Differentially Private Deep Q-Learning for Pattern Privacy Preservation in MEC Offloading

Shuying Gan[†], Marie Siew[¶], Chao Xu[†§], Tony Q.S. Quek[*]

[†]School of Information Engineering, Northwest A&F University, Yangling, Shaanxi, China

[¶]Department of Electrical and Computer Engineering, Carnegie Mellon University, USA

[*]Information System Technology and Design Pillar, Singapore University of Technology and Design, Singapore

Email: {ganshuying99, cxu}@nwafu.edu.cn, msiew@andrew.cmu.edu, tonyquek@sutd.edu.sg

Abstract—Mobile edge computing (MEC) is a promising paradigm to meet the quality of service (QoS) requirements of latency-sensitive IoT applications. However, attackers may eavesdrop on the offloading decisions to infer the edge server's (ES's) queue information and users' usage patterns, thereby incurring the pattern privacy (PP) issue. Therefore, we propose an offloading strategy which jointly minimizes the latency, ES's energy consumption, and task dropping rate, while preserving PP. Firstly, we formulate the dynamic computation offloading procedure as a Markov decision process (MDP). Next, we develop a <u>D</u>ifferential <u>P</u>rivacy <u>D</u>eep <u>Q</u>-learning based <u>O</u>ffloading (DP-DQO) algorithm to solve this problem while addressing the PP issue by injecting noise into the generated offloading decisions. This is achieved by modifying the deep Q-network (DQN) with a Function-output Gaussian process mechanism. We provide a theoretical privacy guarantee and a utility guarantee (learning error bound) for the DP-DQO algorithm and finally, conduct simulations to evaluate the performance of our proposed algorithm by comparing it with greedy and DQN-based algorithms.

Index Terms—mobile edge computing, computation offloading, differential privacy, deep reinforcement learning.

I. INTRODUCTION

The Internet of Things (IoT) integrates a large number of pervasive, connected, and smart devices in the physical world via the Internet, enabling various smart IoT applications, e.g., deep-learning-driven smart video surveillance, flying ad hoc networks for precision agriculture, and e-health [1]. These IoT applications heavily rely on computationally intensive machine learning algorithms, which are typically resource-hungry and cannot be readily supported by resource-constrained mobile devices (MDs) [2]. Meanwhile, offloading computation to the cloud server (CS) may incur a high latency cost due to the long transmission distance [3]. In light of this, a new computing paradigm called mobile edge computing (MEC) has been proposed, to meet the quality of service (QoS) requirements of latency-sensitive IoT applications. Here, edge servers (ESs) are placed in close proximity to MDs [4], at the network edge (e.g. at base stations or access points). For an MEC system, it is of great importance to design efficient computation offloading strategies through achieving appropriate cooperation among ESs and CSs [5]–[7]. While efficient computation offloading algorithms have been developed for MEC systems under various scenarios [5]–[7], privacy

issues, mainly derived from computation cooperation and data sharing in computation offloading, were ignored. This would potentially provide attackers with the opportunity for privacy mining [8].

Recently, researchers begin to focus on addressing privacy issues in MEC computation offloading, and the available work can be broadly categorized into two types: i.e., data privacy protection [9], [10] and pattern privacy (PP) protection [11]. The data privacy issue refers to the case where attackers directly steal users' private information (e.g., account passwords, email addresses, home addresses, etc.) from the transmitted data during computation offloading. Authors in [9] designed a sampling perturbation encryption strategy to protect data privacy against attackers during computation offloading. In [10], to protect data privacy, a trustworthy access control mechanism was developed by utilizing smart contracts. In contrast to data privacy, the PP issue refers to the case where attackers eavesdrop on the offloading patterns and decisions to infer system information, e.g., the size and required computational resources of offloading tasks. Based on this, attackers could further infer users' private information, such as their identities and usage patterns. To the best of our knowledge, there are very few studies on PP-preserving offloading in MEC systems. In [11], to achieve PP protection, the authors proposed to disguise users' offloading tasks by deliberately generating redundant tasks, sent along with the actual tasks to MEC servers. It achieved a tradeoff between the computation rate and privacy preservation but inevitably incurred extra computation and communication costs from the generation, transmission, and processing of the redundant tasks.

In this work, a novel PP-preserving dynamic computation offloading strategy is devised for MEC systems. Particularly, we consider a dynamic computation offloading problem that jointly minimizes the latency, ES's energy consumption, and task dropping rate, and formulates it as a Markov Decision Process (MDP). To solve this problem while addressing the PP issue, we propose a <u>D</u>ifferential <u>P</u>rivacy <u>D</u>eep <u>Q</u>-learning based <u>O</u>ffloading (DP-DQO) algorithm. The core idea of DP-DQO is to inject noise after the output layer of the deep Q-network (DQN) to make adjacent state-action pairs indistinguishable, where the "distance" between two state-action pairs is evaluated by the difference of their corresponding rewards.

Fig. 1. Computation offloading model for MEC.

This strategy adds noise to the generated offloading decisions, which prevents the attacker from inferring information on the edge queue and users' usage patterns, thereby achieving PP protection. For our proposed DP-DQO algorithm, we provide theoretical analysis on both its privacy guarantee and utility guarantee (learning error bound). Finally, we evaluate DP-DQO's performance through simulations, showing that, by suitably choosing the noise level, DP-DQO can achieve both the high return and PP protection during the computation offloading process.

