Energy-Efficient Task Offloading in MEC-Cloud

Hang Liu[1], Haoran Ma[1], Qiang Liu[1], Zaixing Sun[2], Chonglin Gu[1(✉)], and Hejiao Huang[1(✉)]

[1] Harbin Institute of Technology, Shenzhen 518071, China
{guchonglin,huanghejiao}@hit.edu.cn
[2] Peng Cheng Laboratory, Shenzhen 518055, China

Abstract. Mobile edge computing (MEC) is a promising paradigm that brings communication and computing resources closer to mobile users, thereby enabling low-latency and high-quality services. However, the low energy efficiency of edge servers (ESs) remains a critical issue, as they are often kept continuously active to handle irregular and unpredictable task arrivals. In fact, tasks can be offloaded from one ES to another ES or to the cloud, so that the idle ESs can be sleep down to save energy. Unfortunately, existing work often ignores the delay caused by the sleep operations of ESs, which limits their applicability in real-world scenarios. In this paper, we investigate the problem of energy-efficient task offloading in MEC-cloud. Tasks can be dynamically offloaded either among ESs or from ESs to the cloud, depending on the system state. To achieve energy savings, we consider both wake-up and sleep-down delays of the ESs. To address this problem, we first formulate it as a Markov Decision Process (MDP) and propose a Soft Actor-Critic (SAC)-based algorithm to dynamically offload the tasks from ESs to ESs/cloud. Then, we design a heuristic algorithm to wake up or sleep down the ESs for energy saving. Simulation results demonstrate that our method significantly reduce the time-average energy consumption of ESs as well as the time-average transmission and computation delay of all tasks.

Keywords: MEC-cloud · offloading · energy-efficient · sleep

1 Introduction

In recent years, mobile applications have increasingly relied on high computational power to support low-latency and high-quality services. Traditional cloud computing alone cannot meet these requirements due to significant transmission delays. To address this issue, mobile edge computing (MEC) has emerged as a new computing paradigm. It moves cloud data center functions to the network edge near users, forming a CETCN architecture. By offloading computational tasks from mobile devices to base stations (BSs), MEC leverages edge computing and storage resources, thereby enhancing user experience [10]. However, task offloading increases computing energy consumption, making the reduction of edge server (ES) energy usage a major challenge in MEC networks [2].

D.-S. Huang et al. (Eds.): ICIC 2025, LNCS 15859, pp. 295–307, 2025.
https://doi.org/10.1007/978-981-96-9812-7_25

In typical MEC architectures, servers are deployed at base stations to serve mobile users through task offloading. Related research can be categorized into three main directions:

- *Task offloading without ES-ES collaboration:* Wu et al. [9] optimize multi-objective like energy consumption and latency in heterogeneous MEC systems by considering the characteristics of different application types during task offloading. Alhartomi et al. [1] conserve mobile device battery energy through renewable energy harvesting and edge offloading. Chen et al. [4] propose FUNOff for dynamic function-level application offloading in MEC, to reduce response time. While most existing studies focus on offloading tasks to ESs to overcome mobile device limitations, dynamic workloads in real scenarios make ES–ES collaboration increasingly essential [11, 14].
- *Task offloading with ES-ES collaboration:* Zhang et al. [16] propose a DT-driven intelligent task offloading framework for collaborative ESs, where DT maps the physical system into a virtual environment to optimize economic efficiency. Chen et al. [5] achieve load balancing by offloading tasks from overloaded ESs to underutilized ones, thereby minimizing system delay. Maleki et al. [17] consider a three-layer MEC system for resource allocation, partitioning mobile user workloads to minimize average cost.
- *Energy saving in task offloading:* Some studies focus on reducing ES energy consumption in MEC using DVFS or server sleep. Panda et al. [12] propose a deep reinforcement learning and DVFS scheme for IoT device energy consumption. However, DVFS is ineffective for reducing static power consumption, which can instead be addressed by ACPI through controlling processor core states [13]. Amer et al. [2] propose dual-threshold core power state control for energy conservation. Wang et al. [15] study ES offloading and sleep scheduling, but neglect the delay caused by wake-up and sleep transitions, limiting practical applicability.

