Adaptive QoS-Aware Task Offloading in Dynamic Mobile Edge Computing Environment

Jacob Don, Sajib Mistry, Redowan Mahmud(✉), and Aneesh Krishna

School of Electrical Engineering, Computing and Mathematical Sciences, Curtin University, Perth, Australia
mdredowan.mahmud@curtin.edu.au

Abstract. Ensuring Quality of Service (QoS) for real-time applications like Augmented Reality in a Mobile Edge Computing (MEC) setting is both vital and demanding in research. In this work, we propose a novel framework of hybrid ML approaches to enable QoS-aware offloading in dynamic MEC. First, we create a method using deep reinforcement learning to figure out the best way to offload tasks for an application in a new MEC environment, even when we don't have early information about the application's QoS. Then, we deploy a transfer learning approach in the dynamic MEC that transfers knowledge from previously trained deep reinforcement off-loading policies to new optimal policies. Experimental results show that the proposed framework adapts to the dynamic MEC environments efficiently, reducing ML training time and retaining higher accuracy and precision during the task offloading process.

Keywords: Quality of Service · Mobile Edge Computing · Task offloading · Hybrid machine learning

1 Introduction

The advent of the *Internet of Things* has seen a rise in many real-time application services where the Quality of Service (QoS) is vital; one such service is *Augmented Reality* (AR). AR is an interface between a user and the natural world that artificially augments the user's experience by adding computer-generated content [3]. The content is provided in real-time and is dependent on the context of the user's environment. Some of the most well-known uses of AR today are on modern smartphones, with the AR game Pokemon Go receiving millions of downloads and the social media app Snapchat providing face-altering AR [12].

Despite AR being a functional area of computing, there have been few real-world implementations. This is due to AR requiring devices to be small and portable but also a large amount of RAM, power and fast CPUs to run efficiently [11]. While modern-day smartphones have powerful cameras and are highly portable, the intensive requirements of AR mean that mobile implementations are slow, poor quality, and power-draining. This is just one of many applications unable to be fully functional due to the divide between their computational complexity and the capabilities of modern devices.

A. Zaslavsky et al. (Eds.): MobiQuitous 2023, LNICST 594, pp. 341–352, 2024.
https://doi.org/10.1007/978-3-031-63992-0_22

A potential solution to this issue is utilising *cloud computing* to solve the problem of user devices not having enough computational resources for the AR tasks [9]. The use of cloud computing caused a new bottleneck, the network connecting the cloud to the user. These networks cannot keep up with the demands of the computations resulting in significant lag [15]. This bottleneck means that real-time applications such as AR that cannot run natively on mobile devices experience substantial latency problems when offloading to the cloud [16]. Hence, the use of Mobile Edge Computing (MEC) is becoming prevalent, which focuses on moving the computation from central cloud servers to mobile edge computing nodes at the user premises [16]. Task offloading with MEC and 5G allows devices to compute tasks with improved response times and QoS.

Utilising MEC improves latency by reducing the distance the data travels, and combined with 5G technology, results in even lower latency. The use of MEC brings up the issue of how to best distribute the tasks among the Mobile Edge Servers. This problem of optimal task distribution is known to be NP-Hard [10]. The NP-Hard nature makes using heuristic algorithms to solve this problem difficult. Machine Learning based solutions are suitable candidates, e.g. Artificial Neural Networks (ANN), to address the NP-Hard problem of trask offloading [2]. Due to ANN's fixed input and output sizes, it is challenging to fit ANNs in the dynamic environment as a new model must be trained for every new environment and task offloading.

Deep Q Networks (DQNs) are a type of *reinforcement* learning that utilizes ANNs to learn optimal actions to take in a dynamic environment. DQNs and algorithms derived from them have shown strong performances in the area of dynamic task distribution [2,7,18]. *We have identified the following limitations of DQN-based solutions in the dynamic MEC task offloading problem*:

- **Fluid Adaptivity**: Solutions require retraining on every possible environment change due to the neural network used to estimate values which is not optimal in real-world applications.
- **Faster Convergence**: Training DQNs has a slow convergence, meaning it takes a considerable amount of time and computing capability to develop a DQN that provides strong placements of tasks in the runtime.