II. SYSTEM MODEL AND PROBLEM FORMULATION

A. Network Model

As shown in Fig.1, we consider an MEC system consisting of N mobile devices (MDs) served by an edge server (ES) and a cloud server (CS), where the set of MDs is denoted by $\mathcal{N} = \{1, 2, \ldots, N\}$. Compared with the CS, the ES is closer to MDs but does not have sufficient computing resources to serve all MDs simultaneously. In this paper, we consider a time-slotted system $\mathcal{T} = \{1, 2, \ldots, T\}$ where each timeslot's duration is Δ_T (in seconds). At the beginning of each time slot, the ES may receive offloading tasks from MDs. We consider the task arrivals from MD n following an independent Poisson process with parameter λ_n [12]. At the beginning of time slot t, if there is an offloading task sent from MD n, we denote it by $\mathbf{x}_n(t) \triangleq (\rho_n(t), \beta_n(t))$ with $\rho_n(t)$ and $\beta_n(t)$ respectively denoting its data size and required CPU cycles for task computation. For a generic task $\mathbf{x}_n(t) \triangleq (\rho_n(t), \beta_n(t))$, we assume that it is atomic and can not be divided further. Besides, we consider that the data size $\rho_n(t)$ and the required CPU cycles $\beta_n(t)$ follow independent uniform distributions [5].

To buffer the arriving offloading tasks, the ES maintains a task request queue (TRQ) whose size is denoted by q_{max}^{TR}. The newly arriving tasks would be buffered in the TRQ if there is enough space there, and dropped otherwise. After the TRQ is updated in time slot t, we denote $I(t)$ as the total number of offloading tasks that have arrived at the ES, and $K(t)$ as the combined size of the tasks buffered at the ES, i.e., $K(t) \leq q_{max}^{TR}$. For the updated TRQ, the ES would deal with the task appearing at its head, denoted by $\mathbf{x}_H(t) \triangleq (\rho_H(t), \beta_H(t))$, deciding between computing it locally (at the ES), or delivering it to the CS via one of M orthogonal channels. At the beginning of time slot t,

let $\omega(t)$ denote the number of available orthogonal channels, i.e., $\omega(t) \leq M$. Besides, let $A(t) \in \{0, 1\}$ denote the ES's computation offloading decision at time slot t, i.e., $A(t) = 1$ if the task is offloaded to the CS for processing, and $A(t) = 0$ otherwise. It is worth noting that if there is no free channel between the ES and CS at the beginning of time slot t, then the ES has to process the task locally, i.e., $A(t) = 0$ if $\omega(t) = 0$.

The ES maintains another queue for the tasks to be processed locally, called the local computing queue (LCQ), according to the first-come-first-serve (FCFS) discipline. We denote LCQ's size by q_{max}^{LC}. At the beginning of time slot t, we denote the total number of CPU cycles required by the tasks waiting in the LCQ by $\hat{P}(t)$, and the combined size of these tasks by $\hat{K}(t)$ with $\hat{K}(t) \leq q_{max}^{LC}$. In each time slot, the arriving task $\mathbf{x}_H(t) \triangleq (\rho_H(t), \beta_H(t))$ (if $A(t) = 0$) would be buffered only if there is enough space in the LCQ, i.e., $\hat{K}(t) + \rho_H(t) \leq q_{max}^{LC}$. At the end of time slot t, we denote by $I_d(t)$ the number of tasks that are dropped due to the overflow of the TRQ or LCQ.

In this paper, we aim at optimizing the computation offloading decisions to jointly minimize the incurred latency, energy consumption of the ES, and task dropping rate. Note that the waiting time of each task in the TRQ is determined by the task arrival processes and is independent of the ES's offloading decisions. As such, we equivalently focus on minimizing the latency, ES's energy consumption, and task dropping rate of the tasks appearing at the head of the TRQ, while ignoring their waiting time in the TRQ.

B. Latency and Energy Consumption Model

For the task at the head of the TRQ in time slot t, i.e., $\mathbf{x}_H(t) \triangleq (\rho_H(t), \beta_H(t))$, the incurred latency, and energy consumption are determined by the offloading decision made by the ES. In this subsection, we specify the latency and energy consumption models regarding local computing at the ES and offloading computation to the CS, respectively.

1) Local Computing at the ES: For this case, the latency $L_H^e(t)$ consists of two parts, i.e., the waiting time in the LCQ $L_H^w(t)$ and the computation execution time $L_H^c(t)$, i.e.,

$$L_H^e(t) = L_H^w(t) + L_H^c(t) = \left(\hat{P}(t) + \beta_H(t) \right) / f_e, \quad (1)$$

in which the waiting time $L_H^w(t)$ is the sum of the computation execution time of all tasks before the task $\mathbf{x}_H(t)$ in the LCQ, i.e., $L_H^w(t) = \hat{P}(t)/f_e$ with f_e (in CPU cycles per second) denoting the ES's computation capacity. Meanwhile, the energy consumption for the local computing at the ES can be expressed as [13]

$$E_H^e(t) = \kappa_1 \left(f_e \right)^2 \beta_H(t), \quad (2)$$

where κ_1 is an effective capacitance coefficient dependent on the ES's chip architecture, and $\kappa_1 \left(f_e \right)^2$ denotes the energy consumption per CPU cycle for the local computing.

