Task Offloading and Scheduling Based on Mobile Edge Computing and Software-defined Networking

Fatimah Azeez Rawdhan

Mustansiriyah University, Baghdad, Iraq

https://doi.org/10.26636/jtit.2025.1.1941

Abstract — When integrated with mobile edge computing (MEC), software-defined networking (SDN) allows for efficient network management and resource allocation in modern computing environments. The primary challenge addressed in this paper is the optimization of task offloading and scheduling in SDN-MEC environments. The goal is to minimize the total cost of the system, which is a function of task completion lead time and energy consumption, while adhering to task deadline constraints. This multi-objective optimization problem requires balancing the trade-offs between local execution on mobile devices and offloading tasks to edge servers, considering factors such as computation requirements, data size, network conditions, and server capacities. This research focuses on evaluating the performance of particle swarm optimization (PSO) and Q-learning algorithms under full and partial offloading scenarios. Simulation-based comparisons of PSO and Q-learning show that for large data quantities, PSO is more cost efficient than the other algorithms, with the cost increase equaling approximately 0.001% per kilobyte, as opposed to 0.002% in the case of Q-learning. As far as energy consumption is concerned, PSO performs 84% and 23% better than Q-learning in the case of full and partial offloading, respectively. The cost of PSO is also less sensitive to network latency conditions than GA. Furthermore, the results demonstrate that Q-learning offers better scalability in terms of execution time as the number of tasks increases, and exceeds the outcomes achieved by PSO for task loads of more than 40. Such observations prove that PSO is better suited for large data transfers and energy-critical applications, whereas Q-learning is better suited for highly scalable environments and large numbers of tasks.

Keywords — energy efficiency, MEC, PSO, Q-learning, scalability, scheduling, SDN

1. Introduction

The rapid development of mobile devices and the increasing importance of computationally-intensive services present numerous difficulties to mobile computing [1]. Mobile edge computing (MEC) has been developed to handle the problems in question by providing the necessary computational capabilities closer to the needs [2]. At the same time, software-defined networking (SDN) significantly altered the nature of networks by offering the ability to manage them through a logically centralized control interface separated from the data plane of the forwarding devices, thus ensuring flexibility and programmability for the management of the networks [3].

The combination of MEC and SDN is promising to be a solution that could improve the efficiency of mobile computing even further. SDN provides centralized control of the network, while MEC systems will be able to make better decisions concerning task offloading and resource management.

MEC provides a function known as task offload which involves reallocation of computational tasks from mobile devices that are constrained in terms of resources to edge servers with a relatively higher quantity of resources available. Efficient offloading of tasks is complicated and requires that a number of factors be considered, such as network availability, server capacity, energy consumption time, and deadlines [4].

Classical approaches to offloading have drawbacks when applied in mobile networks when it comes to achieving efficient resource allocation, especially when the task-related requirements vary. However, due to the existence of a large number of different mobile devices and edge servers, the problem of task offloading also faces another challenge. Mobile equipment can produce different levels of computation and consume various amounts of power, while edges servers might vary in their computational capacity and available resources [5].

This heterogeneity makes task offloading and scheduling a more complex issue, as the technique should be able to accommodate a wide range of device characteristics and network conditions. Due to the employment of new machine learning techniques, there are new prospects for solving these issues. Some reinforcement learning techniques, such as Q-learning algorithms, have been found to be effective in optimizing decisions in dynamic environments [6]. Similarly, other bio-inspired metaheuristic algorithms, such as particle swarm optimization (PSO), have been used to solve complex problems, such as scheduling [7].

The framework proposed in this paper introduces a new solution based on integrating SDN, MEC, and state-of-the-art machine learning approaches for efficient offloading and scheduling of tasks in the context of MEC. The proposed approach takes advantage of the global view of the network that SDN provides to collect information about the conditions in the network and the availability of resources in real-time. This

information is relied upon by a combination of Q-learning and PSO algorithms to determine the resource allocation plan.