Without addressing the issues of adaptivity, the performance of a network can be greatly reduced, resulting in lag and poor QoS. *To address this, we introduce a hybrid DQN-transfer learning approach in which learned domain knowledge is kept when the environment changes.* The main **contributions** of the paper are:

- A hybrid DQN-transfer learning framework for dynamic MEC environment.
- The formulation of MEC task offloading environment as a Markov Decision Problem through which reinforcement learning algorithms can be trained.
- The performance evaluation of the proposed framework with synthetic data, comparing it to the ideal results using the baseline brute force algorithm.

The rest of the paper is organised as follows. Section 2 highlights the literature on task offloading in MEC. Section 3 discusses the proposed adaptive QoS-aware task offloading policy. Section 4 outlines the experimental setups to evaluate the performance of our proposed solution. Finally, Sect. 5 concludes the paper.

2 Related Work

Deep Q learning is a type of reinforcement learning that has seen significant research in the area of task offloading for MEC architectures due to its ability to learn to navigate a given environment to maximize an arbitrary reward function [14]. Unlike its predecessor, tabular Q learning, DQN is able to learn environments with arbitrarily large state spaces. X. Chen, et al., [2] uses deep reinforcement learning to solve energy-efficient task distribution. They use a MADDPG deep learning algorithm as the basis of their artificial intelligence. Reinforcement learning means the model can be taught without knowing the ideal outcome beforehand. Combined with deep learning means, the model can learn very complicated patterns that even a human would miss. Promising experimental results, power consumption was down 10% in the multi-MEC system, with the delay staying well below the desired threshold.

K. Zhang, et al., [18] and M. Hossain, et al., [7] use Deep Q learning to produce a model that can offload tasks in an efficient manner. The model is one of the most simple forms of deep Q learning, meaning that the implementation is easy. Deep Q learning has issues with these stochastic problems as the state space can rapidly expand to larger than the deep Q learning algorithm can handle [1,4,13] meaning that this algorithm will not work with multiple users.

Transfer learning is a method of translating knowledge that a machine learning algorithm has in one task space into another closely related space [19]. There are many methods for each machine learning algorithm. The most common use for transfer learning is reducing the required training time. This is a useful area because training machine learning algorithms can often take multiple hours to multiple days. Therefore, any reduction in that time is ideal, especially when fine-tuning models, which can require them to be trained multiple times. Transfer learning methods are effective in deep reinforcement learning [19] and more generally in deep learning [17].

Z. Zhu et al., [19], provided a survey of the current research into transfer learning in the area of deep reinforcement learning. Providing reviews and descriptions of current methods such as Reward Shaping, Learning from Demonstrations and Representation Transfer. Our method focuses on the use of the Representation Transfer method. Representation transfer is an area of methods that employs the ability of DNNs to abstract information into representations and then transfer those learned representations.

JT. Huang et al., [8] was analyzed in C. Tan et al., [17] and shows how a DNN can be split into two parts, the first a reusable set of hidden layers that extracts features from a language, and the second portion that is retrained for each possible language and acts as a language classifier. The transferable first layer is a strong example of transfer learning.

One of the main issues in the existing research is that they have limited focus on fluid adaptability to a frequently changing MEC environment. To address this issue we propose a hybrid DQN-transfer learning approach for the task offloading incorporating fluild adaptivity and faster convergence in the runtime.

3 Adaptive QoS-Aware Task Offloading Framework

3.1 Motivation Scenario

Let's assume, George is a user of an augmented reality application running on his smartphone that provides information on local flora. However, while looking up information on plants once they are known is an easy task that any smartphone can complete, localising and identifying the plant in real-time is more complicated. The smartphone, however, has access to nearby servers with far more processing power through a 5G radio connection, which provides low latency. Utilising these servers to execute computationally intensive tasks results in a faster application. An example of an off-loadable task is finding the screen locations of any flowers. Figure 1 shows a simple situation of MEC task offloading. However, the number of tasks required for computation is far more dynamic in many situations, as is the number of compute servers that the user has available. Therefore, a method that can both provide intelligent task offloading and the ability to handle a dynamic changing environment must be selected.

