electronics

Article

Cooperative Service Caching and Task Offloading in Mobile Edge Computing: A Novel Hierarchical Reinforcement Learning Approach

Tan Chen [1], Jiahao Ai [1], Xin Xiong [2] and Guangwu Hu [3,*]

1 College of Computer Science, Beijing University of Technology, Beijing 100124, China; chentan@bjut.edu.cn (T.C.); aijiahao2021@emails.bjut.edu.cn (J.A.)
2 School of Information Technology, Beijing City University, Beijing 101309, China; xiongxin@bcu.edu.cn
3 School of Computer Sciences, Shenzhen Institute of Information Technology, Shenzhen 518172, China
* Correspondence: hugw@sziit.edu.cn

Abstract: In the current mobile edge computing (MEC) system, the user dynamics, diversity of applications, and heterogeneity of services have made cooperative service caching and task offloading decision increasingly important. Service caching and task offloading have a naturally hierarchical structure, and thus, hierarchical reinforcement learning (HRL) can be used to effectively alleviate the dimensionality curse in it. However, traditional HRL algorithms are designed for short-term missions with sparse rewards, while existing HRL algorithms proposed for MEC lack delicate a coupling structure and perform poorly. This article introduces a novel HRL-based algorithm, named hierarchical service caching and task offloading (HSCTO), to solve the problem of the cooperative optimization of service caching and task offloading in MEC. The upper layer of HSCTO makes decisions on service caching while the lower layer is in charge of task offloading strategies. The upper-layer module learns policies by directly utilizing the rewards of the lower-layer agent, and the tightly coupled design guarantees algorithm performance. Furthermore, we adopt a fixed multiple time step method in the upper layer, which eliminates the dependence on the semi-Markov decision processes (SMDPs) theory and reduces the cost of frequent service replacement. We conducted numerical evaluations and the experimental results show that HSCTO improves the overall performance by 20%, and reduces the average energy consumption by 13% compared with competitive baselines.

Keywords: mobile edge computing (MEC); hierarchical reinforcement learning (HRL); service caching; task offloading

check for updates

Academic Editor: Mehdi Sookhak

Received: 28 November 2024
Revised: 14 January 2025
Accepted: 17 January 2025
Published: 19 January 2025

Citation: Chen, T.; Ai, J.; Xiong, X.; Hu, G. Cooperative Service Caching and Task Offloading in Mobile Edge Computing: A Novel Hierarchical Reinforcement Learning Approach. *Electronics* **2025**, *14*, 380. https://doi.org/10.3390/electronics14020380

1. Introduction

In recent years, as a promising technology, mobile edge computing (MEC) brings computation, storage, and software resources from the cloud center to the edge of network. By shortening communication times and reducing the energy consumption of mobile terminals (MTs), it greatly facilitates users near edge servers (ESs) [1]. Various applications exist that run on mobile terminals, and some tasks can be fully or partially offloaded to edge servers to enhance the quality-of-use experience (QoE) [2–5]. Obviously, the corresponding service, which is the server-side component of the application composed of executable code, library, and database, should first be loaded on ESs to construct a running environment for incoming tasks [6,7].

In order to simplify the model, some previous research works have made the primary assumption that all tasks request the same type of service. Apparently, this assumption is invalid in most cases. For example, the process of handling an augmented reality (AR) task may typically need an action tracking module, a visual processing module, and a light prediction module. An image processing task may consist of a preprocessing module, a feature extraction module, and a classification module [8]. Moreover, an intelligent healthcare task application [9], which usually collects data from wearable devices, may normally require a disease diagnosis module, a health alert and prediction module, and a security and privacy module. Different functions cause different needs for various types of resources; therefore, these services exhibit significant heterogeneity in terms of computing resource requirements, storage space occupation, and energy consumption. Unfortunately, one ES cannot cache all types of services at the same time due to its limited resources. Hence, an ES should establish its service caching policy to decide which services will be hosted according to its available storage space and predictions of future task offloading. The combination of user mobility, the randomness in the type of incoming task, the heterogeneity of service, and the time-dependent state of the underlying wireless network causes highly dynamic and diverse demands for computing resources. Therefore, in order to enhance the performance of the MEC system for various applications, including delay-sensitive applications and computationally intensive applications, the service caching decision and task offloading decision should be made cooperatively and collaboratively.

Numerous works have been devoted to the research area of the joint optimization of service caching and task offloading, and these have applied a Lyapunov-based optimization framework [10,11], convex optimization technology [12,13], multi-agent and hierarchical reinforcement learning [14–20], or a hybrid scheme [21–23] in various MEC scenarios to address different optimization problems with different goals, including minimizing delay, decreasing overall energy consumption, or maximizing the utility function, which involves a trade-off between delay and energy consumption. Although these works have made great contributions in this area, this article endeavors to tackle three open challenges.

First, with the expansion of the MEC scale and the rise of service types, the dimensions of decision variables, simultaneously containing service caching and task offloading, will grow exponentially, causing an increase in computational complexity. The hierarchical reinforcement learning (HRL) algorithm [24–27] is an ideal way to combat the curse of dimensionality, which can effectively reduce the dimensionality of each decision by partitioning the complex problem into several subproblems to solve separately, and then learn an optimal strategy to integrate them together. However, the majority of current typical research works on HRL primarily focuses on handling sparse and delayed feedback, as well as improving generalization. These algorithms commonly learn temporal abstractions over action space at first to achieve intrinsically constructed goals, and then generate a multi-layer hierarchical structure based on the theory of semi-Markov decision processes (SMDPs). Due to the complicated computational process, these HRL algorithms are unsuitable to be directly applied in real engineering scenarios. The main method to enable HRL in the MEC scenario is currently relaxing the coupling degree between the higher and lower layer in HRL and training them separately. Sometimes, the coupling of two layers is maintained, solely relying on a common metric such as the user number served by one ES. Such a compromise results in low performance. Obviously, there still exists a large gap between the typical design of HRL and the real complex engineering environment. The lack of a more effective joint policy to improve the performance while alleviating the curse of dimensionality poses a considerable challenge.

Second, with the development of artificial intelligence, trained deep neural network (DNN) models are gradually becoming the main component of service. Meanwhile, these models maintain an increasing trend in size with the goal to ensure the high accuracy of inference results. Aiming to meet the needs of users' requests without violating their deadline, ESs should proactively download services from the cloud center and load them in GPU memory [28]. Such an operation will incur extra energy consumption, particularly when services switch frequently. Therefore, it is another important challenge to carefully design an intelligent policy of deploying and running services to save energy.

Third, there may be some limitations in a practice scenario that prevent the deployment of services as freely as in previous research works. For instance, under the protection of intellectual property rights, commercial software venders usually apply a software license to regulate user rights and specify authorized usage and limitations. Edge computing enables new IoT capabilities while posing software license compliance risks. Thus, the service caching policy should take into account compliance within the range of software license agreement during the process of deploying services, while maintaining high availability and high performance. The issue has been studied in cloud computing [29,30]. Such constrains are rarely mentioned in the MEC field, and are challenging because services cannot be arbitrarily and widely placed close to users and the algorithm will become more complex.

We highlight our contributions as follows:

- We propose a novel HRL algorithm hierarchical service caching and task offloading (HSCTO) to solve cooperative service caching and task offloading problem in MEC. The lower layer is in charge of task offloading strategies, while the upper layer makes decisions on service caching. We design the algorithm based on analyzing the intrinsic structure and a priori knowledge of the MEC scenario, and mostly focus on specificity rather than generality. In contrast to current HRL algorithms, the action of the upper-layer agent in HSCTO not only represents the temporal and behavioral abstractions of the primitive actions of the lower-layer agent, but also corresponds to concrete meaning, i.e., service caching decisions which can be manually defined in advance. Within the hierarchical organization, the lower-layer module learns ordinary offloading policies by interacting with its environment, and the upper-layer module learns policies directly by exploiting the Q value functions of the lower-layer agents. The tightly coupled training method guarantees the algorithm performance.
- HSCTO arranges two layer agents to work at different time granularities. The lower-layer agent takes action upon the MEC environment in each time slot, and the upper-layer agent makes a decision over multiple time slots. In addition, we adopt a fixed multiple-time-step approach in the upper layer, so as to not rely on SMDP theory. We theoretically analyze that the upper layer is also an MDP and derive the Bellman optimal equations. As a result, current DRL algorithms can be employed in our hierarchical architecture directly and seamlessly. Furthermore, this two-time-scale framework ensures the stability of service deployment, while reducing the cost of frequently switching services in terms of energy consumption.
- In terms of extra limitation in a practical scenario, we consider separate protection and licensing methods in intellectual property, i.e., the number of MEC servers that can legally host the same type of service at the same time is limited. We integrate this constraint into our MEC model and address it.
- We conduct extensive numerical experiments to verify the performance of HSCTO, and the results show that our algorithm can effectively reduce computational delay while optimizing energy consumption, achieving significant performance improvements over multiple baseline approaches.

The rest of this paper is organized as follows: Section 2 introduces the mathematical model and problem formulation. The proposed HRL architecture is analyzed in Section 3 and the HSCTO algorithm is elaborated in Section 4. Section 5 shows evaluation experiments and results. Related works are discussed in Section 6. Finally, Section 7 concludes our work.

2. Mathematical Model and Problem Formulation

In this section, we discuss the system model and problem formulation. We discrete the infinite time horizon $\mathcal{T}$ into slots with equal size, and make an assumption that system states remain unchanged in one time slot.