The action selection approach the presented solution is based on includes also a Q-learning component, thus improving performance of the system by monitoring changes in network conditions and task characteristics over time. This allows to make dynamic decisions as to when the fully or partially off-loaded approaches should be used, depending on the conditions of the network and task-related demands. Furthermore, the proposed technique uses the PSO algorithm to optimize the schedule of the performing offloaded tasks in multiple edge servers in order to prevent resource waste and uneven load distribution. This framework also includes a dynamic cost model taking into account energy consumption, processing time, network latency, and the completion lead time required for a specific task.

The remainder of this paper is organized as follows. Section 2 presents the relevant literature on multiedge computing, software-defined networking, and various task offloading approaches. Section 3 describes the model of the system and formulates the problem. Section 4 presents a hybrid Q-learning and PSO-based offloading and scheduling plan. In Section 5, a detailed description of the simulation environment and the results that were obtained are presented. Lastly, conclusion are drawn and suggestions for future research are presented in Section 6.

2. Related Works

The authors of [8] proposed an integrated approach to task and resource allocation for MEC in an IoT network, using deep reinforcement learning. Implementation of a deep Q network leads to an overall energy output decrease of 15%, in addition to a 10% increase in output rate, as opposed to conventional methods. The scheme has shown adequate performance for various types and densities of networks and tasks. [9] proposed a metaheuristic approach to task scheduling for MEC based on a combination of genetic and PSO algorithms. The approach demonstrated an 18% reduction in total latency along with a 12% better efficiency of resources for various types of work. One of the algorithm's main strengths was being able to calibrate itself to the varying capabilities of edge servers.

The authors of [10] proposed a federated learning-based method to design the offload of protective tasks in an SDN-supported MEC environment. They obtained 22% less network overhead and 17% better privacy of control compared to the conventional centralized learning technique. The said approach offered great scalability as well. In other words, it was capable of addressing scenarios with a significant number of edge devices. A multi-objective optimization in MEC for energy-sensitive task offloading was proposed in [11], where an improved version of ant colony optimization was employed. The approach worked towards attaining near-optimal solutions for both power consumption and time required to complete a task, with energy consumption reduced by 20% and with a 14% increase in accomplishments per task under dynamic networks.

In article [12], the authors developed a new edge intelligence concept that combines blockchain and deep reinforcement learning to ensure safe and optimal task relocation in MEC. This strategy indicated that there is a 25% improvement in security measures and that the latency of the end-to-end technique is 16% lower than that of traditional techniques. That overarching framework was shown to work well in scenarios ranging from low to high levels of distrust among the edge nodes.

The authors of [13] proposed a context-aware task offloading scheme for MEC in a 5G network based on LSTM and the Q-learning algorithm. Their approach achieved a remarkably high-level of improvement in QoS satisfaction (19%) and a 13% reduction in energy consumption. According to more recent investigations, the scheme proved to be more efficient with regard to forecasting and managing user mobility.

2.1. Limitations of Related Research Work

In the existing literature concerned with task offloading in MEC and SDN, the following limitations are observed. Most of the existing approaches apply classic algorithms that were not capable of adjusting to the dynamic natures of network conditions and task load variations at all times. This led, in many cases, to suboptimal resource allocation and higher latency. Additionally, some algorithms incorporate reinforcement learning methods. However, those may not aim to increase both energy efficiency and performance scalability in most applications, especially when the amounts of resources are limited. The presented approach will integrate both PSO and Q-learning, so that energy efficiency will be combined with scalability of resources as the number of tasks increases. Furthermore, the new method introduces a more flexible task sharing solution that may fit both complete and partial sharing techniques, offering a higher level of flexibility.

3. System Model

In this section, we present the mathematical models and equations used in our MEC-SDN integrated system for task offloading and scheduling. The architecture of an SDN-based MEC environment consists of three main layers (Fig. 1):

- **Cloud computing layer.** It is located at the top and comprises centralized cloud facilities with a core network connected to cloud data centers.
- **Edge computing layer.** The middle layer is located between the cloud and the infrastructure and has an SDN global controller to control the entire network. It is implemented in the form of numerous edge computing zones with their own SDN local controllers. Each zone includes MEC servers used as computation facilities located at the edge of the network and OpenFlow switches for SDN-based network management.

