*Intelligent and Converged Networks*
**2020, 1(2): 181–198**

**ISSN 2708-6240**
**DOI: 10.23919/ICN.2020.0014**

# Deep reinforcement learning for dynamic computation offloading and resource allocation in cache-assisted mobile edge computing systems

**Samrat Nath and Jingxian Wu***

**Abstract:** Mobile Edge Computing (MEC) is one of the most promising techniques for next-generation wireless communication systems. In this paper, we study the problem of dynamic caching, computation offloading, and resource allocation in cache-assisted multi-user MEC systems with stochastic task arrivals. There are multiple computationally intensive tasks in the system, and each Mobile User (MU) needs to execute a task either locally or remotely in one or more MEC servers by offloading the task data. Popular tasks can be cached in MEC servers to avoid duplicates in offloading. The cached contents can be either obtained through user offloading, fetched from a remote cloud, or fetched from another MEC server. The objective is to minimize the long-term average of a cost function, which is defined as a weighted sum of energy consumption, delay, and cache contents' fetching costs. The weighting coefficients associated with the different metrics in the objective function can be adjusted to balance the tradeoff among them. The optimum design is performed with respect to four decision parameters: whether to cache a given task, whether to offload a given uncached task, how much transmission power should be used during offloading, and how much MEC resources to be allocated for executing a task. We propose to solve the problems by developing a dynamic scheduling policy based on Deep Reinforcement Learning (DRL) with the Deep Deterministic Policy Gradient (DDPG) method. A new decentralized DDPG algorithm is developed to obtain the optimum designs for multi-cell MEC systems by leveraging on the cooperations among neighboring MEC servers. Simulation results demonstrate that the proposed algorithm outperforms other existing strategies, such as Deep Q-Network (DQN).

**Key words:** Mobile Edge Computing (MEC); caching; computation offloading; resource allocation; Deep Reinforcement Learning (DRL); Deep Deterministic Policy Gradient (DDPG); multi-cell

## 1 Introduction

With the advancement of Smart Mobile Devices (SMDs) and the emergence of the Internet of Things (IoTs), many new types of mobile applications, such as virtual and augmented reality, face and gesture recognition, online interactive gaming, etc., are rapidly gaining shares in the mobile computing market. Typically, these applications are computation-intensive and delay-sensitive, and require high energy consumption. However, due to limited battery life and computational capacity (processing speed) of an SMD, it is often very difficult for the SMD to meet the requirements and Quality of Experience (QoE) of these mobile applications. To bridge the gap between the resource-limited SMDs and the computation-intensive and delay-sensitive applications, Mobile Edge Computing (MEC) has recently emerged as a promising technology[1].

• Samrat Nath is with Walmart Inc., Bentonville, AR 72716, USA. E-mail: samrat.nath@walmart.com.
• Jingxian Wu is with the Department of Electrical Engineering, University of Arkansas, Fayetteville, AR 72701, USA. E-mail: wuj@uark.edu.
∗ To whom correspondence should be addressed.
Manuscript received: 2020-05-30; revised: 2020-09-15; accepted: 2020-10-22

The conventional Mobile Cloud Computing (MCC) system relies on remote public clouds, such as Amazon Web Services and Microsoft Azure. Usually, the cloud servers are spatially far from the SMDs, which causes high transmission delay. Unlike MCC, MEC augments the computational capability at the edge of mobile networks by deploying densely distributed high-performance servers close to the Mobile Users (MUs)[1]. It enables MUs to offload computing tasks to the MEC server connected with a Base Station (BS) via the wireless network. Through the computation offloading of delay-sensitive and computation-intensive tasks, MUs can significantly reduce the computation latency and energy consumption, and thus improving the QoE of mobile applications. Hence, the interests on computation offloading in MEC systems have been growing rapidly.

The efficiency of computation offloading depends largely on how the limited computation, power, and communication resources are managed in an MEC system. Various computation offloading algorithms with different design objectives and resource allocation schemes have been studied extensively in the literatures[2–16]. In Ref. [3], the joint optimization problem of computation offloading and resource allocation is solved by applying the Alternating Direction Method of Multipliers (ADMM). The problem studied in Ref. [9] considers a multi-user multi-channel MEC system, where the goal is to jointly minimize the energy consumption, delay, and deadline penalty of all the users and determine the optimal offloading, computational resource, and channel allocation. An online algorithm based on Lyapunov optimization is developed in Ref. [13] for jointly managing the radio and computational resources of multi-user MEC systems. However, most of the above-mentioned studies do not consider dynamic task arrivals or/and dynamic channel conditions. In practice, the MEC systems have time-varying task arrivals and stochastic channel conditions.

Although in the MEC framework, the MUs are located in proximity to the MEC server, they still experience delay and consume energy due to computation offloading. In order to further improve the performance of MEC systems, content caching or task caching has been proposed as a promising technique[2, 4, 12, 14–16]. Task caching usually refers to caching task applications and related data in the MEC server. If a task is cached, it can be executed directly in the MEC server, whereas uncached tasks either need to be executed locally at the MU or need to be offloaded to the server before execution. Thus, caching popular tasks in the MEC server can further reduce the delay and energy consumption by avoiding unnecessary duplicates in data transmissions. An MEC architecture that combines caching, cooperative task offloading, and security service assignment is proposed in Ref. [4] for multi-cell IoT networks, and the architecture is designed to obtain both stringent security protection and energy savings. The works in Refs. [14–16] study the problem of dynamic caching with the consideration of time-varying popularity of the tasks. However, these works only consider the storage capacity of the MEC server and assume that the server has enough computational capacity to support all the tasks that are offloaded. This assumption is impractical, because both the storage and computational capacity of the MEC server are limited. Both storage and computation constraints are considered in Refs. [2, 12]. However, they do not consider time-varying system dynamics and provide only offline solutions. Therefore, it is imperative to study the joint problem of online caching, offloading, and resource allocation policy in dynamic MEC systems.

All the above-mentioned works formulate the complicated joint caching, computation offloading, and resource allocation as optimization problems, which are generally non-convex and very challenging to solve. In order to tackle this challenge, a Reinforcement Learning (RL)-based algorithm can be an effective approach that can provide online solutions to the complicated sequential decision-making problems without requiring any prior knowledge of the system[17]. Moreover, with the explosive growth of interest in the Deep Neural Networks (DNNs), researchers have recently started adopting Deep Reinforcement Learning (DRL) algorithms to solve these problems[5–11]. DRL can be considered as an advanced RL technique implemented with DNNs. By exploiting the function approximation property of DNNs, DRL can provide solutions to large-scale problems, where the conventional RL

methods become infeasible[18]. A DRL-based State-Action-Reward-State-Action (SARSA) algorithm is proposed in Ref. [5] to resolve the classical problem of task offloading and resource allocation in multi-cell MEC systems. The solutions proposed in Refs. [6–8] utilize the Deep Q-Network (DQN)[19], while the Deep Deterministic Policy Gradient (DDPG) algorithm[20] is adopted in Refs. [9–11]. However, the work in Ref. [9] assumes the channel conditions to be quasi-static. Despite considering dynamic channel conditions and stochastic task arrivals, the problem in Ref. [10] does not consider caching, while no constraint on the computational capacity of MEC server is considered in Ref. [11].

Even though task caching and computation offloading can reduce the delay and energy consumption of MUs, how to design the optimal strategy for caching, offloading, and resource allocation in a dynamic MEC system is a very challenging problem when considering the time-varying nature of the system, the heterogeneity of the tasks, and the limited resources at the MEC server. Therefore, in this paper, we propose to develop an online DRL-based scheme for dynamic caching, computation offloading, and resource allocation in a resource-constrained multi-user MEC system by addressing four key questions:

(1) Whether a given task should be cached in the MEC server?

(2) Whether a given uncached task should be executed locally at an MU or offloaded to the MEC?

(3) How much transmission power should be allocated to a given MU for task offloading? and

(4) How much computational resources should be allocated by the MEC server for a given task?

In the proposed MEC framework, the time is divided into slots of equal length, and the channel conditions, task popularity, and task arrivals are assumed to be time-varying and stochastic. At the beginning of each time slot, the caching decision, offloading decision, power allocation, and computational resource allocation are determined centrally by the BS, and then the results are forwarded to the MU. Our objective is to develop an online DRL-based solution for efficient caching, computation offloading, and resource allocation. The key contributions of this paper are summarized as follows.

- The optimum designs of both single-cell and multi-cell MEC systems are studied in this paper by solving the joint problem of task caching, offloading, and resource allocation in a dynamic setting, which considers time-varying stochastic system conditions, such as channel conditions, task popularity, and task arrivals. To the best of our knowledge, no prior works in the literature consider such a comprehensive setup under stochastic system conditions.
- The problem is solved by adopting a DDPG-based method, which can deal with the continuous space of optimization variables, yet many other DRL-based solutions, such as DQN, rely on discretization of the continuous state and action space. Discretization of optimization variables results in loss of precision. Simulation results demonstrate the superiority of proposed solution against other existing strategies, such as DQN.
- In the design of the multi-cell MEC system, a new decentralized DDPG solution is developed to leverage the cooperations among neighboring MEC servers. In the decentralized design, the MEC servers can communicate and share resources among each other to further improve the performance of the system. Simulation results have shown that the proposed decentralized DDPG significantly outperform non-cooperative DDPG, and it can achieve a performance that is very similar to centralized DDPG but with a much lower complexity and overhead.

The rest of this paper is organized as follows. The system model and problem formulation for the dynamic caching, computation offloading, and resource allocation of the single-cell MEC system are presented in Section 2. In Section 3, some preliminaries on RL and DRL are introduced and the design of the DRL-based algorithm for the single-cell MEC system is proposed. Section 4 extends the work to multi-cell MEC network, where a cooperative decentralized DRL-based solution is presented. Simulation results are illustrated in Section 5. Section 6 concludes this paper. A list of notations is summarized in Table 1.

**Table 1 List of notations.**

| Notation | Definition |
|---|---|
| $\mathcal{T}(\mathcal{K})$ | Set of discrete time slots (tasks) |
| $\mathcal{N}(\mathcal{S})$ | Set of MUs (BSs) |
| $D(F)$ | Cache (computational) capacity of the MEC server |
| $b_k(d_k)$ | Data size (computational requirement) of $k$-th task |
| $\boldsymbol{c}_t(\boldsymbol{x}_t)$ | Caching (computation offloading) decision vector at slot $t$ |
| $\boldsymbol{b}_t(\boldsymbol{f}_t)$ | Transmission power (computational resource) allocation vector |
| $\mathcal{K}_t^o(\mathcal{K}_{t+1}^c)$ | Set of tasks offloaded to (cached in) the MEC server at slot $t$ |
| $\boldsymbol{k}_t$ | Tasks requested by all the MUs at the start of slot $t$ |
| $\boldsymbol{H}_t$ | Channel matrix at slot $t$ |
| $\mathcal{N}_t^c$ | Set of MUs with requested task available in the cache |
| $\bar{J}(\bar{J}')$ | Long-term average cost of single-cell (multi-cell) MEC system |
| $\omega_n(\omega_c)$ | Weight for delay-energy tradeoff (fetching cost) |
| $\mathcal{R}(\mu)$ | Reward (policy) function |
| $\boldsymbol{\theta}^Q(\boldsymbol{\theta}^\mu)$ | Neural network weights of critic (actor) network |
| $\lvert\mathcal{N}^s\rvert$ | Number of MUs associated with the $s$-th BS |

## 2 System model and problem formulation for single-cell MEC network

Consider a multi-user MEC system that consists of one BS with $M$ antennas, one MEC server, and a set of $N \leqslant M$ single-antenna MUs denoted by $\mathcal{N} = \{1, 2, \ldots, N\}$. The BS is connected directly with the MEC server with a cache size of $D$ (in bits) and the computing capacity of $F$ (in CPU cycles per second). A discrete-time model is adopted for the MEC system, where the time is divided into slots with equal length $T_s$ (in seconds) and indexed by $\mathcal{T} = \{0, 1, \ldots\}$. The system model is shown in Fig. 1.