2.1. MEC Scenario

Figure 1 demonstrates the scenario considered in this paper. Generally speaking, the three-tier architecture consists of one cloud computing center, several edge computing servers, and a group of mobile terminals. The cloud center has huge computational and storage resources and hosts services for all kinds of applications. Edge servers are connected to the cloud center through a wired backbone network, and hence, a variety of services may be downloaded from the cloud center and deployed on each edge server under the intelligent instruction of the system controller. By equipping a base station, an edge server can communicate with mobile terminals via wireless communication, and the edge network is divided into several disjoint regions centered around these base stations. At the beginning of each time slot, based on user operations, mobile terminals generate various computing tasks with different service requirements and computing resource demands. Because of their limited computing resources and real-time task requirements, mobile terminals need to make decisions to offload all or part of the tasks to edge servers with the goal of minimizing latency and energy consumption. If and only if the required services have already been placed on that edge server will the offloading process succeed. In the case that the edge server does not cache the corresponding service, the offloaded task will be relayed to the cloud center to complete the computation.

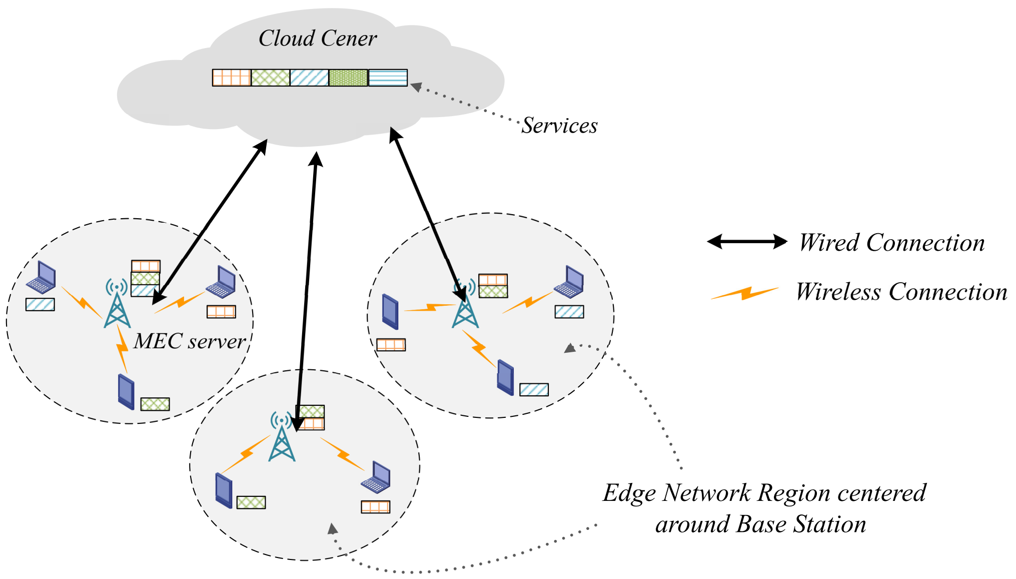

Figure 1. System model.

We use $\mathcal{E}$ and e to denote the set of ES and individual ES, use $\mathcal{D}$ and d denote the set of MT and individual MT, and similarly use $\mathcal{V}$ and v to denote the set of service and individual service. In addition, we use $\mathcal{D}_e$ to denote the set of MT which connects with server e directly.

2.2. Decision Model

We consider partial offloading in this paper, i.e., MT may offload all or part of its computing task to ESs in order to enhance the efficiency of the computation process. We use $\mu_{d,e,t} \in [0,1]$ to denote the proportion of MT d's task to ES e in the time slot t. Also, we use the vector $\boldsymbol{\mu}_{e,t} = \{\mu_{d,e,t} | \forall d \in \mathcal{D}_e\}$ to describe the whole offloading decision of ES e. In terms of service caching, due to the limited storage space, each ES cannot cache all services at the same time. Therefore, the caching policy has to judiciously decide which services to cache on each ES in the next time slot. Let binary variable $w_{v,e,t} \in \{0,1\}$ indicate whether the service v is deployed onto the ES e in the time slot t or not. If ES e has service v, $w_{v,e,t} = 1$, otherwise, $w_{v,e,t} = 0$. In addition, we use the vector $\boldsymbol{w}_{e,t} = \{w_{v,e,t} | \forall vs. \in \mathcal{V}\}$ to describe the service cache status on ES e. Let $\mathcal{C}_v$ be the storage requirement of service v, and then $\sum_{v \in \mathcal{V}} w_{v,e,t}\mathcal{C}_v$ can represent the storage space that ES e actually uses to cache services in the time slot t after the making service caching decision. Let $\mathcal{C}_e$ be the storage resources allocated for service caching on ES e. Therefore, the following inequation should be satisfied to guarantee that the storage needs of all cached services will not exceed the available storage resources on ES e.

$$\sum_{v \in \mathcal{V}} w_{v,e,t}\mathcal{C}_v \leq \mathcal{C}_e, \forall e \in \mathcal{E} \tag{1}$$

Let Lic_v denote the number of authorized uses described in the software license agreement of service v; obviously, it is an integer number. $\sum_{e \in \mathcal{E}} w_{v,e,t}$ presents the number of services v deployed in the entire MEC system in time slot t. To ensure proper use, we have

$$\sum_{e \in \mathcal{E}} w_{v,e,t} \leq Lic_v, \forall vs. \in \mathcal{V} \tag{2}$$

2.3. Computation Delay and Energy Consumption Model

2.3.1. Transmission Model

We consider orthogonal frequency-division multiple access (OFDMA) technology, in which subchannels are assigned to each pair of MT and ES exclusively, eliminating co-channel interference. Let $l_{d,e,t}$ and $b_{d,e,t}$ denote the distance and bandwidth between ES e and MT d, respectively, and the channel gain can be estimated by

$$h_{d,e,t} = \frac{\psi_{d,e,t}}{{l_{d,e,t}}^{\frac{\chi}{2}}} \tag{3}$$

where χ is the large scale path loss parameter, and $\psi_{d,e,t}$ is the small scale path loss parameter between ES e and MT d which obeys Rayleigh distribution. Suppose that the transmission power of MT d in the current time slot is denoted by $p_{d,t}$; then, the transmission rate between ES e and MT d can be given by

$$r_{d,e,t} = b_{d,e,t} log_2 (1 + \frac{p_{d,t}{h_{d,e,t}}^2}{\sigma}) \tag{4}$$

where σ is Gaussian noise.

2.3.2. Computation Model

We use the tuple $\zeta_{d,t} = \{z_{d,t}, f_{d,t}, v_{d,t}\}$ to denote the computing task generated by MT d at time slot t, where $z_{d,t}$ is the size of the chunk data of the application following an exponential distribution with mean Θ_d, $f_{d,t}$ is the computing resource demand of the task, and $v_{d,t}$ is the corresponding service. We assume that task arrival follows a Poisson process.

Moreover, $\mathbf{Y}_{e,t} = \{\zeta_{d,t}, \forall d \in \mathcal{D}_e\}$ stands for all tasks observed by the ES e. We decompose the offloading process into several subprocesses.

1. Local computing
   A computational task will be partitioned into two parts for local computing and offloading computing, respectively, based on offloading decisions. The local part is directly processed in MT, and thus, the computing delay can be calculated by

$$T_{d,t}^{local} = \frac{(1-\mu_{d,e,t})f_{d,t}}{F_d} \tag{5}$$

   where F_d represents the local computing resources of MT d in CPU frequency. The energy consumption equation for the local computing process can be written as

$$E_{d,t}^{local} = (1-\mu_{d,e,t})f_{d,t}\varphi_d \tag{6}$$

   where φ_d is the energy consumption coefficient per unit of computing resources in d.
2. Offloading to edge server
   Given the transmission rate $r_{d,e,t}$ by Equation (4), the time delay of offloading from MT d to ES e is shown in next equation.

$$T_{d,t}^{d2e} = \frac{\mu_{d,e,t}z_{d,t}}{r_{d,e,t}} \tag{7}$$

   Suppose the power of MT d remains unchanged in one time slot, the energy consumption of the transmitting task between MT d and ES e can be estimated as:

$$E_{d,t}^{d2e} = p_{d,t}T_{d,t}^{d2e} \tag{8}$$

   where $p_{d,t}$ is the transmission power of MT d in that time slot.
3. Computing in edge server
   Let F_e denote the computational resource allocated for MEC computing in ES e, and the time of computing the task from MT d is given by

$$T_{d,t}^{es} = \frac{\mu_{d,e,t}f_{d,t}}{\zeta_{d,e,t}F_e} \tag{9}$$

   where $\zeta_{d,e,t} \in (0,1]$ denotes the proportion of computing resources of ES e allocated for MT d. Moreover, the energy consumption of the MEC server is related to the required computing resources and can be calculated by

$$E_{d,t}^{es} = k_e\mu_{d,e,t}f_{d,t} \tag{10}$$

   where k_e is the energy consumption coefficient of ES e.
4. Offloading to the cloud center
   If there is no required service $v_{d,t}$, the ES e will transmit the task to the cloud computing center for computation. Since the wired backbone network is adopted between the edge servers and cloud center, the transmission rate is supposed to be a constant r_c. Therefore, in time slot t, the transmission delay can be estimated as follows:

$$T_{d,t}^{e2c} = \frac{\mu_{d,e,t}z_{d,t}}{r_c} \tag{11}$$

Furthermore, let $p_{e,t}$ denote the transmission power of ES e, and the energy consumption of the previous relay process can be written as

$$E_{d,t}^{e2c} = p_{e,t} T_{d,t}^{e2c} \tag{12}$$

5. Service caching
   To simplify, we use o_v and μ_v to denote the transmission delay and energy consumption of downloading and deploying service $v_{d,t}$, respectively. Thus, the delay and energy required to update the caching situation of $v_{d,t}$ will be

$$T_{v,e,t}^{sc} = o_v \times (w_{v,e,t-1} \oplus w_{v,e,t}) \tag{13}$$

$$E_{v,e,t}^{sc} = \mu_v \times (w_{v,e,t-1} \oplus w_{v,e,t}) \tag{14}$$

   where $\oplus$ is the XOR operator. Obviously, $w_{v,e,t-1} \oplus w_{v,e,t} = 1$ indicates that ES e will cache a new service $v_{d,t}$.
   Consequently, the time and energy consumption of the offloading process can be calculated by the next two equations, respectively.

