploiting a significantly smaller memory footprint [37]. The update frequency means the number of timesteps between consecutive SGD updates. Since the computational cost of an update to the neural network is more expensive than that of a forward pass through the neural network, the larger value of the update frequency saves computation and increases training speed [38]. Therefore, in the edge device's settings, we gradually increase the value of the update frequency (4, 8, 16, and 32) and gradually decrease the mini-batch size (32, 16, 8, and 4) to study how performance changes as the two hyper-parameters vary. The experimental results of *Case 1* and *2* are shown in Section V-B1 and V-B2, respectively.

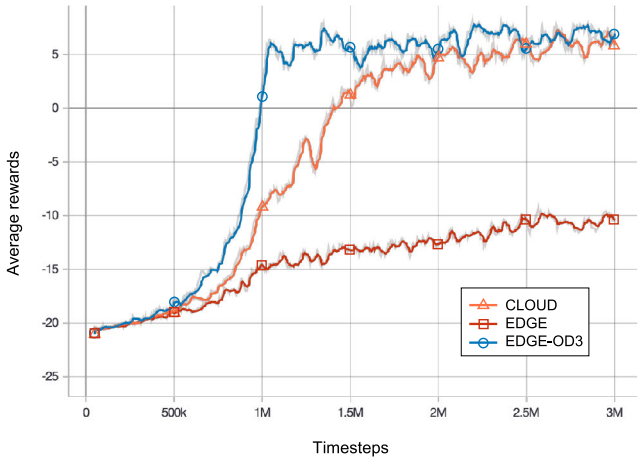

FIGURE 4. Average rewards with respect to the number of timesteps. The gray lines depict the actual average reward values measured from the experiments. The colored lines show the smoothed ones, which can be useful for illustrating the overall trend.

TABLE 3. Comparison of the major experimental results between the cloud policy network and two variants of the edge policy network. It includes the number of parameters, policy network size, and training time elapsed until first reaching the convergence reward point. This excludes the result of the training time elapsed for *EDGE* because it does not converge within the timesteps defined in Table 2.

	The number of parameters	Network size (%)	Time elapsed (Hour)	Time elapsed (%)
CLOUD	6,716,748	100%	9h 44m 7s	100%
EDGE	432,492	6.3%	-	-
EDGE-OD3	432,492	6.3%	11h 12m 39s	115%

B. EXPERIMENTAL RESULTS

1) CASE 1: RESULTS FOR THE SAME HYPER-PARAMETER SETTING BETWEEN THE CLOUD AND THE EDGE DEVICE

Figure 4 shows the result of the average rewards with respect to the number of timesteps in the policy training processes. *EDGE-OD3* significantly outperforms *EDGE* in terms of average rewards although they train the same architecture of the *smallnet*. The *smallnet* of the edge policy network is trained by using only 6.3% parameters over the *largenet* of the cloud policy network, as shown in Table 3. Nevertheless, the average rewards in the edge policy training reach very close to that in the cloud policy training while the edge device has only significantly limited hardware resources. The learning curve of *EDGE-OD3* converges at the 1M timestep point while that of *CLOUD* converges at the 3M timestep point. *EDGE-OD3* achieves about 66% improvement compared to *CLOUD* in terms of the number of timesteps for convergence.

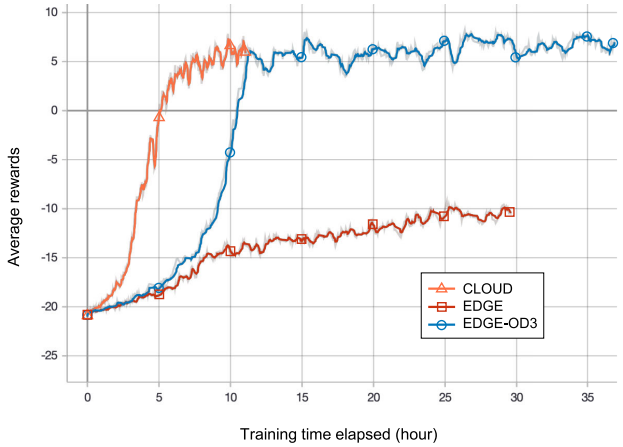

FIGURE 5. Average rewards with respect to the actual time elapsed for training.