2) Offloading Computation to the CS: For wireless MEC systems, the task transmission time from the ES to the CS is generally much larger compared to the computation time at the CS [6]. Therefore, by ignoring the task computation time, we specify the offloading latency $L_H^o(t)$ as

$$L_H^o(t) = \rho_H(t)/r^{tr}. \qquad (3)$$

Wherein, r^{tr} denotes the transmission rate from the ES to the CS, which is considered to be constant [14]. Besides, the energy consumption of the ES $E_H^o(t)$ can be expressed as

$$E_H^o(t) = p^{tr}L_H^o(t), \qquad (4)$$

where p^{tr} denotes the uplink transmission power for the ES.

C. Dynamic Computation Offloading Problem Formulation

Firstly, to balance between the incurred latency and energy consumption, we define the task execution cost $C_0(t)$ of the offloading decision in time slot t, i.e, $A(t)$, as

$$C_0(t) = \begin{cases} L_n^e(t) + \psi E_n^e(t), & \text{if } A(t) = 0 \\ L_n^o(t) + \psi E_n^o(t), & \text{if } A(t) = 1 \end{cases} \qquad (5)$$

where ψ is a parameter introduced to balance the trade-off between lowering the latency and reducing the energy consumption. Meanwhile, due to the limited buffer size and channel resources, some tasks may be dropped from TRQ or LCQ. Bearing this in mind, to further reduce the number of dropped tasks, we define the overall cost in time slot t as

$$C(t) = C_0(t)/[1 - I_0(t)] = C_0(t)/[1 - I_d(t)/I(t)] \qquad (6)$$

where $I_0(t) = I_d(t)/I(t)$ denotes the task dropping rate at the end of the time slot.

In light of this, the dynamic computation offloading problem can be formulated as

$$\mathbf{P}: \min_{\mathbf{A}^T} \lim_{T \to \infty} \sum_{t=1}^{T} \gamma^{t-1}C(t) \qquad (7)$$

$$\text{s.t. } A(t) \in \{0, 1\}, \quad \forall t \in \mathcal{T} \qquad (8)$$

$$\omega(t) \leq M, \quad \forall t \in \mathcal{T} \qquad (9)$$

where $\mathbf{A}^T = (A(1), A(2), \ldots, A(T))$ denotes the sequence of offloading decisions made by the ES from time slot 1 to T, and the discount factor $\gamma \in [0, 1)$ is introduced to give importance to the present cost and to make the long-term cumulative cost finite. Besides, constraint (9) indicates that there are at most M orthogonal channels that can be used by the ES in each time slot.

Essentially, problem $\mathbf{P}$ can be formulated as an MDP and solved by reinforcement learning (RL)/ deep reinforcement learning (DRL) algorithms, where the action-value function (or policy) should be learned through trial-and-error interactions with the environment [15]. In this case, by observing the offloading decisions made by the ES, the attacker (e.g., malicious MDs) can use tools such as inverse reinforcement learning [16] to infer, for instance, the action-value function. With the inferred action-value function and the observed offloading decision, the attacker can infer the ES's state information [17], e.g., the size of buffered tasks and the required CPU cycles. On this basis, the attacker could further obtain the usage patterns of MDs, causing the PP issue. For instance, considering an MEC system consisting of MDs with different usage patterns, the attacker can identify a specific MD from a set of anonymous ones. To address this issue, we propose a novel PP-preserving dynamic computation offloading algorithm, called

DP-DQO, so as to minimize the incurred long-term cumulative cost (7) while achieving PP protection during the computation offloading process.

III. PATTERN PRIVACY (PP) PRESERVING SOLUTION

A. MDP Formulation and Differential Privacy

We formulate the dynamic offloading problem as an MDP consisting of a tuple $(\mathbb{S}, \mathcal{A}, R(\cdot, \cdot))$ and depicted as follows:

1) State space $\mathbb{S}$: The state $\mathcal{S}(t)$ at the beginning of time slot t is defined as $\mathcal{S}(t) = (K(t), \hat{K}(t), \hat{P}(t), \omega(t))$, where $K(t)$ denotes the combined size of tasks buffered in the TRQ, $\hat{K}(t)$ and $\hat{P}(t)$ the combined size and required CPU cycles of tasks in the LCQ, and $\omega(t)$ the number of available channels. Denote the space of all possible states by $\mathbb{S}$.

2) Action space $\mathcal{A}$: The action of the agent at time slot t is the offloading decision $A(t)$ i.e., $\mathcal{A} = \{0, 1\}$.

3) Reward function $R(\cdot, \cdot)$: In each time slot, the reward obtained by the agent is dependent on the state $\mathcal{S}(t)$ and executed action $A(t)$, which is defined as the negative of the overall cost, i.e., $R(\mathcal{S}(t), A(t)) = -C(t)$.

