Computation Offloading in Multi-access Edge Computing using Deep Sequential Model based on Reinforcement Learning

Jin Wang[1], Jia Hu[1], Geyong Min[1], Wenhan Zhan[2], Qiang Ni[3], Nektarios Georgalas[4]
[1]College of Engineering Mathematics and Physical Sciences, University of Exeter, UK
[2]School of Computer Science and Engeeneering, University of Electronic Science and Technology of China
[3]School of Computing and Communications and Data Science Institute, Lancaster University, UK
[4]Applied Research Department, British Telecom, UK
Email: {jw855, j.hu, g.min}@exeter.ac.uk, zhanwenhan@uestc.edu.cn,
q.ni@lancaster.ac.uk, nektarios.georgalas@bt.com

***Abstract*—Multi-access Edge Computing (MEC) is an emerging paradigm which utilizes computing resources at the network edge to deploy heterogeneous applications and services. In the MEC system, mobile users and enterprises can offload computation-intensive tasks to nearby computing resources to reduce latency and save energy. When users make offloading decisions, the task dependency needs to be considered. Due to the NP-hardness of the offloading problem, the existing solutions are mainly heuristic, and therefore have difficulties in adapting to the increasingly complex and dynamic applications. To address the challenges of task dependency and adapting to dynamic scenarios, we propose a new Deep Reinforcement Learning (DRL) based offloading framework, which can efficiently learn the offloading policy uniquely represented by a specially designed Sequence-to-Sequence (S2S) neural network. The proposed DRL solution can automatically discover the common patterns behind various applications so as to infer an optimal offloading policy in different scenarios. Simulation experiments were conducted to evaluate the performance of the proposed DRL-based method with different data transmission rates and task numbers. The results show that our method outperforms two heuristic baselines and achieves nearly optimal performance.**

I. Introduction

The technological evolution of smartphones, tablets, and wearable devices is driving the emergence of novel computationally demanding services and applications such as virtual reality (VR), augmented reality (AR), face recognition, and mobile healthcare. Although new generations of mobile devices possess more computing power, they are still unable to run the computation-intensive applications efficiently. To address this challenge, Multi-access Edge Computing (MEC) [1] has been proposed to move computing and storage out of the remote cloud and closer to the user. Therefore, the computation-intensive applications in MEC can be offloaded to computing resources at the network edge in order to achieve high Quality-of-Service (QoS) and low energy consumption.

An MEC application must be able to decide which tasks should be offloaded to nearby computing resources. A good task offloading strategy can save energy on devices and reduce the response time of applications. On the contrary, an inappropriate offloading policy can cause high energy consumption and poor response time. The existing works generally consider independent tasks [2] [3] offloading or dependent tasks offloading [4] [5]. For example, Lin et al. [4] propose an offloading method with task dependency on different processors based on Heterogeneous Earliest Finish Time (HEFT). De Maio et al. [5] propose a heuristic based approach to find a trade-off solution among application runtime, battery lifetime and user cost. Most of the existing works on offloading strategies are based on heuristic algorithms, because of the NP-hardness of MEC offloading. However, with the increasing complexity of MEC applications and wireless network architecture, it is hard for any heuristic offloading strategy to fully adapt to the various scenarios in MEC.

In order to tackle this issue, we propose a deep reinforcement learning (DRL) based offloading framework. Recent breakthroughs in DRL have been achieving great successes in different areas including gaming, robotics, resource allocation, computer systems, etc. One of the most well-known achieve-

Corresponding authors: Jia Hu; Geyong Min

ment is AlphaGo, which beats the best human players in the game of Go [6]. DRL has a strong ability to handle complex problems by efficiently learning from experiences, and therefore it is naturally suited to obtain the optimal offloading policies in various complicated MEC scenarios.

Using DRL methods in MEC has multiple benefits. First, the dynamics of MEC systems can be highly complex and hard to model. DRL methods treat the complicated system as a black box and interact with it to learn the optimal policies without modelling the system dynamics. Second, DRL methods combine Deep Neural Network (DNN) which can effectively extract hidden patterns from large and complex datasets of heterogeneous MEC applications.