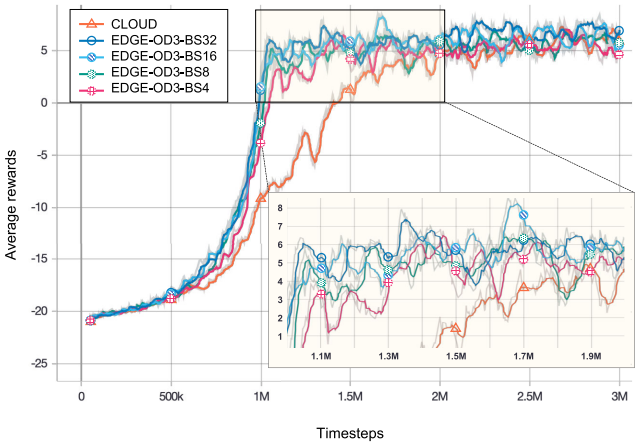

FIGURE 6. Average rewards with respect to the number timesteps with varying the mini-batch size of the edge device.

Since the actual time to execute a timestep can vary greatly depending on the hardware resource budgets, we rescale the result of the average rewards with the *x*-axis of the actual time elapsed for policy training, as shown in Fig. 5. The learning curve of *EDGE-OD3* with the *smallnet* successfully converges as quickly as that of *CLOUD* with the *largenet* although the edge device suffers from the significantly limited hardware resources. Table 3 also shows the actual time elapsed until the average rewards of *EDGE-OD3* and *CLOUD* reach the convergence point (6.2 reward points) for the first time. Using the proposed OD3, the edge policy training performs on par with the cloud policy training spending only 15% more time.

The overall performance shown in *Case 1* experiments demonstrates that OD3 performs on par with cloud policy training in terms of average rewards although the size of the policy network used in the edge device is much smaller compared to that used in the cloud. Moreover, the edge policy training with the OD3 not only decreases the training time elapsed significantly compared to edge policy training from scratch but also converges as quickly as the cloud policy training. We can see that OD3 allows an edge device to successfully perform knowledge transfer and policy model compression together under a given limited resource budget. Consequently, our results confirm the effectiveness of OD3.

2) CASE 2: RESULTS WITH VARYING HYPER-PARAMETER SETTINGS OF THE EDGE DEVICE

To study the effect of varying a hyper-parameter related to a memory constraint of the edge device, we vary the mini-batch size with a sequence of four values (i.e., 32, 16, 8, and 4) and refer these variants to as *EDGE-OD3-BS32*, *EDGE-OD3-BS16*, *EDGE-OD3-BS8*, and *EDGE-OD3-BS4*, respectively. Figure 6 shows the result of the average rewards with respect to the number of timesteps in the policy training processes in the cloud and the four *BS* variants of the edge device. The result indicates that the performance differences between four variants of the edge policy training are slight from the perspective of the performance of the average rewards with respect to the timestep. Varying the mini-batch size does not have a significant impact on performance in terms of the average rewards since the OD3 can transfer more information about action knowledge for a single update, compared to the policy training without distillation. However, a decrease in the mini-batch size significantly reduces the training time elapsed for edge policy training, as shown in Fig. 7. Thus the OD3 using a small batch allows an edge device to train its policy more quickly with negligible performance loss.

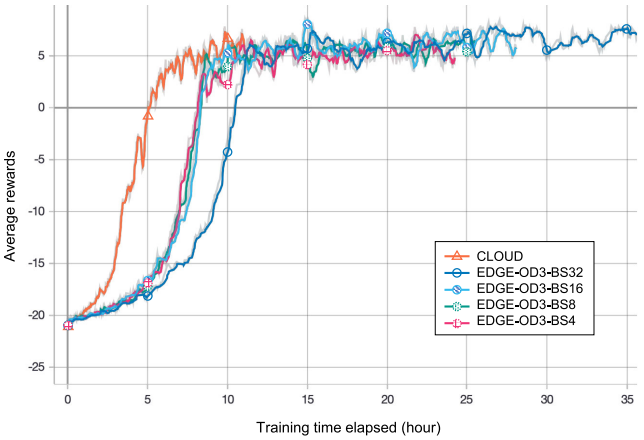

FIGURE 7. Average rewards with respect to the actual time elapsed for training with varying the mini-batch size of the edge device.

We also vary the update frequency for the OD3 with a sequence of four values (i.e., 4, 8, 16, and 32) and refer these variants to as *EDGE-OD3-LF4*, *EDGE-OD3-LF8*, *EDGE-OD3-LF16*, and *EDGE-OD3-LF32*, respectively. Figure 8 depicts the result of the average rewards with respect to the number of timesteps in the policy training processes in the cloud and the four *LF* variants of the edge device. The result shows that the performance of the average rewards decreases considerably as the update frequency of the edge device increases. We believe this is because the number of updates to the edge policy decreases and the sufficient distilled knowledge is not transferred to the edge device.