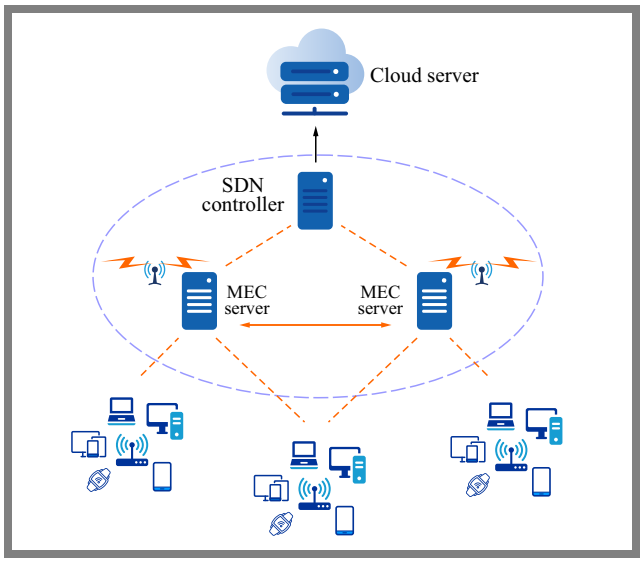

Fig. 1. Structure of an SDN-based MEC environment.

- **Infrastructure layer.** The end-user access layer is the lowest layer of the model. It includes base stations and access points (APs) and supports various user devices: desktops, servers, tablets, mobile phones, smart devices, cars, etc.

In the network model, let $N = \{1, 2, \ldots, n\}$ be the set of mobile devices and $M = \{1, 2, \ldots, m\}$ be the set of edge servers. The SDN controller manages the network topology $G = (V, E)$, where $V = N \cup M$ and E represents the set of communication links.

In the task model, each task T_i is characterized by a tuple (c_i, d_i, τ_i), where c_i is computation requirement (CPU cycles), d_i is data size (bits), and τ_i is the deadline.

Next, in the communication model, the data transmission rate between device i and server j is given by:

$$R_{ij} = B_{ij} \log_2 \frac{P_i \, h_{ij}}{N_0 \, B_{ij}} \,, \tag{1}$$

where B_{ij} is channel bandwidth, P_i stands for transmission power of device i, h_{ij} denotes channel gain, and N_0 is noise power spectral density.

In the computation model, the local execution time for task T_i on device i is:

$$T_{local_i} = \frac{c_i}{f_i} \,, \tag{2}$$

where f_i is the CPU frequency of device i.

The execution time on edge server j is formulated as:

$$T_{edge_{ij}} = \frac{c_i}{f_j} \,, \tag{3}$$

where f_j is the CPU frequency of server j.

For the energy consumption model, the energy consumption for local execution is:

$$E_{local_i} = \kappa \, c_i \, f_i^2 \,, \tag{4}$$

where κ is the energy coefficient.

Energy consumption for offloading is defined as follows:

$$E_{off_{ij}} = P_i \frac{d_i}{R_{ij}} + \varepsilon \, d_i \,, \tag{5}$$

where ε is the coefficient of the circuit power.

The decision variables are defined in the following way:

$$x_{ij} = \begin{cases} 1, & \text{if task } T_i \text{ is offloaded to server} \\ 0 & \text{otherwise} \end{cases} \,, \tag{6}$$

$$y_i = \begin{cases} 1, & \text{if task } T_i \text{ is executed locally} \\ 0 & \text{otherwise} \end{cases} \,. \tag{7}$$

3.1. Problem Formulation

We define the task of offloading and scheduling performed in the MEC-SDN integrated environment as a multi-objective optimization problem. The objective is to reduce the total cost of the system, which is a function of the total task completion time and energy consumption subject to the task deadline constraints.

Let $x_{ij} \in \{0, 1\}$ denote the offloading decision variable, where $x_{ij} = 1$ if task T_i is offloaded to server j, and 0 otherwise. Similarly, let $y_i \in \{0, 1\}$ represent the local execution decision, where $y_i = 1$ if task T_i is executed locally, and 0 otherwise.