In this work, we aim at deriving a policy π^* that maximizes the discounted accumulative rewards (i.e., the return) given an initial state $\mathcal{S}(1)$, i.e.,

$$\pi^* = \arg\max_{\pi} \lim_{T \to \infty} \mathbb{E}\Big[\sum_{t=1}^{T} \gamma^{t-1}R(\mathcal{S}(t), A(t)) \,|\, \mathcal{S}(1) \Big]$$

$$= \arg\min_{\pi} \lim_{T \to \infty} \mathbb{E}\Big[\sum_{t=1}^{T} \gamma^{t-1}C(t) \Big]. \qquad (10)$$

To solve this problem while addressing the PP issue, we devise a novel computation offloading algorithm by modifying the DQN-based DRL algorithm with the Function-output Gaussian process mechanism (FGPM). Particularly, the core idea of our proposed algorithm is to inject the generated noise into the output of the DQN to provide differential privacy (DP) throughout the offloading process. Before formally presenting our proposed algorithm, we start by introducing some necessary definitions.

Definition 1: $((\epsilon, \delta)$-DP [18]): A random mechanism $\mathcal{M}$: $\mathcal{R} \to \mathcal{U}$ with domain $\mathcal{R}$ and range $\mathcal{U}$ satisfies (ϵ, δ)-DP, if for any two adjacent inputs $r, r' \in \mathcal{R}$ and for any subset of outputs $\mathcal{X} \subseteq \mathcal{U}$ we have

$$\Pr[\mathcal{M}(r) \in \mathcal{X}] \leq e^{\epsilon}\Pr[\mathcal{M}(r') \in \mathcal{X}] + \delta \qquad (11)$$

where ϵ denotes the privacy budget and δ the relaxation factor.

Definition 2: For a random mechanism $\mathcal{M}$: $\mathcal{R} \to \mathcal{U}$, its sensitivity $\Delta_{\mathcal{M}}$ is defined as the maximum difference between the query results of two adjacent inputs $r, r' \in \mathcal{R}$, i.e, [18]

$$\Delta_{\mathcal{M}} = \sup_{r, r' \in \mathcal{R}} \|\mathcal{M}(r) - \mathcal{M}(r')\| \qquad (12)$$

where $r, r' \in \mathcal{R}$ denotes a pair of adjacent inputs, and $\|\cdot\|$ the an norm function defined on $\mathcal{U}$.

It is noteworthy that for a random mechanism $\mathcal{M}$, the sensitivity is used to quantitatively assess its effect on data privacy. Particularly, the larger sensitivity of the mechanism $\mathcal{M}$, the larger probability of privacy leakage, and the allocated privacy budget ϵ to make the mechanism $\mathcal{M}$ satisfy DP is also

larger. By considering the action-value function as a random mechanism and the state-action pairs as its inputs, it seems reasonable to develop a DP-based DRL algorithm to solve MDPs while making the learned action-value function satisfy DP. To achieve this, the traditional action-value function defined in RL needs to be modified with the noise mechanism [18]. In this paper, we choose the Function-output Gaussian process mechanism (FGPM) to deal with the continuous inputs, which extends the Gaussian mechanism with the reproducing kernel Hilbert space (RKHS).

Definition 3: (Function-output Gaussian process mechanism (FGPM) $F(\cdot)$ [19]): Given a random mechanism $\mathcal{M}: \mathcal{R} \to \mathcal{U}$ with sensitivity $\Delta_{\mathcal{M}}$, the FGPM $F(\cdot)$ is defined as:

$$F(r) = \mathcal{M}(r) + g, \forall r \in \mathcal{R} \tag{13}$$

where g denotes the noise sampled from the Gaussian process noise $\mathcal{N}\left(0, \sigma^2 K\right), \mathcal{U}$ an RKHS with kernel function K, and σ the noise level that can be specified according to the allocated privacy budget.

Proposition 1: If the privacy budget ϵ satisfies $0 < \epsilon < 1$, the relaxation factor δ and the noise level σ satisfy $\sigma \geq \sqrt{2 \ln(1.25/\delta)}\Delta_{\mathcal{M}}/\epsilon$, then the FGPM $F(\cdot)$ satisfies (ϵ, δ)-DP.

Proof 1: We refer readers to [19] for the detailed proof.

In the following two subsections, the details of our proposed algorithm and the theoretical analysis on its privacy and utility Guarantees will be elaborated on respectively.

B. Differentially Private Deep Q-learning based Offloading (DP-DQO) Algorithm

The details of our proposed DP-DQO algorithm are presented in Algorithm 1. At the beginning of DP-DQO, the experience replay buffer $\mathbb{D}$ is cleared out, the parameters of the Q-network θ are randomly initialized, and the parameters of the target Q-network are set as $\hat{\theta} = \theta$. When the initialization is completed, the algorithm goes into a loop and the learning process is divided into Γ episodes, each of which comprises T time slots. At beginning of each episode, the initial state $\mathcal{S}(1)$, sorted state sets $\{\mathbb{S}_0\}_{A \in \mathcal{A}}$, and noise dictionaries $\{\mathcal{G}_A\}_{A \in \mathcal{A}}$ are respectively initialized. Then, in time slot t, an action $A(t)$ is chosen by following the $\mathcal{E}$-greedy policy. After the chosen action is executed, the corresponding experience tuple $((\mathcal{S}(t), A(t), R(\mathcal{S}(t), A(t)), \mathcal{S}(t + 1))$ is observed and further stored into the replay buffer $\mathbb{D}$.

