A Deep Reinforcement Learning approach for hierarchical edge devices collaboration

Mohamed Amine Ghamri[1,*,†], Badis Djamaa[1,†], Mohamed Akrem Benatia[1,†] and Issam Eddine Lakhlef[1,†]

[1]*Ecole Militaire Polytechnique, Bordj El Bahri, Algiers, Algeria*

Abstract

Artificial intelligence has become an integral part of modern life, with pervasive AI leveraging interconnected devices to deliver intelligent services in real-time environments with the most challenging aspects emerging in the domain of mobile edge computing (MEC). Deploying deep neural networks (DNNs) on such devices and orchestrating the necessary communication require dynamic coordination, an area where reinforcement learning (RL) has shown significant promise. In this paper, we propose a Double Deep Q-Network (DDQN) approach for DNN inference offloading within a mobile edge computing system, efficiently addressing the challenges of dynamic orchestration while optimizing hierarchical collaboration between edge devices. Our experimental results demonstrate the strength of this approach, achieving superior performance in optimizing inference offloading compared to existing solutions.

Keywords

Reinforcement Learning, Deep Neural Network, task offloading, DDQN

1. Introduction

Artificial intelligence (AI) has found applications in numerous domains, offering transformative solutions across industries. In particular, deep neural networks (DNNs) have proven effective in tackling complex challenges, driving innovation in areas such as healthcare [1], autonomous systems [2], and smart cities [3]. To fully harness the power of these models, we aim to make AI pervasive—deploying these sophisticated models across interconnected devices to provide intelligence anytime and anywhere [4]. This vision of pervasive AI involves deploying DNNs at different levels of the Internet of Things (IoT) stack [5], enabling distributed intelligence through the use of connected objects.

One key challenge in deploying DNNs in IoT environments is the tradeoff between model accuracy and the computing requirements associated with the limited hardware capabilities of many IoT nodes [6]. An effective strategy to address this challenge is the use of split computing [7], where DNNs are partitioned across different layers of the IoT stack to distribute the computational load efficiently [8]. This approach involves dynamic coordination of data exchanges between different nodes, which can be optimally managed through reinforcement learning (RL). Recognizing the potential of RL, in this paper, we propose a Double Deep Q-Network (DDQN) based approach to make dynamic DNN inference offloading decisions, ensuring efficient operation across resource-constrained environments. Our results demonstrate the effectiveness of this method, showing promising improvements over existing approaches.

The remainder of this paper is structured as follows: Section 2 provides an overview of related work, Section 3 details our proposed approach, and Section 4 presents the results of our experiments. Finally, we conclude by summarizing our contributions and offering directions for future research.

TACC'2024The 4th Tunisian-Algerian Conference on applied Computing, December 17-18, 2024, Constantine, Algeria

*Corresponding author.

†These authors contributed equally.

mohammedamine.ghamri@emp.mdn.dz (M. A. GHAMRI); badis.djamaa@emp.mdn.dz (B. DJAMAA); akrem.benatia@emp.mdn.dz (M. A. BENATIA); issameddine.lakhlef@emp.mdn.dz (I. E. LAKHLEF)

0009-0002-1439-033X (M. A. GHAMRI); 0000-0003-2323-9316 (B. DJAMAA); 0000-0003-1779-2705 (M. A. BENATIA); 0009-0001-3845-891X (I. E. LAKHLEF)

2. Related work

Reinforcement learning (RL) has proven to be an effective approach for handling sequential decision-making problems, especially in dynamic environments. This makes it well-suited for managing system dynamics in DNN inference offloading. In [9], the authors proposed MECI, an RL-based approach built on Q-learning, where each device maintains its own Q-table to dynamically decide on offloading decisions, such as selecting one of several possible cut layers and choosing the target server. While this approach effectively handled the problem, its main limitation lies in the inability of Q-learning to cope with complex state spaces, especially as the number of devices grows, which is a typical scenario as stated in [10]. To address this limitation, the authors of [9] later introduced DMECI, a deep reinforcement learning (DRL) approach based on an actor-critic framework. In this method, the Q-value estimation is replaced by deep neural network (DNN) function approximators, leveraging both target and policy networks. Similarly, the work presented in [11] addressed DNN inference offloading with a simplified setup involving only a single cut layer. They used an improved version of Deep Q-Networks (DQN), which employs two neural networks for value estimation and an additional replay memory to accelerate convergence. Their goal was to carry out classification tasks until a target confidence level was reached.