In this paper, we study energy-efficient task offloading in the MEC-cloud network. We consider a small cell network with ESs at SBSs, multiple SBSs, and a cloud data center. ESs can collaborate via LAN or offload to the cloud via WAN, and idle ESs can sleep. Our model accounts for sleep/wake transition delay and power. In 5G networks, unpredictable task workloads make offloading and ES sleep decisions particularly challenging. Although putting ESs to sleep can save energy, it may lead to resource shortages and increased transmission delays. Moreover, long-term energy consumption is time-coupled due to the sequential nature of decision-making. To address these challenges, we formulate the problem as a Markov Decision Process (MDP) and propose a Soft Actor-Critic (SAC)-based reinforcement learning algorithm for task offloading, along with a heuristic algorithm for ES sleep and wake-up management. The main contributions of this paper are:

- We formulate a long-term energy consumption optimization problem that incorporates ES–ES/cloud collaborative offloading, while explicitly account for the non-negligible transition delays of wake-up and sleep-down switching.

- To solve this problem, we propose a SAC-based reinforcement learning approach for task offloading and develop an effective heuristic algorithm for wake-up and sleep scheduling decisions based on dual thresholds and historical task arrival rates.
- Extensive simulations are conducted to evaluate the proposed algorithms. Compared with existing methods, our approach significantly reduces the time-average energy consumption of ESs, as well as the average transmission and computation delays of tasks.

The remainder of this paper is organized as follows. In Sect. 2, we establish the system model and formulate the problem of long-term energy consumption optimization. Section 3 details the task offloading and sleep scheduling algorithms. Section 4 presents and analyzes the experimental results. Finally, Sect. 5 concludes the paper.

2 System Model

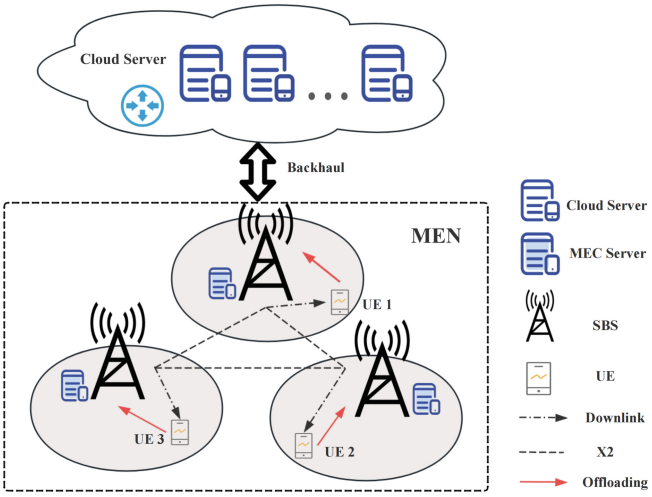


Fig. 1. System model

The overall system architecture is depicted in Fig. 1. In this paper, we consider an edge network comprising a set of SBSs $\mathcal{N} = \{1, \ldots, N\}$ to provide communication and computing services, each equipped with an ES. User equipments can offload communication or computation tasks to these edge nodes. Tasks can be transmitted between edge nodes via LAN [6]. Additionally, the edge network is connected to the cloud data center, which can further offload tasks to more powerful cloud servers for processing. The energy consumption of the MEC can be saved through sleeping down the idle ESs.

2.1 Task Arrival Model

We consider that offloading decisions are made on a per-time-slot basis. At the beginning of each time slot, user tasks arrive at the edge network. Let λ_i^t represent the task arrival

rate (*measured in Mbit*) on the ES i, which is assumed to follow Poisson distribution with a mean of λ_p [9].

2.2 ES Offload Model

ESs within the same LAN can offload tasks to each other. High-load ESs can offload tasks to low-load ESs to achieve load balancing, thereby maximum the utilization of computation capability [17]. Low-load ESs can offload tasks to other ESs, allowing them to enter sleep mode and save energy. When the workload on the entire edge network is relatively high, ESs can offload tasks to the cloud data center to alleviate computational pressure. Let $\theta_i^t = \theta_{i0}^t, \theta_{i1}^t, ..., \theta_{in}^t (i \in (1, 2, ..., n))$ denote the offloading decisions of ES i in time slot t, where $\theta_{i,j}^t$ represents the workload to be offloaded from ES i to ES j ($j \neq 0$) or to the cloud ($j = 0$) in time slot t. When $i = = j$, $\theta_{i,j}^t$ represents the workload left on ES i. When an ES i is sleeping, all of its arriving tasks must be offloaded to other ESs or cloud. Thus, the offloading decision of the entire system in time slot t can be represented as $\theta^t = \left\{\theta_1^t, \theta_{2,}^t ..., \theta_n^t\right\}$. Let v_i^t denote the workload to be processed by ES i after offloading in time slot t. It is composed of the workload offloaded to it from other ESs and that arrived from UE. Therefore, it can be represented as: $v_i^t = \sum_{j=1}^{N} \theta_{ji}^t$.