$$T_{d,t}^{off} = T_{d,t}^{d2e} + (T_{d,t}^{es} + T_{v_{d,t},e,t}^{sc}) \times \mathbf{1}_{w_{v_{d,t},e,t}=1} + T_{d,t}^{e2c} \times \mathbf{1}_{w_{v_{d,t},e,t}=0} \tag{15}$$

$$E_{d,t}^{off} = E_{d,t}^{d2e} + (E_{d,t}^{es} + E_{v_{d,t},e,t}^{sc}) \times \mathbf{1}_{w_{v_{d,t},e,t}=1} + E_{d,t}^{e2c} \times \mathbf{1}_{w_{v_{d,t},e,t}=0} \tag{16}$$

   If the required service $v_{d,t}$ is cached on ES e in time slot t, i.e., $w_{v_{d,t},e,t} = 1$, the indicator function $\mathbf{1}_{w_{v_{d,t},e,t}=1} = 1$, the values of $T_{d,t}^{es} + T_{v_{d,t},e,t}^{sc}$ and $E_{d,t}^{es} + E_{v_{d,t},e,t}^{sc}$ will be added into the $T_{d,t}^{off}$ and $E_{d,t}^{off}$, respectively. Otherwise, the indicator function $\mathbf{1}_{w_{v_{d,t},e,t}=0} = 1$, $T_{d,t}^{e2c}$ and $E_{d,t}^{e2c}$ will be considered. In summary, taking into account both local and offloading computing, the total delay and energy consumption of the task generated by MT d can be estimated by

$$T_{d,t}^{tot} = max\{T_{d,t}^{local}, T_{d,t}^{off}\} \tag{17}$$

$$E_{d,t}^{tot} = E_{d,t}^{local} + E_{d,t}^{off} \tag{18}$$

   Furthermore, let $\mathcal{V}_{e,t}^{idle}$ denote the set of new caching services that no one uses in ES e, and the corresponding energy consumption can be expressed as

$$E_{e,t}^{idle} = \sum_{v \in \mathcal{V}_{e,t}^{idle}} \nu_v (w_{v,e,t-1} \oplus w_{v,e,t}) \tag{19}$$

2.4. Problem Formulation

In this subsection, we formulate the cooperative service caching and task offloading (CSCTO) problem with the objective of minimizing the utility function of weighted delay and energy consumption.

$$\min_{\mu_e, w_e, \zeta_e} \lim_{T \to \infty} \frac{1}{T} \sum_{t=0,}^{T} \sum_{e \in \mathcal{E}} \big(\alpha \sum_{d \in \mathcal{D}_e} T_{d,t}^{tot} + (1-\alpha)(\sum_{d \in \mathcal{D}_e} E_{d,t}^{tot} + E_{e,t}^{idle})\big) \tag{20}$$

$$\begin{aligned}
s.t. \quad & c1: \mu_{d,e,t} \in [0,1], && \forall d \in \mathcal{D}_e, \forall e \in E \\
& c2: w_{v,e,t} \in \{0,1\}, && \forall v \in \mathcal{V}, \forall e \in E \\
& c3: \sum_{v \in \mathcal{V}} w_{v,e,t}\mathcal{C}_v \leq \mathcal{C}_e, && \forall v \in \mathcal{V}, \forall e \in E \\
& c4: \sum_{e \in \mathcal{E}} w_{v,e,t} \leq Lic_v, && \forall v \in \mathcal{V}, \forall e \in E \\
& c5: \zeta_{d,e,t} \in (0,1], && \forall d \in \mathcal{D}_e, \forall e \in E \\
& c6: \sum_{d \in \mathcal{D}_e} \zeta_{d,e,t} \leq 1, && \forall d \in \mathcal{D}_e, \forall e \in \mathcal{E}
\end{aligned}$$

$c1$ indicates that each MT may offload part of its computing task to ES, $c2$ states that ES may or may not host each service; $c3$ ensures that the total amount of resources for caching service cannot exceed the maximum available storage capacity of ES, while $c4$ guarantees that the number of deployments of the same service is within the legal range specified in the software license. $c5$ and $c6$ demonstrate that the sum of computing resources allocated to each ES cannot exceed its total computing resource.

3. Hierarchical Reinforcement Learning Framework

There is a significant gap between the design goal of typical HRL algorithms and the characteristic of the MEC environment we discussed above. First, current typical HRL algorithms mainly focus on handling a short-term mission with sparse and delayed rewards, like video games, but the MEC scenario consists of a long-term and stochastic mission with instant rewards. Second, the higher-layer action of HRL usually refers to a temporal abstraction of a group of primitive actions, namely option, and thus the algorithm needs to learn how to first find the optimal option, and then exploit it. On the contrary, in the MEC scenario, by having prior knowledge, a hierarchical algorithm may define super action manually; moreover, these actions are concrete rather than abstraction. Although typical HRL algorithms are not directly suitable for MEC environment, their hierarchical idea inspires us to decompose complex problems into related subproblems to combat the curse of dimensionality. In this section, we propose a novel fixed multiple time step hierarchical reinforcement learning architecture (FSHRL). The upper layer makes decisions over a longer time scale and the upper and lower layers are closely coupled to achieve the overall optimal value. Figure 2 illustrates our HRL architecture.

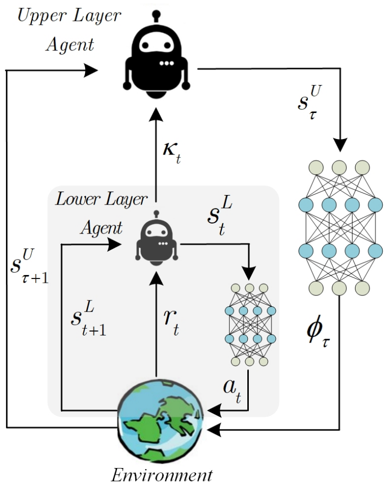

Figure 2. Architecture of hierarchical reinforcement learning.

3.1. MDP

3.1.1. The Lower Layer

Similar to traditional RL algorithms, the lower-layer agent performs actions upon the environment and directly obtains a reward. Based on the main assumption that a highly stochastic and uncertain MEC scenario evolves in an ergodic manner, the state system is supposed to have a Markov property. We use s_t to denote the system state in time slot t and $\mathcal{S}$ to be the state space. The lower layer can be modeled by an MDP, which is composed of tuple $< \mathcal{S}, \Phi, \mathcal{A}, \mathbb{P}, \gamma, r >$. Furthermore, in order to emphasize the impact of the higher-layer actions on the lower-layer states, we use $s_t^L = \{s_t, \phi_t\}$ to denote the augmented state, as well as $\mathcal{S}^L$ to be the augmented state space, where ϕ_t and Φ represent the super action of the upper layer at that time and the super action space, respectively. Moreover, $\mathcal{A}$ is an action space, $\mathbb{P}$ is the state transition function, γ is the discount factor, and r is the reward function.

3.1.2. The Upper Layer

The term super action is utilized to describe the action of the upper-layer agent. This agent makes a decision over a broader range of time scales, i.e., η slots, where η is a constant integer. We use τ to denote the sequence number of the upper time step, and then we have $\tau = t/\eta$, where the operator / means integer division. A tuple $< \mathcal{S}^U, \Phi, \boldsymbol{P}, \gamma, \kappa >$ is adopted to describe the decision process of the upper layer. $\mathcal{S}^U$ represents the state space; clearly, for each $s_\tau^U \in \mathcal{S}^U$, we have $s_\tau^U = s_{\tau\times\eta}$. Φ is a super action space and $\kappa : \mathcal{S}^U \times \Phi \to \mathbb{R}$ is the upper layer's reward function. In addition, we use $\Pi(\phi_\tau | s_\tau^U)$ to denote the policy function. The upper-layer agent does not receive any reward by directly interacting with the environment, but calculates the expectation of rewards of the lower-layer agents during the period of η time slots using Equation (21).

$$\kappa(s_\tau^U, \phi_\tau) = \sum_{i=0}^{\eta-1} \sum_a P(s_{t+i}^L | s_{t+i-1}^L, a_{t+i-1}) \pi(s_{t+i}^L, a_{t+i}) r(s_{t+i}^L, a_{t+i}) + \Psi(\phi_\tau) \tag{21}$$

where $\Psi : \Phi \to \mathbb{R}$ is the extra reward function and $t = \tau \times \eta$. It is worth noting that this reward function ensures that the two layers are tightly coupled. Additionally, the lower and upper layers are relatively independent, and each layer regards the other as a part of the environment; therefore, the curse of dimensionality will be effectively alleviated.

Due to the design of a fixed step size, the upper-layer decision process is also an MDP, rather than SMDP. The main reasons are three-fold. First, the state transition has a Markovian property, i.e.,

$$\begin{aligned} \boldsymbol{P}(s_{\tau+1}^U | s_\tau^U, s_{\tau-1}^U, s_{\tau-2}^U, \cdots) &\overset{(a)}{=} \boldsymbol{P}(s_{\tau\times\eta+\eta} | s_{\tau\times\eta}, s_{\tau\times\eta-\eta}, s_{\tau\times\eta-2\times\eta}, \cdots) \\ &\overset{(b)}{=} \boldsymbol{P}(s_{t+\eta} | s_t, s_{t-\eta}, s_{t-2\eta}, \cdots) \\ &\overset{(c)}{=} \boldsymbol{P}(s_{t+\eta} | s_t) \\ &\overset{(d)}{=} \boldsymbol{P}(s_{\tau+1}^U | s_\tau^U) \end{aligned}$$

According to the relationship of t and τ, i.e., $\tau - k = t - k \times \eta$, and the definition of s_τ^U mentioned above, we have $s_{\tau-k}^U = s_{t-k\times\eta}$, then $\overset{(a)}{=}$, $\overset{(b)}{=}$ and $\overset{(d)}{=}$ hold. Moreover, because of the fact that the k-step transition Markov chains still possess Markov properties, $\overset{(c)}{=}$ holds. Second, the state-value function also only depends on the current state s_τ^U.

$$V(s_\tau^U) = \mathbb{E}_t(G_\tau^U | s = s_\tau^U) = \mathbb{E}_t(\kappa_\tau + \gamma\kappa_{\tau+1} + \gamma^2\kappa_{\tau+2} + \cdots | s = s_\tau^U) \tag{22}$$

where G^U_τ is the decaying sum of all rewards in the decision process, sampled from the current state s^U_τ to the termination state. Third, for the policy function Π, we make Markov assumptions as usual. The probability of taking action ϕ_τ in state s^U_τ is only related to the current state s^U_τ and is independent of other factors.