In [12], the authors addressed DNN inference distribution in an IoT environment with a focus on security and hardware limitations. Their innovative approach involved distributing portions of intermediate feature maps across different devices for a collaborative inference process, which they optimized using the DQN algorithm, showing promising results. Additionally, in [13], the authors framed the inference distribution as a Markov decision process (MDP) in a mobile edge computing (MEC) environment for signal classification, targeting distributed inference with high accuracy. They adopted an actor-critic framework with the Deep Deterministic Policy Gradient (DDPG) algorithm, which produced favorable outcomes. In the context of vehicular edge computing, the work in [14] tackled task offloading by employing a deep RL approach using an actor-critic structure with the DDPG algorithm. Moreover, in another MEC environment, [15] framed the task offloading problem as a multi-objective optimization targeting quality of service (QoS) requirements using the DQN algorithm.

Table 1
Task offloading works using RL

Category	Work	Env	Algorithm
Policy-based	[9]	MEC	A2C
	[14]	VEC	DDPG
	[13]	MEC	DDPG
Policy-free	[9]	MEC	Q-Learning
	[12]	IoT	DQN
	[11]	MEC	DQN
	[15]	MEC	DQN

As summarized in Table 1, existing studies commonly address the distributed inference offloading problem by leveraging reinforcement learning (RL) approaches and their deep variants, which have proven effective in handling complex state spaces. These RL methods involve formulating the problem as a Markov Decision Process (MDP), requiring a well-designed state space, action space, and a reward function aligned with the desired objectives.

These studies can be categorized into policy-based and value-based approaches. Policy-based methods, such as those employing the actor-critic framework [14, 13, 9], use two neural networks: the critic network for estimating value functions and the actor network for estimating the policy. Value-based approaches, on the other hand, focus on estimating the quality or value function for a given state and action, and then follow a greedy policy to derive the optimal course of action. This approach resembles the trial-and-error process of an agent learning to navigate a stochastic environment. One notable algorithm in the value-based category is the Deep Q-Network (DQN), which utilizes two sets of neural networks. Improvements to DQN, such as the addition of a replay memory in [11], have significantly

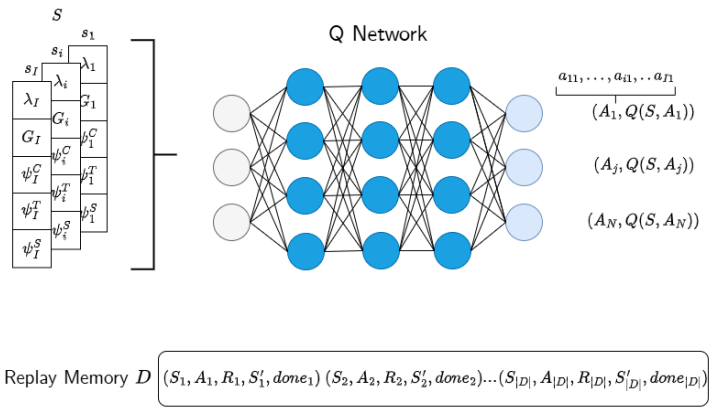

Figure 1: Q-Network illustration

enhanced its performance. A further advancement is the Double Deep Q-Network (DDQN), introduced in [16], which has demonstrated considerable success in both edge and cloud environments [17, 18]. Given its effectiveness, DDQN forms the foundation of our work. In our study, we tackle the problem of DNN inference offloading using the DDQN algorithm that will be detailed in next section.

3. DDQN approach

3.1. DDQN algorithm

The Double DQN algorithm [16] aims to mitigate the overestimation problem found in traditional Q-learning. In standard Q-learning and DQN, the same parameters (the online Q network) are used for both action selection and evaluation, which can result in inflated Q-value estimates, especially for actions consistently given higher values. Double DQN addresses the overestimation issue in traditional Q-learning by introducing "target networks" Instead of using a single set of parameters for both action selection and evaluation, Double DQN uses two separate networks: the online network for selecting actions and the target network for evaluating Q-values. The target network, updated periodically with the online network's parameters, provides more stable and less optimistic estimates, reducing overestimation during evaluation.