2.3 Power Consumption Model

The power consumption of the system primarily consists of following two parts:

1) *Communication power consumption*

Let $\sigma_i^t = c\lambda_i^t$(c is a constant) denote the downlink traffic of the SBS i, and let $P_{tx,i}$ denote the downlink transmission power of the ES i and r_0 is the minimum transmission rate required to meet the quality of service. The communication power consumption of the SBS i can be represented as: $P_{tx,i} \frac{\sigma_i^t}{r_0}$.

2) *ES power consumption*

The computing power consumption of ES i in the active state can be expressed as:

$$P_0 = \alpha P_{peak} + (1 - \alpha) P_{peak} u_i$$

where P_{peak} is the peak power that ES is fully utilized, and α is the proportion of static power to peak power. u_i is the CPU utilization. Specifically, $u_i^t = v_i^t / \beta_i$, where v_i^t is the workload of the ES i after offloading in time slot t, and β_i is the maximal serving rate of the ES i. According to ACPI, multiple sleep states are possible for an ES. In our model, we select one sleep state k. Let P_k, $P_{0\to k}$ and $P_{k\to 0}$ denote the power of an ES in sleep state k, the transitioning power from sleep state k to active state, and that from active state to sleep state k, respectively.

2.4 Delay Model

The delay of tasks in the entire system typically includes following four parts:

1) The transmission delay from UEs to SBS can be modeled as an M/G/1 queue [15]. Based on queuing theory, the delay corresponding to each 1 Mbit workload can be expressed as:

$$D_{tx,i}^{t} = \frac{1}{r_0}\left(1 + \frac{\rho_i^t}{2\left(1-\rho_i^t\right)}\right)$$

where $\rho_i^t = \left(\mathbb{E}\left(\lambda_i^t\right)/r_0\right)$, r_0 is the minimum transmission rate that meets the quality of service.

2) The offloading delay for transmitting tasks between ESs can be modeled as an M/M/1 queue [15]. Based on queuing theory, the delay corresponding to each 1 Mbit workload can be expressed as:

$$D_{LAN}^{t} = \frac{\tau}{1-\tau\varphi^t}, \varphi^t < \frac{1}{\tau}$$

where τ is the average transmission time required for a unit workload to be transmitted back and forth without blocking the LAN. φ^t is the sum of all loads that need to be transmitted in the LAN.

3) The offloading delay for transmitting tasks from ES to the cloud can also be modeled as an M/M/1 queue [15]. Thus, we have:

$$D_{bh}^{t} = \frac{\gamma\tau}{1-\gamma\tau\psi^t}, \psi^t < \frac{1}{\gamma\tau}$$

where $\psi^t = \sum_{i=1}^{N}\theta_{i0}^t$ is the sum of all ES loads that need to be transmitted to the cloud. The average transmission time required per unit workload to be transmitted to the cloud is γ times that to other ESs.

4) Computation delay for tasks on ESs: It is assumed that the computation delay at the cloud server can be negligible [15]. Therefore, we focus solely on the computation delay of tasks processed by ES. We model each ES as M/M/1 queuing system [15]. The computation delay at ES i in time slot t can be expressed as:

$$D_{c,i}^{t} = \frac{1}{\beta_i^t - v_i^t}$$

where β_i^t is the maximal serving rate of ES i, and v_i^t is the workload to be processed by ES i after offloading.

2.5 Sleep Control of ES

Let $S^t = \left\{S_1^t, S_2^t, ..., S_n^t\right\}$ denote the sleep status of each ES in time slots t, where S_n^t represents the sleep state of ES n. When $S_n^t = 0$, it indicates that ES n is active in time slot t; otherwise, it is in a sleep state or a transition state.