Although there are emerging DRL-based offloading methods [7], [8], [9], they assume that the tasks are independent. In our work, we consider the general task dependency and model it as a Directed Acyclic Graph (DAG). To the best of our knowledge, this work is the first of its kind to solve the offloading problem in MEC considering the general task. Inspired by the previous work [10] on Travelling Salesman Problem (TSP), we utilize the Sequence-to-Sequence (S2S) neural network with DRL training to solve the challenging problem of task offloading in MEC.

Although there are outstanding merits in DRL methods, there remain significant challenges when applying DRL to solve task offloading problems in MEC. One key challenge is how to represent the state and action space as well as the reward function in the RL framework for MEC offloading. Moreover, an appropriate DNN structure is needed for representing an effective offloading policy (i.e., a mapping from state to action). This paper proposes effective DRL methods to address these challenges for task offloading in MEC. The main contributions are concluded as follows:

- The offloading problem in MEC is formulated as a Markov Decision Process (MDP). More specifically, we model the offloading plan and the DAG of tasks as the state, the offloading decision for each task as the action and the negative increment of the running cost as the reward.
- To capture the characteristics of DAGs, a new S2S neural network is proposed for state representation. We encode the vertices and adjacent information into embedding vectors, which are used as the input of the S2S neural network. The output is the sequence vectors representing the offloading policies for DAGs.

The rest of this article is organized as follows. The proposed DRL-based offloading architecture is described in Section II. The model details of the offloading solution are presented in Section III. Simulation results are presented and discussed in Section IV. Finally, Section VI concludes this article.

II. Overall Design of the DRL-based Offloading Framework in Multi-Access Edge Computing

In this section, we firstly introduce the basic concept of MEC and the formulation of the task offloading problem. The proposed DRL-based offloading framework is then presented in detail.

A. Multi-access Edge Computing (MEC)

MEC utilizes computing and storage resources deployed at the network edge for running applications and computation tasks. Numerous MEC applications have emerged in different areas, which improve the quality of living and enhance productivity. Meanwhile, a variety of applications such as social networking, gaming, and AR become increasingly complex, and therefore demand more computing resources and energy. Offloading some tasks in these applications to MEC servers can increase the QoS and reduce energy consumption. Another attractive application scenario of MEC offloading is in the intelligent transportation system, where vehicles can offload the tasks of sensory data analysis and navigation path identification to road side units (RSUs) for fast processing. However, the diversity of application scenarios brings about unpredicted challenges to offloading methods. Therefore, efficient and flexible task offloading methods are crucial to the success of MEC.

B. Task offloading in MEC: Problem Formulation

In general, a mobile application consists of different tasks with dependencies. For example, a face recognition application consists of two main logical blocks: a face detector and a classifier, which can be composed into several dependent tasks [11]. We