During training, the parameters of the online network are updated by minimizing the loss function, commonly using the Mean Squared Error (MSE) between the Q-value estimated by the online network and the target value generated by the target network. This process begins by randomly sampling state transitions from a replay memory buffer, which holds a collection of past actions and states. Unlike traditional Q-learning that relies on matrix-based operations, these updates are processed by neural networks guided by the Bellman equation.

Figure 1 illustrates the Q-Network and its connection to the replay memory buffer, which stores tuples representing individual experiences. Each tuple contains the current state, the chosen action, the resulting new state after action execution, the observed reward, and an indication of episode termination. These stored experiences are sampled to compute the Bellman equation term using the target network's inference. The online network is then optimized through gradient descent to approach this target Bellman value. As shown in Figure 1, both the online and target networks share identical architectures. Each network is composed of multilayer perceptrons (MLPs) that receive a flattened representation of the state space (as detailed in Section 3.3) and output value estimates for each possible action.

By decoupling the action selection and evaluation processes, Double DQN reduces the likelihood of overestimating Q-values, leading to more stable and accurate learning. This approach has been shown to improve the performance and training stability of deep reinforcement learning algorithms, particularly in environments with large state spaces or complex dynamics.

3.2. System model

We operate in a mobile edge computing (MEC) system comprising multiple edge devices of number I, each equipped with a deployed DNN model. All devices are connected to an edge server via wireless communication, which also hosts a DNN model to handle any remaining processing tasks. The DNN models are partitioned into L_i segments, indexed by j for each device i, with each segment generating a number of floating-point operations denoted as F_j^i. The partitioning is achieved using cut layers, which in our current work are selected as pooling layers due to their smaller output sizes, facilitating more efficient wireless transmission. When a device i performs inference up to a selected cut layer $l_i(t)$, the computing delay is directly proportional to the number of layers processed and the device's hardware characteristics, such as CPU frequency f^i. This delay is determined by the following formula:

$$d_c^i(t) = \sum_{j=0}^{L_i-1} \frac{u(l_i(t) - j) \times \phi \times F_j^i}{f^i} \tag{1}$$

ϕ denotes the device's computing performance (number of cycles per flop), and u is defined as:

$$u(x) = \begin{cases} 0, & \text{if } x < 0 \\ 1, & \text{if } x >= 0 \end{cases}$$

The remaining computation is carried out on the server side, where the CPU frequency is f^s, as determined by the following formula:

$$d_c^s(t) = \frac{\sum_{j=0}^{L_i-1} u(j - l_i(t)) \times \phi \times F_j^s}{f^s} \tag{2}$$

We adopt a linear model to estimate the energy consumption during computation, as outlined in [19]. Given the energy efficiency coefficient $\rho_c^i$the energy consumption is defined as:

$$e_c^i(t) = d_c^i(t) \times (f^i)^3 \times \rho_c^i \tag{3}$$

For the communication delay, we use Shannon's formula, which relates the transmission delay to the output data size from the partial inference n_j^i, the transmission power of the device ρ^i, and the network bandwidth W, resulting in the following formula:

$$d_T^i(t) = \frac{n_j^i}{\frac{W}{I} Log(1 + \frac{\rho_T^i g_i(t)}{\sigma^2})} \tag{4}$$

Here, σ^2 represents the spectral noise, while $g_i(t)$ denotes the time-varying channel gain between the device and the server, capturing the fluctuating network conditions over time. It is important to note that the available bandwidth is equally divided among all devices, which is not the most efficient allocation strategy, as the computing demands of individual devices usually vary. Nevertheless, we adopt this approach since our current focus is on the reinforcement learning method rather than optimizing resource allocation. We will also note that n_0^i is equal to the initial input size multiplied by the compression rate $c_i(t)$. And the transmission energy consumption formula is given by :

$$e_t^i(t) = d_T^i(t)\rho_T^i \tag{5}$$

Given λ_i arrived input data, and a corresponding selected cut layer $l_i(t)$, the total delay is equal to:

$$D_i(t) = \lambda_i(t) \sum_{j=0}^{L_i-1} \left(\frac{\phi u\left(l_i(t) - j\right) F_j^i}{f^i} + \frac{\phi u\left(j - l_i(t)\right) F_j^s}{\frac{f^s}{I}} + \frac{n_j^i}{\frac{W}{I} \log(1 + \frac{\rho_T^i g_i(t)}{\sigma^2})} \right) \tag{6}$$