### 2.1 Task model

Assume there are $K$ heterogeneous tasks denoted by the set $\mathcal{K} \triangleq \{1, 2, \ldots, K\}$. Each task $k \in \mathcal{K}$ is characterized by two parameters; $d_k$ (in cycles per second) denotes the amount of computing resource required for the task and $b_k$ (in bits) denotes the size of computation input data, such as program codes and input parameters. One example of application scenario for such tasks is Virtual Reality (VR), where the MUs need to execute various types of computation-intensive VR application tasks, such as rendering scenes, recognizing and tracking objects, etc.

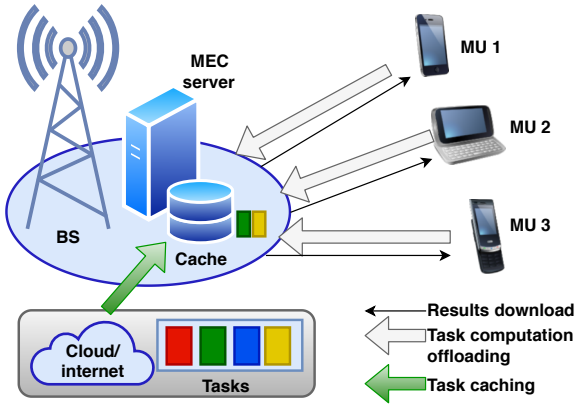



**Fig. 1 System model for single-cell MEC network.**

The number of MUs is assumed to be more than the number of tasks ($N \geqslant K$). This is because some computing tasks have higher popularity (e.g., rendering scenes in VR), which are repeatedly requested and executed multiple times[2]. At the beginning of each time slot, each MU requests one task from the set $\mathcal{K}$, where multiple users may simultaneously request a particular task. Denote $k_t^n \in \mathcal{K}$ as the task requested by the $n$-th MU at the start of slot $t$ and $\boldsymbol{k}_t \triangleq [k_t^1, \ldots, k_t^N]^{\mathrm{T}}$ as the task request vector for all MUs.

The popularity of each task $\phi_{k,t}$ is dynamic and follows Zipf distribution[21]. The popularity profile vector is defined as $\boldsymbol{\phi}_t \triangleq [\phi_{1,t}, \ldots, \phi_{K,t}]^{\mathrm{T}}$. Given that the popularity rank of task $k$ during slot $t$ is $z_{k,t} \in \mathcal{K}$, the popularity of the corresponding task can be expressed as

$$\phi_{k,t} = \frac{z_{k,t}^{-\eta}}{\sum_{l=1}^{K} z_{l,t}^{-\eta}} \tag{1}$$

where the Zipf parameter $\eta \geqslant 0$ controls the skewness of popularity. Specifically, $\eta = 0$ yields a uniform spread of popularity among the tasks, and the popularity difference among the tasks becomes larger with a larger $\eta$.

To model the time-varying nature of task popularity, the popularity profile $\boldsymbol{\phi}_t$ is modeled by a $V$-state Markov chain[22], represented by $V$ different popularity profiles $\boldsymbol{\phi}^{(1)}, \ldots, \boldsymbol{\phi}^{(V)}$. Each profile is modeled by Zipf distributions with parameters $\eta_v$. So, at each time slot $t$, the popularity profile $\boldsymbol{\phi}_t$ will follow one of these $V$ states and each task $k \in \mathcal{K}$ will be assigned popularity

ranks $z_{k,t}$ randomly.

### 2.2 Caching model

Based on the definition of task caching, the application programs are all cached in the MEC server, and a caching policy is used to decide whether to cache the input data of the computing task in the MEC server[2, 12]. Task caching can reduce task latency and energy consumption of MU, because there is no need for transmitting the data of a task already available in the cache. However, how to assign the limited caching capacity is a challenging problem, which depends on the dynamic popularity ($\phi_{k,t}$), size ($b_k$), and computational requirement ($d_k$) of each task.

Denote $c_{k,t} \in \{0, 1\}$ as the binary variable that represents the caching decision for task $k$ in slot $t$ and denote $\boldsymbol{c}_t = [c_{1,t}, \ldots, c_{K,t}]^{\mathrm{T}}$ as the corresponding decision vector for all tasks. If $c_{k,t} = 1$, the input data for computing task $k$ is cached in the MEC server at the end of slot $t$ and the corresponding data can be utilized in the next time slot $t + 1$ for executing the task entirely at the server. Consequently, the system will not experience any delay and will not incur any energy cost corresponding to the data transmission from MUs to the server for that particular task. Therefore, the users' QoE will be significantly improved. However, all the tasks cannot be cached due to the limited storage size of the MEC server. Therefore, the caching decision variable must meet the following constraint:

$$\sum_{k=1}^{K} \mathbf{1}(c_{k,t} = 1) b_k \leqslant D, \quad \forall t \in \mathcal{T} \tag{2}$$

where $\mathbf{1}(\mathcal{E})$ is the indicator function with $\mathbf{1}(\mathcal{E}) = 1$ if the event $\mathcal{E}$ is true and 0 otherwise.

Moreover, it is assumed that the input data of all the tasks are available in a remote cloud (as shown in Fig. 1). The BS can download/fetch the data for any task from the cloud via a back-haul link by incurring fetching cost. In addition, the MEC can also cache the data of one or more offloaded tasks in the current slot for future use without incurring the fetching cost. Given the caching decision vector $\boldsymbol{c}_t$ at slot $t$ and $\boldsymbol{c}_{t-1}$ at slot $t - 1$, the overall fetching cost of the MEC system associated with the caching decision at slot $t$ is defined as

$$C_t = \sum_{k=1}^{K} \mathbf{1}(c_{k,t-1} = 0, c_{k,t} = 1, k \notin \mathcal{K}_t^o) g_k \tag{3}$$

where $\mathcal{K}_t^o$ is the set of tasks offloaded to the MU at the $t$-th slot, and $g_k$ (in bits) is the fetching cost for downloading the input data for task $k$ from the remote cloud server. Naturally, $g_k$ depends on the size of task data $b_k$.

Define $\mathcal{K}_t^c \subset \mathcal{K}$ as the set of tasks that have been cached during time slot $t - 1$ and are available for utilization at the beginning of slot $t$, i.e., $\mathcal{K}_t^c \triangleq \{k : c_{k,t-1} = 1\}$. If a task requested by the $n$-th MU is not cached, i.e., $k_t^n \notin \mathcal{K}_t^c$, either the task needs to be executed in the MU locally, or the task data need to be uploaded to the MEC server for remote execution. Details regarding task execution and computation are presented in Subsection 2.4.

### 2.3 Communication model

The BS and the MUs form a multi-user Multiple-Input Multiple-Output (MIMO) system, with $M$ antennas at the BS serving $N$ single-antenna MUs. The channel conditions between the BS and MUs are described by the $M \times N$ channel matrix $\boldsymbol{H}_t = [\boldsymbol{h}_{1,t}, \ldots, \boldsymbol{h}_{N,t}]$, where $\boldsymbol{h}_{n,t} \in \mathbf{C}^{M \times 1}$ is the channel vector of the $n$-th MU. In order to characterize the temporal channel correlation between consecutive slots, the Gaussian Markov block fading autoregressive model[23] is employed. The channel vector for the $n$-th MU can be expressed as

$$\boldsymbol{h}_{n,t} = \rho_n \boldsymbol{h}_{n,t-1} + \sqrt{1 - \rho_n^2} \boldsymbol{e}_t \tag{4}$$

where $\rho_n$ is the normalized channel correlation coefficient for the $n$-th MU, the error vector $\boldsymbol{e}_t \in \mathbf{C}^{M \times 1}$ is uncorrelated with $\boldsymbol{h}_{n,t}$, and it is complex Gaussian distributed with zero mean and covariance matrix $\sigma_e^2 \boldsymbol{I}_M$, with $\boldsymbol{I}_M$ being a size $M$ identity matrix. Here, $\sigma_e^2 \triangleq h_0 (d_0/d_m)^{\beta}$, where $h_0$ is the path-loss at reference distance $d_0$, $d_m$ is the maximum coverage radius of the BS, and $\beta$ is the path-loss exponent. Moreover, $\rho_n = J_0(2\pi f_n^d T_s)$ according to Jake's fading spectrum, where $f_n^d$ is the Doppler frequency of the $n$-th MU, $T_s$ is the slot duration, and $J_0(\cdot)$ is the zero-order Bessel function of the first kind[24].

Denote the transmission power of the $n$-th MU at the $t$-th slot as $p_{n,t} \in [0, P_n^{\max}]$, where $P_n^{\max}$ is the

maximum transmission power of the $n$-th MU. The BS manages the uplink transmissions of multiple single-antenna MUs by adopting the linear detection algorithm Zero-Forcing (ZF)[11,25]. With the ZF detector at the BS, the Signal-to-Interference-plus-Noise Ratio (SINR) for the signal from the $n$-th MU is

$$\gamma_{n,t} = \frac{p_{n,t}}{\sigma^2[(\sqrt{\boldsymbol{P}_t}^{\mathrm{T}}\boldsymbol{H}_t^{\mathrm{H}}\boldsymbol{H}_t\sqrt{\boldsymbol{P}_t})^{-1}]_{nn}} \tag{5}$$

where $\sigma^2$ is the noise power, $\boldsymbol{P}_t = \mathrm{diag}\{\boldsymbol{b}_t\}$ is a diagonal matrix with $\boldsymbol{b}_t = [p_{1,t}, \ldots, p_{N,t}]^{\mathrm{T}}$ on its main diagonal, the operators $\boldsymbol{A}^{\mathrm{T}}$ and $\boldsymbol{A}^{\mathrm{H}}$ represent the matrix transpose and Hermitian operations, respectively, and $[\boldsymbol{A}]_{mn}$ denotes the $(m,n)$-th element of the matrix $\boldsymbol{A}$. For those MUs that do not offload in a given slot, their transmission power in that slot will be set to 0. The transmission data rate from the $n$-th MU to the BS at slot $t$[11] can be expressed as

$$r_{n,t} = W\log_2(1+\gamma_{n,t}) \tag{6}$$

where $W$ is the system bandwidth.