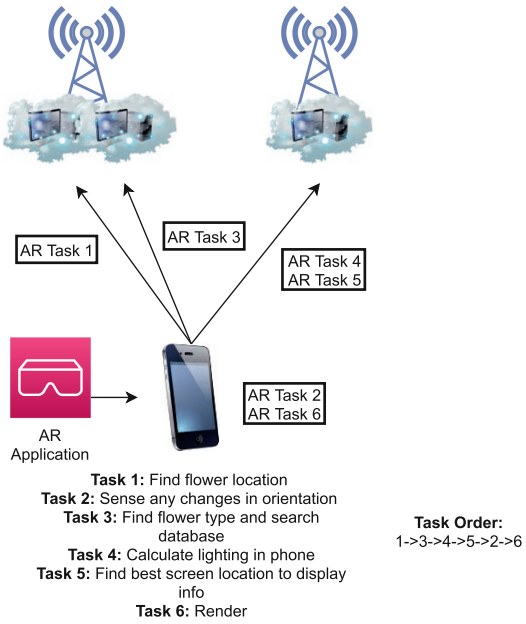

Fig. 1. AR Task Offloading

3.2 Problem Definition

Intelligent task offloading is a problematic issue that can be solved with reinforcement learning [2]. However, it is not feasible to train individual deep rein-

forcement models for each possible environment in a dynamically changing environment with current methods. The environment notations are given below:

- Task (T): A Task consists of an input size (T_i) in Mb, a processing requirement (T_p) in seconds based on a 1Ghz computer and an execution location (T_l) which is always initially set to C
- Edge Servers (S): An Edge Server is a computing location that can process tasks, it has a network connectivity speed (S_c) to the client in MBps, a processing speed (S_p) in Ghz and a count of tasks on the server (S_n)
- Clients (C): A client is the start location for all the tasks and another potential processing location with no network connection and therefore is not limited by input size and has a processing speed (C_p) in Ghz. The client also keeps track of the count of tasks running on the client (C_n).
- QoS: Quality of service is a set of factors that determines how well a distribution performs.

Assuming there is a set of tasks: $\{T_1, T_2, ...T_N\}$ and $\{S_1, S_2, ...S_M\}$ edge servers. The goal is to select a distribution of tasks that increases QoS to a maximum in the small time frame required by augmented reality applications.

3.3 Hybrid Framework for Task Offloading

In Fig. 2, the framework begins with the client device determining available edge servers and the task count. It checks pretrained models to find a suitable fit for the current environment. If a model fits, it's chosen; if not, a new model is trained. The client then utilizes the decision model for task offloading and location selection. The process repeats as the environment remains stable with infrequent changes in edge servers and task count.

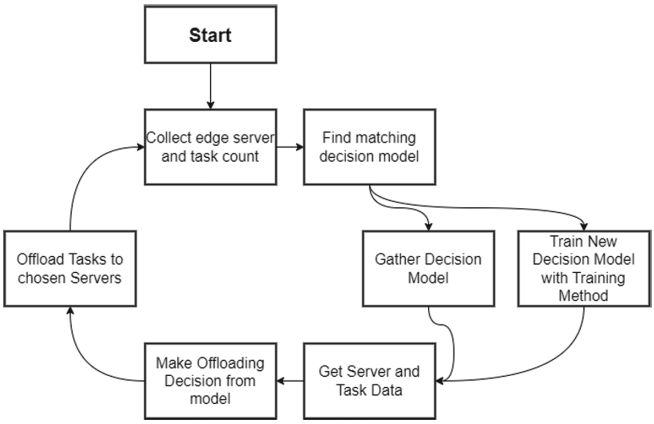

Fig. 2. Proposed Framework

We selected a DQN for the decision model despite its limitation with multiple users due to its large state space [1,4,13]. Its simplicity and proven effectiveness in research, along with its role in building other deep reinforcement

algorithms like MADDPG, led to the choice. We opted for the network-based transfer learning method of hidden weight copying due to its adaptability to deep neural network models, a modern AI paradigm.