Our framework is different from the option presented in [24]. Because there exist many options not being Markov, it requires that all the moments it has passed are recorded—not just the previous moment. As a result, the option of this technology is based on SMDP theory. On the contrary, the decision process in our framework is an MDP, and we can adopt various existing reinforcement learning algorithms in the upper layer.

3.2. Bellman Optimality Equation

In reinforcement learning, the Bellman optimality Equation [31] reveals the fact that the state value of the optimal policy must be equal to the expected return of the best action from the state, and is the fundamental of a value iterative operation. Let V^* and Q^* denote the optimal state value function and optimal action value function, respectively, and the traditional Bellman optimality equations are defined as

$$V^*(s_t) = \max_{\alpha} \left(r(s_t, a_t) + \gamma \sum_{s_{t+1}} \mathbb{P}(s_{t+1}|s_t, a_t) V^*(s_{t+1})\right) \tag{23}$$

$$Q^*(s_t, a_t) = r(s_t, a_t) + \gamma \sum_{s_{t+1}} \mathbb{P}(s_{t+1}|s_t, a_t) \max_{a_{t+1}} Q^*(s_{t+1}, a_{t+1}) \tag{24}$$

Next, we discuss the Bellman optimality equations of our hierarchical framework. We use V^{L*} and Q^{L*} to present an optimal state value function and optimal action value function of the lower layer, respectively, and the Bellman optimality equations can be expressed as:

$$V^{L*}(s_t) = \max_{\alpha \in \mathcal{A}} \left(r(s_t, \phi_{t/\eta}, a_t) + \gamma \sum_{s_{t+1} \in \mathcal{S}} \mathbb{P}(s_{t+1}|s_t, \phi_{t/\eta}, a_t) V^{L*}(s_{t+1})\right) \tag{25}$$

$$Q^{L*}(s_t, \phi_{t/\eta}, a_t) = r(s_t, \phi_{t/\eta}, a_t) + \gamma \sum_{s_{t+1} \in \mathcal{S}} \mathbb{P}(s_{t+1}|s_t, \phi_{t/\eta}, a_t) \max_{a_{t+1} \in \mathcal{A}} Q^{L*}(s_{t+1}, \phi_{(t+1)/\eta}, a_{t+1}) \tag{26}$$

It should be noted that, since we take the super action of the upper layer as part of the lower-layer state, we use $(s_t, \phi_{t/\eta})$ to represent the augmented state of the lower layer in the two above equations. Furthermore, because we use different time scales for the upper and lower layers, in order to obtain the time number τ in the super action ϕ_τ of the upper layer, we adopt operation $\tau = t/\eta$ to convert the sequence number from the lower-layer time space to the upper-layer time space in the calculation process.

Let V^{U*} be the optimal state value function, Q^{U*} be the optimal action value function, and the Bellman optimality equations of the upper layer are the same as the traditional form, and given by

$$V^{U*}(s^U_\tau) = \max_{\phi \in \Phi} \left(\kappa(s^U_\tau, \phi_\tau) + \gamma \sum_{s^U_{\tau+1} \in \mathcal{S}^U} P(s^U_{\tau+1}|s^U_\tau, \phi_\tau) V^{U*}(s^U_{\tau+1})\right) \tag{27}$$

$$Q^{U*}(s^U_\tau, \phi_\tau) = \kappa(s^U_\tau, \phi_\tau) + \gamma \sum_{s^U_{\tau+1} \in \mathcal{S}^U} P(s^U_{\tau+1}|s^U_\tau, \phi_\tau) \max_{\phi_{\tau+1} \in \Phi} Q^{U*}(s^U_{\tau+1}, \phi_{\tau+1}) \tag{28}$$

4. Algorithm

With the help of the FSHRL framework, we introduce an algorithm named HSCTO. Apparently, HSCTO will learn two policies, one of which is the upper policy Π_θ for handling service caching issues, the other is the lower policy π_θ to be responsible for making offloading decisions, where θ and ϑ are parameters. In theory, the goal pursued by the overall strategy is to maximize the expected return given the initial state s_0 and super action ϕ_0, i.e., the service caching decision ω_0.

$$\rho(\theta, \vartheta, s_0, \phi_0) = \mathbb{E}_{\theta,\vartheta}(\sum_{\tau=0}^{\infty} \gamma^{\tau} \kappa_{\tau} | s_{\tau}, \phi_{\tau}) \tag{29}$$

In the context of the MEC specific algorithm, we replace the lower and the upper layer with task offloading and service caching algorithm; moreover, we use service caching time slot length (SCTSL) to represent the time slot length of the service caching, i.e., η, in Section 3.

4.1. Task Offloading Algorithm

The task offloading algorithm is employed on each ES, consequently, multiple agents will exist that are working on the MEC scenario, running the algorithm simultaneously and independently.

- Observation
  Firstly, we discuss the observation of each agent in multi-agent environment. $\boldsymbol{s}_{e,t}^{L} = \{\boldsymbol{l}_{e,t}, \boldsymbol{h}_{e,t}, \boldsymbol{F}_d, F_{e,t}, \boldsymbol{b}_{e,t}, \mathbf{Y}_{e,t}, \boldsymbol{w}_{e,t}\}$ is the partial observation of ES e; moreover, the total state at each time step t is denoted by $\boldsymbol{s}_t^L = \{\boldsymbol{s}_{e,t}^L | \forall e \in \mathcal{E}\}$, and $\boldsymbol{l}_{e,t} = \{l_{d,e,t} | \forall d \in \mathcal{D}_e\}$, $\boldsymbol{h}_{e,t} = \{h_{d,e,t} | \forall d \in \mathcal{D}_e\}$ and $\boldsymbol{b}_{e,t} = \{b_{d,e,t} | \forall d \in \mathcal{D}_e\}$ denote the set of distance, channel gain and bandwidth between ES e and each MT d in $\mathcal{D}_e$, respectively. $\boldsymbol{F}_d = \{f_d | \forall d \in \mathcal{D}_e\}$ is the computing capacity of each MT in $\mathcal{D}_e$. As defined in the previous section, $\mathbf{Y}_{e,t}$ is the set of all tasks observed by ES e. In addition, $\boldsymbol{w}_{e,t} = \{w_{v,e,t} | \forall vs. \in \mathcal{V}\}$ is the current service set cached in ES e. Obviously, $\boldsymbol{w}_t = \{\boldsymbol{w}_{e,t} | \forall e \in \mathcal{E}\}$ is the current super action $\phi_{t/\eta}$ of the upper-layer algorithm.
- Action
  As mentioned before, we consider a partial offloading method and the corresponding decision for MT d is $\mu_{d,e,t} \in [0, 1]$, and thus, $\boldsymbol{\mu}_{e,t} = \{\mu_{d,e,t} | \forall d \in \mathcal{D}_e\}$ is the action of ES e. Moreover, we use $\boldsymbol{\mu}_t = \{\boldsymbol{\mu}_{e,t} | \forall e \in \mathbb{E}\}$ to be the action of the lower-layer algorithm.
- Reward
  Since the optimization objective of the task offloading decision is to minimize the average delay and energy consumption of all mobile terminal tasks, the reward function for each ES e is defined as the weighted value of the overall energy consumption for computation and the penalty for latency.

  $$r_{e,t} = -\frac{1}{|\mathcal{D}_e|} \sum_{d \in \mathcal{D}_e} (\alpha T_{d,t}^{tot} + (1-\alpha) E_{d,t}^{tot}) \tag{30}$$

  The smaller the weighted sum of the completion delay and energy consumption, the greater the reward of the task offloading decision.

Because the deterministic policy gradient algorithm (DDPG) [32] can effectively handle continuous actions, we design a task offloading algorithm based on DDPG. Our algorithm maintains two neural networks to learn policy and Q value separately, i.e., the actor network and critic network; furthermore, each has a target network to alleviate the overestimation by differentiating the calculation of target Q value and the selection of target Q value actions. Given the TD target $y_t = r_t + \gamma Q(s_{t+1}^L, \pi(s_{t+1}^L | \theta^{\pi'}) | \theta^{Q'})$, where $\theta^{\pi'}$ and $\theta^{Q'}$ are the parameters of the target actor network and target critic network, respectively; the loss function, which estimates how well the critic network matches the desired outcome, is defined as follows:

$$L(\theta^Q) = \mathbb{E}\big[(y_t - Q(s_t^L, a_t | \theta^Q))^2\big] \tag{31}$$

where θ^Q is the parameter of the critic network. In addition, the critic network updates its parameters based on gradient descent.

$$\theta^Q \leftarrow \theta^Q + \lambda_Q \nabla_{\theta^Q} L(\theta^Q) \tag{32}$$

where λ_Q is learning rate.

In order to improve the exploration efficiency of our algorithm and avoid getting stuck in local optimal solutions, it is necessary to add exploration noise to action a_t, i.e., $a_t = \pi_\theta(s_t^L) + \sigma_n$, σ_n is a random noise.

