Integrating Future Smart Home Operation Platform With Demand Side Management via Deep Reinforcement Learning

Tan Li, *Student Member, IEEE*, Yuanzhang Xiao, *Member, IEEE*, and Linqi Song, *Member, IEEE*

***Abstract*—Residential demand side management (DSM) is a promising technique in smart grids to improve the power system robustness and to reduce the energy cost. However, the ongoing paradigm shift of computation, such as mobile edge computing for smart home, poses a big challenge to residential DSM. Therefore, it is important to schedule the new smart home computing tasks and traditional DSM in a smart way. In this paper, we investigate an integrated home energy management system (HEMS) that participates in a DSM program and implements smart home computation tasks by offloading tasks with the help of a Smart Home Operation Platform (SHOP). The goal of HEMS is to maximize the user's expected total reward, defined as the reward from completing computing tasks minus the cost of energy consumption, execution delay, running the SHOP servers, and the penalty of violating the DSM requirements. We solve this task scheduling based DSM problem using a deep reinforcement learning method. The DSM program considered in this paper requires the household to reduce a certain amount of energy consumption within a specified time window, which, in stark contrast to the well-studied real-time pricing, results in a long-term temporal interdependence and thus a high-dimensional state space in our formulated problem. To address this challenge, we use the Deep Deterministic Policy Gradient (DDPG) method to characterize the high-dimensional state space and action space, which uses deep neural networks to estimate the state and to generate the action. Experimental results show that our proposed method achieves better performance gains over reasonable baselines.**

***Index Terms*—Demand side management, edge computing, task offloading, deep reinforcement learning.**

I. Introduction

LEVERAGING bidirectional information and power flows between the utilities and consumers, smart grids are deployed to diversify the power supply, to reduce green house gas emissions, and to improve the power efficiency [2], [3]. In smart grids, demand side management (DSM) is an important mechanism which conducts efficient customer-side energy management to reduce the peak-hour energy supply of the power grid and hence the operational cost in the power grid. As a result, the DSM program has been widely adopted by residential customers [4].

Residential DSM strategies motivate consumers to re-shape their load profiles and limit peak energy demands through smart meters based on real-time pricing or incentives [5]. However, it is facing challenges under the ongoing paradigm shift of intelligent computation. A rapidly increasing number of Internet of Things (IoT) devices are deployed, such as smart appliances (e.g., smart TV and smart refrigerators), healthcare monitoring devices, and surveillance networks [6], [7]. These devices often require high-performance computation to handle complex computing tasks, such as voice recognition and image processing. Above tasks are preferred to be done on or near the devices rather than being offloaded to the remote cloud servers due to the latency requirement and privacy concerns [8]. However, the surge in local device computing will definitely increase the household energy consumption and make it more challenging for residential DSM. Smart Home Operation Platform (SHOP), offering the storage and computation resources at the edge computing server, is considered as a promising solution to fix the weakness of long-distance cloud computing [9] as well as the poor computing capacity of local equipment. In future smart home, SHOP will be deployed as a key component to support various intelligent household applications. It allows users to offload some computation-intensive tasks to the SHOP servers for execution. Since the SHOP server is often deployed closer to the edge and trustworthy, it will not introduce as large transmission delay as remote cloud computing or reveal user's privacy. Considering the restriction of residential DSM, we can choose to offload some non-urgent computing tasks to the SHOP server during peak hours at the expense of introducing some transmission delays and server payment compared to local computing. Therefore, it is important to jointly consider DSM and SHOP in an integrated framework. In other words, we have to balance a trade-off between satisfying user's requirements, complying with the utility company's DSM restrictions, and reducing user's energy consumption cost. In this paper, we study for the first time how to perform DSM in an integrated network with SHOP by deciding how to offload computational tasks. We consider an

Manuscript received December 20, 2020; revised March 17, 2021; accepted April 11, 2021. Date of publication April 19, 2021; date of current version May 20, 2021. This work was supported in part by the Hong Kong RGC under Grant ECS 21212419; in part by the Guangdong Basic and Applied Basic Research Foundation through Key Project under Grant 2019B1515120032; in part by the City University of Hong Kong SRG-Fd under Grant 7005561; and in part by NSF IIP under Grant 1822213. This article was presented in part at the 2019 IEEE International Conference on Communications, Control, and Computing Technologies for Smart Grids (SmartGridComm), Beijing, China. The editor coordinating the review of this article was B. Kantarci. *(Corresponding authors: Tan Li; Yuanzhang Xiao; Linqi Song.)*

Tan Li and Linqi Song are with the Department of Computer Science, City University of Hong Kong, Hong Kong, and also with the City University of Hong Kong Shenzhen Research Institute, Shenzhen, China (e-mail: tanli6-c@my.cityu.edu.hk; linqi.song@cityu.edu.hk).

Yuanzhang Xiao is with the Department of Electrical Engineering, University of Hawaii at Mānoa, Honolulu, HI 96822 USA (e-mail: yxiao8@hawaii.edu).

Digital Object Identifier 10.1109/TGCN.2021.3073979

incentive-based DSM program, which requires the household to reduce a certain amount of energy consumption within a specified time window. However, the traditional DSM program may not be widely accepted by consumers [10] since power shifting often leads to poor user satisfaction. Here we consider a different scheme from power shifting. The integrated home energy management system (HEMS) allows users do not need to (or slightly) perform power load shifting during the peak hours, instead they could offload tasks to the SHOP in order to realize DSM. This framework was first presented in our previous work [1]. The SHOP is equipped with more powerful computing capacity compared with local devices and is considered to be reliable. Users can choose to allocate the computational task load among the local devices and the SHOP, where the computation on local devices will result in more power consumption and the computation in the SHOP will lead to some transmission delay and monetary cost. This task allocation problem can be formulated as an interactive process between the HEMS and smart home environment. In the environment, smart appliances randomly generate tasks with different features. For example, face recognition task requires a large amount of calculation (usually processed by the neural network inference) and a short response time, while the heart rate monitoring requires less computation and no need for rapid response. When computing loads are allocated by the HEMS after observing the task features, the environment will react with the energy consumption, execution time and whether it violates the requirements of DSM. Then the HEMS needs to further adjust the task allocation strategy based on the feedback. In smart home, HEMS will continuously observe new tasks, such as tasks generated from a new household appliance, thus online learning is suitable for solving this problem. Our goal is to design an efficient online learning algorithm to cope with complex smart home environment and obtain the optimal task allocation policy. Our main contributions are summarized as follows.

1) We investigate an integrated HEMS framework who participates in a DSM program and can realize task offloading with the help of the SHOP. We analyze the interaction between the HEMS and the smart home environment by dividing the system into local computing, SHOP computing, battery and demand response model. The HEMS aims to maximize the user's expected total reward induced by these models by deciding how to offload computational tasks and charge/discharge the battery.

2) We formulate the above reward maximum problem as a Markov Decision Process (MDP) and use a deep reinforcement learning approach to solve it. The proposed method is able to overcome the challenges of a high-dimensional state space and a large continuous action space in our model. The key is to use the critic network to estimate the action-value function and the actor network to output a parameterized policy.

3) Experimental results show that under the same environmental setting, our algorithm outperforms other baselines. In addition, it could achieve better performance gains over other task offloading strategies in different environmental scenarios.

The rest of the paper is organized as follows. Related work is discussed in Section II. System model and problem formulation is introduced in Section III. A deep reinforcement learning (DRL)-based approach is proposed in Section IV. In Section V, several experiments are presented. Section VI concludes the paper.

II. Related Work

There has been extensive research on how to perform residential DSM, which can be roughly categorized into price-driven and incentive-driven mechanisms [5]. Price-driven DSM mechanisms provide time-varying prices, encouraging the customers to reduce their energy consumption when prices are high. Consumers are rewarded when signing up for the incentive-driven DSM program and may be penalized if they did not adjust their energy consumption when asked by the utility company to do so [11]. Online learning methods are used to learn the strategies for realizing residential DSM. The learning process is often modeled as the interaction between the users and environment (including electricity consumption, electricity price, etc.), and the set of decisions may include how to shift peak power consumption and schedule the working time of household appliances. For example, reinforcement learning (RL) is utilized for online scheduling of building energy [12] and electric vehicle charging systems [13]. However, most of these works did not consider the urgent requirements of smart devices for high-performance computing, nor combine with the edge computing framework.

More general paradigm of mobile edge computing (MEC) have been widely studied in [14]–[18] for task offloading scheduling. Some works restrict the task execution to be on the local devices only or on the server only [19], while others allow the tasks to be decomposed and executed locally and online in parallel [20]. However, these works mainly focus on the communication and networking features of MEC, such as bandwidth and computing resource limitations in cellular networks, and hope to find an optimal decision that minimizes the overall offloading cost in terms of computation cost, and delay cost.