2.6 Problem Formulation

In this paper, we consider a network of MEC-cloud collaboration. Our objective is to minimize the time-average energy consumption of ESs while simultaneously reducing the time-average transmission and computation delay of all tasks. The decisions include task offloading, as well as wake- up and sleep-down operations in each time slot. The problem formulation is given as follows:

$$\textbf{Minimize}: \quad \sum_{t=1}^{T} \sum_{i=1}^{N} \left(P_{tx,i} \frac{\sigma_i^t}{r_0} + P_i^t \right) \tag{1}$$

$$\textbf{Subject to}: \quad D_i^t \leq D_{i,max}^t$$

$$\theta_{i,j}^t \geq 0 \tag{2}$$

$$\lambda_i^t = \sum_{j=0}^{N} \theta_{ij}^t \tag{3}$$

$$\sum_{j=1}^{N} \theta_{ji}^t \leq \beta_i \tag{4}$$

$$S_i^t \in \{0, 1, 2, 3\} \tag{5}$$

$$\forall i \in \{1, 2, ..., n\}, \forall j \in \{0, 1, ..., n\}$$

The objective of this problem is to minimize the long-term time-average energy consumption of ESs. Constraint (1) ensures that in each time slot, the delay of an ES i should meets the requirement of service quality for users, where D_i^t is the total delay of ES i in time slot t. This total delay is composed of four parts as mentioned above, so $D_i^t = \lambda_i^t D_{tx,i}^t + \varphi_i^t D_{LAN,i}^t + \theta_{i0}^t D_{bh,i}^t + v_i^t D_{c,i}^t$. Constraint eq. (2) stipulates that the offloading decisions should be non-negative. Constraint eq. (3) ensures that the total workload arriving at ES i in t equals the sum of the workloads offloaded to other ESs or the cloud and that to be processed by itself. Constraint eq. (4) ensures that after offloading, the workload to be processed on ES i does not exceed its maximal serving rate. Constraint eq. (5) specifies that the ES state should be one of the following states: active state, sleep state, wake-up transition or sleep-down transition. It should be noted that decisions made in a time slot may affect that in the subsequent time slots (e.g., an ES in transition between active and sleep states cannot change states immediately and may remain in transition for a period).

Algorithm 1: SAC-based offloading training

Randomly initialize parameters: Ω, $\hat{\Omega}$, $\emptyset$
for *episode* ← *1 to NumEpisode* **do**
  Reset environment;
  for *time* ← *0 to T* **do**
    Obtain *a(t)* from Actor network based on *s(t)*;
    Obtain *s(t+ 1)*, offloading decisions, total energy consumption, total delay, reward *r(t)*;
    Store [*s(t), a(t), r(t), s(t + 1)*] in the experience buffer;
    Randomly sample batches of empirical samples from the empirical buffer;
    Calculate the loss function and update the critic network parameters Ω, actor network parameters $\emptyset$, and soft update critie target network parameters $\hat{\Omega}$, respectively.

3 Algorithm Design

In this section, we first introduce our SAC-based reinforcement learning algorithm for task offloading decisions. We then provide details of our heuristic algorithm for sleep scheduling of ESs in our system.

3.1 SAC-Based Reinforcement Learning for Task Offloading

The training process is described in Algorithm and the various elements of the SAC-based offloading algorithm are defined as follows:

1) Environment States: For the entire system, each ES will have a task arrival rate at the beginning of each time slot *t,* which serves as the environment states. Thus, the environment state is $s^t = \lambda^t = \left\{\lambda_1^t, ..., \lambda_n^t\right\}$.
2) Actions: The agent will adopt an action based on the state of the environment, expressed as actions: $a^t = \{a_0^t, a_1^t, ..., a_n^t\}$. It is the proportion of the workload (against the total arriving workload in time slot t) in cloud and that in each ES after offloading, such that $a_0^t + a_1^t + ... + a_n^t = 1$. Based on this proportion, the actual offloading decisions regarding the workload to be offloaded from ESs to ESs/cloud can easily be calculated as follows:

   We first classify ESs into three types: a) The ESs that do not need to offload tasks to/from other ESs or the cloud. b) ESs that need to offload tasks to other ESs/cloud. c) ESs that need to receive offloading tasks from other ESs. After classification, offload the tasks from the ESs in the second type to the third type. If there are still remaining tasks, they are offloaded to the cloud. For the ESs in sleep state, all incoming tasks should first be offloaded to the cloud if not exceeding a certain threshold; otherwise, the remaining tasks are offloaded to other ESs. To ensure load balancing, tasks are

always offloaded to the ESs with the least workload, and the number after offloading should not exceed that of the ES with second least number. Repeat this process until all offloading is complete.