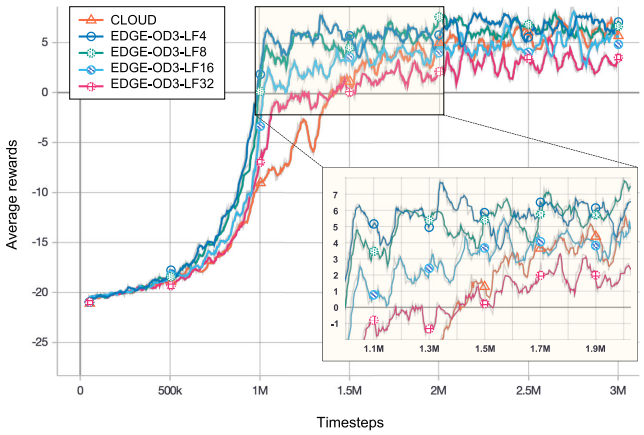

FIGURE 8. Average rewards with respect to the number timesteps with varying the update frequency of the edge device.

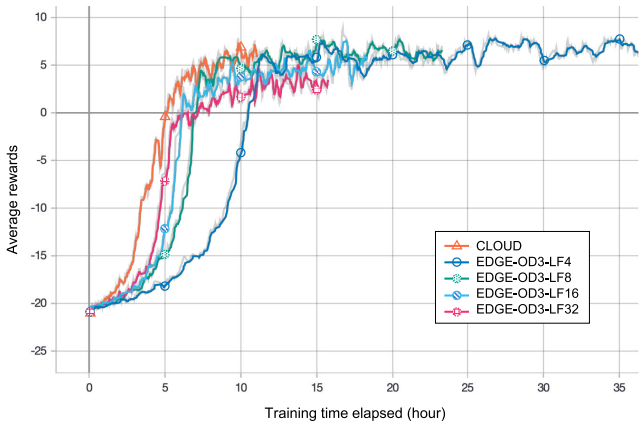

FIGURE 9. Average rewards with respect to the actual time elapsed for training with varying the update frequency of the edge device.

Due to the reduced number of the policy updates in the OD3, the performance of a lower update frequency decreases while the training is finished quickly, as shown in Fig. 9.

The overall performance shown in *Case 2* experiments indicates that it is important to choose an appropriate hyper-parameter setting to efficiently perform the proposed OD3. A comprehensive empirical study needs to be conducted thoroughly to further improve performance.

VI. CONCLUSION

In this paper, we proposed a method to efficiently transfer distilled knowledge for DRL based edge device control in resource-constrained edge computing systems, referred to as OD3. The goal of the OD3 is to conduct knowledge transfer and policy model compression simultaneously within a single training process on edge devices by leveraging a knowledge distillation technique. We analyzed the performance of the proposed method by implementing it on a commercial embedded system-on-module equipped with limited hardware resources. The experimental results showed that OD3 achieved near-cloud-performance although it utilizes only the significantly smaller policy network than the cloud. Also, the training time elapsed for edge policy training with our methods was reduced significantly compared to edge policy training from scratch. The results well confirmed the effectiveness of the OD3 in resource-constrained edge computing systems. In future work, we plan to investigate an effective method for generating optimal neural network architectures by considering the balance between the performance of the distilled policy and the resource constraints of the heterogeneous edge devices, as well as hyper-parameter optimization for DRL in resource-constrained edge environments.