The total task completion time (TCT) is given by:

$$TCT = \sum_{i \in N} \left(y_i \, T_{local_i} + \sum_{j \in M} x_{ij} \left(T_{trans_{ij}} + T_{edge_{ij}} \right) \right), \tag{8}$$

where $T_{trans_{ij}} = \dfrac{d_i}{R_{ij}}$ is the transmission time from device i to server j.

Total energy consumption (TEC) is expressed as:

$$TCT = \sum_{i \in N} \left(y_i \, E_{local_i} + \sum_{j \in M} x_{ij} \, E_{off_{ij}} \right) \tag{9}$$

The optimization problem may be formulated as follows:

$$\text{minimize } \; \alpha \cdot TCT + \beta \cdot TEC$$

with the following constraints:

Task allocation:

$$\sum_{j \in M} x_{ij} + y_i = 1, \quad \forall i \in N \tag{10}$$

Task deadline:

$$y_i \, t_{local_i} + \sum_{j \in M} x_{ij} \left(t_{trans_{ij}} + t_{edge_{ij}} \right) \leqslant \tau_i, \quad \forall i \in N \tag{11}$$

Server capacity:

$$\sum_{i \in N} x_{ij} \, c_i \leqslant C_j, \quad \forall j \in M \tag{12}$$

Decision variable:

$$x_{ij}, y_i \in \{0, 1\}, \quad \forall i \in N, \quad \forall j \in M \tag{13}$$

where α and β are time and energy weighting factors, respectively.

Constraint (10) ensures that each task is either offloaded to one of the servers or executed locally. Constraint (11) guarantees that the task completion time does not exceed the deadline.

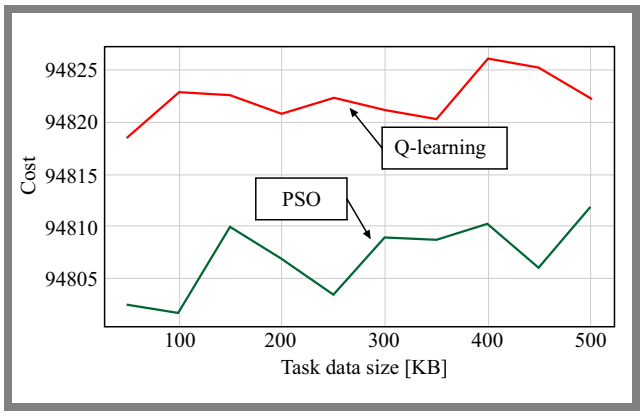

Fig. 2. Offloading cost performance versus task data size.

Constraint (12) ensures that the total computational load on each server does not exceed its capacity.

This formulation emphasizes the details of the task-offloading problem in MEC-SDN environments, considering both time and energy efficiency and respecting system-related constraints [14].

The two methods that have been introduced Algorithm 1 include PSO and Q-learning. The goal is to reduce the total cost of the system, which is a function of the time taken to complete tasks and the energy consumed while running on a dynamic network.

The algorithm starts by setting up the system's variables for the tasks to be performed, the capabilities of the edge server, and the state of the network under control of an SDN controller. Then, PSO is applied, followed by Q-learning, with both methods applied independently to determine their efficiency.

PSO is highly effective at reducing energy consumption during task scheduling, making it particularly suitable for environments with limited resources, where energy efficiency is crucial. In contrast, Q-learning is adept at handling dynamic task loads and showcases excellent scalability as the number of tasks grows.

In the case of PSO, the algorithm adjusts the positions and velocities of particles, reflecting the tasks assigned to servers. It considers the cost of every particle solution, including the local execution and the offloading strategies, which can be full or partial, and then updates the personal best and global best.

For Q-learning, the algorithm gives an agent experience in different episodes. For each episode, it selects actions (servers) for each task from the state, computes rewards according to the cost, and updates the Q-table for better decision-making in the future.

On the same note, another advantage of this algorithm is its versatility in handling both full and partial offloading techniques and its flexibility in adjusting to the dynamic network conditions controlled by the SDN. This makes it possible to assess offloading approaches under different circumstances and conditions.