3) Reward: After offloading, the system reward is determined. If the delay constraint is violated ($D_i^t > D_{i,max}^t$), the reward is $-\sum_{i=1}^{N} E_{max}$, where E_{max} is the maximum energy consumption budget of an ES. Otherwise, the reward is the negative of the energy consumption $-\sum_{i=1}^{N} E_i^t$.

3.2 Heuristic Based Sleep Scheduling Algorithm

The sleep scheduling determines whether to sleep down or wake up an ES, and which ES to select. Note that, the sleep scheduling algorithm is executed every 5 time slots. Considering the uneven workload, we avoid making sudden changes to the ESs. In each scheduling period (5 time slots), we sleep down or wake up at most one ES. Algorithm 2 presents our proposed dual-threshold algorithm to control sleep-down and wake-up operations.

We first initialize the sleep-down and wake-up decisions, denoted as X and Y, respectively. In current time slot t, we count the total number of active ESs *activeNum* and their total serving rate β_{total} (here $S_i^t = 0$ means ES i is in active state). If the total serving rate is less than a threshold $\beta_{threshold}$, we do not perform any sleep-down operations. Next, we calculate the average arrival rate λ_{avg} of tasks over the most recent *slotNum* time slots. R is an array recording the relative resource utilization in each time slot. In time slot t, it can be calculated using λ_{avg} divided by β_{total}. We then traverse back *slotNum* time slots from the current time slot. If the relative resource utilization in any of these time slots exceeds the sleep-down threshold $Sleep_{threshold}$, we do not perform any sleep-down operations. Additionally, if the total number of active ESs is less than a threshold $activeNum_{threhold}$, we also do not perform any sleep-down operation. If X is true, sleep down the ES with the lowest total task arrival rate over the past $T_{observed}$ (here it is set to be 30 time slots) time slots. Similarly, we traverse back *slotNum* time slots from the current time slot. If the relative resource utilization in any of these time slots is less than a wake-up threshold $Active_{threshold}$ (it is set to be 2), we do not perform any wake-up operations. Meanwhile, if any ES is being wake up in time slot t, we also do not perform any wake-up operation. If Y is true, wake up the ES with the highest total task arrival rate in the past $T_{observed}$ time slots.

4 Performance Evaluation

4.1 Simulation Setup

In our MEC-cloud network, we assume there are 10 SBSs connected through LAN, and one cloud data center. We consider all the tasks are of web request workloads with uncertain pattern, arriving at each ES every second. Offloading decisions are made on a per-second time slot basis, while wake-up and sleep-down operations are performed every 5 time slots. τ is set as 0.002 s. β is set as 30Mbit/S. γ is set as 10. α is set as 0.5.

r_0 is set as 30 MB/s. T is set as 5000. λ_p is set as 25 Mbit. h_i is set as exp (1). W is set as 20 MHz. σ^2 is set as 10^{-10} W/Hz. Since ACPI can provide multi-sleep modes, we only select the most energy-efficient one as our sleep mode, provided its delay is not excessively large. The power and delay characteristics of different sleep modes are listed in Table 1.

Table 2 shows the time-average energy consumption and delay for different sleep states S_1–S_5. Generally, deeper sleep modes achieve higher energy efficiency at the expense of longer wake-up times. However, as shown in Table 2, S_5 consumes the least energy while maintaining similar delay levels to other sleep modes, despite being the deepest sleep mode with the largest wake-up delay. The primary reason is that during the wake-up/sleep-down transitions from/to S_5, ES can not process any computation tasks but instead offload the arriving tasks to other ESs or to the cloud. This collaboration between ESs and the cloud ensures that the wake-up and sleep-down transitions have minimal impact on delay while significantly reducing energy consumption. Therefore, we select S_5 as our sleep mode in the following experiments for energy savings.

To evaluate the performance of our method, we compare it with the following state-of-the-art algorithms: *Reactive, SoftReactive* [7], *Classification* [15], *Priority* [3] and *Proportion* [17].