Although MEC can tackle the problem of insufficient local computing resources, it may compromise the efficiency of DSM under a paradigm shift of household energy consumption patterns [21]. In fact, the usage of household appliances is heavily dependent on the real-time electricity price, which should not be neglected when making task offloading decisions. The importance of integrating MEC and smart grid framework has been recognized by some researchers [6], [7]. Indeed, MEC offers a platform for collecting and storing data from smart meters and other sensors, as well as for the associated computational tasks, and thus can act as a bridge of the smart grid and the household, resulting in reduced latency, increased privacy and locality for smart grids. These benefits prompt more works that consider the problem of joint task scheduling, computational offloading, and energy management under the edge computing architecture [22], [23]. The survey [24] summarized state-of-the-art research works on energy-aware edge computing, including architectures, operating systems, applications services, and computational offloading. These works usually have different decision rules and learning objectives. Some works focus on maximizing the number of finished tasks

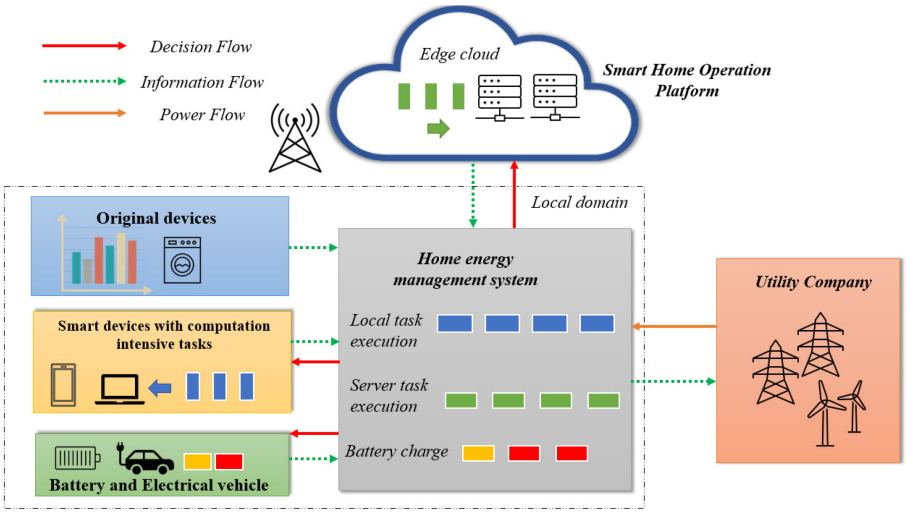


Fig. 1. Residential demand side management with edge computing.

with limited battery capacity of edge devices [25], [26]. Other works investigate the goal of optimization in the smart grid [27], [28].

III. System Model

A. System Setup

In this section, we propose the integrated system with local and SHOP domain. The system model is illustrated in Fig. 1. We consider a residential user who runs several household appliances and participates in a DSM program. In the local domain, there are two types of devices: 1) the smart devices with computationally intensive tasks to be completed and 2) the ordinary device with non-time-constrained tasks. In the SHOP domain, the SHOP deploys several edge servers with more powerful computation and storage resources compared with the local domain. A base station is connected directly with the SHOP through the wireless channel. The household employs an intelligent HEMS, which schedules the energy consumption of the entire household and acts as the interface between the user and the utility company. Besides, the HEMS is equipped with a battery that can purchase and sell electricity from/to the grid. The HEMS can monitor the state of tasks, battery, and DSM restrictions. Based on the above observations, it determines how to perform the tasks on devices (i.e., local computing) or in the SHOP server (i.e., mobile edge computing) and informs the battery to buy or sell electricity. The whole system consists of three flows: 1) the decision flow indicates the task assignment and dis/charge action decided by the HEMS; 2) the information flow contains the state information of tasks, the battery, and DSM requirements; and 3) the power flow includes electricity sold from the grid to the end user or the opposite way. We adopt a discrete-time system model where time is divided into slots of equal length T_s (in seconds) and indexed by $t = \{1, 2, \ldots\}$. Next, we describe the models for devices, tasks, battery, and demand response programs in detail.

B. Smart Device Model

There are a growing number of computationally intensive tasks within a household. Examples of such tasks include smart devices such as smart refrigerators, voice assistant speakers and entertainment devices like smart TVs. Common features of these tasks include: 1) requirement of high-performance computing and storage for voice recognition or image processing and 2) desire for local execution due to time-sensitivity and privacy issues; high energy consumption induced from high-performance computing.

Bearing these common features in mind, we model these tasks mathematically as follows. At each time *t*, the HEMS maintains a sequence of $n^S \in \mathbb{Z}_+$ devices requesting to execute tasks. Each device needs to complete one task k_i, $i = 1, 2, \ldots, n^S$, which is characterized by a tuple (C_i, E_i), where $C_i \in \mathbb{R}_+$ is the total amount of computing resources required for completing task *i* (i.e., task computing load) and $E_i \in \mathbb{R}_+$ is the time before the task expires. We use the CPU cycles to measure the workload of computational tasks C_i, as is widely used in works on resource allocation and task offloading. The state of each task k_i at time slot *t* includes the remaining task load $c_i(t) \in [0, C_i]$ and the elapsed time of the task $e_i(t) \in [0, E_i]$. For example, in incremental learning, it usually takes several hours every day to update the neural network model with the newly generated data.

At the beginning of each time slot *t*, the HEMS makes the task allocation decisions and forward them to the devices and the SHOP server. In our paper, we adopt the latter setting, namely the task loads of the local device and the cloud server are additive. Specifically, HEMS determines the amount $a_i^L(t) \in \mathbb{R}_+$ of task executed on local devices and the amount $a_i^S(t) \in \mathbb{R}_+$ of task offloaded to the SHOP server for task k_i in time slot *t*. To show the details of such task offloading, we first introduce the wireless network environment settings where there is a multi-user SHOP system with one base station (BS). We denote *W* as the bandwidth of the wireless channel, which is equally allocated to the n^S devices offloading tasks to the SHOP server simultaneously at time slot *t*. Thus, according to the Shannon channel capacity, the achievable upload data rate for device *i* is:

$$r_i = \left(W/n^S\right) \cdot \log\left(1 + \frac{P_i^U h_i}{\left(W/n^S\right) N_0}\right), \tag{1}$$

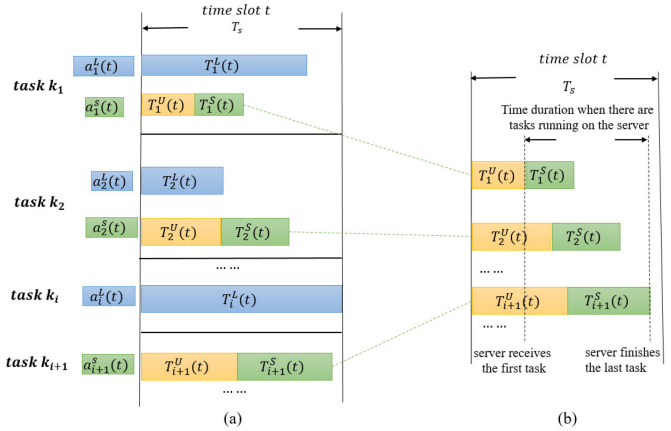


Fig. 2. (a) Task execution time of smart device *i* during time slot *t*. (b) Server running time during time slot *t*.

where P_i^U is the transmission power of device i when uploading data, h_i is the channel gain of device i and N_0 is the variance of white Gaussian channel noise. We next analyze the required time and energy cost for computing $a_i^L(t)$ and $a_i^S(t)$ respectively.

1) Task Execution Time: In the local domain, the execution time of $a_i^L(t)$ amount of task k_i is,

$$T_i^L(t) = a_i^L(t)/f_i^L, \tag{2}$$

where f_i^L is the computation capacity of local processor (i.e., CPU cycles per second) of device i. Clearly, the computing capacity differs among devices. Devices with image processing capabilities (e.g., mobile phones) are enabled with more powerful CPUs, or even GPUs.

In contrast to local computing, $a_i^S(t)$ amount of task k_i needs to be offloaded and executed by the SHOP server. The whole SHOP computing procedure consists of three steps. First, device i uploads the required data to the HEMS through the wireless channel and the HEMS forwards the data to the SHOP server. The corresponding transmission time is:

$$T_i^U(t) = b_i^U(t)/r_i = \alpha \cdot a_i^S(t)/r_i. \tag{3}$$

where $b_i^U(t)$ is the data size (bits) of task $a_i^S(t)$, which can be calculated by a coefficient α. Second, the SHOP server executes the task k_i for:

$$T_i^S(t) = a_i^S(t)/f^S, \tag{4}$$

Finally, the SHOP server returns the computational result to device i. The time can be reasonably ignored of this step due to the small size of the computational result. Thus, the time cost by SHOP computing is the sum of data uploading stage and server computing stage: $T_i^U(t) + T_i^S(t)$. Since the local and server computing are performed simultaneously, see Fig. 2, the total execution time is the maximum of these two: $T_i(t) = \max\{T_i^L(t), T_i^S(t) + T_i^U(t)\}$.

2) Energy Consumption: Since DSM is only related to the local energy consumption, we only need to consider the energy consumed in local domain.

We first use the features of the CPU to characterize the energy consumption of local task execution:

$$C_i^L(t) = \epsilon_i a_i^L(t), \tag{5}$$

where ϵ_i is the energy consumption per CPU cycle.[1]

Then, the data transmission leads to the following energy consumption:

$$C_i^U(t) = P_i^U T_i^U(t) \tag{6}$$

The total energy consumption at time slot t of device i is the sum of these two stage: $C_i(t) = C_i^L(t) + C_i^U(t)$.

C. Demand Response Model

The user participates in a demand response program that may require the user to consume a reduced amount of electricity within a specified time frame (e.g., a few hours). Such a demand reduction is usually mandated by a contract signed by the user and the utility company. The user is rewarded when signing up for the program and may need to pay a penalty if failing to fulfill the load reduction requirement. At a certain time slot t, the utility company may send a signal ($l(t)$, $d(t)$) to the user, which requires the user to restrict its electricity load to a total amount of $l(t)$ kilowatt hour (kWh) in the next $d(t)$ time frame. The demand response events will not overlap, namely a new event will always happen after the previous event has ended. In other words, we can view $l(t)$ as the "quota" for electricity consumption in the next $d(t)$ time frame, starting from time slot t. This quota decreases over time as the user consumes more electricity. Note that the computing resources in the SHOP server will not be counted when calculating the local electricity consumption.

D. Ordinary Device Model

In order to make our model more practical, in addition to the smart devices listed above, we also consider some ordinary household appliances. We assume that these appliances neither have intelligent computing functions, nor scheduled tasks. Consider there are totally n^O ordinary devices in our system, where device j only has two states: {idle, work} at each time slot t. The corresponding energy consumption in time slot t of device j is:

$$C_j(t) = \begin{cases} P_j^I T_s & \text{if } j \text{ is in idle state} \\ P_j^W T_s & \text{if } j \text{ is in work state} \end{cases} \tag{7}$$

where P_j^I, P_j^W are the power of idle and working state respectively. Our strategy does not require scheduling the state of these devices. However, their energy consumption affects the remaining quote of DSM and thus our decision-making. For example, when most original devices are in work state, in order not to violate DSM, we may need to offload more computational intensive tasks to the SHOP for completion.