3.3. Problem formulation

By presenting our system model, we arrive at the formulation of the optimization problem, which involves:

$$\text{Minimize} \quad \frac{1}{E}\sum_{t=0}^{E}\sum_{i=1}^{I} D_i(t)$$

This is achieved by determining the decision variables $l_i(t)$ and $c_i(t)$, while accounting for the stochastic variations in network conditions $gi(t)$ and the data arrival rate $\lambda_i(t)$. Consequently, solving this problem is NP-hard, necessitating the use of a reinforcement learning approach that sequentially optimizes the decision variables through a learned policy. However, prior to this, we need to model our problem as a Markov Decision Process (MDP), which involves defining the state, action, and reward structures.

- State : We need to monitor the data arrival rate $\lambda_i(t)$, the network conditions through $g_i(t)$, and the current workloads of both the device and server. These will be represented as the available computing and transmission queues sizes $\Psi_i^C(t)$ and $\Psi_i^T(t)$ respectively, and also the computing queue size for the server $\psi^S(t)$. The partial observations from each device will be consolidated into a global observation. This aggregation will consequently define our state space as follows:

$$S(t) = (\lambda(t), G(t), \psi^C(t), \psi^T(t), \psi^S(t)) \tag{7}$$

- Action : Each device must determine both the cut layer at which to perform data inference and the compression rate to apply. Consequently, the action space will encompass the collective actions of all devices, resulting in:

$$A(t) = (C(t), L(t)) \tag{8}$$

  with $C(t) = (c_1(t), ...c_i(t), ...c_I(t))$ and $L(t) = (l_1(t), ..., l_i(t), ..., l_I(t))$
- Reward : Since our objective is to optimize the average service delay, we choose to maximize its dual aspect, known as throughput. In this context, we reward the agent for each data completion within the system, specifically the total number of completed tasks $\mathcal{T}^i(t)$ across all devices:

$$R(t) = \sum_{i=1}^{I} \mathcal{T}^i(t) \tag{9}$$

3.4. Algorithm description

In our work, we consider a single agent operating on the server side, which manages two Q networks (the online and target networks) and maintains a memory buffer D for storing experiences. This agent is responsible for observing the system state by gathering information from the devices and the environment, formatting this data in accordance with Section 3.3, and subsequently determining the actions for each device using an epsilon-greedy policy. This approach involves selecting random actions with a probability of epsilon to encourage exploration, while also enabling exploitation by utilizing the online Q network to estimate the Q values for each action combination, ultimately selecting the action with the highest Q value [20]. Subsequently, the corresponding actions are dispatched to each device, prompting them to execute these tasks. This includes compressing the input data at the specified rate and carrying out DNN inference up to the designated cut layer. Afterward, the devices send the intermediate inference results to the server, which completes the remaining computations and returns the final results to the respective devices. Once this cycle is completed, each device receives a reward based on the formulation outlined in Section 3.3, and the results are transmitted to the server for aggregation. The entire environment then transitions to a new state s'. This experience is recorded in the replay buffer for future use in updating the Q networks. The online Q network is updated once a

sufficient number of experiences have been gathered, specifically a batch size of stored experiences. This update utilizes the standard Bellman equation, which establishes the relationship between the Q value of the current state s and that of the subsequent state s'. The key distinction in the DDQN approach lies in applying this equation using the target network, leading to the following formulation:

$$Q_{\text{online}}(s, a|\theta) \sim (1-\alpha) \cdot Q_{\text{online}}(s, a|\theta) + \alpha \cdot \left[R(s, a) + \gamma \cdot \max_{a'} Q_{\text{target}}(s', a'|\theta) \right] \tag{10}$$

Backpropagation is then performed to minimize the mean squared error between the calculated value and the stored value. After a predetermined number of iterations, determined by a constant C, the parameters of the target network are updated by copying those of the online Q network. To facilitate the agent's transition from exploration to increased exploitation, the epsilon value will be gradually decreased in a linear manner until it reaches a predetermined minimum. This iterative process will persist for a defined number of training episodes. The algorithm is outlined in Algorithm 1.