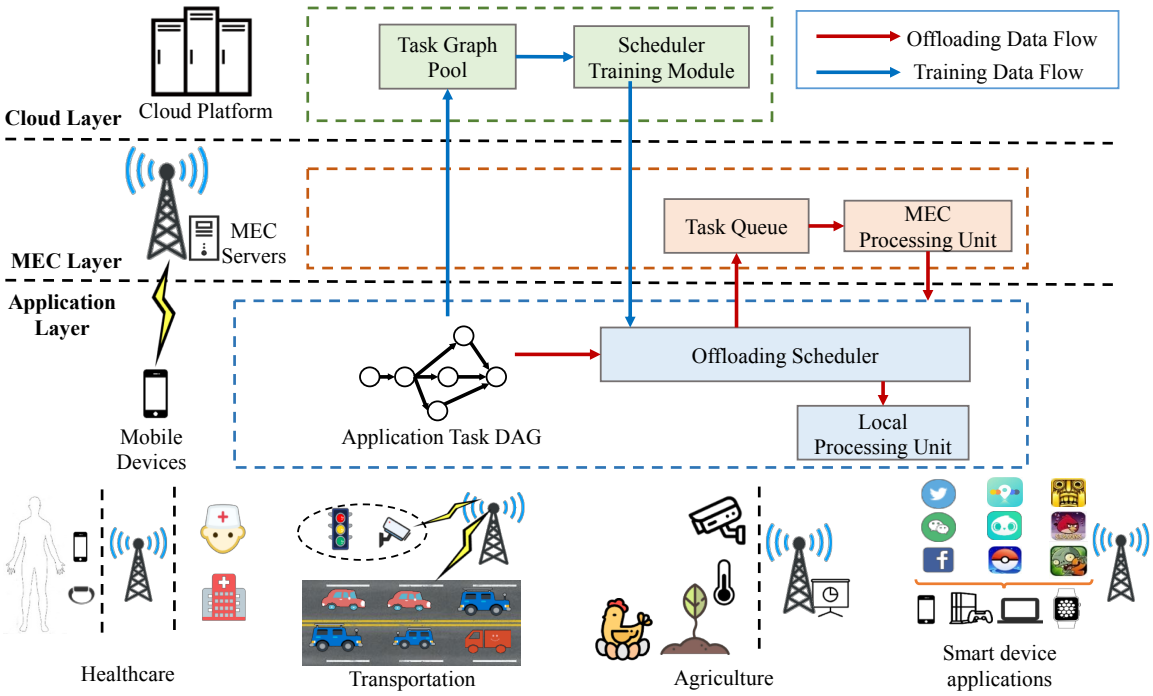

Fig. 1. The proposed DRL-based computation offloading framework in MEC systems.

model a mobile application by using a DAG, where a vertex represents a task in the application and a directed edge represents the dependency between tasks. A task can start to run only when all of its predecessors are finished.

We assume all tasks in the application DAG can be offloaded to an MEC server or run locally on mobile devices. The MEC server receives offloaded tasks and executes them one by one. After finishing executing each task, the processing results will be returned to mobile devices. Therefore, the offloading cost of a task includes the cost of uploading the task data, executing the task on the server, and downloading the execution results.

We assume that the required CPU cycles and transmission data sizes (both uplink and downlink) for each task in a given DAG are known as a prior. A scheduling plan of a DAG is defined as a sequence of offloading decisions for all the tasks in the DAG. Solving the offloading problem means to find an optimal scheduling plan where the total running cost is minimal. The running cost is a general definition depending on the specific service request. For example, the total running cost can be latency if it is a delay-sensitive service request, or energy consumption if it is an energy-efficient service request, or the combination of both if it is a hybrid service request. In this work, we solely consider latency as the running cost, thus the target is to minimize the overall latency of the service.

C. The proposed DRL-based Framework

As shown in Fig. 1, the DRL-based offloading framework is integrated into the MEC system. In the application layer, an offloading scheduler is embedded to each mobile device, which will decide the scheduling targets for tasks. A key component for the offloading scheduler is an S2S neural network, which can make offloading decisions through network inference. In the MEC layer, a task queue is used to receive offloaded tasks from mobile devices while an MEC processing unit executes those tasks. The Cloud layer includes two components: one is the task graph pool which is used to gather application DAGs from mobile devices; the other is the scheduler training module which conducts DRL training based on the gathered DAGs and the environment. In the following, we explain the training and offloading data flows of our framework.

- **Training Data Flow:** Application DAGs are gathered into the task graph pool through MEC servers. Next, the scheduler training module starts to train the S2S neural network based on the task graph pool. Afterwards, the trained

neural network parameters are sent back to mobile devices. The training procedure is done in cloud because the training for deep neural networks requires powerful computing, while we need to make full use of the limited resources in MEC servers to provide better QoS for applications.

- **Offloading Data Flow:** With the trained neural network, forward propagation is run to make offloading decisions for tasks. Next, the tasks are either offloaded to the task queue and then processed by the MEC processing unit, or executed by the local processing unit.

III. Models and Algorithms for the DRL-based Task Offloading Solution

In this section, we present the details of our DRL-based offloading solution. We first briefly introduce the key concepts of DRL. The offloading problem is then formulated as an MDP and a new S2S neural network is proposed to model the offloading policy. Finally, the methods of the policy network inferencing and DRL training are presented.