[1]We set $\epsilon_i = 10^{-27}(f_i^L)^2$ in the experiment according to some previous works [29].

E. Battery Model

With the rapid development of smart grids and electric vehicles (EVs), the batteries have become the most significant energy storage equipments and have attracted widespread attention. Whether to charge/discharge the battery can be controlled by the HEMS wisely based on the current electricity price and the remaining capacity, e.g., charging the battery when the electricity price is low and discharging the battery when the price is on-peak.

In this paper, we consider a battery with capacity C_B. Let $B(t)$ represent the state of charge (SOC) of the battery at time slot t. At time slot t, the HEMS chooses the charging/discharging action $a_B(t)$, where $a_B(t) \in \{-1, 0, 1\}$ is positive when the battery is charging and negative when discharging. We further denote the charge and discharge currents and voltages by $I^{in}, I^{out}, U^{in}, U^{out}$. To model the energy loss during charging and discharging, we define the charging and discharging efficiency of the battery as $\beta^{in} \in (0, 1)$ and $\beta^{out} \in (0, 1)$, respectively. Thus, the change of the energy level in the battery at time slot t can be computed as:

$$b(t) = \begin{cases} \beta^{in} I^{in} U^{in} T_s & \text{if } a_B(t) = 1 \\ -\left(1/\beta^{out}\right) I^{out} U^{out} T_s & \text{if } a_B(t) = -1 \\ 0 & a_B(t) = 0. \end{cases} \tag{8}$$

The net amount of electricity purchased from the utility at time slot t is:

$$C_b(t) = \begin{cases} I^{in} U^{in} T_s & \text{if } a_B(t) = 1 \\ -I^{out} U^{out} T_s & \text{if } a_B(t) = -1 \\ 0 & \text{otherwise,} \end{cases} \tag{9}$$

where a negative $C_b(t)$ indicates energy export to the grid.

From (8) and (9), we can see that due to efficiency loss of the battery, the actual increase of the SOC is less than the energy purchased from the grid, and the actual reduction of the SOC is greater than the energy sold to the grid.

F. Problem Formulation

We define $J(t)$ as the overall system cost at time slot t,

$$J(t) = \sum_{i=1}^{n^S} \left[\omega^E C_i^{local}(t) + \omega^T C_i^{time}(t) + \omega^S C_i^{server}(t) \right] + \sum_{j=1}^{n^O} \omega^E C_j(t) + \omega^E C_b(t) \tag{10}$$

where the first term demonstrates the total cost of tasks executed by the n^S smart devices. In particular, C_i^{local}, C_i^{time} and C_i^{server} are the cost of local energy, task completing time and the server rental fee of task k_i at time slot t. We give their specific definitions in the next section. The second term illustrates the energy cost of n^O original devices while the last term states the energy cost of battery charging. The coefficients $\omega^E, \omega^T, \omega^S$ demonstrates the trade-off between the cost of energy (including the consumption of smart devices, original devices and the battery), time-delay and the server.

TABLE I
DEFINITION OF NOTATIONS

Notion	Definition
n^S, n^O	Number of smart and original devices
C_i	Computing load of task k_i
E_i	Expiration time of task k_i
T_s	Duration of time slot t (in seconds)
$c_i(t)$	Remaining task load of task k_i at slot t
$e_i(t)$	Elapsed time if task k_i at slot t
$a_i^L(t)$	Amount of task k_i executed locally at slot t
$a_i^S(t)$	Amount of task k_i offloaded at slot t
$a_B(t)$	Battery charge/discharge decision at slot t
$B(t)$	Battery storage
f_i^L	Local computation capacity of device i
ϵ_i	Energy cost per cycle of device i
r_i	Upload data rate of device i
P_i^U	Transmission power of device i
P_i^I	Power of device i in idle state
P_i^W	Power of device i in work state
α	Coefficient between CPU cycle and data size
$l(t), d(t)$	DSM requirements
$P_E(t)$	Electricity price at slot t
$P_S(t)$	Server rent fee at slot t
η	Minimum battery percentage level

The objective of this paper is to minimize the long-term discounted sum cost of the system, which is computed as

$$\begin{aligned} \min_{\boldsymbol{a}^L, \boldsymbol{a}^S, a_B} \quad & J^\gamma = \mathbb{E}\left[\sum_{t=1}^{\infty} \gamma^t J(t) \right] \\ & C1\text{: } a_i^L(t) + a_i^S(t) \le c_i(t), \sum_{i=1}^{n^S} \alpha a_i^S(t) \le C_S, \\ & \qquad a_i^L(t) \le C_i^L \ \forall\ i \in n(t) \\ & C2\text{: } T_i(t) \le T_s \ \forall\ i \in n(t) \\ & C3\text{: } e_i(t) \le E_i(t), \ \forall\ i \in n(t) \\ & C4\text{: } l(t) > 0, \ \text{for } d(t) = 0 \\ & C5\text{: } \eta \times C_B \le B(t) \le C_B, \end{aligned} \tag{11}$$

γ is a discount factor. $C1 - C5$ are constraints. $C1$ represents that the constraints on allocated computation resource. In particular, the amount $a_i^L(t)$ should be no more than the total computation capacity C_i^L of device i. And all offloaded amount of task load $\sum_{i=1}^{n^S} \alpha a_i^S(t)$ should be limited by the server cache size C_S. Besides, since each device must execute its allocated amount of task either locally or in the SHOP within one time slot, we have C_2. $C3$ indicates that the task must be completed before the expired time. $C4$ is on behalf of the DMS constraints that the energy consumption should be lower than the limitation of the utility company. $C5$ states that the remaining capacity of the battery must not be less than a certain percentage η of the total capacity C_B and not larger than C_B. Here, C_S, C_i^L and C_B are both constants. We summarize the parameters in Table I for ease of reading.

The optimal solution to Eq. (11) requires complete information regarding the mathematical models of the system, such as the statistical distributions of the tasks and expired time. Such information is in general not available in a practical

system. One feasible approach to overcome these challenges is to design an online solution that can efficiently make the decisions regarding charging and computation resource allocation in real time through interactions with the system. Therefore, instead of applying conventional optimization methods to solve the problem Eq. (11), we propose a DRL-based method to find the optimal $\boldsymbol{a}^L$, $\boldsymbol{a}^S$, a_B.

IV. Deep Reinforcement Learning-Based Solution

In this section, we first convert the above optimization problem into a Markov decision process (MDP) which can be solved by the deep reinforcement learning algorithm.

A. Formulation of Markov Decision Process

As we can see that the current HEMS states are completely determined by the previous states and task assignment decisions. Therefore, we can formulate the decision making problem as an MDP with infinite time horizon. An MDP is a five-tuple $(\boldsymbol{S}, \boldsymbol{A}, \mathbb{P}(\boldsymbol{s}|\boldsymbol{s'}, \boldsymbol{a}), \boldsymbol{R}(\boldsymbol{s}, \boldsymbol{a}), \gamma)$, where $\boldsymbol{S}$ is the set of states, $\boldsymbol{A}$ is a set of actions, $\mathbb{P}(\boldsymbol{s}|\boldsymbol{s'})$ represents the state transition probability, $\boldsymbol{R}(\boldsymbol{s}, \boldsymbol{a})$ is the immediate reward and γ is a discount factor. The details about the MDP formulation are shown as follows.

1) States: For compact presentation, we define the vectors $\boldsymbol{c}(t) = [c_1(t), \ldots, c_{n^S}(t)]$ and $\boldsymbol{e}(t) = [e_1(t), \ldots, e_{n^S}(t)]$ as the states of all the tasks at time slot *t*. Then the state $\boldsymbol{s}(t)$ encapsulates five types of information: 1) the price of electricity $p_E(t) \in \mathbb{R}_+$; 2) the price of mobile server computing $p_S(t) \in \mathbb{R}_+$; 3) the state of demand response $(l(t), d(t)) \in \mathbb{R}_+ \times \mathbb{R}_+$; 4) the status of tasks $\boldsymbol{c}(t), \boldsymbol{e}(t) \in \mathbb{R}_+^{n^S} \times \mathbb{R}_+^{n^S}$; and 5) the state of the battery $B(t) \in \mathbb{R}_+$. We define the collection of all the above status as the state of the MDP.

$$\boldsymbol{s}(t) = (p_E(t), p_S(t), l(t), d(t), \boldsymbol{c}(t), \boldsymbol{e}(t), B(t)). \quad (12)$$

2) Actions: The task offloading action at each time slot *t* specifies how much computing resources, both on the devices and on the SHOP server, allocated to each task. We define the vectors $\boldsymbol{a}^L(t) = [a_1^L(t), \ldots, a_{n^S}^L(t)]$ and $\boldsymbol{a}^S(t) = [a_i^S(t), \ldots, a_{n^S}^S(t)]$ as the task offloading decisions. Besides, the action $a_B(t)$ represents the charged ($a_B(t) = 1$) or discharged ($a_B(t) = 1$) decision in the battery during time slot *t*. Considering the constraint $C5$ in Eq. (11), $a_B(t)$ will always be set as 1 until $B(t)$ achieve the threshold ηC_B. Then the whole action vector can be written as

$$\boldsymbol{a}(t) = \Big(\boldsymbol{a}^L(t), \boldsymbol{a}^S(t), a_B(t)\Big) \in \mathbb{R}_+^{n^S} \times \mathbb{R}_+^{n^S} \times \mathbb{Z}_+. \quad (13)$$

From the constraints $C1 - C2$ in Eq. (11), we can obtain the set of feasible actions at each time slot *t*

$$\begin{aligned}\mathcal{A}(t) = \Big\{ \Big(\boldsymbol{a}^L(t), \boldsymbol{a}^S(t), a_B(t)\Big) \mid a_i^L(t), a_i^S(t) \geq 0, \\ a_i^L(t) + a_i^S(t) \leq c_i(t), \sum_{i=1}^{n^S} \alpha a_i^S(t) \leq C_S \\ a_i^L(t) \leq C_i^L, \ \forall \ i \in n^S \Big\}. \end{aligned} \quad (14)$$

3) Rewards: Designing a good reward function is essential to achieve high-quality policies. The reward function must be consistent with our optimization goal Eq. (11), besides that it satisfies all the constraints. Given a state-action pair ($\boldsymbol{s}(t)$, $\boldsymbol{a}(t)$), the reward at each time slot *t* has five components.