The gradient for the actor parameters can be approximated by

$$\nabla_{\theta^\pi} J(\theta^\pi) \approx \mathbb{E}\big[\nabla_{\theta^\pi} \pi(s^L|\theta^\pi)|_{s^L=s_t^L} \nabla_a Q(s^L, a|\theta^Q)|_{a=\pi(s_t^L|\theta^\pi)}\big] \tag{33}$$

Thus, the parameters of actor network updates can be updated by:

$$\theta^\pi \leftarrow \theta^\pi + \lambda_\pi \nabla_{\theta^\pi} J(\theta^\pi) \tag{34}$$

where λ_π is the learning rate.

To accelerate the gradient policy, target networks utilize the following equation to perform soft updates of their parameters, where $\lambda_t < 1$ is an adjustable hyperparameter.

$$\theta^{Q'} \leftarrow \lambda_t \theta^Q + (1 - \lambda_t)\theta^{Q'} \tag{35}$$

$$\theta^{\pi'} \leftarrow \lambda_t \theta^\pi + (1 - \lambda_t)\theta^{\pi'} \tag{36}$$

The pseudocode of the task offloading algorithm is illustrated in Algorithm 1.

Algorithm 1 Task offloading algorithm

```
1: Initialize network parameters $\theta^\pi, \theta^Q$ and target network parameters $\theta^{\pi'}, \theta^{Q'}$
2: Initialize experience replay buffer $\mathcal{D}_d, \lambda_t, \lambda_\pi$ and $\lambda_Q$
3: for $episode = 0 \to N$ do
4:     randomly initialize the MEC environment
5:     obtain state $s_0^L$ from the system
6:     for $t$=0 $\to T$ do
7:         select action to explore with the policy $a_t = \pi_\theta(s_t^L) + \sigma_n$
8:         execute action $a_t$ and obtain reward $r_t$ and new state $s_{t+1}^L$
9:         save transition $(s_t^L, a_t, r_t, s_{t+1}^L)$ in $\mathcal{D}_d$
10:        select a minibatch of $K$ transitions $\{(s_k^L, a_k, r_k, s_{k+1}^L)\}_{k=0,\dots,K-1}$ from $\mathcal{D}_d$
11:        for $k = 0 \to K-1$ do
12:            calculate TD target $y_k = r_k + \gamma Q(s_{k+1}^L, \pi(s_{k+1}^L|\theta^{\pi'})|\theta^{Q'})$
13:        end for
14:        calculate $L(\theta^Q) = \frac{1}{K}\sum_{k=0}^{K-1}[(y_t - Q(s_k, \pi(s_k|\theta^Q)))^2]$
15:        update critic network parameter $\theta^Q \leftarrow \theta^Q - \lambda_Q \nabla_{\theta^Q} L(\theta^Q)$
16:        calculate the gradient
               $\nabla_{\theta^\pi} J(\theta^\pi) = \frac{1}{K}\sum_{j=0}^{K-1}[\nabla_{\theta^\pi}\pi(s^L|\theta^\pi)|_{s^L=s_k^L}\nabla_a Q(s^L, a|\theta^Q)|_{a=\pi(s_k^L|\theta^\pi)}]$
17:        update actor network parameter $\theta^\pi \leftarrow \theta^\pi - \lambda_\pi \nabla_{\theta^\pi} J_i(\theta^\pi)$
18:        update target networks
                   $\theta^{Q'} \leftarrow \lambda_t \theta^Q + (1 - \lambda_t)\theta^{Q'}$
                   $\theta^{Q'} \leftarrow \lambda_t \theta^Q + (1 - \lambda_t)\theta^{Q'}$
19:        $s_t^L = s_{t+1}^L$
20:    end for
21: end for
```

4.2. Service Caching Algorithm

In contrast to the task offloading algorithm, the service caching algorithm is supposed to run on a central controller, which is capable of monitoring the global status of the entire MEC system. Considering that the service caching action is discrete, we implement the high-layer algorithm based on DDQN [33].

- State

  According to the design principle of FSHRL, the state observed by the service caching layer is $\boldsymbol{s}_{\tau}^{U} = \{\boldsymbol{s}_{e,\tau}^{U}, \forall e \in \mathcal{E}\}$, where $\tau = t \times \eta$ and

$$\boldsymbol{s}_{e,\tau}^{U} = \{\boldsymbol{l}_{e,\tau}, \boldsymbol{h}_{e,\tau}, \boldsymbol{F}_{d}, F_{e,\tau}, \boldsymbol{b}_{e,\tau}, \boldsymbol{\Upsilon}_{e,\tau}\}$$

- Super action

  Obviously, the super action is the service caching operation in next macro time step, i.e.,

$$\boldsymbol{\phi}_{\tau} = \{\boldsymbol{w}_{e,\tau} | \forall e \in \mathcal{E}\}$$

  where $\boldsymbol{w}_{e,\tau} = \{w_{v,e,\tau} | \forall v \in \mathcal{V}\}$ and $w_{v,e,\tau} \in \{0, 1\}$.

- Reward

  The reward function κ_{τ} is the sum of the lower-layer agents' rewards of each time step t within the current macro time step τ, and an extra cost of super action ϕ_{τ}, i.e., $\Psi_{e}(\boldsymbol{\phi}_{\tau})$.

$$\kappa_{\tau} = \sum_{e \in \mathcal{E}} \Big(\sum_{t=\tau \times \eta}^{(\tau+1) \times \eta - 1} r_{e,t} + \Psi_{e}(\boldsymbol{\phi}_{\tau}) \Big) \tag{37}$$

$$\Psi_{e}(\boldsymbol{\phi}_{\tau}) = \sum_{v \in \mathcal{V}} \mu_{v} \mathbf{1}_{w_{v,e,\tau} - w_{v,e,\tau-1} = 1} \tag{38}$$

  where μ_{v} is the energy cost when deploying the service v to the edge server.

  The service caching algorithm makes caching decisions for all MEC servers. The agent consists of the current Q network Q_{ϑ} and its corresponding target Q network $Q_{\vartheta'}$, where ϑ and ϑ' are parameters, respectively. The loss function of the current Q network is defined as

$$L(\vartheta) = \mathbb{E}\big[(Q(s_{\tau}^{U}, \phi_{\tau} | \vartheta) - \kappa_{\tau} - \gamma \max_{\phi_{\tau+1} \in \Phi} Q(s_{\tau+1}^{U}, \phi_{\tau+1} | \vartheta'))^{2}\big] \tag{39}$$

  The agent selects the service caching decision w_{τ}, i.e., super action ϕ_{τ}, by employing an ϵ-greedy strategy, and $\phi_{\tau+1}$ is the super action corresponding to the max value selected in the target Q network using the *argmax* function. The current Q network updates its parameters based on gradient descent.

$$\vartheta \leftarrow \vartheta + \lambda_{s} \nabla_{\vartheta} L(\vartheta) \tag{40}$$

where λ_{s} is the learning rate.

Moreover, the target Q network soft updates its parameter as follows

$$\vartheta' \leftarrow \lambda_{t} \vartheta + (1 - \lambda_{t}) \vartheta' \tag{41}$$

where $\lambda_{t} < 1$ is an adjustable hyperparameter. The pseudocode of the service caching algorithm is illustrated in Algorithm 2.

Algorithm 2 Service Caching Algorithm

```
1:  Initialize networks parameters $\vartheta, \vartheta'$
2:  Initialize experience replay buffer $\mathcal{D}_u, \lambda_t$ and $\lambda_s$
3:  for $episode = 0 \to N$ do
4:      generate authorized super action $\Phi$ set using $Lic = \{Lic_v | \forall v \in \mathcal{V}\}$
5:      randomly initialize the MEC environment
6:      get state $s_0^U$ from the system
7:      for $\tau = 0 \to T/\eta$ do
8:          select super action $\phi_\tau$ based on Q value and $\epsilon$-greedy strategy
9:          execute super action $\phi_\tau$
10:         collect each ES's rewards in next $\eta$ slots and then get the state $s_{\tau+1}^U$
11:         calculate $\Psi(\phi_\tau)$
12:         calculate $\kappa_\tau$ using Equation (37)
13:         save transition $(s_\tau^U, \phi_\tau, \kappa_\tau, s_{\tau+1}^U)$ in $\mathcal{D}_u$
14:         select a minibatch of $K$ transitions $\{(s_k^U, \phi_k, \kappa_k, s_{k+1}^U)\}_{k=0,\ldots,K-1}$ from $\mathcal{D}_u$
15:         for $k = 0 \to K-1$ do
16:             calculate $Y_k = \kappa_k + \gamma \max_{\phi_{k+1}} Q(s_{k+1}^U, \phi_{k+1} | \vartheta')$
17:         end for
18:         calculate $L(\vartheta) = \frac{1}{K} \sum_{k=0}^{K-1} [(Q(s_k^U, \phi_k | \vartheta) - Y_k)^2]$
19:         update Q network parameter $\vartheta \leftarrow \vartheta + \lambda_s \nabla_\vartheta L(\vartheta)$
20:         update actor network parameter $\theta^\pi \leftarrow \theta^\pi - \lambda_\pi \nabla_{\theta^\pi} J_i(\theta^\pi)$
21:         update target networks parameter $\vartheta' \leftarrow \lambda_t \vartheta + (1 - \lambda_t)\vartheta'$
22:         $s_\tau^U = s_{\tau+1}^U$
23:     end for
24: end for
```

5. Evaluation

5.1. Evaluation Configuration

In order to verify the performance of our hierarchical algorithm, we conducted extensive simulations. We developed a simulator utilizing PyTorch [34] , and adopted the scenario containing 3 edge servers, 27 mobile terminals, 5 types of service, and 1 cloud center. Moreover, the communication range of each MEC server is set to be 1000 m without intersections between the different servers. Mobile terminals are randomly distributed, and each user will move randomly. Other detailed parameters are shown in Tables 1 and 2.