### 2.4 Computation model

Define $\mathcal{N}_t^c$ as the set of MUs that have requested the tasks at time slot $t$, with the requested tasks available in the cache of the MEC server, i.e., $\mathcal{N}_t^c \triangleq \{n : k_t^n \in \mathcal{K}_t^c\}$. Similarly, define $\mathcal{N}_t = \mathcal{N} \setminus \mathcal{N}_t^c$ as the set of MUs with requested tasks at time slot $t$ unavailable in the cache. The computation tasks for the MUs belonging to the set $\mathcal{N}_t^c$ are executed in the MEC server during slot $t$ by default. Thus the offloading decision only needs to be performed for $n \in \mathcal{N}_t$. Denote $x_{n,t} \in \{0,1\}$ as the computation offloading decision variable of the $n$-th MU at slot $t$, where $n \in \mathcal{N}_t$. Specifically, if $x_{n,t} = 0$, the $n$-th MU decides to execute its task locally, and if $x_{n,t} = 1$, the $n$-th MU decides to offload the data of its current computation task to the MEC server via the wireless link. The computation offloading decision vector for all MUs is denoted by $\boldsymbol{x}_t \triangleq [x_{n,t}]_{n\in\mathcal{N}_t}$.

#### 2.4.1 Local execution

In the local execution approach, the $n$-th MU executes its computation task $k_t^n$ locally using its own CPU. Denote $f_n^l$ as the computation capability (in CPU cycles per second) of the $n$-th MU. Computational capabilities may differ across various MUs. The computation time of task $k_t^n$ by local execution can then be expressed as

$$T_{n,t}^l = \frac{d_{k_t^n}}{f_n^l} \tag{7}$$

The corresponding energy consumption is

$$E_{n,t}^l = \zeta_n d_{k_t^n} \tag{8}$$

where the coefficient $\zeta_n$ denotes the energy consumption per CPU cycle, which depends on the chip architecture at the MU. In this paper, we set $\zeta_n = 10^{-27}(f_n^l)^2$ according to Ref. [26].

#### 2.4.2 MEC server execution

In this approach, the MEC server executes the computation task on behalf of the MU. This approach consists of three steps. First, the $n$-th MU uploads its task data of size $b_{k_t^n}$ to the BS through the wireless channel, and the BS forwards that data to the MEC server. Second, the MEC server allocates part of its computational resources to execute the task. Finally, the MEC server returns the execution results of the task to the $n$-th MU.

In the first step, the transmission delay incurred by task offloading by the $n$-th MU during slot $t$ can be computed as

$$T_{n,t}^x = \frac{b_{k_t^n}}{r_{n,t}} \tag{9}$$

where $r_{n,t}$ is the uplink data rate of the $n$-th MU as shown in Eq. (6). The corresponding energy consumption due to transmission is expressed as

$$E_{n,t}^x = p_{n,t}T_{n,t}^x = \frac{p_{n,t}b_{k_t^n}}{r_{n,t}} \tag{10}$$

In the second step during task execution, the processing delay incurred by the MEC server is computed as

$$T_{n,t}^p = \frac{d_{k_t^n}}{f_{n,t}} \tag{11}$$

where $f_{n,t}$ denotes the computational resource (in CPU cycles per second) allocated by the MEC server to the $n$-th MU during slot $t$. During this step, the $n$-th MU is assumed to be in idle state and the energy consumption of the MU is considered as negligible. The energy consumption of the MEC server is not considered either.

Denote $\boldsymbol{f}_t \triangleq [f_{1,t}, \ldots, f_{N,t}]^{\mathrm{T}}$ as the MEC computational resource allocation vector for all the MUs. The total amount of allocated resource can not

exceed the entire computational resource at the MEC server, i.e.,

$$\sum_{n=1}^{N} \mathbf{1}(x_{n,t} \neq 0 \vee n \in \mathcal{N}_t^c) f_{n,t} \leqslant F, \quad \forall t \in \mathcal{T} \quad (12)$$

where $\vee$ represents the logical "or" operation. It is worth noting that, the MEC server will not allocate any computational resource to an MU unless it offloads its computation task to the server or the corresponding task data are available in the server cache.

In the final step, the MU downloads the output data from the MEC server. In general, the size of the computation output data is much smaller than that of the computation input data for many applications. Besides, the download data rate is in general much higher compared to the uplink data rate. Hence, similar to many studies[27–29], the delay and energy consumption during the final step are not considered in this paper.

### 2.5 Problem formulation

Given the computation offloading decision vector $\boldsymbol{x}_t$, the energy consumption and computation delay for the $n$-th MU can be calculated, respectively, as

$$E_{n,t} = \mathbf{1}(n \notin \mathcal{N}_t^c)[\mathbf{1}(x_{n,t}=1)E_{n,t}^x + \mathbf{1}(x_{n,t}=0)E_{n,t}^l] \quad (13)$$

$$T_{n,t} = \mathbf{1}(n \notin \mathcal{N}_t^c)[\mathbf{1}(x_{n,t}=1)(T_{n,t}^x + T_{n,t}^p) + \mathbf{1}(x_{n,t}=0)T_{n,t}^l] + \mathbf{1}(n \in \mathcal{N}_t^c)T_{n,t}^p \quad (14)$$

The overall cost of all MUs in the MEC system during slot $t$ is defined as

$$J_t = \sum_{n=1}^{N} E_{n,t} + \sum_{n=1}^{N} \omega_n T_{n,t} + \omega_c C_t \quad (15)$$

where $\omega_n$ (in W) denotes the weight parameter associated with the delay at the $n$-th MU, and $\omega_c$ (in J/bit) denotes the weight for the fetching cost associated with caching. The weigh parameters $\omega_n$ and $\omega_c$ control the tradeoff among delay-energy and fetching cost, respectively. Different MUs might have different requirements regarding the task execution delay. For example, for MUs prioritizing faster execution, $\omega_n$ can be set to a large value. Similar types of weighted system cost formulation have also been observed in the literatures[30–33].

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

$$\bar{J} = \mathbb{E}\left[\lim_{|\mathcal{T}|\to\infty} \frac{1}{|\mathcal{T}|} \sum_{t\in\mathcal{T}} J_t\right] \quad (16)$$

where $\mathbb{E}(\cdot)$ denotes mathematical expectation. The optimization problem is formulated as follows.

**P1:** $\min_{\boldsymbol{c},\boldsymbol{x},\boldsymbol{b},\boldsymbol{f}} \bar{J}$

s.t. C1: $c_{k,t} \in \{0,1\},\ \forall k \in \mathcal{K},\ \forall t \in \mathcal{T}$;

C2: $x_{n,t} \in \{0,1\},\ \forall n \in \mathcal{N}_t,\ \forall t \in \mathcal{T}$;

C3: $p_{n,t} \leqslant P_n^{\max},\ \forall n \in \{n : x_{n,t} = 1\},\ \forall t \in \mathcal{T}$;

C4: $\sum_{n=1}^{N} \mathbf{1}(x_{n,t} = 1 \vee n \in \mathcal{N}_t^c) f_{n,t} \leqslant F, \forall t \in \mathcal{T}$;

C5: $\sum_{k=1}^{K} \mathbf{1}(c_{k,t} = 1) b_k \leqslant D,\ \forall t \in \mathcal{T}$;

C6: $T_{n,t} \leqslant T_s,\ \forall n \in \mathcal{N},\ \forall t \in \mathcal{T}$.

Here, C3 represents the maximum transmission power constraint imposed on each MU that offloads computation, C4 indicates that the total amount of allocated resources can not exceed the total computational resource at the MEC server, C5 specifies that the total amount of cached data cannot exceed the cache size at the MEC server, and C6 represents the constraint that each MU must execute its task either locally or in the MEC server within one time slot.

The optimization is performed with respect to the binary caching decision vector $\boldsymbol{c} \in \mathcal{B}^K$, the binary offloading decision vector $\boldsymbol{x} \in \mathcal{B}^{|\mathcal{N}_t|}$, the power allocation vector $\boldsymbol{b} \in \mathcal{P}^N$, and the MEC computation resource allocation vector $\boldsymbol{f} \in \mathcal{F}^N$, where $\mathcal{B} = \{0,1\}$, $|\mathcal{N}_t|$ is the cardinality of the set $\mathcal{N}_t$, $\mathcal{P} = \{u \in \mathbf{R} | 0 < u \leqslant P_n^{\max}\}$ is the set of real numbers, $\mathcal{F} = \{v \in \mathbf{R} | 0 < v \leqslant F\}$, and $\mathbf{R}$ is the set of real numbers.

The optimal solution to **P1** requires complete information regarding the mathematical models of the system, such as the statistical distributions of the requests and channel conditions. Such information is in general not available in a practical system. Moreover, **P1** is a mixed-integer nonlinear programming, which is challenging to solve even with all the statistical distributions. One feasible approach to overcome these challenges is to design an online solution that can efficiently make the decisions regarding caching,

computation offloading, and resource allocation in real-time through interactions with the system. Therefore, instead of applying conventional optimization methods to solve the Non-deterministic Polynomial (NP)-hard problem **P1**, we propose a DRL-based method to find the optimal $\boldsymbol{c}, \boldsymbol{x}, \boldsymbol{b}$, and $\boldsymbol{f}$.

## 3 DRL-based solution for dynamic caching, computation offloading, and resource allocation

DRL can be considered as a combination of DNN and RL. In order the solve **P1** with a DRL-based method, we will first reformulate **P1** under the RL framework. Then the details of the proposed DRL-based solution for the single-cell MEC network are presented.

### 3.1 RL framework

The RL framework is usually defined based on Markov Decision Process (MDP) with the underlying Markov property, which states that the evolution of the Markov process in the future depends only on the present state and does not depend on the past history. MDP can be solved by using classical dynamic programming. One of the main differences between dynamic programming and RL is that the latter does not require knowledge of the underlying mathematical model of the MDP, such as the Markovian transition probabilities. The RL can implicitly learn the underlying model by interacting with the environment. Generally, the RL framework is well-suited for providing online solutions to complicated sequential decision-making problems and near-optimal solutions to large-scale MDPs where exact dynamic programming methods become infeasible[17].

The RL framework consists of an agent, an environment, and three key elements: a set of possible states $\mathcal{S}'$, a set of available actions $\mathcal{A}$, and a reward function $\mathcal{R} : \mathcal{S}' \times \mathcal{A} \to \mathbf{R}$. The RL agent continually learns and makes decisions through the interactions with the environment in discrete time steps. In each time step $t$, the agent observes the state of the environment $\boldsymbol{s}_t \in \mathcal{S}'$ and takes an action $\boldsymbol{a}_t \in \mathcal{A}$. The agent's behavior is defined by a policy. In this paper, we consider a deterministic policy $\mu$, which deterministically maps a state to a specific action, i.e., $\mu : \mathcal{S}' \to \mathcal{A}$.

After executing the action, the environment returns a scalar reward $r_t = \mathcal{R}(\boldsymbol{s}_t, \boldsymbol{a}_t)$ and makes a transition of state from $\boldsymbol{s}_t$ to $\boldsymbol{s}_{t+1}$.