The first component is the positive reward of completing the task on time. For task k_i, the reward function is written as:

$$u_{T,i} : (c_i(t), e_i(t)) \mapsto u_{T,i}(c_i(t), e_i(t)). \quad (15)$$

In general, the reward is realized only when the task is completed before the deadline ($C3$). An example reward function could be

$$u_{T,i}(c_i(t), e_i(t)) = \begin{cases} \gamma_i & \text{if } c_i(t) = 0 \text{ and } e_i(t) \leq E_i \\ 0 & \text{otherwise,} \end{cases} \quad (16)$$

where $\gamma_i > 0$ is the reward of completing task k_i.

The second component of the reward is the cost of local electricity consumption. It is main incurred by smart devices for executing task, original devices for staying in work/idle state and battery's dis/charging decision. As suggested by [30], the price of selling electricity to the grid is the same as purchasing. Thus the whole local energy cost is defined as:

$$C^{energy}(t) = p_E(t) \cdot \left(\sum_{i=1}^{n^S} C_i(t) + \sum_{j=1}^{n^O} C_j(t) + C_b(t) \right) \quad (17)$$

The third component of the reward is the cost of running the computation in the SHOP server. Here we do not need to consider the energy consumption of the SHOP server, as this is constant over time. Instead, we consider the running time of the computing task on the SHOP server since the server is always charged by the rent time. Since all offloaded tasks are executed in parallel, we use the time between the start of the first task and the completion of the last task to indicate the time duration when there is a task running on the server, shown in Fig. 2. Then the cost can be computed as:

$$\begin{aligned} C^{server}(t) = p_C(t) \cdot \Big(\max_i \Big\{ T_i^U(t) + T_i^S(t) \Big\} \\ - \min_i \Big\{ T_i^U(t) \Big\} \Big). \end{aligned} \quad (18)$$

The fourth component of the reward is the time cost of all n^S tasks:

$$C^{time}(t) = \sum_{i=1}^{n^S} T_i(t) \quad (19)$$

The final component of the reward, following the constraint $C4$, is the potential penalty of violating the demand response requirement, which is defined as

$$u_D(l(t), d(t)) = \begin{cases} \rho & \text{if } d(t) = 0 \text{ and } l(t) < 0 \\ 0 & \text{otherwise,} \end{cases} \quad (20)$$

where $\rho > 0$ is the (usually large) penalty of violating the DSM requirement. In other words, the user will be charged by ρ if the local energy consumption exceeds $l(t)$ when the time frame $d(t)$ ends.

According to the definition of the above reward functions, we change the problem of cost minimization (11) into the reward maximization problem. As such, the reward function is then

$$u(\boldsymbol{s}(t), \boldsymbol{a}(t)) = \sum_{i=1}^{n^S} u_{T,i}(c_i(t), e_i(t)) - w^E C^{energy}(t) - w^T C^{time}(t) - w^S C^{server}(t) - u_D(l(t), d(t)). \quad (21)$$

Here, $\omega^E, \omega^T, \omega^S$ are coefficients, which indicate the trade-off relation ship between the energy consumption, time-delay cost and server cost.

4) State Transitions: We specify the state transition for each component in (12). Given the task allocation decision, the state of task k_i will be updated in time slot $t+1$ according to

$$c_i(t+1) = c_i(t) - a_i^L(t) - a_i^S(t), \quad (22)$$

$$e_i(t+1) = e_i(t) + T_s. \quad (23)$$

Given the charging decision, the battery SOC transits according to

$$B(t+1) = B(t) + b(t). \quad (24)$$

The state of the demand response event transits according to

$$l(t+1) = l(t) + B(t) - \sum_{i=1}^{n^S} C_i(t) - \sum_{j=1}^{n^O} C_j(t) + b(t), \quad (25)$$

$$d(t+1) = d(t) - T_s. \quad (26)$$

$B(t)$ can be regarded as an additional supplementary power. Noting that, $l(t)$ is only related to the local energy consumption rather than the server. The pricing states $p_E(t)$ and $p_S(t)$ are drawn randomly and independently of each other and other state components.

From (22)-(25) we can see that there is some certain stable state transition probability $\mathbb{P}(\boldsymbol{s}|\boldsymbol{s}', \boldsymbol{a})$ for the system starting from state $\boldsymbol{s}'$ to state $\boldsymbol{s}$ when taking action $\boldsymbol{a}$.

The joint decision π is defined as the mapping from the state vector $\boldsymbol{s}(t)$ to the action vector $\boldsymbol{a}(t)$:

$$\pi : \boldsymbol{s}(t) \mapsto \boldsymbol{a}(t), \quad (27)$$

and the set of all policies is defined as Π_M.

Our goal is to find the optimal joint policy that maximizes the expected total reward starting from any initial state $\boldsymbol{s}(1)$ over horizon T, namely

$$\pi^* = \arg \max_{\pi \in \Pi_M} \mathbb{E}^{\pi} \left\{ \sum_{t=1}^{\infty} \gamma^t u(\boldsymbol{s}(t), \boldsymbol{a}(t)) \right\}, \quad (28)$$

where $\gamma \in (0,1)$ is the discount factor, and the expectation $\mathbb{E}^{\pi}$ depends on the policy π. Intuitively, there is a trade-off between the local execution and task offloading. Constrained by the poor local computing capacity, executing all tasks locally may cause the task expires. It also may introduce high local energy consumption which results in the violation of DSM requirements. In contract, offloading all tasks to the SHOP server means the huge transmission delays and expensive server rental fees. Our following algorithm is to find an optimal joint allocation scheme between the trade-off.

B. DDPG-Based Solutions

After formulating the objective problem as an MDP, we can obtain the optimal policy through the reinforcement learning methods. Many reinforcement learning algorithms utilize the action-value function $Q^{\pi}(\boldsymbol{s}(t), \boldsymbol{a}(t))$ to solve (28), which is defined as the total reward under policy π choosing action $\boldsymbol{a}(t)$ at state $\boldsymbol{s}(t)$. It also can be written as following recursive form including $Q^{\pi}(\boldsymbol{s}(t+1), \boldsymbol{a}(t+1))$, known as the Bellman equation:

$$Q^{\pi}(\boldsymbol{s}(t), \boldsymbol{a}(t)) = u(\boldsymbol{s}(t), \boldsymbol{a}(t)) + \gamma \mathbb{E}_{\boldsymbol{s}(t+1), \boldsymbol{a}(t+1) \sim \pi} \times [Q^{\pi}(\boldsymbol{s}(t+1), \boldsymbol{a}(t+1))]. \quad (29)$$

where $\boldsymbol{s}(t+1)$ denotes the next state being transitioned from state $\boldsymbol{s}(t)$ under $\boldsymbol{a}(t)$. Given the optimal action-value function $Q^*(\boldsymbol{s}(t), \boldsymbol{a}(t))$, the optimal Markov policy π^* can be defined as

$$\pi^*(\boldsymbol{s}(t)) = \arg \max_{\boldsymbol{a}(t) \in \mathcal{A}(t)} Q^*(\boldsymbol{s}(t), \boldsymbol{a}(t)). \quad (30)$$

Traditional reinforcement learning algorithms, such as Q-learning [31], maintain a "Q table" to compute and store the Q-value by enumerating each state-action pair. The scale of the "Q table" is related to the size of state and action space. Recall that in our problem formulation, the joint decision at time slot t $\boldsymbol{a}(t) = \{a_1^L(t), \ldots, a_{n^S}^L(t), a_1^S(t), \ldots, a_{n^S}^S(t), a_B(t)\}$ is a $(2n^S+1)$-dimensional vector, and the state $\boldsymbol{s}(t) = (p_E(t), p_S(t), l(t), d(t), \boldsymbol{c}(t), \boldsymbol{e}(t), B(t))$ is a $(2n^S+5)$-dimensional vector.

Therefore, the state space grows exponentially in the number of tasks, resulting in the "curse of dimensionality" and failure to maintain the "Q table." Besides, the joint actions $a_i^L(t)$, $a_i^S(t)$ and $a_B(t)$ are continuous values (or discrete values with high granularity), thus searching through the large action space to find the optimal action with the highest Q value may lead to very low convergence.

To address this problem, some deep learning based methods, such as deep Q network (DQN) [32], utilize deep neural networks as function approximators of the action-value function. The DQN takes state and action vectors as input and outputs the corresponding estimated value. When there are sufficient state-action samples, that is, after sufficiently sampling of the entire environment, the output of the DQN can be approximated as a Q value to help us to choose actions. However, the DQN mainly focus on the high-dimensional state space rather than overcome the continuous action space problem. It still needs to compare the Q value under different actions to choose the best one at current state.