REFERENCES

[1] A. Zanella, N. Bui, A. Castellani, L. Vangelista, and M. Zorzi, "Internet of Things for smart cities," *IEEE Internet Things J.*, vol. 1, no. 1, pp. 22–32, Feb. 2014.
[2] J. A. Guerrero-ibanez, S. Zeadally, and J. Contreras-Castillo, "Integration challenges of intelligent transportation systems with connected vehicle, cloud computing, and Internet of Things technologies," *IEEE Wireless Commun.*, vol. 22, no. 6, pp. 122–128, Dec. 2015.
[3] H. Ma and W. Liu, "A progressive search paradigm for the Internet of Things," *IEEE MultimediaMag.*, vol. 25, no. 1, pp. 76–86, Jan. 2018.
[4] Z. Bi, L. Da Xu, and C. Wang, "Internet of Things for enterprise systems of modern manufacturing," *IEEE Trans. Ind. Informat.*, vol. 10, no. 2, pp. 1537–1546, May 2014.
[5] J. Pan and J. McElhannon, "Future edge cloud and edge computing for Internet of Things applications," *IEEE Internet Things J.*, vol. 5, no. 1, pp. 439–449, Feb. 2018.
[6] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE Internet Things J.*, vol. 3, no. 5, pp. 637–646, Oct. 2016.
[7] N. Liu, Z. Li, J. Xu, Z. Xu, S. Lin, Q. Qiu, J. Tang, and Y. Wang, "A hierarchical framework of cloud resource allocation and power management using deep reinforcement learning," in *Proc. IEEE 37th Int. Conf. Distrib. Comput. Syst. (ICDCS)*, Jun. 2017, pp. 372–382.
[8] T. Inoue, G. De Magistris, A. Munawar, T. Yokoya, and R. Tachibana, "Deep reinforcement learning for high precision assembly tasks," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Sep. 2017, pp. 819–825.
[9] D. Isele, R. Rahimi, A. Cosgun, K. Subramanian, and K. Fujimura, "Navigating occluded intersections with autonomous vehicles using deep reinforcement learning," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2018, pp. 2034–2039.
[10] M. Mohammadi, A. Al-Fuqaha, S. Sorour, and M. Guizani, "Deep learning for IoT big data and streaming analytics: A survey," *IEEE Commun. Surveys Tuts.*, vol. 20, no. 4, pp. 2923–2960, 4th Quart., 2018.
[11] A. A. Rusu, S. G. Colmenarejo, C. Gulcehre, G. Desjardins, J. Kirkpatrick, R. Pascanu, V. Mnih, K. Kavukcuoglu, and R. Hadsell, "Policy distillation," in *Proc. ICLR*, May 2016, pp. 1–13.
[12] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, 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. 518, pp. 529–533, Feb. 2015.
[13] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, Jan. 2016.
[14] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," in *Proc. ICLR*, May 2016, pp. 1–14.
[15] C. H. Liu, Z. Chen, J. Tang, J. Xu, and C. Piao, "Energy-efficient UAV control for effective and fair communication coverage: A deep reinforcement learning approach," *IEEE J. Sel. Areas Commun.*, vol. 36, no. 9, pp. 2059–2070, Sep. 2018.
[16] C. Savaglio, P. Pace, G. Aloi, A. Liotta, and G. Fortino, "Lightweight reinforcement learning for energy efficient communications in wireless sensor networks," *IEEE Access*, vol. 7, pp. 29355–29364, 2019.
[17] X. Chen, H. Zhang, C. Wu, S. Mao, Y. Ji, and M. Bennis, "Optimized computation offloading performance in virtual edge computing systems via deep reinforcement learning," *IEEE Internet Things J.*, vol. 6, no. 3, pp. 4005–4018, Jun. 2019.