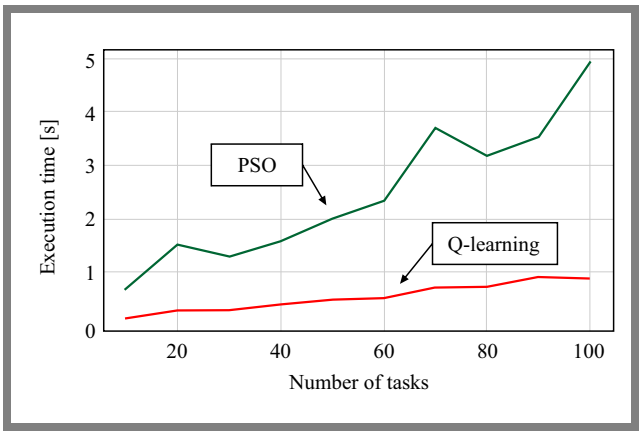

Fig. 3. Scalability test as a function of execution time versus number of tasks.

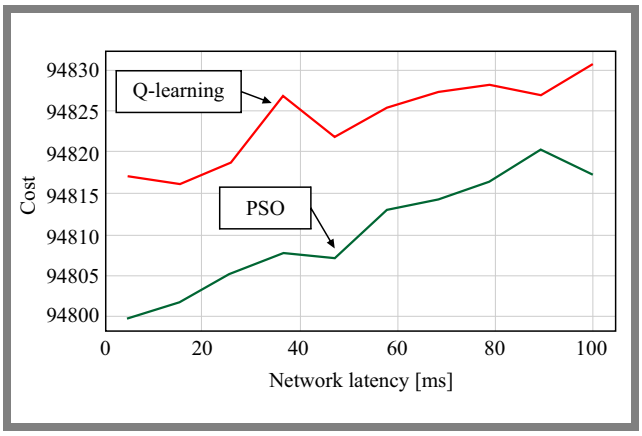

Fig. 4. Effects of network latency on offloading performance.

4. Results and Discussion

Here, simulation results related to PSO-Q are presented and compared with other conventional approaches and advanced algorithms. Such parameters as energy efficiency, task execution time, workload balance, and the capacity for expansion are assessed.

4.1. Offloading Performance and Task Data Size

Figure 2 shows the trends in the costs of PSO and Q-learning, as the task data size increases.

It is observed that both algorithms generate higher costs as the size of the task data increases, which may be attributed to the time taken to transmit the data and the amount of energy consumed. However, the result of PSO is always better than that of Q-learning, in terms of the cost for each learning process in all data sizes.

When data size increases, the difference in performance between PSO and Q-learning grows as well, which indicates that PSO is more suitable for solving large-scale data problems. This could be especially crucial in circumstances where a significant amount of data is required to be transmitted or processed, including multimedia and big data applications relying on edge computing.

Algorithm 1 Energy-efficient task offloading using PSO and Q-learning with SDN

Input: `num_tasks, num_devices, num_servers, task_computation, task_data_size, task_deadlines, server_capabilities, offloading_strategy`

Output: Optimized offloading decisions and system costs for both PSO and Q-learning

```
Initialize SDN_controller, task parameters, and network conditions
Initialize PSO particles and Q-learning agent                          ▷ PSO Optimization
for iteration = 1 to max_iterations do
    SDN_controller.update_network_conditions()
    for each particle do
        total_cost = 0
        for device_id = 1 to num_devices do
            for task_id = 1 to num_tasks do
                server_id = particle.position[task_id]
                local_time, local_energy = local_execution_cost(device_id, task_id)
                if offloading_strategy == full then
                    offloading_time, offloading_energy = full_offloading_cost(device_id,
                    server_id, task_id, SDN_controller)
                else if offloading_strategy == partial then
                    offloading_time, offloading_energy = partial_offloading_cost(device_id,
                    server_id, task_id, SDN_controller)
                end if
                 offloading_cost = offloading_time + offloading_energy
                total_cost += min(local_time + local_energy, offloading_cost)
            end for
        end for
        Update particle's personal best and global best based on total_cost
    end for
    Update particle velocities and positions
end for
                                                                       ▷ Q-learning optimization
for episode = 1 to num_episodes do
    SDN_controller.update_network_conditions()
    total_reward = 0
    for device_id = 1 to num_devices do
        for task_id = 1 to num_tasks do
            state = device_id
            action = Q_agent.choose_action(state)
            local_time, local_energy = local_execution_cost(device_id, task_id)
            if offloading_strategy == full then
                offloading_time, offloading_energy = full_offloading_cost(device_id,
                action, task_id, SDN_controller)
            else if offloading_strategy == partial then
                offloading_time, offloading_energy = partial_offloading_cost(device_id,
                action, task_id, SDN_controller)
            end if
            reward = -min(local_time + local_energy, offloading_time + offloading_energy)
            next_state = (device_id + 1) % num_devices
            Q_agent.learn(state, action, reward, next_state)
            total_reward += reward
        end for
    end for
    Store total_reward for this episode
end for
Return PSO_best_position, PSO_best_cost, Q_Learning_q_table, Q_Learning_rewards
```