Table 1. Power and delay for different sleep modes.

State	Power(W)	State	Power(W)	Delay(s)
Peak	471	$S_0 \rightarrow S_1$	302	2
Idle	270	$S_0 \rightarrow S_2$	285	3
S_1	245	$S_0 \rightarrow S_3$	203	4
S_2	209	$S_0 \rightarrow S_4$	130	9
S_3	160	$S_0 \rightarrow S_5$	66	17
S_4	95	$S_1 \rightarrow S_0$	309	2
S_5	12	$S_2 \rightarrow S_0$	292	6
–	-	$S_3 \rightarrow S_0$	212	25
–	-	$S_4 \rightarrow S_0$	154	48
-	-	$S_5 \rightarrow S_0$	102	75

4.2 Result Analysis

1) *The impact of traffic composition*: Fig. 2 illustrates the energy consumption and delay for different task proportions under varying ES idle power ratios α. Typically, α ranges from 0.5 to 0.7, corresponding to (a1)(b1), (a2)(b2), and (a3)(b3) in the figure, respectively. As observed, energy consumption increases with α for all algorithms except *Classification*, which will be discussed later. The workload of mobile devices is divided into computation and communication workloads, and different compositions

Table 2. The time-average delay and energy consumption of different sleep states.

State	Delay (S)	Energy (J)
S_1	11.65	2522.28
S_2	11.67	2428.07
S_3	11.65	2295.49
S_4	11.65	2122.83
S_5	11.68	1905.85

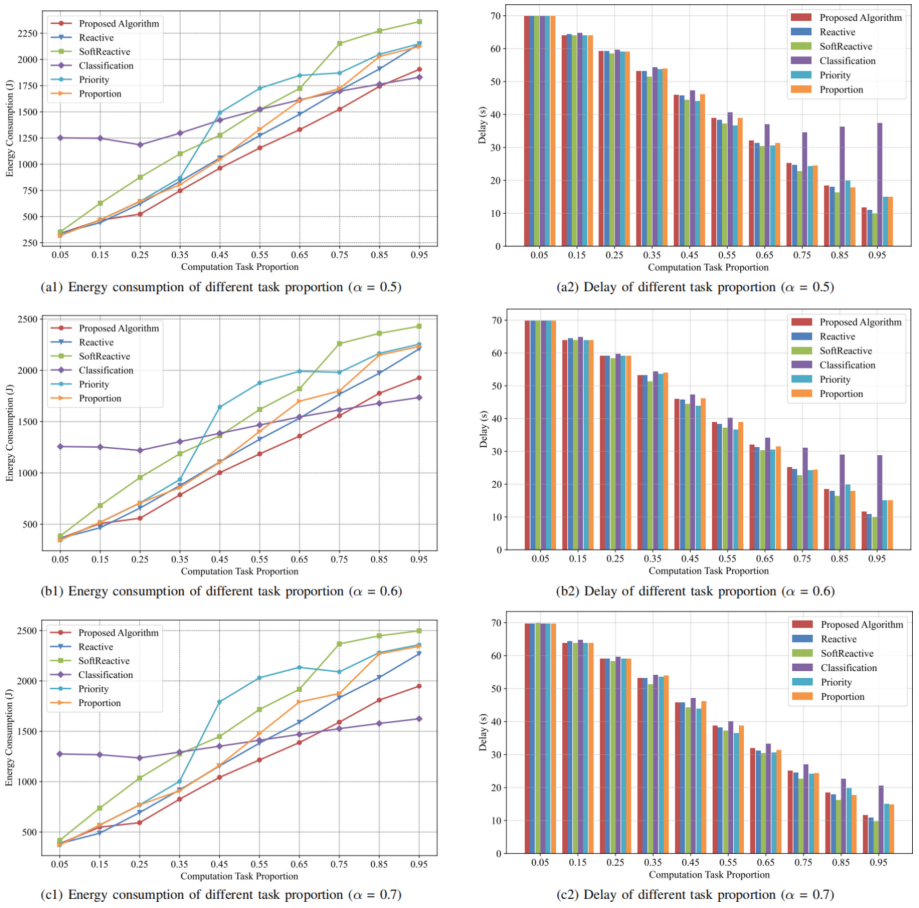
*$\alpha = 0.5, Sleep_{threshold} = 1, Active_{threshold} = 1.1, \gamma = 0.95$.

of these workloads affect energy consumption and latency to varying extents. In Fig. 2, we also investigate the effects of different workload compositions on energy consumption and delay. Assume the proportion of computation tasks is $\xi \in (0, 1)$, then the proportion of communication tasks is $1 - \xi$. In literature [15], it is assumed that the energy consumption and communication delay generated by unit flow of communication tasks are fixed with parameters $D_{comm} = 0.3$ and $E_{comm} = 0.3$. Thus, the energy and delay for communication tasks can be easily calculated.