[18] M. Min, L. Xiao, Y. Chen, P. Cheng, D. Wu, and W. Zhuang, "Learning-based computation offloading for IoT devices with energy harvesting," *IEEE Trans. Veh. Technol.*, vol. 68, no. 2, pp. 1930–1941, Feb. 2019.
[19] X. Chen, Z. Zhao, C. Wu, M. Bennis, H. Liu, Y. Ji, and H. Zhang, "Multi-tenant cross-slice resource orchestration: A deep reinforcement learning approach," *IEEE J. Sel. Areas Commun.*, vol. 37, no. 10, pp. 2377–2392, Oct. 2019.
[20] H. Zhu, Y. Cao, W. Wang, T. Jiang, and S. Jin, "Deep reinforcement learning for mobile edge caching: Review, new features, and open issues," *IEEE Netw.*, vol. 32, no. 6, pp. 50–57, Nov. 2018.
[21] B. Fang, X. Zeng, and M. Zhang, "NestDNN: Resource-aware multi-tenant on-device deep learning for continuous mobile vision," in *Proc. 24th Annu. Int. Conf. Mobile Comput. Netw. MobiCom*, 2018, pp. 115–127.
[22] M. Xu, F. Qian, Q. Mei, K. Huang, and X. Liu, "DeepType: On-device deep learning for input personalization service with minimal privacy concern," *ACM IMWUT*, vol. 2, no. 4, pp. 1–26, 2018.
[23] R. Sharma, S. Biookaghazadeh, B. Li, and M. Zhao, "Are existing knowledge transfer techniques effective for deep learning with edge devices?" in *Proc. IEEE Int. Conf. Edge Comput. (EDGE)*, Jul. 2018, pp. 29–42.
[24] H. Li, K. Ota, and M. Dong, "Learning IoT in edge: Deep learning for the Internet of Things with edge computing," *IEEE Netw.*, vol. 32, no. 1, pp. 96–101, Jan. 2018.
[25] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," in *Proc. NIPS*, Dec. 2015, pp. 1135–1143.
[26] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding," in *Proc. ICLR*, May 2016, pp. 1–14.
[27] Y. Guo, A. Yao, and Y. Chen, "Dynamic network surgery for efficient dnns," in *Proc. NIPS*, Dec. 2016, pp. 1379–1387.
[28] J.-H. Luo, J. Wu, and W. Lin, "ThiNet: A filter level pruning method for deep neural network compression," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Oct. 2017, pp. 5058–5066.
[29] C. Buciluǎ, R. Caruana, and A. Niculescu-Mizil, "Model compression," in *Proc. ACM KDD*, Aug. 2006, pp. 535–541.
[30] J. Ba, and R. Caruana, R, "Do deep nets really need to be deep," in *Proc. NIPS*, Dec. 2014, pp. 2654–2662.
[31] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," 2015, *arXiv:1503.02531*. [Online]. Available: http://arxiv.org/abs/1503.02531
[32] Q. Lu, W. Jiang, X. Xu, Y. Shi, and J. Hu, "On Neural Architecture Search for Resource-Constrained Hardware Platforms," in *Proc. IEEE/ACM ICCAD*, Nov. 2019, pp. 1–8.
[33] C. Gong, Z. Jiang, D. Wang, Y. Lin, Q. Liu, and D. Z. Pan, "Mixed precision neural architecture search for energy efficient deep learning," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, Nov. 2019, pp. 1–7.
[34] Y. Xiong, R. Mehta, and V. Singh, "Resource constrained neural network architecture search: Will a submodularity assumption help?" in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, Oct. 2019, pp. 1–10.
[35] M. Abadi *et al.*, "TensorFlow: Large-scale machine learning on heterogeneous distributed systems," 2016, *arXiv:1603.04467*. [Online]. Available: http://arxiv.org/abs/1603.04467
[36] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *Proc. ICLR*, May 2015, pp. 1–15.
[37] D. Masters and C. Luschi, "Revisiting small batch training for deep neural networks," 2018, *arXiv:1804.07612*. [Online]. Available: http://arxiv.org/abs/1804.07612
[38] M. Roderick, J. MacGlashan, and S. Tellex, "Implementing the deep Q-network," 2017, *arXiv:1711.07478*. [Online]. Available: http://arxiv.org/abs/1711.07478

INGOOK JANG received the B.S. degree (summa cum laude) in computer science and engineering from Chung-Ang University, Seoul, South Korea, in 2008, and the Ph.D. degree in computer science from the Korea Advanced Institute of Science and Technology (KAIST), Daejeon, South Korea, in 2016. Since 2016, he has been with the Electronics and Telecommunications Research Institute (ETRI), Daejeon. His research interests include machine learning, deep learning, edge computing, and the intelligent IoT systems.

HYUNSEOK KIM received the B.A. degree in electronics engineering from Dong-A University, South Korea, and the M.S. and Ph.D. degrees from the Korea Advanced Institute of Science and Technology (KAIST). From 2001 to 2003, he worked with Samsung Electronics Company Ltd. From 2004 to 2009, he was with LG Electronics Inc. He is currently a Senior Researcher with the Electronics and Telecommunications Research Institute (ETRI), South Korea. His research interest includes developing innovative behavior intelligence in multi-agent systems.

DONGHUN LEE received the B.A. degree in computer science from the University of Illinois at Urbana-Champaign, USA, and the M.S. degree from Seoul National University, South Korea. From 2012 to 2013, he worked as a Software Engineer with Google Korea. He is currently a Researcher with the Electronics and Telecommunications Research Institute (ETRI), South Korea. His research interests include variational inference, model-based reinforcement learning, and meta learning.

YOUNG-SUNG SON is currently a Principal Researcher with the Electronics and Telecommunications Research Institute. He is also a Professor with the University of Science and Technology. His research interests include context awareness computing, the intelligent IoT systems, and autonomous intelligent systems.

SEONGHYUN KIM received the B.S. and Ph.D. degrees in electrical engineering from Yonsei University, Seoul, South Korea, in 2009 and 2016, respectively. Since 2016, he has been with the Electronics and Telecommunications Research Institute (ETRI), Daejeon, South Korea. His research interests include machine learning, deep learning, cloud computing, and cellular systems.