When the size of the task data increases from 50 KB to 500 KB, both algorithms demonstrate an increase in costs. As for the cost, PSO clearly shows the lowest result varying from 94800 to 94827. The costs of Q-learning are higher, and they rise from approximately 94819 to 94830. The performance difference is even more pronounced at larger data sizes, and in the case of PSO the cost increase equals approximately 0.003% per kilobyte, compared to 0.002% in the case of Q-learning, and this shows that PSO is more efficient when larger amounts of data are transferred.

4.2. Scalability Test

Figure 3 illustrates the number of tasks and the corresponding execution time of the PSO and Q-learning algorithms. One may notice an interesting balance between the two algorithms. The time it takes PSO to execute its tasks increases at a faster gradient with a growing number of tasks. On the other hand, Q-learning has a comparatively constant execution time, which increases only slightly with the addition of tasks.

Hence, for a small number of tasks, i.e., less than approximately 40, PSO performs better than Q-learning. However, as the number of tasks increases above this point again, Q-learning is more efficient in terms of the time taken to execute the tasks. This crossover point is important for system designers selecting these algorithms.

The execution time of both algorithms escalates from 10 to 100 tasks, as shown in Fig. 3. The execution time of PSO rises more sharply with the number of iterations, from 0 to 5 seconds (614%). Q-learning is more scalable, as evidenced by the fact that the execution time increases from 0.2 to 0. 9 seconds (350%). Q-learning is found to be superior to PSO in terms of execution time for large task numbers.

4.3. Network Latency Effects on Offloading Performance

This graph illustrates the impact of network latency on the offloading costs of the PSO and Q-learning algorithms. As expected, both algorithms demonstrate costs that increase as a function of network latency. This is reasonable, because higher latency would mean that the transmission would take more time to complete, and in some cases the power consumption could be high as well.

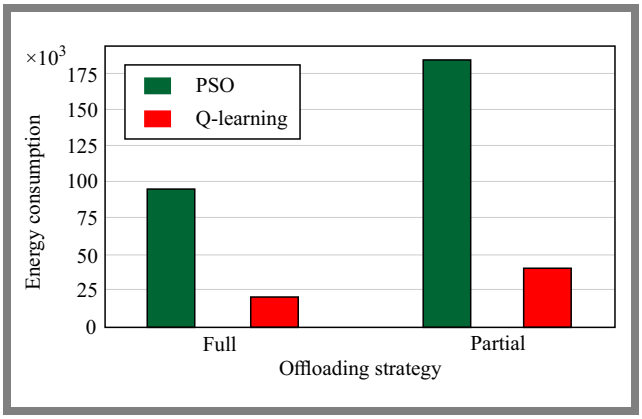

Fig. 5. Energy consumption comparison.

PSO performs better compared to Q-learning at all latency values and has lower costs throughout the range. This implies that PSO might work better in scenarios in which there are fluctuations in network latency in edge computing.