In Fig. 2, energy consumption increases with the proportion of computation tasks as they consume more computing energy and lead to fewer ESs sleeping, with active ESs consuming significant basic power. Among comparison algorithms, *Classification* is unstable, having high energy consumption when the proportion of computation tasks is low. *SoftReactive* has low delay but high energy consumption.

Furthermore, Fig. 2 shows that the delay decreases as the proportion of computation tasks (0.05 - 0.95) mainly due to the reduction in communication tasks which have a higher delay cost per unit flow. Although computation tasks may increase offloading delay, it's offset by the low proportion of communication tasks. Our proposed algorithm has slightly lower delay than *Reactive* and *SoftReactive*. *Reactive*, *SoftReactive*, *Priority* and *Proportion* have similar delays. *Classification* has similar delay to others when the proportion is low but its delay rises sharply as the proportion increases because it uses a single threshold for sleep/wake decisions and can't adapt to workload fluctuations. In summary, our method can save energy and maintain low delay through intelligent task offloading and ES sleeping.

2) *The impact of sleep operation:* Tables 3 and 4 present the energy consumption and delay comparisons between our proposed algorithm and the scenario where ESs do not utilize sleep operations. When $\xi = 0.05$, the maximum energy saving ratio reaches 78.1%. Even when $\xi = 0.95$, the energy saving ratio remains significant at 19.3%. Notably, the delay introduced by sleep-down operations does not increase significantly, regardless of the workload composition. Therefore, enabling ESs to enter sleep mode for energy savings while facilitating task offloading between ESs and the cloud is crucial for providing high-quality services.



(a1) Energy consumption of different task proportion (α = 0.5)

(a2) Delay of different task proportion (α = 0.5)

(b1) Energy consumption of different task proportion (α = 0.6)

(b2) Delay of different task proportion (α = 0.6)

(c1) Energy consumption of different task proportion (α = 0.7)

(c2) Delay of different task proportion (α = 0.7)

Fig. 2. Energy consumption and delay of different task proportion under different α.

Table 3. Comparisons of energy consumption (J) of our proposed algorithm with that of ESs without sleeping.

ξ	0.05	0.15	0.25	0.35	0.45	0.55	0.65	0.75	0.85	0.95
P	340.05	465.42	523.64	746.89	962.66	1155.56	1330.66	1525.58	1743.81	1905.85
N	1555.36	1652.55	1740.22	1829.52	1919.01	2007.03	2096.23	2185.29	2273.22	2360.86
R	78.1%	71.8%	70%	59.2%	49.8%	42.4%	36.5%	30.2%	23.3%	19.3%

* P = Proposed Algorithm, N = No Sleep, R = Energy Saving Ratio

Table 4. Comparisons of delay (s) of our proposed algorithm with that of ESs without sleeping.

ξ	0.05	0.15	0.25	0.35	0.45	0.55	0.65	0.75	0.85	0.95
Proposed Algorithm	69.79	63.92	59.14	53.17	45.86	38.85	32.01	25.20	18.44	11.68
No Sleep	70.09	62.92	55.79	48.71	41.70	34.87	28.46	22.42	16.28	9.86

5 Conclusion

In this paper, we investigate energy-efficient offloading in the MEC-cloud environment by managing ESs' wake and sleep states. We propose a SAC-based reinforcement learning algorithm for task offloading between ESs and the cloud, along with a dual-threshold heuristic strategy for transitioning idle servers into sleep mode. Despite the delays introduced by server state transitions, our approach maintains low-latency and high-quality service through collaborative offloading among ESs and the cloud. Simulation results demonstrate that our method achieves at least a 19.3% reduction in energy consumption compared to the baseline where all ESs remain active, while still maintaining low-latency and high-quality service.