After Γ_{eps} episodes are completed, functional noise generating and parameter updating are performed. Specifically, a mini-batch of Ω experience tuples are randomly sampled from the replay buffer, the set of which is denoted by $\{(\mathcal{S}_i, A_i, R_i, \mathcal{S}_i') | 1 \leq i \leq \Omega\}$. Then, the sampled tuples are utilized in sequence to update the noise dictionaries and Q-network. Firstly, for the i-th experience tuple, the next state $\mathcal{S}_i'$ is inserted into the set $\mathbb{S}_{A_i}$ in the ascending order of the reward. Secondly, a Gaussian process $\mathcal{N}\left(\mu_{\mathcal{S}_i'}, \sigma d_{A,\mathcal{S}_i'}\right)$ is constructed to generate noise for each action A ($\forall A \in \mathcal{A}$)of state $\mathcal{S}_i'$, where $\mu_{A,\mathcal{S}_i'}$ and $\sigma d_{A,\mathcal{S}_i'}$ respectively denote its mean and variance. The expressions of $\mu_{A,\mathcal{S}_i'}$ and $d_{A,\mathcal{S}_i'}$ are presented as (14) and (15), where $\Psi = (4\alpha(z+1)/\Omega)^{-1}$ with α, z, and Ω respectively denoting the learning rate, balance factor, and

---

Algorithm 1 DP-DQO Algorithm

1: **Initialization:** Initialize replay buffer $\mathbb{D}$, Q-network Q_θ, target Q-network $\hat{Q}_{\hat{\theta}}$, and training start time as Γ_{eps}.
2: **for** $episode = 1, \Gamma$ **do**
3:     Set the state $\mathcal{S}(1) = (0, 0, 0, M)$, state sets $\{\mathbb{S}_0\}_{A \in \mathcal{A}} = \{\emptyset\}_{A \in \mathcal{A}}$, and noise dictionaries $\{\mathcal{G}_A\}_{A \in \mathcal{A}} = \{\emptyset\}_{A \in \mathcal{A}}$.
4:     **for** $t = 1, T$ **do**
5:         **Action selection:** With probability $\mathcal{E}$ select a random action $A(t)$, otherwise choose the action $A(t) = \arg\max_{A \in \mathcal{A}} Q_\theta\left(\mathcal{S}(t), A(t)\right)$.
6:         **Acting and observing:** Execute the action $A(t)$, receive reward $R(\mathcal{S}(t), A(t))$, and obtain the new state $\mathcal{S}(t + 1)$.
7:         **Refreshing replay buffer:** Store the new transition $((\mathcal{S}(t), A(t), R(\mathcal{S}(t), A(t)), \mathcal{S}(t + 1))$ into $\mathbb{D}$.
8:         **if** $episode > \Gamma_{eps}$ **then**
9:             **Training:** Sample a mini-batch of transitions $\{(\mathcal{S}_i, A_i, R_i, \mathcal{S}_i') | 1 \leq i \leq \Omega\}$ from $\mathbb{D}$.
10:             **for** $i = 1, \Omega$ **do**
11:                 Insert $\mathcal{S}_i'$ into the sorted state set $\mathbb{S}_{A_i}$.
12:                 **for** $A \in \mathcal{A}$ **do**
13:                     Build a Gaussian process $\mathcal{N}\left(\mu_{A,\mathcal{S}_i'}, \sigma d_{A,\mathcal{S}_i'}\right)$ (14) and (15), sample the noise $g_{A,\mathcal{S}_i'} \sim \mathcal{N}\left(\mu_{A,\mathcal{S}_i'}, \sigma d_{A,\mathcal{S}_i'}\right)$, update the dictionary $\mathcal{G}_A$.
14:                 **end for**
15:                 Calculate the target y_i with (16).
16:             **end for**
17:             Update the Q-network Q_θ with (18).
18:             **Update target network:** $\hat{\theta} = \theta$ every Γ_0 episodes.
19:         **end if**
20:     **end for**
21: **end for**
22: **Output:** The Q-network Q_θ.

---

batch size. Besides, $\zeta = \left\|\mathcal{S}_i' - \mathcal{S}_i'^{-}\right\|_2$, $\nu = \left\|\mathcal{S}_i'^{+} - \mathcal{S}_i'\right\|_2$ and $\Lambda = \left\|\mathcal{S}_i'^{+} - \mathcal{S}_i'^{-}\right\|_2$, in which $\mathcal{S}_i'^{+}$ and $\mathcal{S}_i'^{-}$ are two adjacent states of $\mathcal{S}_i'$ in the ordered set $\mathbb{S}_{A_i}$. And, $G_A(S)$ represents the value of the key $\mathcal{S}$ in the dictionary $\mathcal{G}_A$.

$$\mu_{A,\mathcal{S}_i'} = \frac{\left(e^{\Psi\zeta} - e^{-\Psi\zeta}\right) G_A\left(\mathcal{S}_i'^{+}\right) + \left(e^{\Psi\nu} - e^{-\Psi\nu}\right) G_A\left(\mathcal{S}_i'^{-}\right)}{-e^{-\Psi\Lambda} + e^{\Psi\Lambda}} \tag{14}$$

$$d_{A,\mathcal{S}_i'} = 1 - \frac{\left(e^{\Psi\zeta} - e^{-\Psi\zeta}\right) e^{\Psi\zeta} + \left(e^{\Psi\nu} - e^{-\Psi\nu}\right) e^{\Psi\nu}}{-e^{-\Psi\Lambda} + e^{\Psi\Lambda}} \tag{15}$$