3.4 DQN-Based Task-Offloading Decision Model

A DQN is a type of off-policy reinforcement learning that is derived from Tabular Q-Learning [14], wherein the Q-Table that is normally present is replaced with a Neural Network that learns to predict the Q Values of the current state and associated actions. It can be said that the neural network, which is a universal function approximator, attempts to approximate the function Q where $Q(S, A) = Qr$, S is the current state, A is an action that can be taken in that state and Qr is the resulting Q value of taking that action in the state. A Q value refers to the discounted cumulative reward that is derived from that action being taken in that state, the calculation for this is shown in 1.

$$Q = \sum_{t=t_0}^{\infty} (\gamma^{t-t0} r_t) \tag{1}$$

where γ refers to the discount value of the function, which controls how strongly the future reward is considered compared to the reward generated at the current step. This stops the DQN from becoming a greedy algorithm as in many cases a greedy method is not ideal, because the greatest reward can require following a path that has delayed reward.

The loss of the neural network is calculated by using the Bellman equation [14] and results in the loss Eq. 2, known as the temporal difference.

$$\delta = Q(s_t, a) - (r + \gamma(Q_1(s_{t+1}, a))) \tag{2}$$

In DQN, r is the reward, π is the policy (often a max function), and Q_1 is the target network [5]. The deep neural network approximating Q is updated via gradient descent. DQN is off-policy, enabling the exploration of unselected state spaces and preventing premature convergence. Typically, π is a greedy function selecting actions with the highest potential reward. To stabilize the target network during training, Q_1 is updated at intervals based on hyperparameter X. The DQN takes input in the form of an observation array, explained in Sect. 3.4.

Observation Array for DQN Decision Model. The DQN's observation array, a 1D array, contains vital environmental information. It encompasses the current task distribution, server and client CPU/network speed, task processing requirements, input size, and the current task. When the task distribution changes, we update the array by shifting the current task's execution location, modifying the server CPU and network speed to adapt to the new load, and increasing the current task index by 1. Frequent CPU and network speed updates ensure diverse observation arrays, preventing the neural network from consistently choosing the same task and, subsequently, poor performance.

3.5 Proposed Hybrid Learning Model for Training Method

Transfer learning is a way of using a pre-trained model to decrease the training time of a new model. In the case of DQN, this can be done in many ways [19]. However, in this implementation, a simple weight transfer and layer lock will be used as described by H. George, et al. [6] as the goals in each of the networks are very similar. Certain layers' weights will be loaded in from a previously trained DQN, these layers will then be locked so they can no longer be trained. This will improve the training speed in two ways. Firstly there will be fewer trainable parameters and therefore the speed of convergence will increase. Secondly, the layers with loaded weights will remember the previous attributes and features they were taught, meaning the DQN will be already partly trained.

QoS Feedback from Edge Services. The simulation used to emulate the environment in which the DQN will be used consists of the three main objects, Client, Severs, and Tasks as described in 3.2. The task distribution is simply represented as a list of integers with size, number of tasks, therefore the total processing time required for any distribution can be calculated as shown in 6.

$$Server(T) = \frac{8T_i S_n(T_l)}{10^3 S_c(T_l)} + \frac{T_p S_n(T_l)}{S_p(T_l)} \tag{3}$$

$$Client(T) = \frac{T_p S_n(T_l)}{S_p(T_l)} \tag{4}$$

$$TTime(T) = \begin{cases} Client(T) & \text{if } T_l = 0 \\ Server(T) & \text{if } T_l \neq 0 \end{cases} \tag{5}$$

$$Time = \sum_{t=0}^{taskNum} TTime(t) \tag{6}$$

This calculation simplifies the process to enable the generalization of task distribution ideas for deep reinforcement learning. It incorporates task-specific weights to influence execution time. The reward function prioritizes overall reduction, summing all task processing times, rather than just the slowest task. This approach encourages a more substantial reduction, albeit not necessarily reaching the global minimum due to unpredictable variables.

Training the Proposed Hybrid Model. An episode is defined as starting with all tasks distributed on the client-side, then allowing the algorithm to pick new locations for the tasks one by one, ending when all tasks have had their distributions chosen. A reward is generated at the end of each task decision. It is calculated as a percentage increase or decrease from the previous steps time. If it is a decrease, the returned reward is positive, however, if it is an increase, the returned reward is negative to force the DQN to aim to reduce the running time. This percentage is then broken down into the tiers shown in 1 which return their associated reward.