The infinite-horizon discounted return is defined as the sum of all rewards ever obtained by the agent, but discounted by how far off in the future they are obtained as

$$R(\xi) = \sum_{t=0}^{\infty} \gamma^t r_t \tag{17}$$

where $\gamma \in [0, 1]$ is the discount factor, and the trajectory $\xi = (\boldsymbol{s}_0, \boldsymbol{a}_0, \boldsymbol{s}_1, \boldsymbol{a}_1, \ldots)$ is a sequence of states and actions leading to the sequence of rewards $\{r_t\}_{t=0}^{\infty}$. The action-value function $Q^{\mu}(\boldsymbol{s}, \boldsymbol{a})$ (also known as Q-function) represents the expected return under the policy $\mu$ with $\boldsymbol{s}$ as the initial state and $\boldsymbol{a}$ as the initial action,

$$Q^{\mu}(\boldsymbol{s}, \boldsymbol{a}) = \mathbb{E}[R(\xi)|\boldsymbol{s}_0 = \boldsymbol{s}, \boldsymbol{a}_0 = \boldsymbol{a}] \tag{18}$$

where the expectation is performed with respect to all randomness in the environment.

The goal of the RL agent is to learn the optimal policy $\mu^*$ that chooses the optimal action greedily in state $\boldsymbol{s}$, such that

$$\mu^*(\boldsymbol{s}) = \underset{\boldsymbol{a}}{\operatorname{argmax}}\ Q^*(\boldsymbol{s}, \boldsymbol{a}) \tag{19}$$

where $Q^*(\boldsymbol{s}, \boldsymbol{a})$ is the optimal Q-function, that is, the Q-function obtained by following the optimum policy.

In order to interpret problem **P1** in the RL framework, we define the key elements according to the system model as follows.

#### 3.1.1 State

The state of a system is considered as a set of parameters that can be used to describe the system. Based on the system model presented in this paper, the system state at an arbitrary time slot $t$ is defined as

$$\boldsymbol{s}_t \triangleq \{\boldsymbol{k}_t, \boldsymbol{H}_t, \boldsymbol{c}_{t-1}\} \tag{20}$$

where $\boldsymbol{k}_t$ is the task request vector, $\boldsymbol{H}_t$ is the channel matrix, and $\boldsymbol{c}_{t-1}$ is the caching decision in the previous slot, which in turn represents the status of the MEC server cache at the beginning of current time slot. The randomness in the system is governed by the state variables $\boldsymbol{k}_t$ and $\boldsymbol{H}_t$. At the start of each slot, $\boldsymbol{k}_t$ is made available to the system, and $\boldsymbol{H}_t$ for the upcoming uplink transmission can be estimated by the channel reciprocity[11]. The dimension of the state vector $\boldsymbol{s}_t$ is $K + (M + 1)N$.

#### 3.1.2 Action

Based on the observed system state $\boldsymbol{s}_t$, the RL agent will choose an action $\boldsymbol{a}_t$ based on the decision variables in **P1**, i.e.,

$$\boldsymbol{a}_t \triangleq \{\boldsymbol{c}_t, \boldsymbol{x}_t, \boldsymbol{b}_t, \boldsymbol{f}_t\} \tag{21}$$

where $\boldsymbol{c}_t$ is the caching decision, $\boldsymbol{x}_t$ is the computation offloading decision, $\boldsymbol{b}_t$ is the MU transmission power allocation decision, and $\boldsymbol{f}_t$ is MEC server computational resource allocation decision. The dimension of the action vector $\boldsymbol{a}_t$ is $K + 3N$.

#### 3.1.3 Reward

Given a particular state $\boldsymbol{s}_t$ and an action $\boldsymbol{a}_t$ at time slot $t$, it is evident that the overall system cost $J_t$ in Eq. (15) can be expressed by the reward function $\mathcal{R}$, which maps the state-action pair to a scalar reward $r_t$, such that

$$r_t = \mathcal{R}(\boldsymbol{s}_t, \boldsymbol{a}_t) = -J_t \tag{22}$$

It is noteworthy to mention that although RL algorithms maximize the infinite-horizon expected discounted return, these algorithms can also be used to approximate the true expected infinite-horizon expected non-discounted return, when the discount factor $\gamma \to 1$[34]. Therefore, the average system cost in Eq. (16) will be minimized by applying the policy learned via RL agent.

Many approaches in RL utilize the recursive relationship of the state-action function known as the Bellman equation. Under the optimal policy $\mu^*$, the Bellman optimality equation for the state-action function can be written as

$$Q^*(\boldsymbol{s}, \boldsymbol{a}) = \mathbb{E}\left[\mathcal{R}(\boldsymbol{s}, \boldsymbol{a}) + \gamma \max_{\boldsymbol{a}'} Q^*(\boldsymbol{s}', \boldsymbol{a}')\right] \tag{23}$$

where $\boldsymbol{s}'$ denotes the next state being transitioned from state $\boldsymbol{s}$ under action $\boldsymbol{a}$.

It is very hard to obtain the exact solution of the RL problem with high-dimensional state and action spaces by directly maximizing the Q-function. We propose to tackle this issue by obtaining an approximate solution of the RL problem using DRL with DDPG. Details are provided in the next subsection.

### 3.2 DRL-based solution with DDPG

A feasible method to solve the RL problem is the well-known Q-learning algorithm[35]. Q-learning comes from a class of model-free RL algorithms known as Temporal-Difference (TD) learning, which combines the concepts of Monte Carlo methods and dynamic programming methods. With Bellman optimality equation as the core of the algorithm, Q-learning solves Q-function through a value iteration updating approach as

$$Q(\boldsymbol{s},\boldsymbol{a}) \leftarrow Q(\boldsymbol{s},\boldsymbol{a}) + \alpha\left[\mathcal{R}(\boldsymbol{s},\boldsymbol{a}) + \gamma \max_{\boldsymbol{a}'} Q(\boldsymbol{s}',\boldsymbol{a}') - Q(\boldsymbol{s},\boldsymbol{a})\right] \tag{24}$$

where $[\mathcal{R}(\boldsymbol{s},\boldsymbol{a}) + \gamma \max_{\boldsymbol{a}'} Q(\boldsymbol{s}',\boldsymbol{a}') - Q(\boldsymbol{s},\boldsymbol{a})]$ is the TD error and $\alpha$ is the learning rate. It is proven that the Q-learning algorithm converges with probability one[17].

As the dimensions of the state space and action space increase, the complexity in solving Eq. (24) grows exponentially, which is known as the curse of dimensionality. In order to address this issue, a DRL algorithm can be an efficient alternative. This is because the powerful function approximation properties of DNNs allow DRL algorithms to learn low-dimensional representations for RL problems. The DQN method[19] exploits the architecture of DNN to approximate the Q function with a finite number of parameters and thus to facilitate solving Eq. (24).

Even though DQN can successfully solve problems in high-dimensional state spaces, it can handle only discrete and low-dimensional action spaces. Specifically, when there are a finite number of discrete actions, finding optimal policy according to Eq. (19) is relatively simple. However, for problems with continuous action spaces such as **P1**, the action space has to be discretized before applying DQN. The discretization of the action space results in loss of precision. Moreover, as the number of discretization levels increases, the computational complexity of DQN grows exponentially.

We propose to address this challenge by applying DDPG[20], which extends DRL algorithms to continuous action space and continuous state space. In DDPG, an actor-critic approach is adopted by using two separate DNNs, where the critic network $Q(\boldsymbol{s}, \boldsymbol{a}|\boldsymbol{\theta}^Q)$ approximates the Q-function, and the actor network $\mu(\boldsymbol{s}|\boldsymbol{\theta}^\mu)$ approximates the policy function $\mu$. Here, $\boldsymbol{\theta}^Q$ and $\boldsymbol{\theta}^\mu$ are the neural network weights of the critic and actor networks, respectively. Therefore, unlike DQN, instead of running an expensive optimization subroutine for computing $\max_{\boldsymbol{a}} Q(\boldsymbol{s}, \boldsymbol{a})$

each time, the DDPG algorithm approximates it with $\max_{\boldsymbol{a}} Q(\boldsymbol{s},\boldsymbol{a}) \approx Q\big(\boldsymbol{s},\mu(\boldsymbol{s}|\boldsymbol{\theta}^{\mu})|\boldsymbol{\theta}^{Q}\big)$. Moreover, $Q'(\boldsymbol{s},\boldsymbol{a}|\boldsymbol{\theta}^{Q'})$ and $\mu'(\boldsymbol{s}|\boldsymbol{\theta}^{\mu'})$ represent the target critic and the target actor networks, respectively. The target networks are used for computing target values, with $\boldsymbol{\theta}^{Q'}$ and $\boldsymbol{\theta}^{\mu'}$ being their corresponding neural network weights. The target networks are time-delayed copies of their original networks that slowly track the learned networks and greatly improve stability in learning.

Details of the proposed solution are described in Algorithm 1, which is used to solve **P1**. The objective function in **P1** is modified by using Eqs. (17) and (22) under the RL framework. The constraints in **P1** are used during the training of DRL framework to determine whether to adopt or discard a certain learning result. In the proposed DDPG framework, four-layer fully connected neural networks with two hidden layers are used for both the actor and critic networks. The number of neurons in the two hidden layers are $8N$ and $6N$, respectively. The neural networks use the Rectified Linear Unit (ReLU) as the activation function for all hidden layers, while the final output layer of the actor network uses a sigmoid layer to bound the actions. Ornstein-Uhlenbeck process[36] is used to provide temporally correlated noise for action exploration, while the Adaptive moment estimation (Adam) method[37] is adopted for updating the neural network parameters.

**Algorithm 1 Proposed solution for single-cell MEC**

**Input:** System model parameters, number of episodes $K_{\max}$, number of time steps in each episode $T_{\max}$, replay buffer size $|\mathcal{R}_B|$, mini-batch size $B$, learning rates for critic network $\alpha^{Q}$ and actor network $\alpha^{\mu}$, and update rates $\zeta^{Q}$ and $\zeta^{\mu}$ for the target critic network and target actor network, respectively.

**Initialization:**

Randomly initialize the critic network $Q(\boldsymbol{s},\boldsymbol{a}|\boldsymbol{\theta}^{Q})$ and actor network $\mu(\boldsymbol{s}|\boldsymbol{\theta}^{\mu})$ with weights $\boldsymbol{\theta}^{Q}$ and $\boldsymbol{\theta}^{\mu}$, respectively from a uniform distribution $[-3\times10^{-3}, 3\times10^{-3}]$.

Initialize associated target networks $Q'$ and $\mu'$ with weights $\boldsymbol{\theta}^{\mu'} \leftarrow \boldsymbol{\theta}^{\mu}, \boldsymbol{\theta}^{Q'} \leftarrow \boldsymbol{\theta}^{Q}$.

Initialize the experience replay buffer $\mathcal{R}_B$ as an empty array.

**for** each episode $k = 1, 2, \ldots, K_{\max}$ **do**

Randomly generate an initial state $\boldsymbol{s}_1$

**for** each episode $t = 1, 2, \ldots, T_{\max}$ **do**

Determine the decision vectors by selecting an action $\boldsymbol{a}_t = \mu(\boldsymbol{s}_t|\boldsymbol{\theta}^{\mu}) + \Delta\mu$ using the current policy $\mu$ and exploration noise $\Delta\mu$, which is generated by following the Ornstein-Uhlenbeck process[36].

Execute action $\boldsymbol{a}_t$ and observe the reward $r_t = \mathcal{R}(\boldsymbol{s}_t,\boldsymbol{a}_t) = -J_t$ and the new state $\boldsymbol{s}_{t+1}$ from the simulation environment.