The DDPG algorithm [33], which combines the actor-critic structure and neural networks, makes up for the lack of DQN. The key is to parameterize both the Q-value function and the policy. As illustrated in Figure 3, the DDPG algorithm is integrated in HEMS. Both the *Actor* and *Critic* networks are composed of deep neural networks. First, the Action network

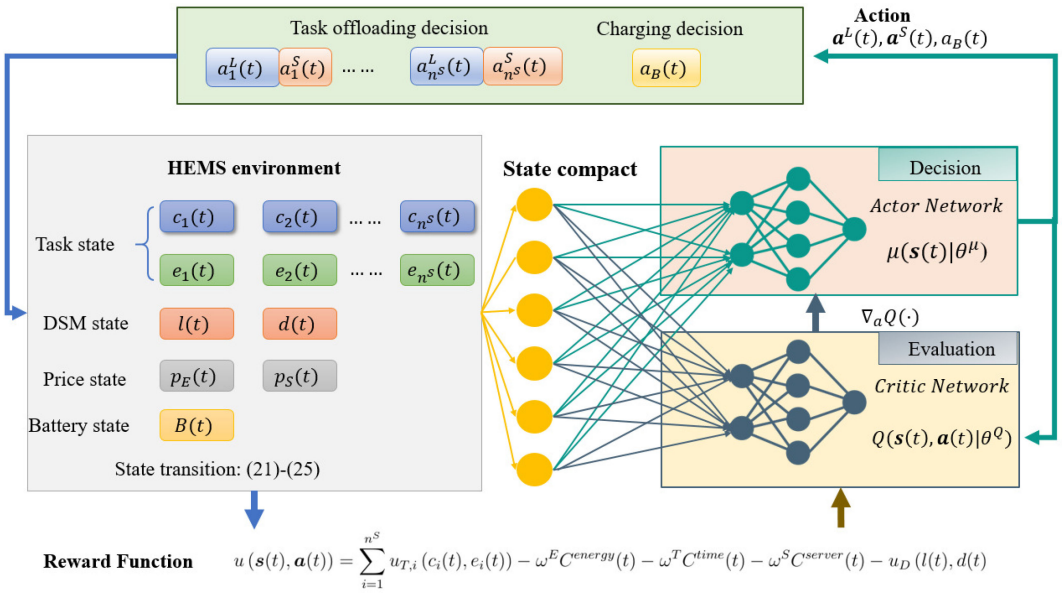


Fig. 3. Deep deterministic policy gradient based residential demand side management framework.

makes the joint decision based on the compacted state from the HEMS under the current policy. Then, the *Critic* network takes the HEMS state, the output of *Actor*, and the immediate reward as input to calculate the approximated Q-value. These two networks are updated alternately until the training process ends. The training details of the DDPG algorithm is shown in Algorithm 1. We define the parameterized *Actor* and *Critic* networks as $\mu(\boldsymbol{s}|\theta^{\mu})$ and $Q(\boldsymbol{s},\boldsymbol{a}|\theta^{Q})$ with parameters θ^{μ} and θ^{Q} respectively. Noting that we do not update the model after we collect one state transition sample. Instead, we initialize a replay buffer with size R. When every training episode starts, the state transition tuple $(\boldsymbol{s}(t),\boldsymbol{a}(t),\boldsymbol{u}(t),\boldsymbol{s}(t+1))$ following current policy θ^{μ} is stored in the replay buffer. Once the replay buffer is up to its capacity limitation, the *Actor* and *Critic* networks are updated by sampling a minibatch of transition tuples from the buffer. The *Critic* network can be optimized using gradient descent $\theta^{Q}\leftarrow\theta^{Q}-r^{Q}\nabla_{\theta^{Q}}L(\theta^{Q})$ by minimizing the Mean Square Error (MSE) loss:

$$L(\theta^{Q})=E_{a(t)\sim\pi,\boldsymbol{s}(t),u(t)}\left[\left(Q\left(\boldsymbol{s}(t),\boldsymbol{a}(t)|\theta^{Q}\right)-y(t)\right)^{2}\right], \tag{31}$$

where $y(t)=u(\boldsymbol{s}(t),\boldsymbol{a}(t))+\gamma Q(\boldsymbol{s}(t+1),\boldsymbol{a}(t+1)|\theta^{Q})$.

The *Actor* network updates the policy parameter θ^{μ} with respect to the direction of the Q-value gradient $\theta^{\mu}\leftarrow\theta^{\mu}-r^{\mu}\nabla_{\theta^{\mu}}J(\theta^{\mu})$:

$$\begin{aligned}\nabla_{\theta^{\mu}}J(\theta^{\mu})&=E\left[\nabla_{\theta^{\mu}}Q\left(\boldsymbol{s},\boldsymbol{a}|\theta^{Q}\right)|_{\boldsymbol{s}=\boldsymbol{s},a=\mu(\boldsymbol{s}(t)|\theta^{\mu})}\right]\\&=E\left[\nabla_{a}Q\left(\boldsymbol{s},\boldsymbol{a}|\theta^{Q}\right)_{\boldsymbol{s}=\boldsymbol{s}(t),a=\mu(\boldsymbol{s}(t)|\theta^{\mu})}\right.\\&\qquad\left.\times\ \nabla_{\theta^{\mu}}\mu(\boldsymbol{s}|\theta^{\mu})_{\boldsymbol{s}=\boldsymbol{s}(t)}\right].\end{aligned} \tag{32}$$

To solve the expectations in (31) and (32), we can use Monte Carlo estimators to obtain estimated values over the mini-batches

$$E_{z\sim q(z|x_i)}[f(z)]\approx\frac{1}{M}\sum_{m=1}^{M}f(z),z\sim q(z|x_i). \tag{33}$$

Algorithm 1 Deep Deterministic Policy Gradient Algorithm

Initialization: Randomly initialize the *Critic* network $Q(\boldsymbol{s},\boldsymbol{a}|\theta^{Q})$, the *Actor* network $\mu(\boldsymbol{s}|\theta^{\mu})$, and the target network and $Q'(\boldsymbol{s},\boldsymbol{a}|\theta^{Q'})$, $\mu'(\boldsymbol{s}|\theta^{\mu'})$

Input: Replay buffer size R, batch size k, number of the episodes m, number of time steps in each episode T, learning rate r^{Q} and r^{μ} for *Critic* and *Actor* network, update rate τ^{Q} and τ^{μ} for target *Critic* and *Actor* network.

Output: The optimal action $\boldsymbol{a}$

for *episode* $=1$ to m **do**
  Initialize a random process for action exploration;
  Receive initial state:
  $\boldsymbol{s}(1)=(p_E(1),p_S(1),l(1),d(1),\boldsymbol{c}(1),\boldsymbol{e}(1),B(1))$
  for $t=1$ to T **do**
    Select task allocation and charging action: $\boldsymbol{a}(t)=\mu(s(t)|\theta^{\mu})+\sigma$ according to the current policy μ and exploration noise σ.
    Execute action $\boldsymbol{a}(t)$, observe the reward $u(\boldsymbol{s}(t),\boldsymbol{a}(t))$ and the next state $\boldsymbol{s}(t+1)$.
    Store transition $(\boldsymbol{s},\boldsymbol{a}(t),u(\boldsymbol{s}(t),\boldsymbol{a}(t)),\boldsymbol{s}(t+1))$ in the replay buffer R
    if Stored transitions $>$ Replay buffer capacity **then**
      Discard the oldest transition samples.
    end if
    Sample a random batch of k transitions from R
    Update *Critic* using the gradient descent:
    $\theta^{Q}\leftarrow\theta^{Q}-r^{Q}\nabla_{\theta^{Q}}L^{k}$ using Eq. (34).
    Update *Actor* using the policy gradient descent:
    $\theta^{\mu}\leftarrow\theta^{\mu}-r^{\mu}\nabla_{\theta^{Q}}J^{k}$ using Eq. (35)
    Update the target networks Q' and μ':
    $\theta^{Q'}\leftarrow\tau\theta^{Q}+(1-\tau)\theta^{Q'}$
    $\theta^{\mu'}\leftarrow\tau\theta^{\mu}+(1-\tau)\theta^{\mu'}$
  end for
end for

For the mini-batch of state transition $\{\boldsymbol{s}_i,\ \boldsymbol{a}_i,\ u(\boldsymbol{s}_i, j\boldsymbol{a}_i), \boldsymbol{s}_{i+1}\}_{i=1}^{k}$ with size k, (31) and (32) can be written as:

$$L^k = \frac{1}{k}\sum_{i=1}^{k}\Big(Q\Big(\boldsymbol{s}_i, \boldsymbol{a}_i|\theta^Q\Big) - y_i\Big)^2 \tag{34}$$

$$\nabla_{\theta^\mu} J^k = \frac{1}{k}\sum_{i=1}^{k}\nabla_a Q\Big(\boldsymbol{s}, \boldsymbol{a}|\theta^Q\Big)_{\boldsymbol{s}=\boldsymbol{s}_i, a=\mu(\boldsymbol{s}_i|\theta^\mu)} \times \nabla_{\theta^\mu}\mu(\boldsymbol{s}|\theta^\mu)_{\boldsymbol{s}=\boldsymbol{s}_i} \tag{35}$$

Since even a small update of θ^Q, θ^μ will lead to a great change in action-value and policy, two target networks with parameters $Q'(\boldsymbol{s}, \boldsymbol{a}|\theta^{Q'})$ and $\mu'(\boldsymbol{s}|\theta^{\mu'})$ are used to give consistent targets during training process. The weights of these target networks are updated slowly by following the learned networks in the way: $\theta^{Q'} \leftarrow \tau^Q\theta + (1-\tau)\theta^{Q'}$, $\theta^{\mu'} \leftarrow \tau^\mu\theta + (1-\tau\mu)\theta\mu'$. As $\tau^Q \ll 1$, $\tau^\mu \ll 1$, the target values change slowly.

V. Experimental Results

In this section, we measure the performance of our proposed algorithm using both simulated and real world data. We first introduce four approaches as comparison methods. The training performance of our proposed model is shown in Section V-B. Then, we carefully compare our algorithm with other methods in terms of the cost weights, number of devices and server cache size in Sections V-C. Finally, we use the real world data to demonstrate the efficiency of our algorithm in Section V-D.

A. Comparison Algorithm

We first introduce four comparison algorithms used in this paper. The first two are popular DRL algorithms and other two are heuristic methods.

1) *A3C* stands for Asynchronous Advantage Actor-Critic algorithm [34], which is a DRL framework. It is originally proposed for acceleration of policy training in parallel. The agent optimizes both policy and approximation of a state-value function. Note that A3C and DDPG are both Actor-Critic based DRL methods, which utilize DNNs to learn both the Q-value and policy networks. The main difference between A3C and DDPG lies in the training process where the former use the asynchronous way.

2) *DQN* stands for the *Deep Q Network* algorithm, which is also a popular DRL method. However, it only uses neural network to estimate the action-value function and selects the optimal action with the highest 'Q-value'. Since DQN can not handle large or continuous action dimensions, we define a small discrete action space: for each task k_i at each time slot t, the HEMS can only chose a value from 0, 1 for $a_i^L(T)$ and $a_i^S(T)$, which stands for either execute the task locally or offload it to the SHOP server. However, the amount of allocated tasks should not exceed the upper limitation of the local or server computing capacity.