For each state-action pair $(\mathcal{S}_i', A), \forall A \in \mathcal{A}$, the noise is generated according to $g_{A,\mathcal{S}_i'}$, which is further utilized to update the dictionary, i.e., $\mathcal{G}_A = \mathcal{G}_A \bigcup \{\mathcal{S}_i', g_{A,\mathcal{S}_i'}\}$, and calculate the target y_i with the target Q-network $\hat{Q}_{\hat{\theta}}$, i.e.,

$$y_i = R(\mathcal{S}_i, A_i) + \gamma \arg\max_{A' \in \mathcal{A}} \left(\hat{Q}_{\hat{\theta}}\left(\mathcal{S}_i', A'\right) + G_{A'}(\mathcal{S}_i')\right). \tag{16}$$

Then, the average of the noised temporal-difference (TD) errors can be expressed

$$\mathcal{L} = 1/\Omega \sum_{i=1}^{\Omega} \left[y_i - \left(Q_\theta\left(\mathcal{S}_i, A_i\right) + G_{A_i}(\mathcal{S}_i)\right)\right]^2 \tag{17}$$

with which the Q-network Q_θ can updated according to

$$\theta = \theta - \alpha \nabla_\theta \mathcal{L}. \quad (18)$$

During the training phase, the target Q-network $\hat{Q}_{\hat{\theta}}$ is updated by setting $\hat{\theta} = \theta$ every Γ_0 epochs.

C. Privacy and Utility Guarantees of DP-DQO Algorithm

In this subsection, we analyze both the privacy guarantee and utility guarantee for our proposed DP-DQO algorithm, as shown in the following two theorems.

Theorem 1: (Privacy Guarantee) The action-value function learned by DP-DQO is $\left(\epsilon, \delta + \exp\left(-(2z - 8.68\sqrt{\Psi}\sigma)^2/2\right)\right)$ -DP, if $2z > 8.68\sqrt{\Psi}\sigma$ and $\sigma \geq J(\alpha, z, D, \Omega)\Delta_F$ $\sqrt{2((\Gamma - \Gamma_{eps})T/\Omega)\ln(e + \epsilon/\delta)}/\epsilon$, in which $J(\alpha, z, D, \Omega)$ $= \left((4\alpha(z+1)/\Omega)^2 + 4\alpha(z+1)/\Omega\right)D^2$, $\Psi = (4\alpha(z+1)$ $/\Omega)^{-1}$, D denotes the Lipschitz constant of the action-value function approximation, Ω the size of mini-batch, and $(\Gamma - \Gamma_{eps})T$ the total number of steps in training. Wherein, the two state-action pairs are called adjacent if their "distance", evaluated by the difference of their corresponding rewards (i.e., R and R'), is less than the sensitivity of the FGPM Δ_F, i.e., $\|R - R'\|_1 \leq \Delta_F$.

Proof 2: Let Q and Q' respectively denote the estimated Q-values given two adjacent state-action pairs, the corresponding rewards of which satisfy $\|R - R'\|_1 \leq \Delta_F$. To establish the DP guarantee, we will check the update step in line 17 of Algorithm 1. Firstly, let $\widetilde{Q}$ denote the Q-value before updating the Q-network. Then, we have

$$\left\|Q - \widetilde{Q}\right\|_1 \leq \alpha D \left(2 + G_A(\mathcal{S}_i') - G_A(\mathcal{S}_i)\right)/\Omega. \quad (19)$$

From Lemma 8 of [19], $\left\|Q - \widetilde{Q}\right\|_1 \leq 2\alpha D(z+1)/\Omega$ is satisfied with probability at least $1 - \exp\left(-(2z - 8.68\sqrt{\Psi}\sigma)^2/2\right)$. Similarly, we have $\left\|Q' - \widetilde{Q}\right\|_1 \leq 2\alpha D(z+1)/\Omega$ is satisfied with probability at least $1 - \exp\left(-(2z - 8.68\sqrt{\Psi}\sigma)^2/2\right)$. Furthermore, by the triangle inequality, it can be derived that $\|Q - Q'\|_1 \leq 4\alpha D(z+1)/\Omega$ for any $\|R - R'\|_1 \leq \Delta_F$. Next, by defining $d = Q - Q'$ and resorting to Lemma 6 in [19], we have

$$\|d\|_{\mathcal{H}}^2 \leq (1 + \Psi/2)(4\alpha D(z+1)/\Omega)^2 + D^2/2\Psi \quad (20)$$

where $\|\cdot\|_{\mathcal{H}}$ denotes the RKHS norm. By setting $1/\Psi = 4\alpha(z+1)/\Omega$, we can transform (20) as

$$\|d\|_{\mathcal{H}}^2 \leq ((4\alpha(z+1)/\Omega)^2 + 4\alpha(z+1)/\Omega)D^2. \quad (21)$$