Save the transition $(\boldsymbol{s}_t,\boldsymbol{a}_t,r_t,\boldsymbol{s}_{t+1})$ into the replay buffer $\mathcal{R}_B$. If $\mathcal{R}_B$ is full, discard the oldest samples.

Randomly sample a mini-batch of $B$ transitions $\{(\boldsymbol{s}_i^{(b)},\boldsymbol{a}_i^{(b)},r_i^{(b)},\boldsymbol{s}_{i+1}^{(b)})\}_{b=1}^{B}$ from $\mathcal{R}_B$.

**for** each transition $b$ in the mini-batch **do**

Compute the target value $y^{(b)}$ as

$$y^{(b)} = r_i^{(b)} + \gamma Q'\big(\boldsymbol{s}_{i+1}^{(b)}, \mu'(\boldsymbol{s}_{i+1}^{(b)}|\boldsymbol{\theta}^{\mu'})|\boldsymbol{\theta}^{Q'}\big) \tag{25}$$

**end for**

Update the critic network $Q(\boldsymbol{s},\boldsymbol{a}|\boldsymbol{\theta}^{Q})$ by one-step gradient descent as $\boldsymbol{\theta}^{Q} \leftarrow \boldsymbol{\theta}^{Q} - \alpha^{Q}\nabla_{\boldsymbol{\theta}^{Q}} L^{Q}$, where the loss $L^{Q}$ is

$$L^{Q} = \frac{1}{B}\sum_{b=1}^{B}\big[y^{(b)} - Q(\boldsymbol{s}_i^{(b)},\boldsymbol{a}_i^{(b)}|\boldsymbol{\theta}^{Q})\big]^2 \tag{26}$$

Update the actor network $\mu(\boldsymbol{s},\boldsymbol{a}|\boldsymbol{\theta}^{Q})$ by using one-step sampled policy gradient ascent as $\boldsymbol{\theta}^{\mu} \leftarrow \boldsymbol{\theta}^{\mu} - \alpha^{\mu}\nabla_{\boldsymbol{\theta}^{\mu}} J^{\mu}$, where $J^{\mu} \triangleq \mathbb{E}_{\boldsymbol{s},\boldsymbol{a}}[Q^{\mu}(\boldsymbol{s},\boldsymbol{a})]$, and

$$\nabla_{\boldsymbol{\theta}^{\mu}} J^{\mu} \approx \frac{1}{B}\sum_{b=1}^{B}\nabla_{\boldsymbol{a}} Q(\boldsymbol{s}_i^{(b)},\boldsymbol{a}|\boldsymbol{\theta}^{Q})\,|_{\boldsymbol{a}=\boldsymbol{a}_i^{(b)}} \times \nabla_{\boldsymbol{\theta}^{\mu}}\mu(\boldsymbol{s}_i^{(b)}|\boldsymbol{\theta}^{\mu}) \tag{27}$$

Update the target networks,

$$\boldsymbol{\theta}^{Q'} \leftarrow \zeta\boldsymbol{\theta}^{Q} + (1-\zeta)\boldsymbol{\theta}^{Q'} \text{ and } \boldsymbol{\theta}^{\mu'} \leftarrow \zeta\boldsymbol{\theta}^{\mu} + (1-\zeta)\boldsymbol{\theta}^{\mu'}$$

**end for**

**end for**

**Output:** Optimal policy $\mu^{*}$.

## 4 Dynamic caching, computation offloading, and resource allocation in multi-cell MEC network

The topic of MEC in a network consisting of multiple cells has been studied widely in the literatures with various design objectives and solution approaches[3–6]. In this section, we extend our problem to the case of multi-cell MEC network and present a DRL-based decentralized solution that utilizes cooperative task caching and execution.

### 4.1 System model

In the multi-cell scenario, we consider a network consisting of multiple small cells, where each small cell

represents a single MEC network as shown in Fig. 1. Each small cell consists of one BS with $M$ antennas, one MEC server, and a set of dedicated MUs. Denote the set of BSs (or equivalently the set of small cells) as $\mathcal{S} = \{1, 2, \ldots, S\}$, where $S$ is the number of BSs in the MEC network. Denote $\mathcal{N}^s$ as the set of MUs associated with the $s$-th BS. One important feature of the multi-cell model is that the BSs can communicate with each other and forward the cached tasks from one's MEC server to another. However, the MUs in set $\mathcal{N}^s$ can only communicate with the $s$-th BS.

It is noteworthy to mention that for the sake of brevity, we have not redefined all the notations of system parameters in the multi-cell model. Instead, we will add the symbol "$s$" as a superscript to the existing notations introduced earlier in order to represent the counterparts of the single-cell model parameters for the $s$-th BS in the multi-cell setting.

The set of tasks $\mathcal{K} = \{1, \ldots, K\}$ is assumed to be uniform throughout the entire network with the same data sizes $[b_k]_{k\in\mathcal{K}}$ and computational requirements $[d_k]_{k\in\mathcal{K}}$. However, the popularity profile of tasks $\boldsymbol{\phi}_t^s$ can vary both across time slots $t \in \mathcal{T}$ and across cells/BSs $s \in \mathcal{S}$. Denote $\boldsymbol{k}_t^s$ as the task request vector for all the MUs in $\mathcal{N}^s$ at the start of slot $t$.

Denote $\boldsymbol{c}_t^s$ as the caching decision vector for all the tasks in the $s$-th BS. In the cooperative multi-cell model, the communication among BSs can provide an additional option for task caching at the MEC servers. In the single-cell network, a BS can either cache the task data from the data offloaded by one of its MUs in the previous time slot or fetch/download from the cloud. With the cooperative model, a BS can also fetch task data from another nearby BS and cache it for possible future use. Therefore, similar to Eq. (3), the overall fetching cost of the $s$-th BS associated with the caching decision in the multi-cell MEC system at slot $t$ is defined as

$$C_t^s = \sum_{k=1}^{K}\big[\mathbf{1}(c_{k,t-1}^s = 0, c_{k,t}^s = 1, k \notin \mathcal{K}_t^{s,o})\times \{\mathbf{1}(\delta_{k,t}^{ss'} = 0)g_k + \mathbf{1}(\delta_{k,t}^{ss'} = 1)g_k^{ss'}\}\big] \quad (28)$$

where $\delta_{k,t}^{ss'}$ is a binary variable with $\delta_k^{ss'} = 1$ indicating that the input data for task $k$ is fetched by the $s$-th BS from the $s'$-th BS during time slot $t$ and 0 indicating no communication, and $g_k^{ss'}$ is the corresponding fetching cost between the two BSs. For simplicity, we assume that the fetching costs between the BSs are mutual, i.e., $g_k^{ss'} = g_k^{s's}$. These fetching costs depend on several factors, such as the size of task data $b_k$, the distance and channel condition between the BSs, etc. However, it is worth noting that the fetching costs between BSs are considerably less than the fetching cost from the cloud for all the tasks, i.e., $g_k^{ss'} < g_k$ ($\forall\, k \in \mathcal{K},\ \forall\, s, s' \in \mathcal{S}$). Therefore, the cooperative multi-cell model will yield lower overall fetching cost compared to the single-cell model.

Similar to the additional task caching option, the cooperative multi-cell MEC network will also provide an additional option for task execution. In the single MEC network, if an MU requests a task that is not available in the MEC server cache, then there are two approaches: local execution and MEC server execution by offloading the task data (as mentioned in Section 2.4). In the multi-cell setting, a BS can fetch the results of task execution from another nearby BS given that the later BS has executed that task in its MEC server through either offloading computation data or leveraging the cached data during same time slot. We denote a binary variable $\gamma_{k,t}^{ss'}$ with $\gamma_{k,t}^{ss'} = 1$ indicating that the result of task $k$ is fetched by the $s$-th BS from the $s'$-th BS during slot $t$ and 0 indicating otherwise. Again we assume that the download/fetching costs (both in terms of energy and delay) for the task results are negligible since the size of the results is usually much smaller than that of the input data.

We define $\mathcal{K}_t^{s,f} \triangleq \{k : \gamma_{k,t}^{ss'} = 1\ \exists\, s' \in \mathcal{S} \setminus s\}$ as the set of tasks whose results have been fetched by the $s$-th BS during slot $t$ from any nearby BS. Denote $\mathcal{N}_t^{s,f} \triangleq \{n \in \mathcal{N}^s : k_t^n \in \mathcal{K}_t^{s,f}\}$ as the set of MUs whose requested tasks are executed by fetching the task results from a nearby BS. It is evident that the MUs in $\mathcal{N}_t^{s,f}$ are indirectly leveraging the computational power of a non-anchor BS by fetching the results of executed tasks in the cooperative model.

### 4.2 Problem formulation

Given the computation offloading decision $\boldsymbol{x}_t^s$, the MU power transmission allocation $\boldsymbol{b}_t^s$, the MEC server's

computational resource allocation $\boldsymbol{f}_t^s$, and the caching decision $\boldsymbol{c}_t^s$ in the $s$-th BS, the energy consumption and computation delay for the $n$-th MU in the $s$-th BS are calculated, respectively, as

$$E_{n,t}^s = \mathbf{1}(n \notin \mathcal{N}_t^{s,c}, n \notin \mathcal{N}_t^{s,f}) \times [\mathbf{1}(x_{n,t}^s = 1) E_{n,t}^{s,x} + \mathbf{1}(x_{n,t}^s = 0) E_{n,t}^{s,l}] \quad (29)$$

$$T_{n,t}^s = \mathbf{1}(n \notin \mathcal{N}_t^{s,c}, n \notin \mathcal{N}_t^{s,f}) \times [\mathbf{1}(x_{n,t}^s = 1)(T_{n,t}^{s,x} + T_{n,t}^{s,p}) + \mathbf{1}(x_{n,t}^s = 0) T_{n,t}^{s,l}] + \mathbf{1}(n \in \mathcal{N}_t^{s,c}, n \notin \mathcal{N}_t^{s,f}) T_{n,t}^{s,p} \quad (30)$$

Similar to Eq. (15), the overall cost of the $s$-th BS during slot $t$ can be defined as

$$J_t^s = \sum_{n \in \mathcal{N}^s} \left[ E_{n,t}^s + \omega_n^s T_{n,t}^s \right] + \omega_c^s C_t^s \quad (31)$$

The optimization problem in the multi-cell MEC system is formulated as follows:

$$\mathbf{P2:} \min_{\boldsymbol{C}, \boldsymbol{X}, \boldsymbol{P}, \boldsymbol{F}} \ \bar{J}' \triangleq \mathbb{E}\left[ \lim_{|\mathcal{T}| \to \infty} \frac{1}{|\mathcal{T}|} \sum_{t \in \mathcal{T}} \sum_{s \in \mathcal{S}} J_t^s \right]$$

s.t.