To extensively compare the performance of our algorithm HSCTO with others, the baseline algorithms used in this experiment are shown below.

- DDPG. A typical single-agent reinforcement learning algorithm. In our experiment, DDPG makes action decisions for task offloading and service caching simultaneously.
- H-DQN [35]. A typical HRL algorithm. In order to cope with the continuous action of task offloading, we first discretize the action space, and then make an offloading decision.
- Random-DQN. An HRL algorithm, in which a random policy is adopted to handle service caching, while task offloading uses DQN for decision making.
- Random-DDPG. An HRL algorithm, similar to the previous one, but using DDPG algorithm for decision making.
- Random. A hierarchical algorithm, which randomly makes both service caching and task offloading decisions.

Table 1. MEC parameters.

Parameter	**Numerical Values**
Number of mobile terminals	27
Number of MEC servers	3
Number of service types	5
Transmission power of MT	[1,2] W
Transmission power of MEC server	10 W
Transmission rate between MEC server and cloud center	20 Mbps
Bandwidth of MEC server	100 MHz
Computational capacity of MT	2 GHz
Computational capacity of MEC server	[15,20] GHz
Number of service authorizations	3
Number of available resources in server	3
Energy consumption of downloading and deploying a service	5
Size of one task	[1,2] MB
CPU cycles for one bit task	[100,200] cycle/bit
Energy consumption coefficient of MT	6×10^{-10} J/Cycle
Energy consumption coefficient of MEC server	5×10^{-10} J/Cycle
Large-scale path loss parameter	3

Table 2. Algorithm parameters

Parameter	**Numerical Values**
Replay buffer size	20,000
Batch size	256
Episode	10,000
Dimension of hidden layer	512
Learning rate	0.0001
Discount factor	0.95
ϵ	0.9
Soft update rate	0.1
Explore noise	0.1

5.2. Performance Analysis

We first verified the convergence of our algorithm. Figure 3 shows the result of the service caching algorithm. The horizontal axis represents the number of training steps, and η is set to be 10, while the vertical axis represents the service cache reward. The dark line denotes the average rewards of the service cache agent in five random experiments, and the shaded area represents the interval between maximum and minimum experimental results. As we can see, the value fluctuates greatly in the initial time interval, then decreases with the increase in training iterations. This phenomenon indicates the good stability of our algorithm. After training about 900 steps, the service caching reward converged at around -29, demonstrating the perfect convergence and effectiveness of our algorithm.

Next, we evaluate the performance of our task offloading agents separately and the results are illustrated in Figure 4. Although we used different time granularity at two levels, in order to compare the results clearly, Figure 4 also used the same time granularity as Figure 3, and displayed the reward value as the sum of rewards within these 10 steps. As we can see, all three of the agent's rewards converge quickly after about 900 steps, this demonstrates that both the upper and lower layers converge synchronously due to the tightly coupled design. Also, the algorithm fluctuation range became smaller as the training times increases. These results indicate that both service caching and task offloading agents have good convergence; moreover, the two layers can collaborate well.

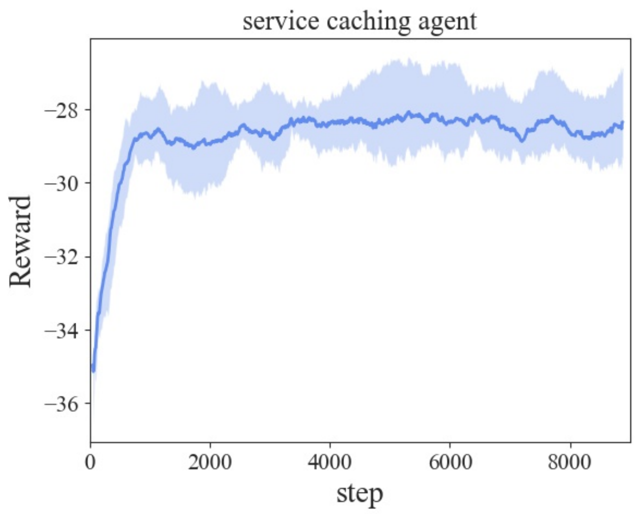

Figure 3. Reward of service caching agent during training process.

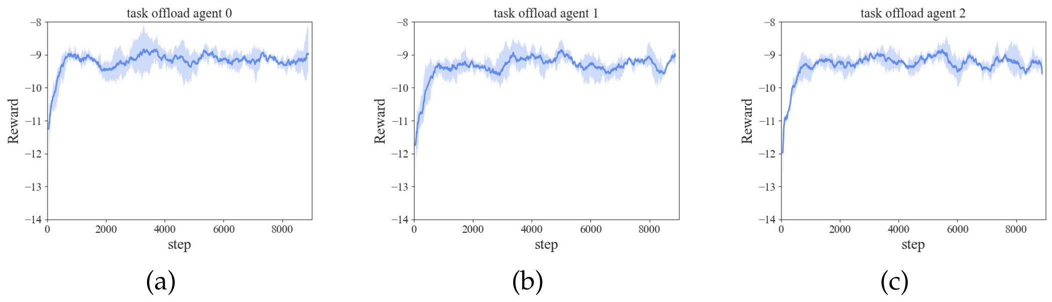

Figure 4. Rewards of task offloading agents during training process. (**a**–**c**) show the reward curves of the 3 agents respectively.

Furthermore, we analyze the impact of different upper time slots' length η on the algorithm. Figure 5 shows the performance of HSCTO with different η values. When η is set to be 10, the maximum reward value can reach around -27; on the contrary, it achieves a relatively poor performance and more severe oscillation if it is 1. When we set the η be 5 or 15, our algorithm can acquire medium results. Clearly, when η is small, the agent is unable to predict hotspot services based on limited information, resulting in the frequent switching of services and increasing total energy consumption. At the same time, this situation also leads to unstable environment observation for the lower-layer agents, causing task offloading decisions to fall into suboptimal results. Additionally, if η is set to be a large number, due to the impossibility of making timely decisions in terms of service caching, the lower-layer agent cannot capture the feature of time varying demand on service. The increase in mismatch will give rise to more computing tasks being offloaded to the distant cloud platforms, resulting in a decrease in algorithm performance.

We evaluate the performance of our algorithm with a different η value in terms of average utility, and Figure 6 demonstrates the result. The horizontal axis represents the values of different η, while the vertical axis corresponds to the average utility function, which consists of computational energy consumption, transmission energy consumption, task completion delay, and service switching energy consumption. The smaller the average utility, the better the task unloading and service caching strategy. It can be seen that the algorithm with a η of 10 has the best average utility, and two items, namely the D2E transmission energy consumption and MEC server computation energy consumption, are relatively higher than the others. The reason for this phenomenon is because the servers make efficient predictions for hotspot services; therefore, the mobile terminal can offload proper tasks to the servers in this situation. The minimum value of E2C also indicates that when $\eta = 10$, the number of tasks offloaded to the cloud platform is the least, revealing a higher hit rate in terms of requesting services. In terms of additional energy consumption, consistent with our theoretical analysis, the value of service caching energy consumption decreases as the value of the service caching time slot length increases. In summary, the effective joint decision of the task offloading and service caching results in a better average system utility.

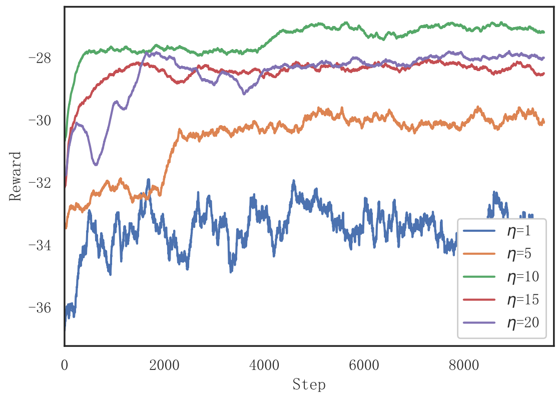

Figure 5. Comparison of rewards with different η.

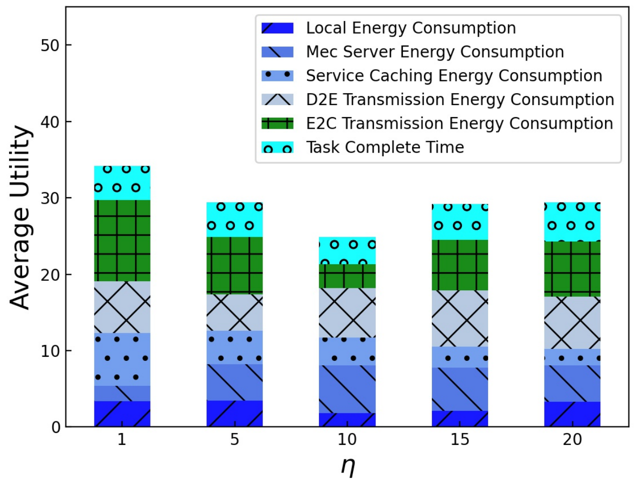

Figure 6. Comparison of average utility with a different η.

Figure 7 illustrates the rewards of HSCTO algorithms and 5 baseline algorithms during the training process. We carried out this experiment 100 times, and show an average result. η is set to be 10 in HSCTO. As we can see, HSCTO algorithm outperforms others with a better performance and faster convergence. Compared with H-DQN and DDPG, HSCTO improves the overall performance by around 20% and 27%, respectively. Moreover, the typical HRL-based algorithms, H-DQN is better than DDPG because it benefits from the hierarchical architecture, which can minimize the curse of dimensionality as much as possible. Both Random-DQN and Random-DDPG make the service caching decision in a random manner, and there is no tight coupling between the two layers, resulting in poor performance. The random algorithm performs the worst due to its completely random task offloading and service caching decisions.