3) *Full local execution* means that all the computation tasks are executed locally; in other words, $a_i^S(t) = 0$. In this way, we have $C_i^{server}(t) = 0$ and $C_i^{time}(t) = T_i^L(t)$ for all task k_i at time slot t.

4) *Full SHOP execution* scheme offloads all tasks to the SHOP server, i.e., $a_i^L = 0$. Similarly, we have $C_i^{time} = T_i^S + T_i^U$ for all tasks k_i. Since except for the smart devices, the original device and the battery contribute to the local energy consumption, we have $C_i^{energy}(t) = P_E(t)\cdot(\sum_{j=1}^{n^O} C_j(t) + C_b(t))$ rather than 0.

B. Training Performance

We firstly present the details about parameter settings to train the DRL model.

1) System and Task Parameters: The number of smart devices, i.e., the number of tasks to be executed n^S, is fixed to be 20. The CPU frequency for each device are uniformly distributed between [1, 2] GHz. Specifically, for each task k_i, the computation requirement C_i (i.e., the CPU cycles) is uniformly distributed between [0, 10] Gigacycles and the data size b_i^S is uniformly distributed between [0, 100] Mbits with $\alpha = 0.001$. The elapsed time E_i is uniformly distributed between [1, 5] seconds. Typically, it need 10s to compute a 10 Gigacycles task using a 1 GHz CPU, which is larger than 5s. Thus, the smart devices need to offload the part of tasks to server before it is expired. The SHOP server, equipped with more powerful capacity and larger storage size, is configured with $C^S = 800$ Mbits storage and $f^S = 5$ Ghz CPU frequency. To characterize the wireless environment, we set the channel bandwidth $W = 10$ MHz.

2) Energy Parameters: We set the transmission power as $P_i^U = 1$W for all smart devices. For the original devices, the idle power is uniformly distributed between [1, 2]W, and the working power is uniformly distributed between [1000, 2000] W. The time that the original devices stayed in idle and work state accounts for 30% and 70% respectively. The DSM signal released by the utility company is modeled as random variables. In particular, it is usually released in advance to cope with the upcoming peak hour, which is usually predicted by the utility company based the historical residential usage. For example, it may require that the total electricity consumption limited to 10 kW before 18:00. Considering that we set each time slot T_s = 1s, it is not conducive to model learning if the DSM signal only appears once in hours (tens of thousands time slots). Therefore, in our training process, we set a frequent DSM signal, such that, the constrained electricity load $l(t)$ is distributed between [0, 0.5] kW with constraint response time $d(t)$ between [5, 10] minutes.

3) Network structure parameter: The details for network structure and training process are summarized in Table II. As we analyzed before, the *state_dim* and *action_dim* here should be a function of the smart device number n^S.

The training processes of our proposed DDPG method and another DRL method-DQN with the number of training episodes over 20 runs are shown in Fig. 4. In a whole, the curves show that DDPG algorithm has a good convergence and gain more reward than DQN. At the beginning of every episode, we reset the environment with a new initial state $\boldsymbol{s}(1)$. For the DDPG algorithm,we compute the cumulative reward and update the model parameters for 10000 steps. In the first 5 episodes, the decision is randomly selected by the Actor network for collecting the memory buffer, resulting in a low reward. After this phase, the Critic and Actor networks begin

TABLE II
NETWORK STRUCTURE PARAMETER

Parameters	Values
Critic	input layer: state_dim+ action_dim hidden layer: 400,300,100 output layer: 1
Actor	input layer: state_dim hidden layer: 400,300,100 output layer: action_dim
batch_size	$k = 64$
learning rate	$r^Q = 0.001, r^\mu = 0.005$
discount factor	$\gamma = 0.99$

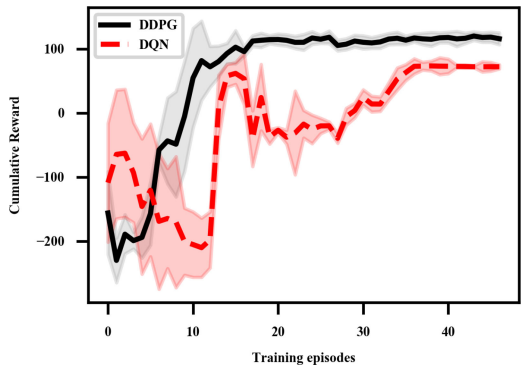


Fig. 4. Training performance of DDPG and DQN algorithm.

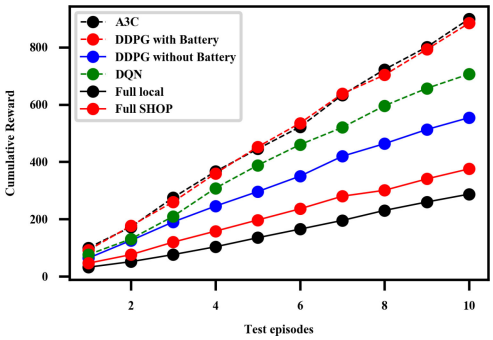


Fig. 5. Testing cumulative rewards of different algorithms.

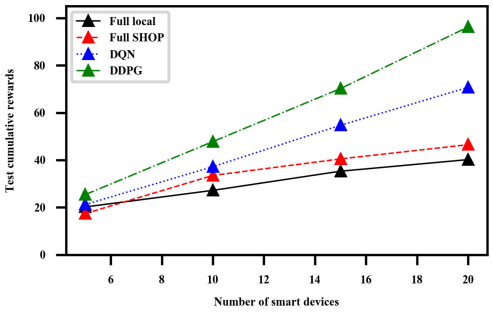


Fig. 6. Testing cumulative rewards of different number of smart devices.

to update using the stored transitions in the memory buffer. It can be observed from Fig. 4 that the training reward starts to increase gradually after episode 5. Then at around episode 20, the cumulative reward converges around 100 with small oscillations. Compared to the DQN algorithm, our proposed algorithm converges faster to a higher average reward with smaller variance. The main reason is that DQN can only select actions in discrete values.

C. Testing Performance

After the training process, the proposed approach can be deployed for energy scheduling. We test the performance of the model from the following aspects:

1) Performance of Cumulative Rewards: We first fix the environmental parameters (i.e., $P_S(t)$ and $P_E(t)$) and compare the cumulative rewards obtained by different algorithms after 10 test episodes. Fig. 5 shows the cumulative rewards of the aforementioned methods. From the parameter setting, we can see that the required computation amount, expired time of each task, and the DSM signal are all concentrated to a narrow interval. Therefore, even though we randomly sample the task and the DSM signal, the cumulative reward we obtain in each episode is close. It leads to some approximate linear results in the Fig. 5.

The A3C (black dash curve) and DDPG (red dash curve) methods, both of which are as Actor-Critic based algorithms, have achieved similar results and are higher than the DQN (green dash curve) methods. This indicates that the policy gradient methods perform better in our problem. The *Full Edge* policy obtains the lowest reward. This is because if all the tasks are executed locally, the rapid increasing $\omega^E C^{energy}$ leads to a significant decrease in $l(t)$, which may become negative before $d(t) = 0$, resulting in the penalty $u_D(l(t), d(t))$ easily. If the user offloads all tasks to the SHOP server, the cost of running the server $\omega^S C^{Server}$ could be higher. Moreover, offloading all tasks introduce a huge transmission delay, which increase the term $\omega^T C^{time}$.

Fig. 5 also demonstrates that the existence of the battery achieve more benefits. Using the same DDPG algorithm, the scenario with a battery can achieve higher cumulative reward than that without a battery (i.e., $a_B(t) = 0, \forall t$) . The main two reasons for the difference are: 1) the battery can sell electricity to the grid when the electricity price is high to obtain more gain and 2) the energy stored in the battery can avoid the possible loss of DSM by increasing the value of $l(t)$.

2) Effect of the Number of Smart Devices: In this section, we use four approaches: 1) DDPG; 2) DQN; 3) full-local; and 4) Full-SHOP to demonstrate the scalability of our proposed methods. We test the case where the number of devices is 5, 10, 15, 20 by setting the task workload generated by some devices as 0. The averaged cumulative rewards in Fig. 6.

Intuitively, more devices generates more tasks, leading to more benefits from completing tasks. The DRL methods gradually outperform the full local and full SHOP schemes, while DDPG achieves higher rewards than DQN. We can also see that the rewards obtained by the DRL methods increase almost linearly with the number of devices, while the rewards obtained by the other two methods tend to saturate as the number of devices grows. The reasons are as follows. For the Full local scheme, the losses caused by violating the DSM offset the part of the gains from completing the tasks. For the Full-SHOP method, limited by the cache size of the server, there is an upper limitation on the amount of tasks that the server can handle during each time slot. Therefore, although the number of generated tasks increases, not all of them can be offload and completed at the server. Besides, too many tasks will also cause an increase in $\omega^T C^{time}$.

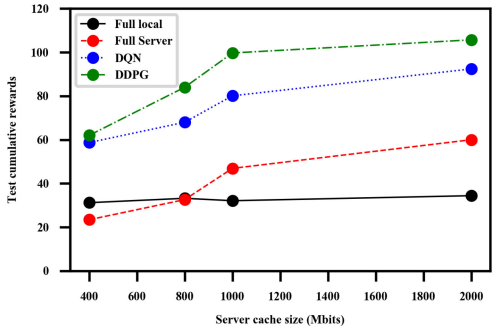


Fig. 7. Testing cumulative rewards of different sever cache size.

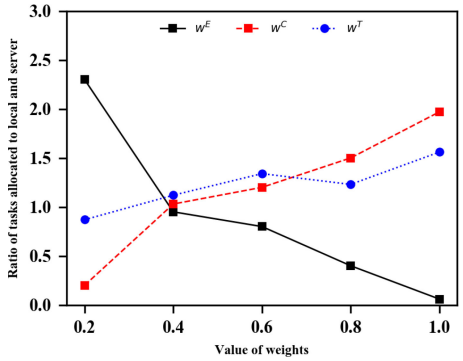


Fig. 8. Task allocation results of different values of weights $\omega^E, \omega^T, \omega^S$.