Algorithm 1 Double DQN (RL module at the server side)

```
Initialize Q-networks Q_online and Q_target for the server with random weights
Initialize replay memory D
Set target network update frequency C
Set discount factor γ
Set exploration parameters, ϵ, ϵ_min, ϵ_decay
for each episode e do
    Observe the initial state from state tracking module
    for each time step t in e do
        Select action using ϵ-greedy policy based on Q_online
        Distribute the actions to the devices
        Execute Inference Tasks
        Receive reward and observe the next state s'
        Store (s, a, r, s', done) in replay memory D
        if length of D ≥ replay batch size then
            Sample a random batch from replay memory D
            for each sample do
                Calculate target Q-values using Q_target and Bellman equation
                Update Q_online using backpropagation
            end for
        end if
        if t is a multiple of C then
            Update Q_target weights with Q_online weights
        end if
        ϵ^i ← epsilon-decay(ϵ^i_decay, ϵ^i_min)
    end for
end for
```

4. Results

4.1. Environment

We developed a simulated environment consisting of a scenario with three edge devices, characterized by the parameters detailed in Table 2. In this setup, the VGG16 classification model is deployed on either the devices or the server. We designated three distinct cut layers—specifically, layers 3, 10, and 18—corresponding to distributed pooling layers, and simulated data collection by randomly sampling from a subset of the ImageNet dataset [21], which consists of 500 images. For the implementation of the

classification model, we utilized TensorFlow [22], which also facilitated the collection of intermediate output sizes and the number of floating-point operations (FLOPs) through the TensorFlow Profiler. The agent was designed to comply with the Gymnasium interface [23], while the DDQN algorithm was implemented using the OpenAI Stable-Baselines3 framework [24].

Table 2
System parameters settings

Parameter Name	Value
Bandwidth (W)	45 MHz
Spectral Noise (σ^2)	-104 dbm
Device transmit power (ρ_T^i)	20 dbm
Device CPU frequency (f^i)	0.45 GHz
Device energy efficiency (ρ_c^i)	10^{-28}
Server CPU frequency (f^s)	20 GHz
CPU cycles per flop (ϕ)	4
Number of bits per float	32
Timestep duration (τ)	900 mS

For the essential hyperparameters required to train the DDQN model, we adhered to the values specified in Table 3.

Table 3
DDQN Learning Hyperpramaters

Parameter Name	Value
Episode Length (E)	16
Learning rate (α)	0.001
Target Network update frequency (C)	160
Batch size	48
Discount factor (γ)	0.9
Initial Exploration probability (ϵ)	1.0
Exploration fraction (ϵ)	0.1
Final Exploration probability (ϵ)	0.01

4.2. Convergence

Figure 2a represents the evolution of rewards over successive episodes. Despite some noise, the trend shows a clear upward trajectory as the episodes progress, indicating that the reinforcement learning agent is successfully improving its policy, resulting in consistently higher rewards. While Figure 2b illustrates the evolution of the loss function, showing a decreasing trend. This indicates that the neural networks are successfully training and adapting well to the accumulated experiences. Through these experiments, we demonstrated the convergence and learning progress of the DDQN algorithm.

4.3. Comparison

In this section, we aim to evaluate the performance of our reinforcement learning approach compared to static configurations, as well as to benchmark it against existing state-of-the-art RL solutions. We use several performance metrics, including the evolution of the episodic cumulative reward, the latency—represented by the average service delivery delay derived from the throughput (i.e., the number of completed inference tasks per episode)—and energy consumption resulting from computation and transmission. The visible plots represent values averaged over 10 episodes. To achieve the first objective, we compare our approach with two benchmarks: "edge," where the inference is performed entirely on the devices, and "central," where the inference is fully offloaded to the edge server. For the second objective, we benchmark against the actor-critic approach (A2C), commonly used in state-of-the-art works [9, 14, 13], as well as the widely adopted DQN [11, 12]. The results are presented in Figure 3.