Surprisingly, the difference in performance between PSO and Q-learning is almost constant as latency increases. The fact that both algorithms exhibit parallel growth in costs proves that neither of them is more efficient in high-latency conditions. From the above results, it is clear that as the network latency increases from 0 to 100 ms, both algorithms will generate higher costs. The costs of PSO increase from 94,800 to 94,817 (by 0.018%), while those of Q-learning increase from 94,817 to 94,830 (by 0.014%). The PSO has a relatively lower cost of approximately 13-15 units, irrespective of latency values, proving its better robustness to network delays.

4.4. Energy Consumption Comparison

Figure 5 presents the energy consumption of PSO and Q-learning for full and partial offloading techniques. The findings show that PSO is characterized by better energy utilization. In the full and partial offloading scenarios, PSO uses much less energy than Q-learning. This energy efficiency may be an essential factor in edge computing applications, in which the battery life of the devices is an issue. Surprisingly, partial offloading consumes more power than full offloading for both algorithms. This may seem counterintuitive, as partial offloading is usually used to share the load between local and edge resources. However, this result implies that the cost of splitting tasks and managing partial offloading could be higher than the energy savings it offers.

The main difference between PSO and Q-learning in terms of energy consumption is even more significant in the case of partial offloading of the tasks. This suggests that PSO may be best applied in cases where partial offloading techniques are to be deployed when energy levels are a concern.

The comparison of energy consumption shows that there is a great difference between the full and partial offloading strategies. For full offloading, PSO uses 95,000 units of energy, while Q-learning uses 175,000 units of energy (i.e. 84% more). The results are similar when in the case of partial offloading, where PSO only used approximately 150,000 units, while Q-learning used approximately 185,000 units, (an increase of 23%). This implies that PSO is more energy efficient than the other algorithms, especially in cases where partial offloading is performed.

5. Discussion

The performance metrics shown in Tab. 1 prove that PSO outperforms Q-learning in terms of cost efficiency and energy consumption under all conditions, and especially when the size of data tasks is large. This finding is of significance for practitioners whose responsibilities include optimizing resource allocation in edge computing environments.

Tab. 1. Summary of results.

Metric	PSO	Q-learning	Key observation
Cost (50 – 500 kB of data)	94 800 – 94 827	94 819 – 94 830	PSO is more efficient with larger data sizes
Execution time (10 – 100 tasks)	0.7 – 5 s (614% increase)	0.2 – 0.9 s (350% increase)	Q-learning is more scalable beyond 40 tasks
Cost (0 – 100 ms latency)	94 800 – 94 817	94 817 – 94 830	PSO is more resilient to network delays
Energy (full offloading)	95 000 units	175 000 units	PSO is 84% more energy-efficient
Energy (partial offloading)	150 000 units	185 000 units	PSO is 23% more energy-efficient

For organizations that rely on edge computing for data-intensive applications, multimedia processing, and big data analytics, PSO would save a lot of money. Practitioners are advised to implement PSO in their task-offloading strategies to reduce operational costs without sacrificing performance.

The large difference in energy consumption observed between PSO and Q-learning shows how important energy efficiency is in mobile and edge computing environments. With increasing energy costs and sustainability concerns, organizations will gain from the use of PSO, as it showed an 84% drop in energy usage under full offloading scenarios.

6. Conclusions

This research offers a comparative analysis of the PSO and Q-learning algorithms for task offloading into SDN-integrated MEC scenarios. The approach adopted takes into account full and partial offloading strategies, depending on network conditions controlled by an SDN controller. The results of the simulations show that PSO offers better cost efficiency than Q-learning while handling growing task data sizes and is characterized by lower energy consumption in full and partial offloading.

PSO also demonstrates greater robustness to variations in network latency. Nevertheless, Q-learning shows better scalability than the other methods, and its performance improves when the number of tasks exceeds a specific value. These results may serve as important guidelines for system designers choosing suitable algorithms in based on the requirements of specific MEC scenarios and network conditions.

Future work may focus on examining the integration of the PSO concept with Q-learning with the aim of achieving improved offloading performance. Future work may also consider the influence of more complex network topologies and various types of fog computing resources on offloading performance.

References

[1] M. Satyanarayanan, "The Emergence of Edge Computing", *Computer*, vol. 50, no. 1, pp. 30–39, 2017 (https://doi.org/10.1109/MC.2017.9).