3) Effect of the Server Cache Size: We then investigate how the server cache size effect the learning performance. We set the server cache size equals to 400Mbits, 800Mbits, 1000Mbits, 2000Mbits. Generally, when the server cache size increases, more tasks can be offloaded and can be executed before expiration, resulting in more rewards. In Fig. 7, we can see that the Full-local method is almost not affected by the server cache size, while other three methods both achieve more rewards with the increase server cache size.

4) Effect of the Cost Weights: We use Fig. 8 to show the effects of trade-off weights ω^E, ω^T and ω^S. We first explore the ratio of the tasks allocated to the edge and to the SHOP with different weights settings. Specifically, we vary ω^E and set $\omega^T = \omega^S = \frac{1-\omega^E}{2}$. We can see that when $\omega^E = 0.2$ is small, the energy cost $\omega^E C^{energy}$ is lower than the cost of time C^{time} and server C^{server}. Our proposed method tends to allocate more task to the edge. With the increase of ω^E, the ratio decreases. The ratio changes with the same trend as ω^S, since when the server cost is large, the algorithm tends to allocate more tasks locally. However, we found that the change of ω^T has little impact on decision-making results. It can be seen that only when ω^T is very small, the algorithm tends to offload more tasks to server. One possible reason is that the designed reward function has already contains the time-tolerance limitation. That is, completing the task as soon as possible leads to more rewards.

The above experimental results cannot directly reflect the impact of the weights on overall performance, so we next carry out experiments to show the cumulative rewards with respect of different weights settings. We consider 3 particular settings: 1) in *env* 1, we set ω^E as 0; 2) in *env* 2, wet set $\omega^E, \omega^T, \omega^S$ are basically equal; and 3) ω^S is set as 0 in *env* 3. The DMS constraints exist in both settings.

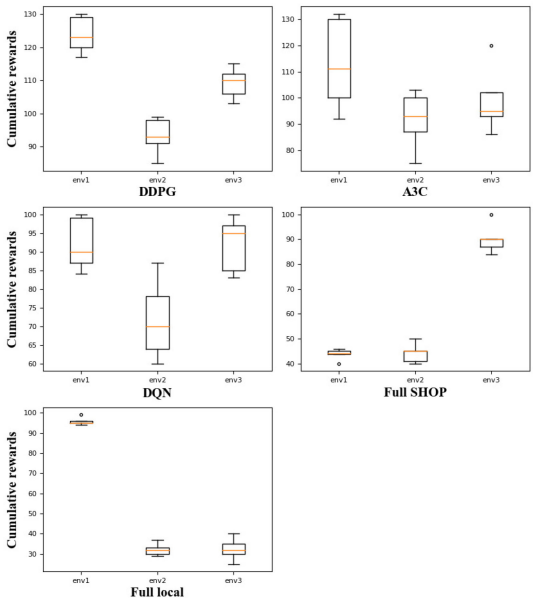


Fig. 9. Testing cumulative rewards of different weights settings.

In *env* 1, the cost of local electricity consumption $\omega^E C^{energy}$ tends to be 0. Therefore, local execution is the better action . Fig. 9 shows that three DRL methods achieve a little more reward than Full local execution, while the DDPG algorithm has the smallest variance and largest mean reward. This indicates that even though local computing can avoid energy cost, the DRL methods chooses to upload some of the tasks to the SHOP server to avoid the DSM penalty. Similarly, in *env* 3, since $\omega^S C^{server} = 0$, the terms $\omega^T C^{time}$ and $\omega^E C^{energy}$ dominate. Thus, the optimal action tends to offload all tasks to the SHOP server. We can see that our method performs better than the Full SHOP execution method, while the Full local execution performs the worst. The Full SHOP execution can minimize the $\omega^E C^{energy}$ but maximize the $\omega^T C^{time}$. DRL algorithms can choose to execute some tasks locally to find the "balance point" minimizing the sum of $\omega^E C^{energy}$ and $\omega^T C^{time}$. *env* 1 and *env* 3 show that our strategy can achieve nearly optimal performance at the extreme situation when energy cost (or SHOP cost) is 0. *env* 2 is more practical and our proposed DDPG strategy achieves similar mean reward with A3C but with smaller variance. Besides, it outperforms DQN and other three heuristic methods.

5) User Side Performance: The above test experiments are analyzed from the perspective of cumulative reward, we further define two components to present user side performance in this subsection:

1) Disappoint coefficient (DC): The value indicates that the loss due to task expired. Let $n_{i,task}$ be the total number of tasks generated by smart devices i over the testing process. Let $n_{i,expired}$ be the total number of time slots that satisfies $e_i(t) = 0, c_i(t) \neq 0$, i.e., $u_{T,i}(c_i(t), e_i(t)) = 0$. Then *DC* can be computed as:

$$DC = \sum_{i}^{n^S} n_{i,expired} / n_{i,task} \tag{36}$$

TABLE III
USER SIDE PERFORMANCE OF DIFFERENT ALGORITHMS

	DDPG	A3C	DQN	Full Local	Full SHOP
DC	0.16	0.12	0.23	0.82	0.62
PC	0.08	0.09	0.13	0.85	0

TABLE IV
IDLE AND WORK POWER OF COMMON ORIGINAL DEVICES

Original Devices	Idel Power	Work Power
Vacuum Cleaner	2.5W	1200W
Water Heater	4.7W	2000W
Microwave Oven	1.6W	1300W
Air Conditioning	2.3W	800W

2) Penalty coefficient (PC): This value indicates the loss due to the violation of DSM signals. We denote n_{DSM} be the total number of times that the DSM signals are generated during test process. And n_V is the total number of time slots that satisfies $d(t) = 0$, $l(t) < 0$, i.e., $u(l(t), d(t)) \neq 0$. Then *PC* can be computed as:

$$PC = n_V / n_{DSM} \tag{37}$$

As shown in Table III, the *DC* for both Full local and Full SHOP are relatively high since it is difficult to complete all tasks with only one computing resource. Three DRL methods obtained smaller *DC* while the A3C perform best. For the Full SHOP scheme, the *PC* is always be 0 since no tasks are performed locally while it is close to 1 for the Full local scheme. Three DRL methods also achieve smaller *PC* while the DDPG and A3C have achieved similar results and are better than DQN.

D. Performance With Real World Data

The real-world hourly electricity price dataset [35] is used to represent dynamic price state. The hourly data file contains load, day-ahead and real-time prices for the ISO New England Control Area in 2016. Specifically, we choose the real time price from 2016/1/1 to 2016/3/31 for testing. The rental price charged by the server on time also comes from the real-world data. We refer to [36], in which a general server with 8 core and 16GB memory is charged by 0.5 USD/hour. We consider a household battery with parameter 200AH and 24V, which can store 4.8kW electricity. We set the charge current as 20A and it can charge 0.48kW (ignoring the conversion loss) for each hour. For the energy consumption of original devices, we refer the following table.

1) Setting: The blue curve in Fig. 10 represents the average value of the hourly electricity price using above dataset. We can roughly see that 18:00-22:00 is the peak period while the minimal electricity price happens from 00:00-6:00. In order to better simulate the real environment, we divide 24 hours into three stages with distinct modes and use red, green, blue regions to illustrate the stages:

Stage 1: In 0:00 - 6:00, we assume that there is no DSM signals. Since most appliances stay idle at night, both the amount of computational intensive tasks and energy generated by the original devices are small.

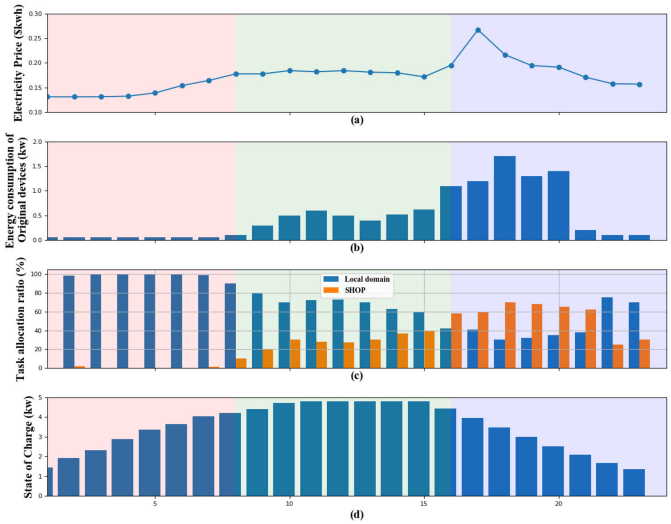


Fig. 10. From top to bottom: (a) electricity price. (b) Energy consumption of original devices. (3) Task allocation percentage of local domain and SHOP. (d) SOC of the battery.

Stage 2: In 6:00 - 16:00, we assume that the DSM signal exists. Both the amount of computational intensive tasks and the energy consumption of original devices increase.

Stage 3: In 16:00 - 24:00, we assume that the DSM signal keeps existing . The amount of intensive tasks and the energy of original devices are more than *Stage* 2 since appliances are more likely to be used during night.

2) Performance Illustration: We use the task allocation percentage and the state of charge to show the efficiency of our methods. In Fig. 10, we can see that the charging and discharging decisions learned by our algorithm are reasonable and intuitive: we charge and discharge to take advantage of the lows and ups of electricity prices while respecting the SOC constraints. We can also see that, in *Stage* 1, almost all tasks are chosen to be computed locally. Starting from *Stage* 2, the number of computing tasks increases, so does the task allocated the SHOP server. In *Stage* 3, we can see that the task allocated of the SHOP server has exceeded local devices. On the one hand, local computing resources are not enough to cope with the large number of computing tasks. Also, the price of electricity has increased significantly at this stage.