Referring to Definition 3 and Proposition 1, we inject noise sampled by the Gaussian process to Q that makes the update step achieve $\left(\epsilon, \delta + \exp\left(-(2z - 8.68\sqrt{\Psi}\sigma)^2/2\right)\right)$-DP, given that $\sigma \geq \sqrt{2\ln(1.25/\delta)}\|d\|_{\mathcal{H}}/\epsilon$. This shows that each iteration of update has a privacy guarantee. Finally, according to the composition theorem in [20], multiple iterations in our proposed algorithm provide a privacy guarantee. $\square$

Next, we establish the utility guarantee of DP-DQO by analyzing the discrepancy between the action-value of some

state-action pairs learned by our algorithm and that regarding the optimal policy, i.e., the learning error (utility loss) [19]. In Theorem 2, we provide the upper bound of the learning error for our algorithm, which tends towards zero as the size of the state space approaches infinity, i.e., a utility guarantee is achieved.

Theorem 2: (Utility Guarantee) Let Q and Q^* respectively denote the action-value of some state-action pairs learned by our algorithm and that regarding the optimal policy, and n the cardinality of the state space $\mathbb{S}$, i.e., $|\mathbb{S}| = n$. In the case $n < \infty$, and $\gamma < 1$, the utility loss (learning error) of the algorithm satisfies

$$\mathbb{E}\left[\|Q - Q^*\|_1\right] \leq \frac{2\sqrt{2}\sigma}{\sqrt{n}\pi(1 - \gamma)}. \quad (22)$$

Proof 3: Due to space limitations, we omit the derivations and refer readers to [19] for the detailed proof. $\square$

IV. SIMULATION RESULTS

In this section, we conduct simulations to evaluate the performance of our proposed DP-DQO algorithm. Consider an MEC system consisting of a CS, an ES, and $N = 5$ MDs. The length of the time slot is set as $\Delta_T = 1$. The data size and the required CPU cycles for arriving tasks are uniformly generated between $[5, 50]$ MB and $[0.5 \times 10^{11}, 2.0 \times 10^{11}]$ cycles, respectively. Besides, we respectively set the size of the TRQ and LCQ as $q_{max}^{TR} = 5$ GB and $q_{max}^{LC} = 2$ GB, and the computation capacity and effective capacitance coefficient of the ES as $f^e = 5 \times 10^{10}$ and $\kappa_1 = 10^{-11}$ [21]. For the link from the ES to CS, we adopt the channel model and parameters as in [14] and correspondingly set the transmission rate as $r^{tr} = 5$ MB/s. The performance of our proposed DP-DQO algorithm is evaluated against two baseline algorithms: 1) the *Greedy algorithm* and 2) the *DQN-based algorithm*. For DRL algorithms, each artificial neural network (ANN) consists of two fully connected hidden layers, each with 128 neurons, where the ReLU function is adopted as the activation function. We set the length of one episode as $T = 100$, the size of replay buffer $\mathbb{D} = 2000$, the mini-batch size $\Omega = 64$, and the target network update frequency $\Gamma_0 = 10$. To achieve exploration, we adopt the $\mathcal{E}$-greedy policy with $\mathcal{E} = 0.02$. The learning rate α and discount factor γ are set to 0.002 and 0.98, respectively. Here, all simulation results are obtained by averaging 10 independent runs with different seeds, and for fair comparisons, the same seed is adopted for all algorithms and policies in one run.

Firstly, we evaluate the convergence of DP-DQO with different noise levels, as shown in Fig. 2. The dark curve (solid or dotted) shows the mean value over runs, and the shaded areas are obtained by filling the interval between the maximum and minimum values over runs. In Fig. 2, it can be seen that the training performance is negatively correlated with the noise level σ. This is mainly due to the fact that a higher level of privacy protection would reduce the difference between the Q-values of adjacent actions more significantly. In return, this may cause more uncertainty in the Q-network's

Fig. 2. Convergence comparison for DP-DQO at noise levels $\sigma = \{0.1, 0.3, 0.5, 0.7\}$ and two baseline algorithms.

Fig. 3. Performance comparison for DP-DQO at noise levels $\sigma = \{0.1, 0.3, 0.5, 0.7\}$ and two baseline algorithms, in terms of the return where the task arrival rate from 0.1 to 0.4.

update direction, thereby making the algorithm's performance more unstable. To further evaluate the effectiveness of DP-DQO, we vary the task arrival rate from 0.1 to 0.4, and present the achieved return in Fig. 3. It can be seen that our proposed DP-DQO algorithm outperforms the greedy algorithm in all cases. Besides, it is noteworthy that when the noise level is not high, e.g., $\sigma = 0.1$, the return achieved by our proposed DP-DQO algorithm is roughly the same as that of the DQN-based algorithm. In other words, by suitably choosing the noise level, a high utility guarantee (low learning error) can be achieved while preserving the PP protection during the computation offloading process.

V. CONCLUSIONS

In this paper, we have investigated and proposed a computation offloading strategy for the MEC system which not only minimizes the offloading cost, consisting of the latency, energy consumption of the edge server, and task dropping rate but also preserves Pattern Privacy (PP) during the offloading process. This prevents attackers from inferring the edge server's queuing information and users' usage patterns through inverse reinforcement learning and other techniques when they observe the offloading decisions. By modifying the vanilla DQN with the Function-output Gaussian Process Mechanism, a novel PP-preserving dynamic computation offloading algorithm, called DP-DQO, was proposed. For DP-DQO, we provided theoretical analysis on both its privacy guarantee and utility (learning error) guarantee. Our simulation results show that DP-DQO with a suitable noise level performs as well as the DQN-based algorithm in terms of its achieved return and significantly outperforms the greedy algorithm.