Table 1. Reward Calculation

Result	0	< 1	< 10	< 20	< 30	< 40	< 50	< 70	< 90	> 90
Decrease	0	1	10	15	35	50	60	90	120	150
Increase	0	-2	-20	-30	-70	-100	-120	-180	-240	-300

We used percentages instead of final processing times due to variable minimum processing times in different environments. Reward is categorized into tiers to reduce the need for precise predictions, promoting a stronger distribution. While accuracy may decrease, the focus is on a robust distribution, not the global minimum. The reward function disregards the one's digit, encouraging positive development. Rewards are provided at every offloading action to reduce sparsity and enhance training stability.

Choosing the training policy is vital for DQN training, affecting its exploration level. There are two common policies, the epsilon greedy policy and the BoltzmannQPolicy. The epsilon greedy policy takes in a parameter ϵ with a value range of 0 to 1 and the Q values generated by the neural network. With probability ϵ it selects a random action and with probability $1 - \epsilon$ it selects the action with the highest Q value. This means the amount of exploration is directly correlated to ϵ. The BoltzmannQ policy however generates a probability distribution based on a temperature value T and the Q values generated by the neural network. This has the same effect as when T is large the DQN experiences large amounts of exploration, then when small. However, the exploration is biased toward actions with high Q values.

4 Experiments

4.1 Experimental Setup

For this experiment, all tasks will have the same emphasis, however, in real-world implementations of this framework, weighted tasks can be extremely beneficial as certain algorithms require certain tasks to be finished quicker. Table 2 outlines the environment variables used during the simulation.

Two environments were used for individual DQN training, each with 4 servers and 15 tasks. The first DQN was trained in an environment with 3 servers and 10 tasks; its weights were transferred to a new DQN for training in the second environment. Notably, weights for the input-to-first-hidden-layer and last-hidden-layer-to-output-layer were not copied due to different dimensions. These DQNs underwent 60 iterations with the same hyperparameters.

During transfer learning, various hidden layer locking patterns were tested, including locking the first hidden layer, no layers, all layers, and the middle three layers. The top-performing pattern was selected and retrained for 200 iterations.

Table 2. Environment Variables

Variable	Value
S_c	1–50 MB/s
S_p	1–5 GHz
T_i	100–1000 MB
T_p	0.0001–4 s/GHz
C_p	1–2 GHz

The epsilon-greedy policy was used during DQN training to encourage exploration. The epsilon value decreased linearly from 0.9 to 0 over 10,000 training steps, allowing the DQN to explore broadly initially and then focus on states closer to the ideal state as it improved. Testing employed a simple greedy algorithm, always selecting the action with the highest value (Table 3).

Table 3. Hyperparameters

Parameter	Value
Policy	Epsilon Greedy
Epsilon	0.9–0
Learning Rate	1e−5
Target Network Update	1000
Training Steps	2,000,000
Hidden Layer Size	$400 \times 400 \times 400 \times 300 \times 200 \times 100 \times 50$

4.2 Analysis

The Baseline Approach. We use a brute force method to calculate the global minimum time, aka the global maximum reward. Using this we can then calculate the Root Mean Square Error (RMSE) of our Deep Q method after training. To compare during training it is not feasible to run the brute force algorithm for every data point due to its high computational complexity, therefore, we use the reward generated by the simulation to compare methods.

Hybrid Task-Offloading Model Results. In Fig. 3, raw data from various locking patterns is presented. The network parameters were initially trained in an environment with 3 servers and 10 tasks, then transferred to a new network with different input and output shapes for retraining in a 4-server, 15-task environment. The locking patterns considered were all-lock, first-lock, half-lock, and no-lock. No-lock performed the best, followed closely by all-lock and first-lock, while half-lock was less effective. For subsequent experiments, the no-lock method will be employed.