VI. CONCLUSION

In this paper, we have studied an integrated smart grid system model for DSM with SHOP. We firstly formulate the task scheduling problem as an MDP problem aiming to maximize the residential user's reward consisting of energy cost, execution time, SHOP server fee and the penalty of DSM. Then we have developed the deep reinforcement learning-based algorithm to solve this problem where we use neural networks to approximate the action-value function and the parameterized optimal action. Experimental results show that our proposed scheme works well and could achieve significant performance gains over other baselines under various environmental parameters.

REFERENCES

[1] T. Li, Y. Xiao, and L. Song, "Deep reinforcement learning based residential demand side management with edge computing," in *Proc. IEEE Int. Conf. Commun. Control. Comput. Technol. Smart Grids (SmartGridComm)*, 2019, pp. 1–6.

[2] X. Fang, S. Misra, G. Xue, and D. Yang, "Smart grid—The new and improved power grid: A survey," *IEEE Commun. Surveys Tuts.*, vol. 14, no. 4, pp. 944–980, 4th Quart., 2012.

[3] Z. Fan *et al.*, "Smart grid communications: Overview of research challenges, solutions, and standardization activities," *IEEE Commun. Surveys Tuts.*, vol. 15, no. 1, pp. 21–38, 1st Quart., 2013.

[4] B. P. Esther and K. S. Kumar, "A survey on residential demand side management architecture, approaches, optimization models and methods," *Renew. Sustain. Energy Rev.*, vol. 59, pp. 342–351, Jun. 2016.

[5] Q. Qdr, "Benefits of demand response in electricity markets and recommendations for achieving them," U.S. Dept. Energy, Lawrence Berkeley Nat. Lab., Berkeley, CA, USA, Rep. 1252, 2006.

[6] R. K. Barik *et al.*, "FogGrid: Leveraging fog computing for enhanced smart grid network," in *Proc. 14th IEEE India Council Int. Conf. (INDICON)*, Dec. 2017, pp. 1–6.

[7] S. Zahoor, N. Javaid, A. Khan, B. Ruqia, F. J. Muhammad, and M. Zahid, "A cloud-fog-based smart grid model for efficient resource utilization," in *Proc. 14th Int. Wireless Commun. Mobile Comput. Conf. (IWCMC)*, Jun. 2018, pp. 1154–1160.

[8] M. Mukherjee *et al.*, "Security and privacy in fog computing: Challenges," *IEEE Access*, vol. 5, pp. 19293–19304, 2017.

[9] N. Abbas, Y. Zhang, A. Taherkordi, and T. Skeie, "Mobile edge computing: A survey," *IEEE Internet Things J.*, vol. 5, no. 1, pp. 450–465, Feb. 2018.

[10] J. Wang, M. Biviji, and W. M. Wang, "Case studies of smart grid demand response programs in North America," in *Proc. ISGT*, Jan. 2011, pp. 1–5.

[11] P. Siano, "Demand response and smart grids—A survey," *Renew. Sustain. Energy Rev.*, vol. 30, pp. 461–478, Feb. 2014.

[12] E. Mocanu *et al.*, "On-line building energy optimization using deep reinforcement learning," *IEEE Trans. Smart Grid*, vol. 10, no. 4, pp. 3698–3708, Jul. 2019.

[13] Z. Wan, H. Li, H. He, and D. Prokhorov, "Model-free real-time EV charging scheduling based on deep reinforcement learning," *IEEE Trans. Smart Grid*, vol. 10, no. 5, pp. 5246–5257, Sep. 2019.

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

[15] J. Li, H. Gao, T. Lv, and Y. Lu, "Deep reinforcement learning based computation offloading and resource allocation for MEC," in *Proc. IEEE Wireless Commun. Netw. Conf. (WCNC)*, 2018, pp. 1–6.

[16] A. Khalili, S. Zarandi, and M. Rasti, "Joint resource allocation and offloading decision in mobile edge computing," *IEEE Commun. Lett.*, vol. 23, no. 4, pp. 684–687, Apr. 2019.

[17] M. Chen and Y. Hao, "Task offloading for mobile edge computing in software defined ultra-dense network," *IEEE J. Sel. Areas Commun.*, vol. 36, no. 3, pp. 587–597, Mar. 2018.

[18] H. A. Alameddine, S. Sharafeddine, S. Sebbah, S. Ayoubi, and C. Assi, "Dynamic task offloading and scheduling for low-latency IoT services in multi-access edge computing," *IEEE J. Sel. Areas Commun.*, vol. 37, no. 3, pp. 668–682, Mar. 2019.

[19] C. Kai, H. Zhou, Y. Yi, and W. Huang, "Collaborative cloud-edge-end task offloading in mobile-edge computing networks with limited communication capability," *IEEE Trans. Cogn. Commun. Netw.*, early access, Aug. 20, 2020, doi: 10.1109/TCCN.2020.3018159.

[20] Z. Kuang, L. Li, J. Gao, L. Zhao, and A. Liu, "Partial offloading scheduling and power allocation for mobile edge computing systems," *IEEE Internet Things J.*, vol. 6, no. 4, pp. 6774–6785, Aug. 2019.

[21] E. Ahvar, A.-C. Orgerie, and A. Lebre, "Estimating energy consumption of cloud, fog and edge computing infrastructures," *IEEE Trans. Sustain. Comput.*, early access, Mar. 18, 2019, doi: 10.1109/TSUSC.2019.2905900.

[22] C. You, Y. Zeng, R. Zhang, and K. Huang, "Asynchronous mobile-edge computation offloading: Energy-efficient resource management," *IEEE Trans. Wireless Commun.*, vol. 17, no. 11, pp. 7590–7605, Nov. 2018.

[23] Z. Chang, Z. Zhou, T. Ristaniemi, and Z. Niu, "Energy efficient optimization for computation offloading in fog computing system," in *Proc. IEEE Glob. Commun. Conf. (GLOBECOM)*, 2017, pp. 1–6.

[24] C. Jiang *et al.*, "Energy aware edge computing: A survey," *Comput. Commun.*, vol. 151, pp. 556–580, Feb. 2020.

[25] Q. Yang and P. Li, "Deep reinforcement learning based energy scheduling for edge computing," in *Proc. IEEE Int. Conf. Smart Cloud (SmartCloud)*, 2020, pp. 175–180.

[26] Y. Chen, Y. Zhang, Y. Wu, L. Qi, X. Chen, and X. Shen, "Joint task scheduling and energy management for heterogeneous mobile edge computing with hybrid energy supply," *IEEE Internet Things J.*, vol. 7, no. 9, pp. 8419–8429, Sep. 2020.

[27] Y. Liu, S. Xie, Q. Yang, and Y. Zhang, "Joint computation offloading and demand response management in mobile edge network with renewable energy sources," *IEEE Trans. Veh. Technol.*, vol. 69, no. 12, pp. 15720–15730, Dec. 2020.

[28] M. Gao *et al.*, "Computation offloading with instantaneous load billing for mobile edge computing," *IEEE Trans. Services Comput.*, early access, May 25, 2020, doi: 10.1109/TSC.2020.2996764.

[29] Y. Wen, W. Zhang, and H. Luo, "Energy-optimal mobile application execution: Taming resource-poor mobile devices with cloud clones," in *Proc. IEEE INFOCOM*, 2012, pp. 2716–2720.

[30] A. R. Khan, A. Mahmood, A. Safdar, Z. A. Khan, and N. A. Khan, "Load forecasting, dynamic pricing and DSM in smart grid: A review," *Renew. Sustain. Energy Rev.*, vol. 54, pp. 1311–1322, Feb. 2016.

[31] C. J. Watkins and P. Dayan, "Q-learning," *Mach. Learn.*, vol. 8, nos. 3–4, pp. 279–292, 1992.

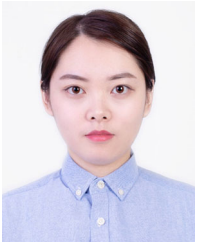
[32] V. Mnih *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, p. 529, 2015.

[33] T. P. Lillicrap *et al.*, "Continuous control with deep reinforcement learning," 2015. [Online]. Available: arXiv:1509.02971.

[34] V. Mnih *et al.*, "Asynchronous methods for deep reinforcement learning," in *Proc. 33rd Int. Conf. Mach. Learn.*, Jun. 2016, pp. 1928–1937. [Online]. Available: http://proceedings.mlr.press/v48/mniha16.html

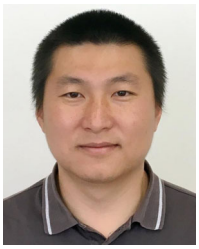
[35] *Hourly Electricity Price Dataset*. Accessed: 2016. [Online]. Available: https://github.com/Electromaxim/electricity_load_forecast

[36] *Huawei Cloud Price*. Accessed: 2021. [Online]. Available: https://www.huaweicloud.com/intl/zh-cn/pricing/index.html#/ecs



Tan Li (Student Member, IEEE) received the B.S. degree from the Central South University, Changsha, China, in 2016, and the M.S. degree from the University of Science and Technology of China, Hefei, China, in 2019. She is currently pursuing the Ph.D. degree with the Department of Computer Science, City University of Hong Kong. Her research interests lie in the edge computing, distributed computing, and machine learning for wireless communication.

Yuanzhang Xiao (Member, IEEE) received the B.E. and M.E. degrees in electronic engineering from Tsinghua University in 2006 and 2009, respectively, and the Ph.D. degree in electrical engineering from UCLA in 2014. He is an Assistant Professor with the University of Hawaii at Mānoa. He was a Postdoctoral Fellow with Northwestern University from 2015 to 2017. His research interests include game theory, mechanism design, and optimization, with applications in socio-technological networks, smart grids, and wireless communication.



Linqi Song (Member, IEEE) received the B.S. and M.S. degrees in electronic engineering from Tsinghua University, China, and the Ph.D. degree in electrical engineering from University of California at Los Angeles (UCLA), Los Angeles, where he was a Postdoctoral Scholar with the Electrical and Computer Engineering Department. He is an Assistant Professor with the Department of Computer Science, City University of Hong Kong. He received a UCLA Fellowship for his graduate studies. His research interests are in algorithms, big data, and machine learning.