[2] Y. Mao *et al.*, "A Survey on Mobile Edge Computing: The Communication Perspective", *IEEE Communications Surveys & Tutorials*, vol. 19, no. 4, pp. 2322–2358, 2017 (https://doi.org/10.1109/COMST.2017.2745201).

[3] D. Kreutz *et al.*, "Software-defined Networking: A Comprehensive Survey", *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, 2015 (https://doi.org/10.1109/JPROC.2014.2371999).

[4] Y. Wang *et al.*, "Mobile-edge Computing: Partial Computation Offloading Using Dynamic Voltage Scaling", *IEEE Transactions on Communications*, vol. 64, no. 10, pp. 4268–4282, 2016 (https://doi.org/10.1109/TCOMM.2016.2599530).

[5] 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 (https://doi.org/10.1109/MCOM.2018.1701069).

[6] Y. Wei, F.R. Yu, M. Song, and Z. Han, "Joint Optimization of Caching, Computing, and Radio Resources for Fog-enabled IoT Using Natural Actor-critic Deep Reinforcement Learning", *IEEE Internet of Things Journal*, vol. 6, no. 2, pp. 2061–2073, 2019 (https://doi.org/10.1109/JIOT.2018.2878435).

[7] L. Yin, J. Luo, and H. Luo, "Tasks Scheduling and Resource Allocation in Fog Computing Based on Containers for Smart Manufacturing", *IEEE Transactions on Industrial Informatics*, vol. 14, no. 10, pp. 4712–4721, 2018 (https://doi.org/10.1109/TII.2018.2851241).

[8] Y. Wang *et al.*, "Cooperative Task Offloading in Three-tier Mobile Computing Networks: An ADMM Framework", *IEEE Transactions on Vehicular Technology*, vol. 68, no. 1, pp. 2763–2776, 2019 (https://doi.org/10.1109/TVT.2019.2892176).

[9] J. Li, H. Gao, T. Lv, and Y. Lu, "Deep Reinforcement Learning Based Computation Offloading and Resource Allocation for MEC", *IEEE Wireless Communications and Networking Conference (WCNC)*, Barcelona, Spain, 2022 (https://doi.org/10.1109/WCNC.2018.8377343).

[10] G. Zhang *et al.*, "Fair Task Offloading Among Fog Nodes in Fog Computing Networks", *IEEE International Conference on Communications (ICC)*, Kansas City, USA, 2018 (https://doi.org/10.1109/ICC.2018.8422316).

[11] L. Tan, Z. Kuang, L. Zhao, and A. Liu, "Energy-Efficient Joint Task Offloading and Resource Allocation in OFDMA-Based Collaborative Edge Computing", in *IEEE Transactions on Wireless Communications*, vol. 21, no. 3, pp. 1960–1972, 2022 (https://doi.org/10.1109/TWC.2021.3108641).

[12] X. Chen *et al.*, "Multi-tenant Cross-slice Resource Orchestration: A Deep Reinforcement Learning Approach", *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 10, pp. 2377–2392, 2022 (https://doi.org/10.1109/JSAC.2019.2933893).

[13] J. Kim *et al.*, "Joint Optimization of Signal Design and Resource Allocation in Wireless D2D Edge Computing", *IEEE INFOCOM 2020 – IEEE Conference on Computer Communications*, Toronto, Canada, 2020 (https://doi.org/10.1109/INFOCOM41043.2020.9155510).

[14] Y. Mao, J. Zhang, S.H. Song, and K.B. Letaief, "Stochastic Joint Radio and Computational Resource Management for Multi-user Mobile-edge Computing Systems", IEEE Transactions on Wireless Communications, vol. 16, no. 9, pp. 5994–6009, 2017 (https://doi.org/10.1109/TWC.2017.2717986).

Fatimah Azeez Rawdhan
Department of Computer Engineering
https://orcid.org/0009-0006-8943-2759
E-mail: fatimah.azeez@uomustansiriyah.edu.iq
Mustansiriyah University, Baghdad, Iraq
https://uomustansiriyah.edu.iq