A. Deep Reinforcement Learning

In the framework of RL, an agent interacts with the environment and learns an optimal policy by trial and error for sequential decision-making. Formally, a policy is defined as the probability of taking an action after observing a state of the environment. When the agent takes an action, the environment provides a numerical reward and shifts its state. The goal of RL is to learn an optimal policy which can get the maximal total reward.

Recently, DRL was proposed to handle large state spaces by combining RL with DNN. The DRL methods can be classified into two categories: value-based and policy-based. In value-based methods, the value function is used to represent how good is the state. Value-based methods indirectly obtain optimal policy via learning the optimal value function. One typical value-based method is deep Q-Learning [12]. For policy-based methods, the optimal policy is trained directly through policy gradient descent. Specifically, the policy is represented through a probability model (e.g. DNN) rather than greedily selecting actions based on the value function.

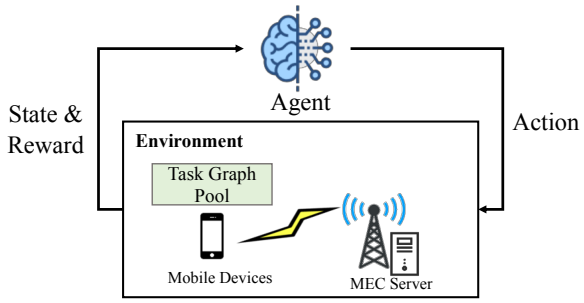

Fig. 2. Modelling the MEC offloading as a RL problem.

B. The MDP model for Task Offloading

The factors that influence the offloading decision of each task include the DAG structure, the task properties and the states of the MEC server and mobile devices. As shown in Fig. 2, the MEC server, mobile devices, mobile applications, and the wireless channel are considered as the environment. The environment is treated as a black box in DRL. Therefore, we can ignore the dynamics of the environment and only observe the reward signals and states from it. The state space, action space and reward function are designed as follows:

- **State Space:** The state space is represented as a combination of the encoded DAG and offloading plan. Formally, we denote a state as $s = (G, A_{1:i})$ where G represents the DAG and $A_{1:i}$ is a vector representing the offloading plan for the first i tasks. More specifically, G consists of a sequence of task embedding vectors, and each of them is a concatenation of three vectors: 1) A vector that embeds the current task index i and the estimated task costs. 2) A vector that contains the direct predecessors' indices. 3) A vector that contains the direct successors' indices.
- **Action Space:** In our setting, a task can be offloaded to an MEC server or executed in a local device, thus we define the action space as $\mathcal{A} = \{1, 0\}$, where 1 represents offloading and 0 stands for local execution.
- **Reward Function:** The objective of the offloading problem is to minimize the total latency of tasks. To achieve this objective, we first design the reward function as the estimated negative increment of the latency after making a decision (offloading or local execution). Next, the DRL method will learn an optimal offloading policy to get the maximal accumulative

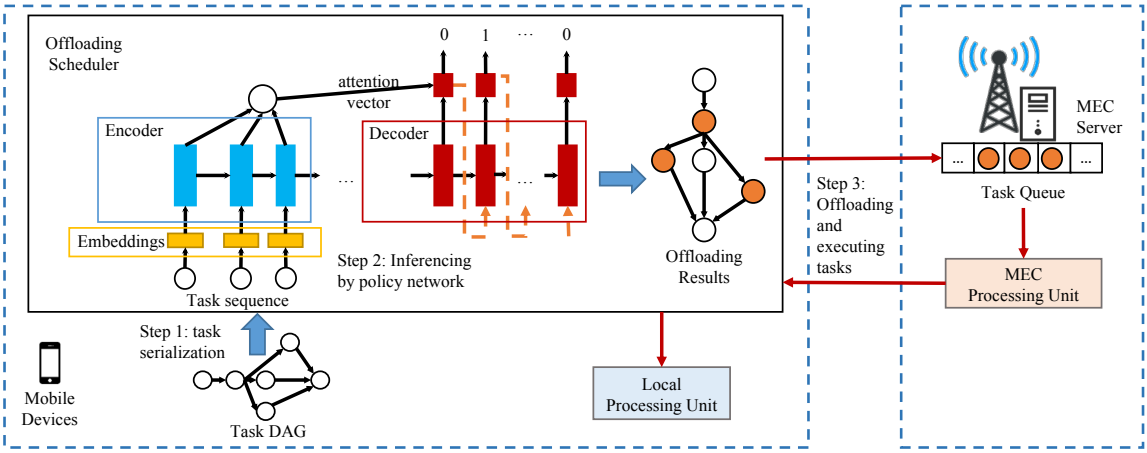

Fig. 3. Detailed structure of the S2S neural network and the process of task offloading.

reward, which is equivalent to the minimal total latency.