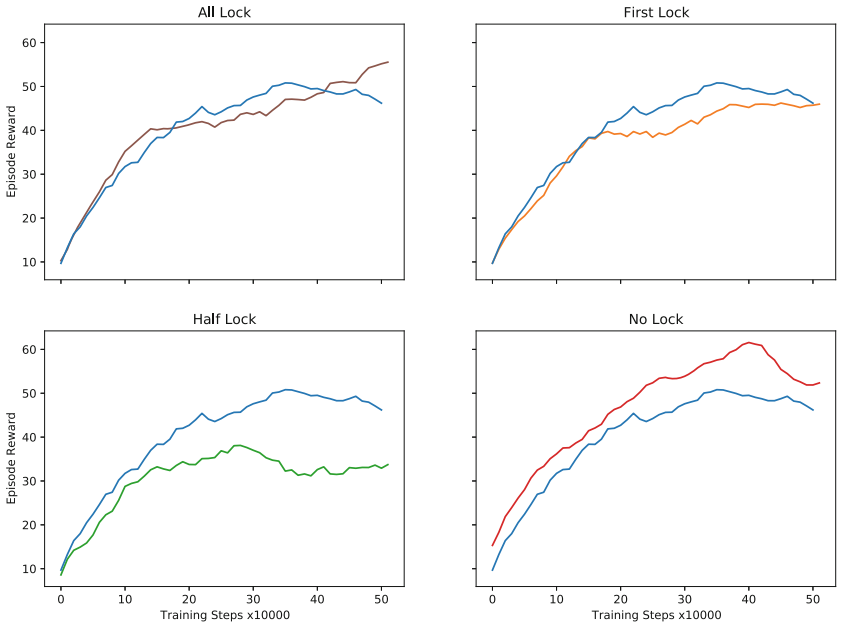

Fig. 3. First 60 Training Steps of Locking Patterns (All-Lock (Brown), First Lock (Orange), Half Lock (Green), No Lock (Red)) Compared with Vanilla DQN (Blue) Training Smoothed (Color figure online)

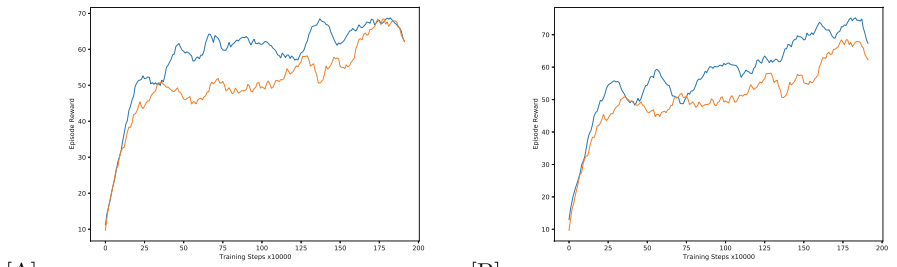

Fig. 4. Transfer Learning DQN Training with No Lock (Yellow) vs Vanilla DQN (Blue) (Color figure online)

Figure 3 compares the locking methods' 60 training steps to the first 60 training steps of the DQN trained in the vanilla DQN method. These graphs further show how the no-locking method performs much better than the other methods.

The DQN was trained using transfer learning with a no-lock method in two runs. Figure 4.A and 4.B display the raw data for both training iterations. The locking method consistently outperformed vanilla DQN training, with the second run finishing closer to 65 than 60, indicating significant improvement.

Comparative Analysis. Distributions are challenging to assess accurately, so they are compared to the brute force algorithm's ground truth. The ground truth averaged a 77% reduction, while the proposed method achieved an average reward of 70, equivalent to a 50–60% reduction in offloading time. These results are robust, given that the brute force algorithm takes about a minute to run, even in relatively small exploration spaces. Table 4 displays root mean square errors, demonstrating that the model performs well in smaller state spaces and that transfer learning and vanilla DQN [7] yield similar results.

Table 4. Root Mean Square Error

Model	RMSE
3×10 Vanilla DQN	0.33
4×15 Vanilla DQN	0.43
4×15 Hybrid Framework	0.43

The DQN, with limited environment information, also achieves a 50–60% reduction in overall running time, while the baseline averages around 80%. The root mean square errors of 0.33 and 0.43 indicate the closeness of the results. Employing DQN for task distribution in Mobile Edge computing with 5G results in an approximate 50% performance boost. Using transfer learning, although end results match, the training curve yields higher rewards, reducing training time compared to vanilla DQN.