REFERENCES

[1] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE Internet Things J.*, vol. 3, no. 5, pp. 637–646, Oct. 2016.

[2] X. Chen, Q. Shi, L. Yang, and J. Xu, "ThriftyEdge: Resource-efficient edge computing for intelligent IoT applications," *IEEE Netw.*, vol. 32, no. 1, pp. 61–65, Jan. 2018.

[3] 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.

[4] Z. Zhou, X. Chen, E. Li, L. Zeng, K. Luo, and J. Zhang, "Edge intelligence: Paving the last mile of artificial intelligence with edge computing," *Proc. IEEE*, vol. 107, no. 8, pp. 1738–1762, Aug. 2019.

[5] J. Ren, G. Yu, Y. He, and G. Li, "Collaborative cloud and edge computing for latency minimization," *IEEE Trans. Veh. Technol.*, vol. 68, no. 5, pp. 5031–5044, May 2019.

[6] G. Peng, H. Wu, H. Wu, and K. Wolter, "Constrained multiobjective optimization for IoT-enabled computation offloading in collaborative edge and cloud computing," *IEEE Internet Things J.*, vol. 8, no. 17, pp. 13 723–13 736, Sept. 2021.

[7] G. Qu, H. Wu, and R. Li, "DMRO: A deep meta reinforcement learning-based task offloading framework for edge-cloud computing," *IEEE Trans. Netw. Service Manag.*, vol. 18, no. 3, pp. 3448–3459, Sept. 2021.

[8] Y. Liu, M. Peng, G. Shou, Y. Chen, and S. Chen, "Toward edge intelligence: Multiaccess edge computing for 5G and Internet of Things," *IEEE Internet Things J.*, vol. 7, no. 8, pp. 6722–6747, Aug. 2020.

[9] S. Chen, X. Zhu, H. Zhang, and C. Zhao, "Efficient privacy preserving data collection and computation offloading for fog-assisted IoT," *IEEE Trans. Sustain. Comput.*, vol. 5, no. 4, pp. 526–540, Oct.-Dec. 2020.

[10] D. C. Nguyen, P. N. Pathirana, M. Ding, and A. Seneviratne, "A cooperative architecture of data offloading and sharing for smart healthcare with blockchain," in *Proc. IEEE ICBC*, May 2021, pp. 1–8.

[11] P. Zhao, J. Tao, L. Kangjie, G. Zhang, and F. Gao, "Deep reinforcement learning-based joint optimization of delay and privacy in multiple-user MEC systems," *IEEE Trans. Cloud Comput.*, pp. 1–1, Jan. 2022.

[12] L. Cui, C. Xu, S. Yang, J. Z. Huang, J. Li, X. Wang, Z. Ming, and N. Lu, "Joint optimization of energy consumption and latency in mobile edge computing for Internet of Things," *IEEE Internet Things J.*, vol. 6, no. 3, pp. 4791–4803, Jun. 2019.

[13] J. Yan, S. Bi, and Y. J. A. Zhang, "Offloading and resource allocation with general task graph in mobile edge computing: A deep reinforcement learning approach," *IEEE Trans. Wireless Commun.*, vol. 19, no. 8, pp. 5404–5419, Aug. 2020.

[14] J. Ren, G. Yu, Y. Cai, and Y. He, "Latency optimization for resource allocation in mobile-edge computation offloading," *IEEE Trans. Wireless Commun.*, vol. 17, no. 8, pp. 5506–5519, Aug. 2018.

[15] S. Manisha, K. T. George, G. Sunil, and R. Santu, "Sympathy-based reinforcement learning agents," in *Proc. AAMAS*, 2022, pp. 1164–1172.

[16] S. Arora and P. Doshi, "A survey of inverse reinforcement learning: Challenges, methods and progress," *Artif. Intell.*, 2021.

[17] X. Pan, W. Wang, X. Zhang, B. Li, J. Yi, and D. Song, "How you act tells a lot: Privacy-leaking attack on deep reinforcement learning," in *Proc. AAMAS*, 2019, pp. 368–376.

[18] C. Dwork and A. Roth, "The algorithmic foundations of differential privacy," *Found. Trends Theor. Comput. Sci.*, vol. 9, no. 3-4, pp. 211–407, 2014.

[19] B. Wang and N. Hegde, "Privacy-preserving Q-learning with functional noise in continuous spaces," in *Proc. Adv. Neural Inf. Process. Syst.*, 2019, pp. 11 327–11 337.

[20] P. Kairouz, S. Oh, and P. Viswanath, "The composition theorem for differential privacy," in *Proc. ICML*, 2015, pp. 1376–1385.

[21] W. Zhang, Y. Wen, K. Guan, D. Kilper, and D. O. Wu, "Energy-optimal mobile cloud computing under stochastic wireless channel," *IEEE Trans. Wireless Commun.*, vol. 12, no. 9, pp. 4569–4581, Sept. 2013.