C. S2S Policy Network Architecture

According to the definition of our state space, we can convert the offloading problem into an S2S prediction problem where the input sequence is the task embedding vectors and the output is the decision sequence for tasks. The policy is defined as the probability distribution over the actions for a task after observing the state. For a DAG with n tasks, an offloading plan represents a decision sequence for all tasks. Therefore, the probability of having the offloading plan can be obtained by applying chain rules of probability on the policy for each task.

In our proposed method, we use an S2S architecture with attention mechanism [13] to model the policy. As shown in Fig. 3, our proposed S2S architecture consists of encoder and decoder parts. Both the encoder and decoder are implemented by a Recurrent Neural Network (RNN). The encoder is used to encode the sequence of input tasks and the decoder outputs the offloading decision for each task.

D. Task Offloading and Model Training Process

Fig. 3 illustrates the details of the offloading process, which consists of three steps:

Step 1: The DAG is transformed into task sequences by a specially designed topological sort referring to the estimated average finish time (EFT). We calculate the EFT of each task by summing up the running cost of the current task and the maximal EFT of previous tasks. The tasks are then indexed in ascending order of EFT.

Step 2: The task sequence is converted into embedding vectors as defined in state space. The embedding vectors are fed into the encoder. After encoding, the output of the encoder will then be fed to the decoder. At each decoding step, the decoder outputs the action for each task and feeds the action as the input of the next decoding step.

Step 3: With the results of offloading decisions, each task is offloaded to the MEC server or executed in the local processing unit of the mobile device.

According to our definition of MDP for this offloading problem, the training goal is to obtain an optimal policy to maximize the expected value of the cumulative sum of rewards. Since we use the S2S neural network with parameters θ to represent the policy, the training goal is then equivalent to obtaining an optimal θ for the S2S neural network.

The S2S neural network is trained via the Proximal Policy Optimization (PPO) [14] algorithm which has a high training performance and stability. At the beginning of PPO, two S2S neural networks are initialized with the same parameters. One neural network is used for sampling, and the other is for updating. In each training loop, the training process can be divided into two stages: exploration and exploitation. At the exploration stage, we use the sampling neural network to sample the action sequence of each DAG based on the sequence of initial states. Rewards are obtained by applying the sampled actions to the environment. At the exploitation stage, the gradients of the loss function

defined in PPO are calculated based on the collected sequences of states, actions and rewards. We then run the minibatch stochastic gradient ascent on the updating neural network for several epochs. At the end of each training loop, the parameters of the two S2S neural networks are synchronized.

IV. Numerical Results

In order to simulate the variety of user applications, we implement a synthetic DAG generator to randomly generate application graphs. The generator produces DAGs with the following parameters:

- *n*: the number of tasks in each DAG.
- *fat*: the width of the DAG. A small value can lead to a thin DAG while a large value results in a fat DAG.
- *density*: the numbers of dependencies between tasks of two consecutive DAG layers.
- *ccr*: communication to computation ratio, which is the ratio of the average communication cost to the average computation cost. A low *ccr* means that the application is computation intensive while a high *ccr* means that it is communication intensive.

Eight DAG sets with different task numbers *n* and various characteristics are generated for evaluating the performance of the proposed offloading method. We increase *n* from 10 to 45 with a step size of 5. We randomly select *fat* from the set [0.3, 0.5, 0.7], *density* from [0.6, 0.7, 0.8] and *ccr* from [0.3, 0.4, 0.5]. Notice that since most of the mobile applications are computation intensive, we set the *ccr* less than 0.5. For each generated DAG, we randomly assign the transmission data size from 5KB to 50KB for each task. The number of CPU cycles required by each task ranges from 10 to 100 megacycles.