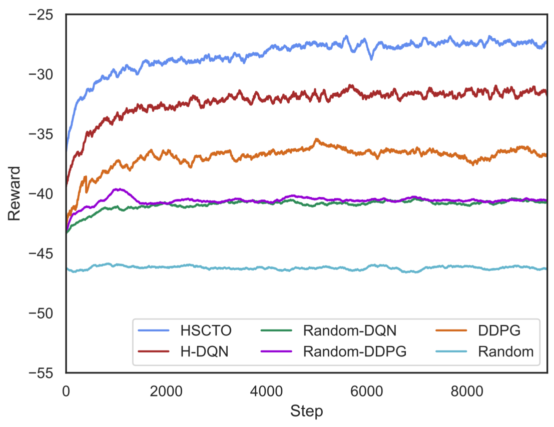

Figure 7. Comparison of the reward for different algorithms.

Figure 8 illustrates more details of the average utility values for different algorithms. Although both our algorithm and H-DQN are tightly coupled algorithms, the higher service caching energy consumption results in H-DQN being inferior to ours. The reason is that H-DQN focuses more on finding the optimal temporal abstraction of primitive actions, neglecting the specific meaning and cost of the super action, which is more suitable for turn-based scenario, rather than infinite horizon MEC scenario. Higher values of the task completion time, D2E communication energy consumption and MEC server energy consumption reveal that HSCTO can offload more tasks for server-side computation while achieving the goals of reducing computational latency and saving energy.

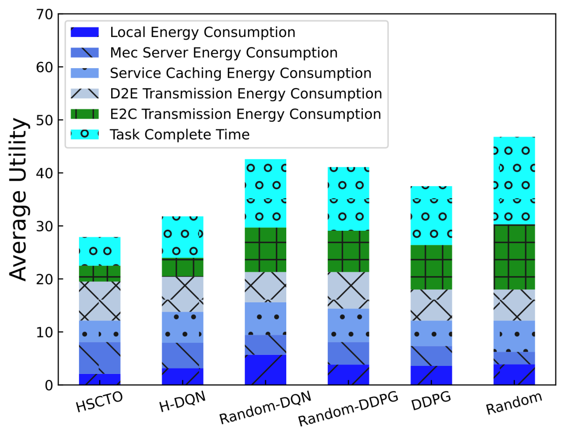

Figure 8. Comparison of average utility for different algorithms.

6. Related Works

In this section, we discuss the related works about the HRL scheme for service caching and task offloading in MEC from the perspective of two categories.

6.1. Service Caching and Task Offloading in MEC

Joint service caching and task offloading approaches are regarded as basic techniques in mobile edge computing and they arise great interest in both academia and industries.

Xu et al. [10] conducted an earlier research work to solve the service caching and task offloading problem in dense mobile edge network, aiming to minimize the computational delay of generating tasks for mobile terminals under long-term energy constraints. They adopted the Lyapunov optimization technique, whose core idea is to stabilize the data queue while maximizing the utility function as much as possible, and obtained a proven close-to-optimal solution. Zhou et al. [11] proposed a Lyapunov optimization-based algorithm to reduce the computational latency of user equipment in the UAV-assisted MEC system through jointly optimizing service caching and task offloading. The similarity with our work is that in order to reduce the cost of the service caching process, they also employed a different time scale principle, updating the service cache of every fixed T time slots while arranging task offloading at each time slot. Ko et al. [12] proposed a joint task offloading and service caching strategy to achieve optimal delay, considering each user's service preference. Premsankar et al. [13] treated deep neural network model as service, considering the fast development of AI applications. They formulated a multiperiod optimization problem which jointly places services and schedules requests to achieve minimized overall energy consumption and low latency. After that, a heuristic algorithm with a lower bound guarantee is proposed by solving a Lagrangian relaxation of the original problem. Yao et al. [16] proposed a deep reinforcement learning with graph attention mechanism for a digital twin-empowered MEC system to improve the performance of task offloading and service caching. In contrast to RL, due to the increasing computational burdens, the traditional optimization algorithm becomes less applicable in a highly dynamic environment with demands for real time. Furthermore, a traditional RL- or DNN-based algorithm will suffer from the curse of dimensionality in solving this complex joint optimization problem, especially in a large-scale environment.

Fang et al. [21] presented joint task offloading and content caching scheme for the NOMA edge network with the goal of minimizing network latency, in which the task offloading subproblem is solved utilizing DQN, and the allocations of network resources are optimized utilizing successive convex approximation (SCA), and the contents cached at the network nodes are updated by the LSTM-based approach. Lin et al. [22] also proposed a hybrid structure to solve the joint problem of service caching and task offloading in the UAV-assisted MEC environment, and a greedy dual-size frequency (GDSF)-based algorithm is utilized to deploy services into the MEC servers, while a single agent RL algorithm is adopted to make the task offloading decision. Xu et al. [23] separated the complex optimization problem of minimizing the delay and computational cost into two subproblems so as to reduce the complexity, and then proposed the hybrid approach in which the RL-based part makes a service caching decision and a convex optimization-based part decides the task offloading policy. Although these schemes divide the joint optimization problem into multiple subproblems to reduce the computational complexity, compared with our HRL-based algorithm, the hybrid scheme does not break away from the limitations of a traditional optimization algorithm.

6.2. Hierarchical Reinforcement Learning in MEC

Hierarchical reinforcement learning is receiving increasing attention from MEC. Existing works have applied different algorithms to tackle different optimization problems with different optimal goals in various MEC scenarios. Shi et al. [18] proposed a trajectory planning and resource allocation algorithm in multi-DC-assisted RANs with the objective of optimizing the accumulative network throughput for high-mobility users. They decomposed the original complex problem into two hierarchical subproblems, global trajectory planning, and local resource allocation. The higher-level algorithm is responsible for planning the trajectories of UAVs, aiming to maximize the number of users who can be served over a longer period of time. After the sequence of served areas is determined, the lower level algorithm addresses the subproblem to allocate the resources for each UVA. Ren et al. [19] proposed a two-layer HRL algorithm to reduce the complexity of a large-scale UAV-assisted MEC network in dynamic environment by partitioning the sophisticated original scheduling problem into several subproblems based on "divide-and-conquer" idea.

Zhang et al. [17] introduced a hierarchical method in a backscatter data collection scenario based on an option idea to first generate multiple clusters of the backscatter sensor node and then schedule multiple UAVs to process the data collection, with the goal of maximize the total flight time under the energy constraints of the UAV. Considering a hybrid wireless network where RF communication and backscatter technology coexist, Zhou et al. [15] proposed an HRL scheme, in which the high-level agent learns to optimize the beamforming of an access point while the lower-level agents solve the problem of efficiently offloading individuals workloads to the edge servers. Geng et al. [14] investigated the cooperative communication with the goal of reducing the outage probability, in order to extend the scale of cooperative network, they divide the problem into two subproblems, i.e., a relay selection and power allocation; after that, they proposed an HRL approach to optimize the subproblems separately. As mentioned above, existing HRL-based algorithms in MEC always relaxed the coupling degree between the higher- and lower-layer, and trained them individually. Such a trade-off will lead to poor performance.

7. Conclusions

In this article, we are devoted to applying the HRL algorithm to the joint optimization problem of service caching and task offloading in practical MEC environments, in order to alleviate the curse of dimensionality while improving the system performance. In order to bridge the gap between the typical HRL algorithm and real-world MEC scenario, we firstly introduce a novel HRL architecture, in which two different layers leverage different time granularities, and differing from other HRL algorithms, the higher layer employs fixed multiple time steps. This two-time-scale design expands the range of algorithms that can be adopted, simplifies algorithm design, and enhances the stability of the upper layer. Then, we propose an algorithm HSCTO based on the above architecture. The upper layer of HSCTO makes service caching decisions by utilizing the rewards of the task offloading agent in the lower layer. The algorithm performance can be guaranteed by this tightly coupled method. Numerical experiments are carried out and the evaluation results show that HSCTO outperforms other baseline algorithms.

Author Contributions: Conceptualization, T.C.; methodology, T.C.; software, J.A.; validation, X.X.; formal analysis, G.H.; investigation, T.C. resources, X.X.; data curation, J.A.; writing—original draft preparation, T.C. writing—review and editing, T.C. visualization, J.A.; supervision, G.H.; project administration, T.C. funding acquisition, G.H. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by The Key Project of the Shenzhen Municipality (Granted No. JSGG20211029095545002), Education and Teaching Reform Research Project of Guangdong Higher Vocational Electronic Information and Communication Teaching Committee, Guangdong Provincial Demonstration Industry College Project (SZIIT-SangFor Cyber-Security Service Project), 2023 Education and Teaching Reform Research and Practice Project of Guangdong Higher Vocational Electronic Information and Communication Education Advisory Committee (Guangdong Higher Vocational Electronic Information and Communication Education Advisory Committee (2023) No. 9), College-level Teacher Teaching Innovation Team Project (Cyber-Security Team), School-level Key Teaching Reform Project of SZIIT (Information Security Vocational Education Undergraduate Talent Training Project), and 2024 College-level Demonstration Virtual Simulation Training Base of SZIIT(Cyber-Range Project).