C7: $c_{k,t}^s \in \{0, 1\}, \ \forall s \in \mathcal{S}, \ \forall k \in \mathcal{K}, \ \forall t \in \mathcal{T}$;

C8: $x_{n,t}^s \in \{0, 1\}, \ \forall s \in \mathcal{S}, \ \forall n \in \mathcal{N}_t, \ \forall t \in \mathcal{T}$;

C9: $p_{n,t}^s \leqslant P_n^{s,\max}, \ \forall n \in \{n : x_{n,t}^s = 1\}$;

C10: $\sum_{n \in \mathcal{N}^s} \mathbf{1}(x_{n,t}^s = 1 \vee n \in \mathcal{N}_t^{s,c}) f_{n,t}^s \leqslant F^s$;

C11: $\sum_{k \in \mathcal{K}} \mathbf{1}(c_{k,t}^s = 1) b_k \leqslant D^s, \ \forall s \in \mathcal{S}, \ \forall t \in \mathcal{T}$;

C12: $T_{n,t}^s \leqslant T_s, \ \forall s \in \mathcal{S}, \ \forall n \in \mathcal{N}, \ \forall t \in \mathcal{T}$,

where $\bar{J}'$ denotes the long-term average system cost of multi-cell MEC network, $\boldsymbol{C}, \boldsymbol{X}, \boldsymbol{P}$, and $\boldsymbol{F}$ are the matrix counterparts of the all the decision vectors as previously mentioned in **P1** and thus represent all the decisions for all the MUs in all the cells in multi-cell model. Moreover, $F^s$ and $D^s$ denote the computational capacity and cache size of the MEC server associated with the $s$-th BS, respectively.

In the next subsection, we propose a DRL-based cooperative decentralized solution with DDPG for **P2**.

### 4.3 Cooperative decentralized solution with DDPG

One obvious approach to solve **P2** is to extend the proposed solution in Algorithm 1 to the multi-cell model by incorporating all the state and action variables from all the cells into the RL framework and yield a centralized decision. However, centralized solutions are usually computationally prohibitive since the system overhead among the MUs and MEC servers grows exponentially as the numbers of MUs and BSs/cells increase[11]. Hence, a decentralized solution at each cell is more favorable for better scalability.

The key elements of RL framework for the proposed decentralized solution are defined as follows.

#### 4.3.1 State

The system state for $s$-th cell at an arbitrary time slot $t$ is defined as

$$\boldsymbol{S}_t^s \triangleq \{\boldsymbol{K}_t, \boldsymbol{H}_t^s, \boldsymbol{C}_{t-1}\} \quad (32)$$

where $\boldsymbol{K}_t \triangleq [\boldsymbol{k}_t^1, \ldots, \boldsymbol{k}_t^S]$ is the $N'$-dimensional task request vector with $N' = \sum_{s \in \mathcal{S}} |\mathcal{N}^s|$ being the total number of MUs in all the cells, $\boldsymbol{H}_t^s$ is the channel matrix of the $s$-th BS/cell, and $\boldsymbol{C}_{t-1} \triangleq [\boldsymbol{c}_{t-1}^1, \ldots, \boldsymbol{c}_{t-1}^S]$ is the $K \times S$ caching decision matrix in the previous slot. The dimension of the state variable for $s$-th cell $\boldsymbol{S}_t^s$ is $N' + M|\mathcal{N}^s| + KS$. Please note that the variables $\boldsymbol{K}_t$ and $\boldsymbol{C}_{t-1}$ are obtained through the cooperative nature of the multi-cell model where these information are shared among all the cells at the beginning of a time slot. However, unlike the centralized approach, we do not need to include the channel information from nearby cells as part of the state variables in the decentralized approach, because they do not influence the decision in the $s$-th cell, thus reduce a large portion of the system overhead.

#### 4.3.2 Action

Based on the observed system state $\boldsymbol{S}_t^s$, the RL agent will choose decentralized actions $\boldsymbol{a}_t^s$ for the $s$-th cell, such that

$$\boldsymbol{a}_t^s \triangleq \{\boldsymbol{c}_t^s, \boldsymbol{x}_t^s, \boldsymbol{b}_t^s, \boldsymbol{f}_t^s\} \quad (33)$$

where $\boldsymbol{c}_t^s, \boldsymbol{x}_t^s, \boldsymbol{b}_t^s$, and $\boldsymbol{f}_t^s$ are the multi-cell model counterparts of the all the decision vectors as previously mentioned in Eq. (21). The dimension of the action vector $\boldsymbol{a}_t^s$ is $K + 3|\mathcal{N}^s|$.

#### 4.3.3 Reward

Similar to the definition of reward given in Eq. (22), the overall cost $J_t^s$ for $s$-th cell in Eq. (31) can be expressed by the reward function $\mathcal{R}^s$, which maps the state-action pair to a scalar reward $r_t^s$, such that

$$r_t^s = \mathcal{R}^s(\boldsymbol{S}_t^s, \boldsymbol{a}_t^s) = -J_t^s \quad (34)$$

Details of the proposed decentralized solution are described in Algorithm 2. In the proposed DDPG framework, there are individual critic network $Q_s(\boldsymbol{S}^s, \boldsymbol{a}^s|\boldsymbol{\theta}_s^Q)$ and actor network $\mu_s(\boldsymbol{S}^s|\boldsymbol{\theta}_s^\mu)$ for each cell $s \in \mathcal{S}$. These networks will be trained in parallel as indicated by Steps (6)–(14) of Algorithm 2. We adopt similar structure, activation functions, and action exploration process for all the neural networks as mentioned at the end of Section 3.2. However, the number of neurons in the two hidden layers of each neural network corresponding to the $s$-th cell are $8|\mathcal{N}^s|$ and $6|\mathcal{N}^s|$, respectively. Once all the networks have been trained, the algorithm will output the final policy function $\mu_s$ for each cell $s \in \mathcal{S}$ that maps the state to actions, i.e., $\mu_s(\boldsymbol{S}_t^s) = \boldsymbol{a}_t^s$, which in turn combines into the solution of **P2**, i.e., decision variables $\boldsymbol{C}, \boldsymbol{X}, \boldsymbol{P}$, and $\boldsymbol{F}$ of the whole system.

**Algorithm 2 Proposed cooperative decentralized solution for multi-cell MEC system using DDPG**

**Input:** System model parameters, number of episodes $K_{\max}$, number of time steps in each episode $T_{\max}$, same size $|\mathcal{R}_B|$ for replay buffers, $\mathcal{R}_B^s$ for all cells $s \in \mathcal{S}$, along with other hyper-parameters as mentioned in the input of Algorithm 1 .

1: **for** each cell $s = 1, 2, \ldots, S$ **do**
2: Randomly initialize the critic network $Q_s(\boldsymbol{S}^s, \boldsymbol{a}^s|\boldsymbol{\theta}_s^Q)$ and actor network $\mu_s(\boldsymbol{S}^s|\boldsymbol{\theta}_s^\mu)$ with weights $\boldsymbol{\theta}_s^Q$ and $\boldsymbol{\theta}_s^\mu$, respectively, from a uniform distribution $[-3, 3] \times 10^{-3}$.
3: Initialize associated target networks $Q_s'$ and $\mu_s'$ with weights $\boldsymbol{\theta}_s^{\mu'} \leftarrow \boldsymbol{\theta}_s^\mu, \boldsymbol{\theta}_s^{Q'} \leftarrow \boldsymbol{\theta}_s^Q$.
4: Initialize the experience replay buffers $\mathcal{R}_B^s$ ($\forall\, s \in \mathcal{S}$) as empty arrays.
5: **end for**
6: **for** each episode $k = 1, 2, \ldots, K_{\max}$ **do**
7: Randomly generate an initial state $\boldsymbol{S}_1 \triangleq \{\boldsymbol{S}_1^s\}_{s\in\mathcal{S}}$ for the whole multi-cell MEC system.
8: **for** each episode $t = 1, 2, \ldots, T_{\max}$ **do**
9: Share the information regarding task requests and cache status among all the BSs.
10: **for** each cell $s = 1, 2 \ldots, S$ **do**
11: Perform Steps (8)–(17) of Algorithm 1 for independently training the networks of cell $s$ with DDPG.
12: **end for**
13: **end for**
14: **end for**

**Output:** Policy $\mu_s$ for all cells $s \in \mathcal{S}$.

## 5 Simulation Result

Simulation results are presented in this section to illustrate the performance of the proposed algorithm with DDPG. Unless specified otherwise, the default settings of the single-cell MEC system are set as follows: the number of MUs is $N = 5$, the number of antennas BS is $M = 6$, the coverage radius of the small cell BS is $d_m = 50$ m, the cache size of the MEC server is $D = 200$ MB, the computational resource of the MEC server $F = 5$ GHz, the CPU frequency of each MU is $f_n^l = 1$ GHz, the channel bandwidth is $W = 20$ MHz, and the duration of a time slot is $T_s = 1$ s.

At the beginning of every episode, the channel vector of each MU is initialized as $\boldsymbol{h}_{n,0} \sim \mathcal{CN}(\boldsymbol{0}, h_0(d_0/d_n)^\beta \boldsymbol{I_M})$, where $h_0 = -30$ dB, $d_0 = 1$ m, the path-loss exponent is $\beta = 3$, and $d_n$ (in meters) denotes the distance from the BS to the $n$-th MU[11]. In each episode, the locations of MUs are randomly set such that they are uniformly scattered throughout the coverage region, and the locations are independent in different episodes. The channel vectors $\boldsymbol{h}_{n,t}$, $\forall n \in \mathcal{N}$, are updated according to Eq. (4), where the channel correlation coefficient is $\rho_n = 0.95$ and the error vector is $\boldsymbol{e}_t \sim \mathcal{CN}(\boldsymbol{0}, h_0(d_0/d_m)^\beta \boldsymbol{I_M})$. The MU's maximum allowed transmission power is $P_n^{\max} = 2$ W, $\forall n \in \mathcal{N}$, and the background noise power is $\sigma^2 = 10^{-9}$ W[11]. The energy-delay tradeoff parameters are $\omega_n = 1$ W for all MUs, and the weight for fetching cost is $\omega_c = 10^{-8}$ J/bit.

There are $K = 4$ computation tasks. The number of CPU cycles required to complete the tasks $d_k$ are uniformly distributed between $[\bar{d} - 0.05, \bar{d} + 0.05]$ Gigacycles with $\bar{d}$ being the average computations per task. The data sizes of the computation tasks $b_k$ are uniformly distributed between $[\bar{b} - 5, \bar{b} + 5]$ MB with $\bar{b}$ being the average data size. Unless specified otherwise, the default values are $\bar{d} = 0.5$ G cycles and $\bar{b} = 75$ MB. The fetching cost of each task is assumed to be the same as the data size of corresponding task, i.e., $g_k = b_k$. Moreover, the popularity profile $\boldsymbol{\phi}_t$ of the tasks is modeled by a three-state Markov chain[22], represented by three different popularity profiles $\boldsymbol{\phi}^{(1)}, \boldsymbol{\phi}^{(2)}$, and $\boldsymbol{\phi}^{(3)}$. These profiles are modeled by Zipf distributions

with parameters $\eta_1 = 1$, $\eta_2 = 1.2$, and $\eta_3 = 1.5$, respectively. So, at each time slot $t$, the popularity profile $\boldsymbol{\phi}_t$ will follow one of these three states and each task $k \in \mathcal{K}$ will be assigned popularity ranks $z_{k,t}$ randomly. Then, MUs will request tasks by sampling Zipf distribution defined by $\boldsymbol{\phi}_t$. The Markov transition probabilities among the three popularity profiles are given by the transition matrix,

$$\boldsymbol{\tau} \triangleq \begin{bmatrix} \tau_{1,1} & \tau_{1,2} & \tau_{1,3} \\ \tau_{2,1} & \tau_{2,2} & \tau_{2,3} \\ \tau_{3,1} & \tau_{3,2} & \tau_{3,3} \end{bmatrix} = \begin{bmatrix} 0.5 & 0.3 & 0.2 \\ 0.1 & 0.6 & 0.3 \\ 0.25 & 0.35 & 0.4 \end{bmatrix} \quad (35)$$

where $\tau_{i,j}$ represents the transition probability from state $i$ to state $j$, for $i, j \in \{1, 2, 3\}$. Please note that these states are different from the system states defined in our problem formulation.