Acknowledgement. This work is financially supported by Shenzhen Science and Technology Program under Grant No. GXWD20220817124827001 and No. JCYJ20210324132406016.

References

1. Alhartomi, M., Salh, A., Audah, L., Alzahrani, S., Alzahmi, A.: Enhancing sustainable edge computing offloading via renewable prediction for energy harvesting. IEEE Access **12**, 74011–74023 (2024). https://doi.org/10.1109/ACCESS.2024.3404222
2. Amer, A.A., Talkhan, I.E., Ismail, T.: Optimal power consumption on distributed edge services under non-uniform traffic with dual threshold sleep/active control. In: Novel Intelligent and Leading Emerging Sciences Conference (NILES), pp. 63–66. IEEE (2021)
3. Amer, A.A., Ismail, T.: Cooperative MEC with non-uniform user density: resource planning and state control optimization. IEEE Commun. Lett. **26**(11), 2700–2704 (2022). https://doi.org/10.1109/LCOMM.2022.3198515
4. Chen, X., Li, M., Zhong, H., Chen, X., Ma, Y., Hsu, C.H.: FunOff: offloading applications at function granularity for mobile edge computing. IEEE Trans. Mob. Comput. **23**(2), 1717–1734 (2023)
5. Chen, L., Zhou, S., Xu, J.: Computation peer offloading for energy-constrained mobile edge computing in small-cell networks. IEEE/ACM Trans. Netw. **26**(4), 1619–1632 (2018). https://doi.org/10.1109/TNET.2018.2841758
6. Chen L, Xu J, Z.S.: Computation peer offloading in mobile edge computing with energy budgets. In: IEEE Global Communications Conference, pp. 1–6. IEEE (2017)
7. Gandhi, A., Harchol-Balter, M., Raghunathan, R., Kozuch, M.A.: Autoscale: dynamic, robust capacity management for multi-tier data centers. ACM Trans. Comput. Syst. **30**(4), 14 (2012)
8. Wu, H.: Multi-objective decision-making for mobile cloud offloading: a survey. IEEE Access **6**, 3962–3976 (2018). https://doi.org/10.1109/ACCESS.2018.2791504

9. Liu, L., Chang, Z., G.X.E.A.: Multi-objective optimization for computation offloading in mobile-edge computing. In: IEEE Symposium on Computers and Communications (ISCC), pp. 832–837. IEEE (2017)
10. Mahmoud, H.H.H. Amer, A.A., Ismail, T.: 6G: a comprehensive survey on technologies, applications, challenges, and research problems. Trans. Emerg. Telecommun. Technol. **32**(4) (2021)
11. Maleki, E.F., Mashayekhy, L., Nabavinejad, S.M.: Mobility-aware computation offloading in edge computing using machine learning. IEEE Trans. Mob. Comput. **22**(1), 328–340 (2021)
12. Panda, S.K., Lin, M., Zhou, T.: Energy-efficient computation offloading with DVFSs using deep reinforcement learning for time-critical IoT applications in edge computing. IEEE Internet Things J. **10**(8), 6611–6621 (2022)
13. Schöne, R., Molka, D., Werner, M.: Wake-up latencies for processor idle states on current X86 processors. Comput. Sci.-Res. Dev. **30**, 219–227 (2015)
14. Tang, F., Fadlullah, Z.M., Mao, B., Kato, N.: An intelligent traffic load prediction-based adaptive channel assignment algorithm in SDN-IoT: a deep learning approach. IEEE Internet Things J. **5**(6), 5141–5154 (2018). https://doi.org/10.1109/JIOT.2018.2838574
15. Wang, S., Zhang, X., Yan, Z., Wenbo, W.: Cooperative edge computing with sleep control under nonuniform traffic in mobile edge networks. IEEE Internet Things J. **6**(3), 4295–4306 (2019)
16. Zhang, Y., Hu, J., Min, G.: Digital twin-driven intelligent task offloading for collaborative mobile edge computing. IEEE J. Select. Areas Commun. **41**(10), 3034–3045 (2023)
17. Zhang, W., Zhang, G., Mao, S.: Joint parallel offloading and load balancing for cooperative-MEC systems with delay constraints. IEEE Trans. Veh. Technol. **71**(4), 4249–4263 (2022). https://doi.org/10.1109/TVT.2022.3143425