Data Availability Statement: The original contributions presented in the study are included in the article, whilst further inquiries can be directed to the corresponding author.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Qiu, T.; Chi, J.; Zhou, X.; Ning, Z.; Atiquzzaman, M.; Wu, D.O. Edge Computing in Industrial Internet of Things:Architecture, Advances and Challenges. *IEEE Commun. Surv. Tutor.* **2020**, *22*, 2462–2488. [CrossRef]
2. Lin, H.; Zeadally, S.; Chen, Z.; Labiod, H.; Wang, L. A Survey on Computation Offloading Modeling for Edge Computing. *J. Netw. Comput. Appl.* **2020**, *169*, 102781. [CrossRef]
3. Guo, F.; Zhang, H.; Ji, H.; Li, X.; Leung, V.C.M. An Efficient Computation Offloading Management Scheme in the Densely Deployed Small Cell Networks With Mobile Edge Computing. *IEEE/ACM Trans. Netw.* **2018**, *26*, 2651–2664. [CrossRef]
4. Hortelano, D.; De Miguel, I.; Barroso, R.J.D.; Aguado, J.C.; Merayo, N.; Ruiz, L.; Asensio, A.; Masip-Bruin, X.; Fernández, P.; Lorenzo, R.M.; et al. A Comprehensive Survey on Reinforcement-Learning-Based Computation Offloading Techniques in Edge Computing Systems. *J. Netw. Comput. Appl.* **2023**, *216*, 103669. [CrossRef]
5. Xu, X.; Liu, K.; Dai, P.; Jin, F.; Ren, H.; Zhan, C.; Guo, S. Joint Task Offloading and Resource Optimization in NOMA-based Vehicular Edge Computing: A Game-Theoretic DRL Approach. *J. Syst. Archit.* **2023**, *134*, 102780. [CrossRef]
6. Wang, L.; Jiao, L.; He, T.; Li, J.; Mühlhäuser, M. Service Entity Placement for Social Virtual Reality Applications in Edge Computing. In Proceedings of the IEEE INFOCOM 2018-IEEE Conference on Computer Communications, Honolulu, HI, USA, 16–19 April 2018; pp. 468–476.
7. Pasteris, S.; Wang, S.; Herbster, M.; He, T. Service Placement with Provable Guarantees in Heterogeneous Edge Computing Systems. In Proceedings of the IEEE INFOCOM 2019-IEEE Conference on Computer Communications, Paris, France, 29 April–2 May 2019; pp. 514–522.
8. Liu, S.; Yu, Y.; Lian, X.; Feng, Y.; She, C.; Yeoh, P.L.; Guo, L.; Vucetic, B.; Li, Y. Dependent Task Scheduling and Offloading for Minimizing Deadline Violation Ratio in Mobile Edge Computing Networks. *IEEE J. Sel. Areas Commun.* **2023**, *41*, 538–554. [CrossRef]
9. Plageras, A.P.; Psannis, K.E. IoT-based Health and Emotion Care System. *ICT Express* **2023**, *9*, 112–115. [CrossRef]
10. Xu, J.; Chen, L.; Zhou, P. Joint Service Caching and Task Offloading for Mobile Edge Computing in Dense Networks. In Proceedings of the IEEE INFOCOM 2018-IEEE Conference on Computer Communications, Honolulu, HI, USA, 16–19 April 2018; pp. 207–215.
11. Zhou, R.; Wu, X.; Tan, H.; Zhang, R. Two Time-Scale Joint Service Caching and Task Offloading for UAV-assisted Mobile Edge Computing. In Proceedings of the IEEE INFOCOM 2022-IEEE Conference on Computer Communications, London, UK, 2–5 May 2022; pp. 1189–1198.
12. Ko, S.W.; Kim, S.J.; Jung, H.; Choi, S.W. Computation Offloading and Service Caching for Mobile Edge Computing Under Personalized Service Preference. *IEEE Trans. Wirel. Commun.* **2022**, *21*, 6568–6583. [CrossRef]
13. Premsankar, G.; Ghaddar, B. Energy-Efficient Service Placement for Latency-Sensitive Applications in Edge Computing. *IEEE Internet Things J.* **2022**, *9*, 17926–17937. [CrossRef]
14. Geng, Y.; Liu, E.; Wang, R.; Liu, Y. Hierarchical Reinforcement Learning for Relay Selection and Power Optimization in Two-Hop Cooperative Relay Network. *IEEE Trans. Commun.* **2022**, *70*, 171–184. [CrossRef]
15. Zhou, H.; Long, Y.; Zhang, W.; Xu, J.; Gong, S. Hierarchical Multi-Agent Deep Reinforcement Learning for Backscatter-aided Data Offloading. In Proceedings of the 2022 IEEE Wireless Communications and Networking Conference (WCNC), Austin, TX, USA, 10–13 April 2022; pp. 542–547.
16. Yao, Z.; Xia, S.; Li, Y.; Wu, G. Cooperative Task Offloading and Service Caching for Digital Twin Edge Networks: A Graph Attention Multi-Agent Reinforcement Learning Approach. *IEEE J. Sel. Areas Commun.* **2023**, *41*, 3401–3413. [CrossRef]
17. Zhang, Y.; Mou, Z.; Gao, F.; Xing, L.; Jiang, J.; Han, Z. Hierarchical Deep Reinforcement Learning for Backscattering Data Collection With Multiple UAVs. *IEEE Internet Things J.* **2021**, *8*, 3786–3800. [CrossRef]
18. Shi, W.; Li, J.; Wu, H.; Zhou, C.; Cheng, N.; Shen, X. Drone-Cell Trajectory Planning and Resource Allocation for Highly Mobile Networks: A Hierarchical DRL Approach. *IEEE Internet Things J.* **2021**, *8*, 9800–9813. [CrossRef]
19. Ren, T.; Niu, J.; Dai, B.; Liu, X.; Hu, Z.; Xu, M.; Guizani, M. Enabling Efficient Scheduling in Large-Scale UAV-Assisted Mobile-Edge Computing via Hierarchical Reinforcement Learning. *IEEE Internet Things J.* **2022**, *9*, 7095–7109. [CrossRef]
20. Birman, Y.; Ido, Z.; Katz, G.; Shabtai, A. Hierarchical Deep Reinforcement Learning Approach for Multi-Objective Scheduling With Varying Queue Sizes. In Proceedings of the 2021 International Joint Conference on Neural Networks (IJCNN), Shenzhen, China, 18–22 July 2021; pp. 1–10.
21. Fang, C.; Xu, H.; Zhang, T.; Li, Y.; Ni, W.; Han, Z.; Guo, S. Joint Task Offloading and Content Caching for NOMA-Aided Cloud-Edge-Terminal Cooperation Networks. *IEEE Trans. Wirel. Commun.* **2024**, *23*, 15586–15600. [CrossRef]
22. Lin, N.; Han, X.; Hawbani, A.; Sun, Y.; Guan, Y.; Zhao, L. Deep Reinforcement Learning Based Dual-Timescale Service Caching and Computation Offloading for Multi-UAV Assisted MEC Systems. *IEEE Trans. Netw. Serv. Manag.* **2024**. [CrossRef]
23. Xu, Y.; Peng, Z.; Song, N.; Qiu, Y.; Zhang, C.; Zhang, Y. Joint Optimization of Service Caching and Task Offloading for Customer Application in MEC: A Hybrid SAC Scheme. *IEEE Trans. Consum. Electron.* **2024**. [CrossRef]

24. Sutton, R.S.; Precup, D.; Singh, S. Between MDPs and Semi-MDPs: A Framework for Temporal Abstraction in Reinforcement Learning. *Artif. Intell.* **1999**, *112*, 181–211. [CrossRef]
25. Parr, R.; Russell, S. Reinforcement Learning with Hierarchies of Machines. *Adv. Neural Inf. Process. Syst.* **1997**, *10*, 1043–1049.
26. Dietterich, T.G. Hierarchical Reinforcement Learning with the MAXQ Value Function Decomposition. *J. Artif. Intell. Res.* **2000**, *13*, 227–303. [CrossRef]
27. Al-Eryani, Y.; Hossain, E. Self-Organizing mmWave MIMO Cell-Free Networks With Hybrid Beamforming: A Hierarchical DRL-Based Design. *IEEE Trans. Commun.* **2022**, *70*, 3169–3185. [CrossRef]
28. Jahanshahi, A.; Sabzi, H.Z.; Lau, C.; Wong, D. Gpu-Nest: Characterizing Energy Efficiency of Multi-Gpu Inference Servers. *IEEE Comput. Archit. Lett.* **2020**, *19*, 139–142. [CrossRef]
29. Caron, E.; Chevalier, A.; Baillon-Bachoc, N.; Vion, A.L. Heuristic for License-Aware, Performant and Energy Efficient Deployment of Multiple Software in Cloud Architecture. In Proceedings of the 2021 12th International Conference on Information and Communication Systems (ICICS), Valencia, Spain, 24–26 May 2021; pp. 297–304.
30. Chevalier, A.; Caron, E.; Baillon-Bachoc, N.; Vion, A.L. Towards Economic and Compliant Deployment of Licenses in a Cloud Architecture. In Proceedings of the 2018 IEEE 11th International Conference on Cloud Computing (CLOUD), San Francisco, CA, USA, 2–7 July 2018; pp. 718–724.
31. Sutton, R.S.; Barto, A.G. *Reinforcement Learning, Second Edition: An Introduction*; MIT Press: Cambridge, MA, USA, 2018.
32. Lillicrap, T.P.; Hunt, J.J.; Pritzel, A.; Heess, N.M.O.; Erez, T.; Tassa, Y.; Silver, D.; Wierstra, D. Continuous control with deep reinforcement learning. *arXiv* **2015**, arXiv:1509.02971.
33. Hasselt, H.v.; Guez, A.; Silver, D. Deep reinforcement learning with double Q-Learning. In Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, Phoenix, AZ, USA, 12–17 February 2016; AAAI Press: Cambridge, MA, USA, 2016; pp. 2094–2100.
34. Paszke, A.; Gross, S.; Massa, F.; Lerer, A.; Bradbury, J.; Chanan, G.; Killeen, T.; Lin, Z.; Gimelshein, N.; Antiga, L.; et al. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In Proceedings of the 33rd International Conference on Neural Information Processing Systems, Vancouver BC, Canada, 8–14 December 2019; pp. 8026–8037.
35. Kulkarni, T.D.; Narasimhan, K.R.; Saeedi, A.; Tenenbaum, J.B. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. In Proceedings of the 30th International Conference on Neural Information Processing Systems, Barcelona, Spain, 5–10 December 2016; pp. 3682–3690.