The hyper-parameters for training the neural networks in the proposed DDPG-based algorithm are described in Table 2. To evaluate the performance of policy $\mu^*$ learned by the proposed algorithm, testing results are averaged over 1000 episodes, with each episode consisting of 100 steps. Results obtained from the proposed algorithm are compared to four baseline strategies for single-cell MEC setting that are described as follows.

**Popularity-based Caching and Local execution (PCL):** The MEC server caches the data of the computing tasks with the maximum number of requests, till reaching the caching capacity. Computational resource $F$ is distributed equally to each MU that requests a cached task. The MUs that request uncached tasks execute their tasks by local CPU.

**Popularity-based Caching and full Offloading (PCO):** The MEC server caches the data of the computing tasks with the maximum number of requests, till reaching the caching capacity. All tasks are executed at the MEC server. That means all MUs who requested uncached tasks will offload their tasks to the MEC server and transmit the corresponding data with the maximum power available. The computational resource $F$ is distributed uniformly across $N$ MUs.

**Randomized Caching, Offloading, and Resource allocation (RCOR):** The MEC server caches the data of computing tasks randomly, till reaching the caching capacity. For MUs requesting uncached tasks, all the offloading and resource allocation decisions are also taken randomly.

**DQN-based solution:** DQN[19] can only be implemented on systems with discrete state and action spaces. The support spaces for $\boldsymbol{b}$ and $\boldsymbol{f}$ are both discretized uniformly into finite $L$ levels each. Therefore, the size of the action space becomes $2^K(2L^2)^N$ for $K$ tasks and $N$ MUs. We arbitrarily set $L = 3$, train the DQN with the same values of hyper-parameters as mentioned in Table 2, and maintain the same neural network architecture as mentioned in Section 3.2. Moreover, $\epsilon$-greedy exploration method is adopted for exploring the actions during network training with $\epsilon = 0.01$.

Figure 2 shows the average system cost ($\bar{J}$) as a function of the computational resource capacity of the MEC server ($F$) with various algorithms. Due to the extra computational resources from the MEC, the performances of all algorithms improve as $F$ increases, but with different slopes. The PCL approach has the smallest absolute slope, because the average number

**Table 2 Hyper-parameters for training neural networks.**

| Parameter | Value |
|---|---|
| Number of training episodes ($K_{\max}$) | 2000 |
| Number of steps in each episode ($T_{\max}$) | 100 |
| Experience replay buffer size ($\lvert\mathcal{R}_B\rvert$) | 50 000 |
| Mini-batch size ($B$) | 128 |
| Learning rate for critic network ($\alpha^{\mathcal{Q}}$) | $10^{-4}$ |
| Learning rate for actor network ($\alpha^{\mu}$) | $10^{-3}$ |
| Soft update rate for target networks ($\zeta$) | $10^{-3}$ |

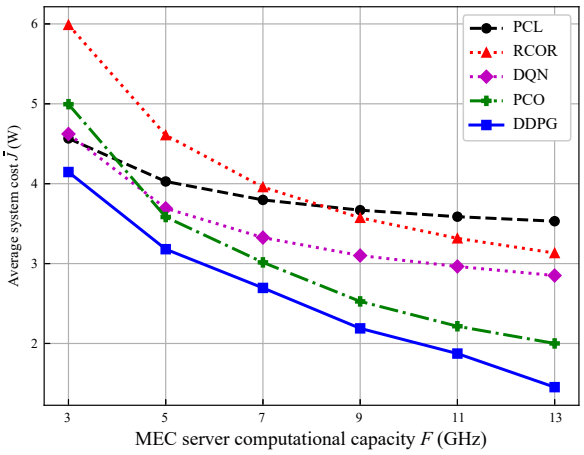



**Fig. 2 Average system cost v.s. computational capacity of the MEC server.**

of MUs using the MEC resources in each time slot is the least among all approaches. The proposed DDPG-based algorithm achieves the best performance. The performance gap between the DDPG-based algorithm and DQN algorithm becomes larger as $F$ increases. Specifically, the performance gap increases from 10.3% when $F = 3$ to 49% when $F = 13$. This means that the DDPG-based algorithm can better utilize the MEC resources. Even though with a larger number of discrete levels $L$, it may be possible to get better results by the DQN approach, it will yield a very high-dimensional action space with prohibitively high computational complexity.

Figure 3 shows $\bar{J}$ as a function of the cache size $D$ of the MEC server. Here, the results obtained at $D = 0$ represents the case without caching. The average system cost decreases as $D$ becomes larger for all approaches except the RCOR strategy. A larger cache size allows the MEC server to cache more tasks, thus reduce the transmission delay and the corresponding energy consumption incurred by the MUs. However, the RCOR strategy demonstrates a different behavior because caching decision in the RCOR strategy is changed frequently in a random manner, which leads to a much larger average fetching cost than other approaches. The larger fetching cost of RCOR negates the advantage of lower delay and energy cost. For the cache-assisted MEC system, the proposed DDPG-based algorithm has the best performance, followed by PCO, DQN, PCL, and RCOR, respectively.

Figure 4 shows the effects of different computation offloading and resource allocation approaches on $\bar{J}$, and the results are shown as functions of the average

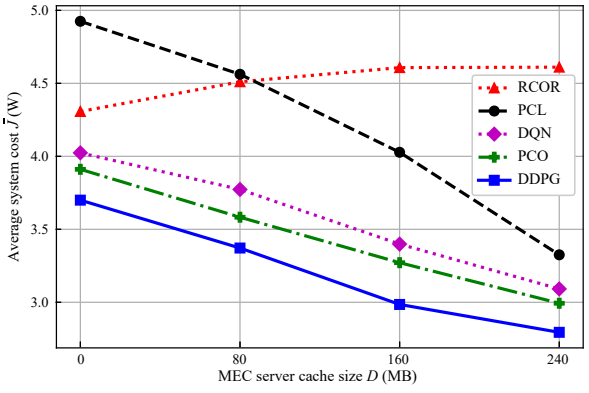



**Fig. 3 Average system cost vs cache size of the MEC server.**

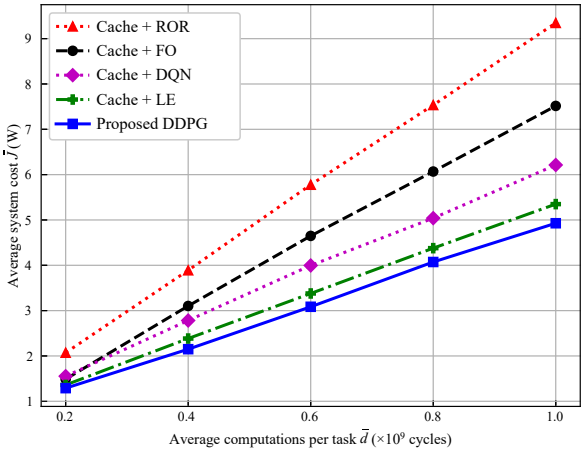



**Fig. 4 Effect of computation offloading and resource allocation.**

computations resources required for each task ($\bar{d}$). In all the approaches, the policy for determining the decision variables $\boldsymbol{x}, \boldsymbol{b}$, and $\boldsymbol{f}$ differs, while the caching decision variable $\boldsymbol{c}$ is obtained according to the proposed DDPG-based solution. Here, LE refers to the local execution approach as defined in the PCL strategy, FO refers to the full offload approach as defined in the PCO strategy, and ROR refers to the randomized offloading and resource allocation strategy. In all the approaches, $\bar{J}$ gets larger as $\bar{d}$ increases, since more computation-intensive tasks incur more processing delay and energy consumption. The proposed DDPG-based solution achieves the best performance, followed by LE, DQN, FO, and ROR, respectively. The performance gaps between the proposed algorithm and all the other approaches become larger as $\bar{d}$ increases, which means the proposed solution can achieve better MEC resource allocation, especially when more computations resources are needed.

Figure 5 demonstrates the tradeoff relationship between average energy consumption and average delay of the system. Various tradeoff points are obtained by changing the values of $\omega_n$ for all $n \in \mathbf{N}$ MUs. Please note that here we consider the time-average not the ensemble average across all the MUs. The average delay experienced by all the MUs can be decreased at the cost of higher energy consumption, and vice versa. Moreover, the proposed DDPG-based algorithm shows better tradeoff performance compared to the DQN-based solution, i.e., the MUs governed by the DDPG-based policy experience comparatively less average delay for

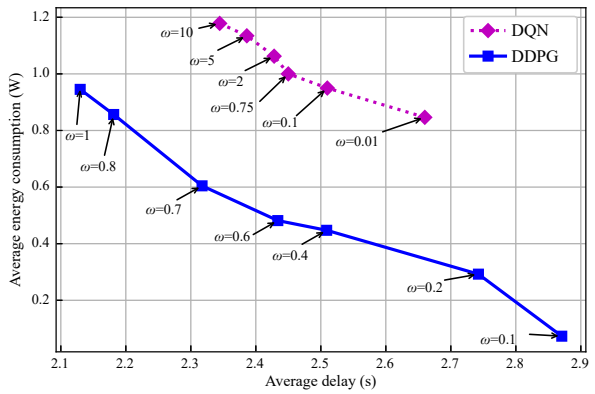



**Fig. 5 Tradeoff between average energy consumption and average delay with tradeoff parameter $\omega_n = \omega$ (in W) $\forall n \in \mathbf{N}$.**

the same average energy consumption. Since all the other baseline approaches have fixed policies, $\omega_n$ does not have any impact on the decision variables. Therefore, no tradeoff relationships are observed in the PCL, PCO, and RCOR approaches.

Figure 6 illustrates the performance of proposed cooperative decentralized solution for multi-cell MEC model in terms of $\bar{J}'$ for different values of the average task data size ($\bar{b}$). In this example, we consider the number of small cells in the network as $S = 3$, with each small cell having the same properties as mentioned at the start of this section, i.e., each cell has the same coverage radius, computational capacity, cache size, etc. In addition, all the BSs are equally distanced from each other and we assume the fetching costs between the BSs for a given task $k$ are constants, such that $g_k^{ss'} = 0.5b_k, \forall\, s, s' \in \mathcal{S}$.