(a) Episode rewards evolution

(b) Q Network training progress

Figure 2: DDQN Training

One notable observation is the marginal superiority of the central approach over the edge solution in terms of system throughput. However, this advantage is contingent upon favorable network conditions, where server performance in computing becomes the guiding factor. As anticipated, the edge solution exhibits stable behavior across varying network conditions. Furthermore, the reinforcement learning solutions (DDQN, DQN and A2C) demonstrate superior adaptability after sufficient training episodes, outperforming the former approaches (edge and central).

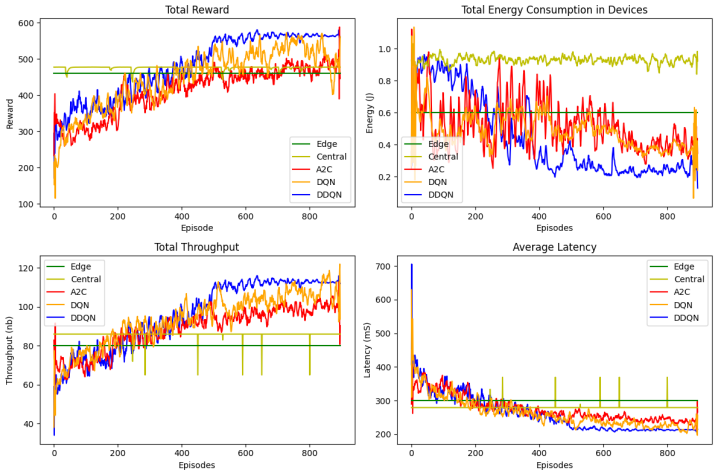

Figure 3: Performance metrics comparison

For the comparison of reinforcement learning approaches, the results demonstrate that the policy-free methods (DQN and DDQN) both outperform the A2C approach in terms of learning behavior, showing greater stability and higher reward values. Additionally, these approaches achieve better performance metrics, such as increased throughput, which leads to reduced latency. Moreover, the DDQN approach exhibits more stability compared to DQN, as observed in the results. This highlights the superiority of the DDQN algorithm over other deep reinforcement learning methods. This experiment serves as empirical validation of the findings in [16] within the context of DNN inference offloading, showing that DDQN improves Q-value estimation stability by reducing overestimation bias.

5. Conclusion

In this work, we tackled the challenging problem of DNN inference offloading using a reinforcement learning approach, specifically leveraging the DDQN algorithm due to its robust capabilities. Our results demonstrated its competitive performance against state-of-the-art solutions. Future research could

focus on refining the MDP design by incorporating additional state and action elements, enhancing the system's adaptability. Additionally, shaping the reward function to encompass a broader range of objectives, such as more efficient resource management or explicit energy optimization, offers promising directions for further improving the system's performance.

Declaration on Generative AI and AI-assisted Technologies

During the preparation of this work, the authors used OpenAI's ChatGPT (GPT-4-turbo, February 2024 version) in order to: Grammar and spelling check, as well as language refinement. After using this service, the authors reviewed and edited the content as needed and take full responsibility for the publication's content.

References

[1] G. S. Nadella, S. Satish, K. Meduri, S. S. Meduri, A systematic literature review of advancements, challenges and future directions of ai and ml in healthcare, International Journal of Machine Learning for Sustainable Development 5 (2023) 115–130.

[2] A. Kondam, A. Yella, Artificial intelligence and the future of autonomous systems, Innovative Computer Sciences Journal 9 (2023).

[3] H. Herath, M. Mittal, Adoption of artificial intelligence in smart cities: A comprehensive review, International Journal of Information Management Data Insights 2 (2022) 100076.

[4] E. Baccour, N. Mhaisen, A. A. Abdellatif, A. Erbad, A. Mohamed, M. Hamdi, M. Guizani, Pervasive ai for iot applications: A survey on resource-efficient distributed artificial intelligence, IEEE Communications Surveys & Tutorials 24 (2022) 2366–2418.

[5] J. Lin, W. Yu, N. Zhang, X. Yang, H. Zhang, W. Zhao, A survey on internet of things: Architecture, enabling technologies, security and privacy, and applications, IEEE internet of things journal 4 (2017) 1125–1142.