We assume a small cell network where mobile devices have different transmission rates depending on their distances from the Access Point (a longer distance means a smaller rate). The transmission rates from a distal end to a proximal end are set as [1.2Mbps, 7Mbps, 20Mbps, 30Mbps, 65Mbps]. The computing capabilities of the mobile device and MEC server are 1GHz and 10GHz, respectively. In order to evaluate the impact of the transmission rate to our proposed method, we generate 100 different DAGs with 10 tasks each and run our method on the fly. We implement the S2S neural networks and

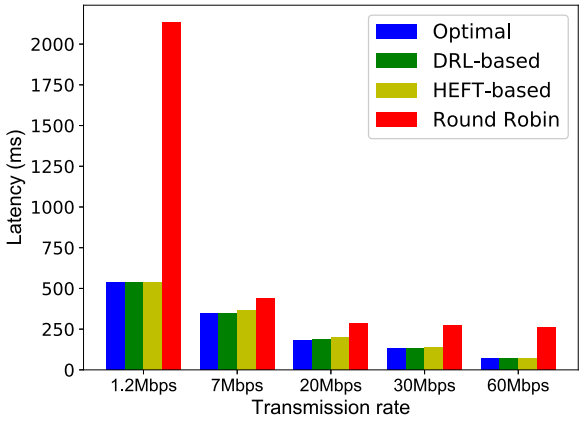

Fig. 4. The average latency of offloading methods with different transmission rates.

TABLE I
THE AVERAGE LATENCY (ms) OF OFFLOADING METHODS WITH DIFFERENT TASK NUMBERS (n).

	Optimal	**DRL-based**	**HEFT-based**	**Round-robin**
n=10	347.14	**349.7**	365.05	436.53
n=15	528.25	**537.63**	559.31	654.88
n=20	616.56	**632.80**	672.57	771.45
n=25	764.98	**790.79**	833.90	948.71
n=30	N/A	**899.7**	968.52	1066.62
n=35	N/A	**1096.86**	1168.28	1283.87
n=40	N/A	**1185.59**	1262.06	1372.12
n=45	N/A	**1358.26**	1450.44	1573.01

PPO algorithm by using Tensorflow. The Optimal algorithm is implemented via exhaustive search in solution space. We also compared our method with two heuristic baselines: the HEFT-based algorithm and Round-robin. Specifically, the HEFT-based algorithm is implemented by two steps: 1) it first prioritizes the tasks in DAG following the strategy in work [4]. 2) the tasks are then scheduled to the resource (local processor or remote MEC server) with earliest estimated finish time.

Fig. 4 shows the comparison results of the average latency of applications between different offloading methods. When the transmission rate is extremely low, the Round-robin algorithm results in huge latency because offloading tasks with low transmission rate can lead to the high communication cost. The HEFT-based algorithm generally performs well via greedy selecting resources according to the estimated finish time. However, greedy selecting can easily lead to local optimal. The DRL-based algorithm outperforms the heuristic

baselines and approximates the optimal solutions in all scenarios.

We further run the experiment on varying DAG sets with different task numbers. The transmission rate is fixed at 7Mbps. The results are shown in Table I. When n is larger than 25, it is impossible to find the optimal solution in a reasonable amount of time. The table shows that the DRL-based method can achieve nearly optimal results with polynomial time complexity. In all cases, the DRL-based method outperforms both the HEFT-based and the Round-Robin algorithms.

V. Conclusion

In this article, we investigate the challenging problem of computation offloading in MEC systems. We first introduce the MEC system and describe the offloading problem in detail. In order to solve the problem, we propose a novel DRL-based offloading framework, where the offloading problem is modelled as an MDP and an S2S neural network is designed to represent the offloading policy. The policy network is trained using a tailored policy gradient method. Numerical results show that our proposed method achieves lower latency than two heuristic baselines in scenarios with different transmission rates and task numbers. Moreover, our method can obtain the nearly optimal results while having polynomial time complexity.