In Fig. 6, the proposed solution is compared with two approaches: cooperative centralized solution and

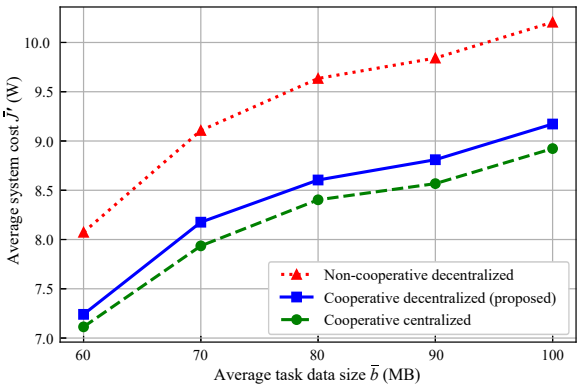



**Fig. 6 Average system cost vs average task data size in the multi-cell MEC model.**

non-cooperative decentralized solution. Naturally, $\bar{J}'$ increases for all the approaches as the average size of the tasks gets larger. The non-cooperative solution is equivalent to implementing Algorithm 1 independently at each cell without utilizing nearby cells' caches and executed task results. Therefore, it performs the worst. As mentioned in Section 4.3, the centralized solution considers all system parameters and state variables from all the cells, thus it yields the optimal decision at the cost of prohibitively high computational complexity, huge system overhead, and slower convergence. Compared to the centralized solution, our proposed cooperative decentralized solution achieves great performance, with the performance gap ranging between 1.8% and 2.8%.

## 6 Conclusion

We have studied the problem of dynamic caching, computation offloading, and resource allocation in cache-assisted MEC systems with stochastic wireless channel conditions. In the MEC system, multiple MUs execute random computation-intensive tasks either locally or remotely in one or more MEC servers. Popular tasks can be fetched from a remote cloud and cached in MEC servers to avoid unnecessary duplicates in transmissions. We have formulated the problem under the MDP framework, which aims at minimizing the long-term average system cost that includes a weighted sum energy consumption, delay, and cache fetching cost, under the constraint of limited storage and computational resource at the MEC server. Centralized and decentralized DDPG-based algorithms have been developed to solve the problems for single-cell and multi-cell MEC systems, respectively. Simulation results have shown that the proposed algorithm outperforms other existing approaches such as DQN.

## References

[1] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, A survey on mobile edge computing: The communication perspective, *IEEE Communications Surveys Tutorials*, vol. 19, no. 4, pp. 2322–2358, 2017.

[2] Y. Hao, M. Chen, L. Hu, M. S. Hossain, and A. Ghoneim, Energy efficient task caching and offloading for mobile edge computing, *IEEE Access*, vol. 6, pp. 11 365–11 373, 2018.

[3] C. Wang, C. Liang, F. R. Yu, Q. Chen, and L. Tang, Computation offloading and resource allocation in wireless cellular networks with mobile edge computing, *IEEE Transactions on Wireless Communications*, vol. 16, no. 8, pp. 4924–4938, 2017.

[4] M. I. A. Zahed, I. Ahmad, D. Habibi, and Q. V. Phung, Green and secure computation offloading for cache-enabled IoT networks, *IEEE Access*, vol. 8, pp. 63 840–63 855, 2020.

[5] T. Alfakih, M. M. Hassan, A. Gumaei, C. Savaglio, and G. Fortino, Task offloading and resource allocation for mobile edge computing by deep reinforcement learning based on SARSA, *IEEE Access*, vol. 8, pp. 54 074–54 084, 2020.

[6] N. Maurice, Q.-V. Pham, and W.-J. Hwang, Online computation offloading in noma-based multi-access edge computing: A deep reinforcement learning approach, *IEEE Access*, vol. 8, pp. 99 098–99 109, 2020.

[7] L. Huang, X. Feng, C. Zhang, L. Qian, and Y. Wu, Deep reinforcement learning-based joint task offloading and bandwidth allocation for multi-user mobile edge computing, *Digital Communications and Networks*, vol. 5, no. 1, pp. 10–17, 2019.

[8] J. Wang, L. Zhao, J. Liu, and N. Kato, Smart resource allocation for mobile edge computing: A deep reinforcement learning approach, *IEEE Transactions on Emerging Topics in Computing*, doi: 10.1109/TETC.2019.2902661.

[9] S. Nath, Y. Li, J. Wu, and P. Fan, Multi-user multi-channel computation offloading and resource allocation for mobile edge computing, doi: 10.1109/ICC40277.2020.9149124.

[10] S. Nath and J. Wu, Dynamic computation offloading and resource allocation for multi-user mobile edge computing, presented at IEEE Global Communications Conf. (GLOBECOM), Taipei, China, 2020.

[11] Z. Chen and X. Wang, Decentralized computation offloading for multi-user mobile edge computing: A deep reinforcement learning approach, *EURASIP Journal on Wireless Communications and Networking*, doi: 10.1186/s13638-020-01801-6.

[12] P. Liu, G. Xu, K. Yang, K. Wang, and X. Meng, Jointly optimized energy-minimal resource allocation in cache-enhanced mobile edge computing systems, *IEEE Access*, vol. 7, pp. 3336–3347, 2018.

[13] Y. Mao, J. Zhang, S. H. Song, and K. B. Letaief, Stochastic joint radio and computational resource management for multi-user mobile-edge computing systems, *IEEE Transactions on Wireless Communications*, vol. 16, no. 9, pp. 5994–6009, 2017.

[14] L. Chunlin and J. Zhang, Dynamic cooperative caching strategy for delay-sensitive applications in edge computing environment, *The Journal of Supercomputing*, vol. 76, no. 1, pp. 1–25, 2020.

[15] J. Xu, L. Chen, and P. Zhou, Joint service caching and task offloading for mobile edge computing in dense networks, in *Proc. of IEEE Conference on Computer Communications* (*INFOCOM*), Honolulu, HI, USA, 2018, pp. 207–215.

[16] P. Yang, N. Zhang, S. Zhang, L. Yu, J. Zhang, and X. Shen, Dynamic mobile edge caching with location differentiation, doi: 10.1109/GLOCOM.2017.8254034.

[17] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. Cambridge, MA, USA: MIT press, 2018.

[18] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, Deep reinforcement learning: A brief survey, *IEEE Signal Processing Magazine*, vol. 34, no. 6, pp. 26–38, 2017.

[19] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, Playing atari with deep reinforcement learning, arXiv preprint arXiv:1312.5602, 2013.

[20] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, Continuous control with deep reinforcement learning, arXiv preprint arXiv:1509.02971, 2015.

[21] K. Shanmugam, N. Golrezaei, A. G. Dimakis, A. F. Molisch, and G. Caire, Femtocaching: Wireless content delivery through distributed caching helpers, *IEEE Transactions on Information Theory*, vol. 59, no. 12, pp. 8402–8413, 2013.

[22] A. Sadeghi, F. Sheikholeslami, and G. B. Giannakis, Optimal dynamic proactive caching via reinforcement learning, doi: 10.1109/SPAWC.2018.8445899.

[23] H. A. Suraweera, T. A. Tsiftsis, G. K. Karagiannidis, and A. Nallanathan, Effect of feedback delay on amplify-and-forward relay networks with beamforming, *IEEE Transactions on Vehicular Technology*, vol. 60, no. 3, pp. 1265–1271, 2011.

[24] M. Abramowitz and I. A. Stegun, *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables*. Washington, DC, USA: US Government Printing Office, 1948.

[25] H. Q. Ngo, E. G. Larsson, and T. L. Marzetta, Energy and spectral efficiency of very large multiuser mimo systems, *IEEE Transactions on Communications*, vol. 61, no. 4, pp. 1436–1449, 2013.

[26] Y. Wen, W. Zhang, and H. Luo, Energy-optimal mobile application execution: Taming resource-poor mobile devices with cloud clones, in *Proc. IEEE Conference on Computer Communications* (*INFOCOM*), Orlando, FL, USA, 2012, pp. 2716–2720.

[27] X. Chen, L. Jiao, W. Li, and X. Fu, Efficient multi-user computation offloading for mobile-edge cloud computing, *IEEE/ACM Transactions on Networking*, vol. 24, no. 5, pp. 2795–2808, 2016.

[28] K. Zhang, Y. Mao, S. Leng, Q. Zhao, L. Li, X. Peng, L. Pan, S. Maharjan, and Y. Zhang, Energy-efficient offloading for

mobile edge computing in 5G heterogeneous networks, *IEEE Access*, vol. 4, pp. 5896–5907, 2016.

[29] J. Li, H. Gao, T. Lv, and Y. Lu, Deep reinforcement learning based computation offloading and resource allocation for MEC, in *Proc. Wireless Communications and Networking Conference* (*WCNC*), Barcelona, Spain, 2018, pp. 1–6.

[30] S. Nath, J. Wu, and J. Yang, Delay and energy efficiency tradeoff for information pushing system, *IEEE Transactions on Green Communications and Networking*, vol. 2, no. 4, pp. 1027–1040, 2018.

[31] S. Nath, J. Wu, and H. Lin, Optimum multicast scheduling in delay-constrained content-centric wireless networks, doi: 10.1109/ICC.2019.8761690.

[32] S. Nath, J. Wu, and J. Yang, Optimum energy efficiency and age-of-information tradeoff in multicast scheduling, doi: 10.1109/ICC.2018.8422521.

[33] S. Nath, J. Wu, and J. Yang, Optimizing age-of-information and energy efficiency tradeoff for mobile pushing notifications, doi: 10.1109/SPAWC.2017.8227712.

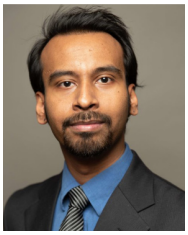
[34] D. Adelman and A. J. Mersereau, Relaxations of weakly coupled stochastic dynamic programs, *Operations Research*, vol. 56, no. 3, pp. 712–727, 2008.

[35] C. J. Watkins and P. Dayan, Q-learning, *Machine learning*, vol. 8, no. 3, pp. 279–292, 1992.

[36] G. E. Uhlenbeck and L. S. Ornstein, On the theory of the brownian motion, *Physical Review*, vol. 36, no. 5, p. 823, 1930.

[37] D. P. Kingma and J. Ba, Adam: A method for stochastic optimization, arXiv preprint arXiv:1412.6980, 2014.

**Jingxian Wu** received the BS (EE) degree from the Beijing University of Aeronautics and Astronautics, Beijing, China in 1998, the MEng (EE) degree from Tsinghua University, Beijing, China in 2001, and the PhD (EE) degree from the University of Missouri at Columbia, Missouri, USA in 2005. He is currently a professor at the Department of Electrical Engineering, University of Arkansas, Fayetteville. His research interests mainly focus on signal processing for large scale networks and wireless communications, cybersecurity for smart grids, statistical data analytics, etc. He served as symposium or track co-chairs for a number of international conferences, such as the 2012 and 2019 IEEE International Conference on Communications, the 2009, 2015, and 2017 IEEE Global Telecommunications Conference, etc. He served as an associate editor of the *IEEE Transactions on Vehicular Technology* from 2007 to 2011, an editor of the *IEEE Transactions on Wireless Communications* from 2011 to 2016, and is now serving as an associate editor of the *IEEE Aeeess*.



**Samrat Nath** received the BS degree in electrical and electronic engineering from the Bangladesh University of Engineering and Technology, Dhaka, Bangladesh in 2014, and the PhD degree in electrical engineering from the University of Arkansas, Fayetteville, USA in 2020. He is currently a data scientist at Walmart Inc. in Bentonville, AR, USA. His research interests include statistical signal analysis, information sensing and processing, optimization, machine learning, and wireless communication.