[6] C. Surianarayanan, J. J. Lawrence, P. R. Chelliah, E. Prakash, C. Hewage, A survey on optimization techniques for edge artificial intelligence (ai), Sensors 23 (2023) 1279.

[7] S.-Y. Kim, H. Ko, Distributed split computing system in cooperative internet of things (iot), IEEE Access (2023).

[8] Y. Matsubara, M. Levorato, F. Restuccia, Split computing and early exiting for deep learning applications: Survey and research challenges, ACM Computing Surveys 55 (2022) 1–30.

[9] Y. Xiao, L. Xiao, K. Wan, H. Yang, Y. Zhang, Y. Wu, Y. Zhang, Reinforcement learning based energy-efficient collaborative inference for mobile edge computing, IEEE Transactions on Communications 71 (2022) 864–876.

[10] S. R. Department, Iot: Number of connected devices worldwide 2012–2025 (2020).

[11] K. Qu, W. Zhuang, W. Wu, M. Li, X. Shen, X. Li, W. Shi, Stochastic cumulative dnn inference with rl-aided adaptive iot device-edge collaboration, IEEE Internet of Things Journal (2023).

[12] E. Baccour, A. Erbad, A. Mohamed, M. Hamdi, M. Guizani, Rl-distprivacy: Privacy-aware distributed deep inference for low latency iot systems, IEEE Transactions on Network Science and Engineering 9 (2022) 2066–2083.

[13] W. Wu, P. Yang, W. Zhang, C. Zhou, X. Shen, Accuracy-guaranteed collaborative dnn inference in industrial iot via deep reinforcement learning, IEEE Transactions on Industrial Informatics 17 (2020) 4988–4998.

[14] W. Shi, L. Chen, X. Zhu, Task offloading decision-making algorithm for vehicular edge computing: A deep-reinforcement-learning-based approach, Sensors 23 (2023) 7595.

[15] I. Rahmati, H. Shah-Mansouri, A. Movaghar, Qoco: A qoe-oriented computation offloading algorithm based on deep reinforcement learning for mobile edge computing, arXiv preprint arXiv:2311.02525 (2023).

[16] H. Van Hasselt, A. Guez, D. Silver, Deep reinforcement learning with double q-learning, in: Proceedings of the AAAI conference on artificial intelligence, volume 30, 2016.

[17] Q. Zhang, M. Lin, L. T. Yang, Z. Chen, S. U. Khan, P. Li, A double deep q-learning model for energy-efficient edge scheduling, IEEE Transactions on Services Computing 12 (2019) 739–749. doi:10.1109/TSC.2018.2867482.

[18] A. Iqbal, M.-L. Tham, Y. C. Chang, Double deep q-network-based energy-efficient resource allocation in cloud radio access network, IEEE Access 9 (2021) 20440–20449. doi:10.1109/ACCESS.2021.3054909.

[19] A. Beloglazov, R. Buyya, Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in cloud data centers, Concurrency and Computation: Practice and Experience 24 (2012). URL: https://api.semanticscholar.org/CorpusID:10061036.

[20] O. Berger-Tal, J. Nathan, E. Meron, D. Saltz, The exploration-exploitation dilemma: a multidisciplinary framework, PloS one 9 (2014) e95693.

[21] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, et al., Imagenet large scale visual recognition challenge, International journal of computer vision 115 (2015) 211–252.

[22] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, X. Zheng, TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL: https://www.tensorflow.org/, software available from tensorflow.org.

[23] M. Towers, A. Kwiatkowski, J. Terry, J. U. Balis, G. De Cola, T. Deleu, M. Goulão, A. Kallinteris, M. Krimmel, A. KG, et al., Gymnasium: A standard interface for reinforcement learning environments, arXiv preprint arXiv:2407.17032 (2024).

[24] A. Raffin, A. Hill, M. Ernestus, A. Gleave, A. Kanervisto, M. Chevalier-Boisvert, J. Kubricht, A. Nichol, B. Hill, J. Pachocki, et al., Stable baselines3, https://github.com/DLR-RM/stable-baselines3, 2020.