VI. Acknowledgement

This work is partially supported by UK EPSRC fund EP/M013936/2.

References

[1] D. Sabell, V. Sukhomlinov, L. Trang, S. Kekki, P. Paglierani, R. Rossbach, X. Li, Y. Fang, D. Druta, F. Giust *et al.*, "Developing software for multi-access edge computing," *ETSI White Paper*, vol. 20, 2019.

[2] T. Q. Dinh, J. Tang, Q. D. La, and T. Q. Quek, "Offloading in mobile edge computing: Task allocation and computational frequency scaling," *IEEE Transactions on Communications*, vol. 65, no. 8, pp. 3571–3584, 2017.

[3] H. Guo, J. Liu, and J. Zhang, "Computation offloading for multi-access mobile edge computing in ultra-dense networks," *IEEE Communications Magazine*, vol. 56, no. 8, pp. 14–19, 2018.

[4] X. Lin, Y. Wang, Q. Xie, and M. Pedram, "Task scheduling with dynamic voltage and frequency scaling for energy minimization in the mobile cloud computing environment," *IEEE Transactions on Services Computing*, vol. 8, no. 2, pp. 175–186, 2015.

[5] V. De Maio and I. Brandic, "First hop mobile offloading of dag computations," in *2018 18th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*. IEEE, 2018, pp. 83–92.

[6] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton *et al.*, "Mastering the game of go without human knowledge," *Nature*, vol. 550, no. 7676, p. 354, 2017.

[7] J. Li, H. Gao, T. Lv, and Y. Lu, "Deep reinforcement learning based computation offloading and resource allocation for mec," in *Wireless Communications and Networking Conference*. IEEE, 2018, pp. 1–6.

[8] X. Chen, H. Zhang, C. Wu, S. Mao, Y. Ji, and M. Bennis, "Performance optimization in mobile-edge computing via deep reinforcement learning," *arXiv preprint arXiv:1804.00514*, 2018, Accessed on Aug. 2018.

[9] L. Huang, S. Bi, and Y.-J. A. Zhang, "Deep reinforcement learning for online offloading in wireless powered mobile-edge computing networks," *arXiv preprint arXiv:1808.01977*, 2018, Accessed on Mar. 2018.

[10] I. Bello, H. Pham, Q. V. Le, M. Norouzi, and S. Bengio, "Neural combinatorial optimization with reinforcement learning," *arXiv preprint arXiv:1611.09940*, 2016, Accessed on Jan. 2018.

[11] M.-R. Ra, A. Sheth, L. Mummert, P. Pillai, D. Wetherall, and R. Govindan, "Odessa: enabling interactive perception applications on mobile devices," in *Proceedings of the 9th international conference on Mobile systems, applications, and services*. ACM, 2011, pp. 43–56.

[12] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, p. 529, 2015.

[13] T. Luong, H. Pham, and C. D. Manning, "Effective approaches to attention-based neural machine translation," *Conference on Empirical Methods in Natural Language Processing*, pp. 1412–1421, 2015.

[14] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017, Accessed on Aug. 2017.

Biographies

Jin Wang is a PhD candidate in Computer Science at the University of Exeter. His research interests include machine learning, reinforcement learning, mobile edge computing, and cloud computing.

Jia Hu is a Lecturer in Computer Science at the University of Exeter, UK. His research interests include performance evaluation, future networking, resource optimization, and network security.

Geyong Min is a Professor of High-Performance Computing and Networking in the Department of Computer Science at the University of Exeter, UK. His research interests include Future Internet, Wireless Communications, Multimedia Systems, High Performance Computing, and Ubiquitous Computing.

Wenhan Zhan is a PhD candidate at the University of Electronic Science and Technology of China (UESTC), Chengdu, China. His research interests include distributed system, cloud computing, and mobile edge computing.

Qiang Ni is a Chair Professor in Communications and Networking at School of Computing and Communications and Data Science Institute at Lancaster University. His research interests include Wireless Networks, Communications, IoT, Data Analytics and Machine Learning techniques.

Nektarios Georgalas is a Principal Researcher at British Telecom's Applied Research department. His research interests include network management, distributed information systems, Cloud and NFV.