5 Conclusion

We addressed the task offloading issue in MECs via 5G. Traditional DQNs and other reinforcement learning approaches struggle with dynamic environments. Our proposal leverages transfer learning to speed training and enhance DQNs' adaptability. We presented a framework for handling task offloading adaptivity using DQNs and transfer learning, explained the simulation environment, and conducted experiments. Results demonstrated that DQNs, with transfer learning, achieve a significant 50–60% reduction in task running time, maintaining learned knowledge across domains for faster convergence and adaptability.

References

1. Chen, X., Zhang, H., Wu, C., Mao, S., Ji, Y., Bennis, M.: Optimized computation offloading performance in virtual edge computing systems via deep reinforcement learning. IEEE Internet Things J. **6**(3), 4005–4018 (2018)
2. Chen, X., Liu, G.: Energy-efficient task offloading and resource allocation via deep reinforcement learning for augmented reality in mobile edge networks. IEEE Internet Things J. 1–1 (2021). https://doi.org/10.1109/JIOT.2021.3050804

3. Furht, B.: Handbook of Augmented Reality. Springer, New York (2011)
4. Hasselt, H.: Double q-learning. Adv. Neural. Inf. Process. Syst. **23**, 2613–2621 (2010)
5. van Hasselt, H., Guez, A., Silver, D.: Deep reinforcement learning with double q-learning. CoRR **abs/1509.06461** (2015). http://arxiv.org/abs/1509.06461
6. He, G., Yang, D., Shen, K.: Improving DQN training routines with transfer learning (2019)
7. Hossain, M.S., Nwakanma, C.I., Lee, J.M., Kim, D.S.: Edge computational task offloading scheme using reinforcement learning for IIoT scenario. ICT Express **6**(4), 291–299 (2020)
8. Huang, J.T., Li, J., Yu, D., Deng, L., Gong, Y.: Cross-language knowledge transfer using multilingual deep neural network with shared hidden layers. In: 2013 IEEE International Conference on Acoustics, Speech and Signal Processing, pp. 7304–7308. IEEE (2013)
9. Huang, Z., Li, W., Hui, P., Peylo, C.: Cloudridar: A cloud-based architecture for mobile augmented reality. In: Proceedings of the 2014 workshop on Mobile Augmented Reality and Robotic Technology-Based Systems, pp. 29–34 (2014)
10. Kartik, S., Siva Ram Murthy, C.: Task allocation algorithms for maximizing reliability of distributed computing systems. IEEE Trans. Comput. **46**(6), 719–724 (1997). https://doi.org/10.1109/12.600888
11. Kounavis, C.D., Kasimati, A.E., Zamani, E.D.: Enhancing the tourism experience through mobile augmented reality: challenges and prospects. Int. J. Eng. Bus. Manag. **4**, 10 (2012)
12. Ling, H.: Augmented reality in reality. IEEE Multimedia **24**(3), 10–15 (2017)
13. Lowe, R., Wu, Y., Tamar, A., Harb, J., Abbeel, P., Mordatch, I.: Multi-agent actor-critic for mixed cooperative-competitive environments. arXiv preprint arXiv:1706.02275 (2017)
14. Mnih, V., et al.: Playing atari with deep reinforcement learning. arXiv preprint arXiv:1312.5602 (2013)
15. Shi, W., Dustdar, S.: The promise of edge computing. Computer **49**(5), 78–81 (2016)
16. Singh, S.: Optimize cloud computations using edge computing. In: 2017 International Conference on Big Data, IoT and Data Science (BID), pp. 49–53. IEEE (2017)
17. Tan, C., Sun, F., Kong, T., Zhang, W., Yang, C., Liu, C.: A survey on deep transfer learning. In: Kůrková, V., Manolopoulos, Y., Hammer, B., Iliadis, L., Maglogiannis, I. (eds.) ICANN 2018. LNCS, vol. 11141, pp. 270–279. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-01424-7_27
18. Zhang, K., Zhu, Y., Leng, S., He, Y., Maharjan, S., Zhang, Y.: Deep learning empowered task offloading for mobile edge computing in urban informatics. IEEE Internet Things J. **6**(5), 7635–7647 (2019)
19. Zhu, Z., Lin, K., Zhou, J.: Transfer learning in deep reinforcement learning: a survey. arXiv preprint arXiv:2009.07888 (2020)