future internet

MDPI

Article

A Learning Game-Based Approach to Task-Dependent Edge Resource Allocation

Zuopeng Li [1,2,*], Hengshuai Ju [1] and Zepeng Ren [1]

[1] School of Information and Electrical Engineering, Hebei University of Engineering, Handan 056038, China; ju1540717348@outlook.com (H.J.); zren6065@gmail.com (Z.R.)
[2] School of Information Engineering, Handan University, Handan 056038, China
* Correspondence: lizuopeng@hebeu.edu.cn

Abstract: The existing research on dependent task offloading and resource allocation assumes that edge servers can provide computational and communication resources free of charge. This paper proposes a two-stage resource allocation method to address this issue. In the first stage, users incentivize edge servers to provide resources. We formulate the incentive problem in this stage as a multivariate Stackelberg game, which takes into account both computational and communication resources. In addition, we also analyze the uniqueness of the Stackelberg equilibrium under information sharing conditions. Considering the privacy issues of the participants, the research is extended to scenarios without information sharing, where the multivariable game problem is modeled as a partially observable Markov decision process (POMDP). In order to obtain the optimal incentive decision in this scenario, a reinforcement learning algorithm based on the learning game is designed. In the second stage, we propose a greedy-based deep reinforcement learning algorithm that is aimed at minimizing task execution time by optimizing resource and task allocation strategies. Finally, the simulation results demonstrate that the algorithm designed for non-information sharing scenarios can effectively approximate the theoretical Stackelberg equilibrium, and its performance is found to be better than that of the other three benchmark methods. After the allocation of resources and sub-tasks by the greedy-based deep reinforcement learning algorithm, the execution delay of the dependent task is significantly lower than that in local processing.

Keywords: edge computing; Stackelberg game; deep reinforcement learning; dependent task; resource allocation

Citation: Li, Z.; Ju, H.; Ren, Z. A Learning Game-Based Approach to Task-Dependent Edge Resource Allocation. *Future Internet* **2023**, *15*, 395. https://doi.org/10.3390/fi15120395

Academic Editor: Antonio Esposito

Received: 8 November 2023
Revised: 1 December 2023
Accepted: 4 December 2023
Published: 7 December 2023
Corrected: 22 April 2024

1. Introduction

With the rapid advancement of hardware and software technologies, there is an increasing demand for computationally intensive real-time applications, such as facial recognition, virtual reality, and augmented reality [1–3]. These applications often require lower response times to meet user experience expectations. However, due to the limitations of device hardware and available resources, the devices cannot complete task computations in the required time [4–6]. Cloud computing is one of the candidate technologies to solve this problem, but the distance between the cloud and the devices is considerable. Transferring a large number of tasks to the cloud imposes a significant communication burden on the entire network. Edge computing (EC) deploys resources closer to devices, thereby enabling devices to offload tasks to a nearby edge server (ES) with less overhead, thus further reducing the time used for computational tasks [7].

Although edge servers (ESs) can provide the necessary resources for devices, the behavior of ESs and the user in practice is usually driven by interests [8,9]. In order to offset operational costs, ESs charge a certain fee for processing tasks, while the user needs to pay for the costs incurred from using these resources, thus ensuring a satisfactory quality of experience (QoE). On the other hand, dependent tasks differ from tasks that can be arbitrarily divided (such as atomic tasks). The order of scheduling of its sub-tasks and

the amount of resources allocated to each sub-task are important factors that affect the execution time of the dependent task [10,11]. Therefore, motivating ESs to engage in task computation and the effective allocation of resources and tasks are key issues in edge computing (EC).

Game theory is a mathematical apparatus that studies the decision-making process of players in a competitive environment. It involves exploration of the interactions between participants, as well as their selection of strategies and expected outcomes [12–14]. Using the strengths of game theory, numerous studies in recent years have applied it to the research of edge resource allocation strategies. For example, Roostaei et al. [15] employed a Stackelberg game for the dynamic allocation and pricing of communication and computational resources in networks, in which they proposed a joint optimal pricing and resource allocation algorithm based on game theory. Chen et al. [16] investigated multiple resource allocation and pricing issues by modeling the problem as a Stackelberg game. Kumar et al. [17] introduced a game model for edge resource allocation with the aim of maximizing the utilization of computational resources. However, these studies all rest on a strong assumption that the game participants should share their information, such as the user needing to disclose resource preference parameters and the edge servers (ESs) required to reveal their cost parameters. Yet, game participants are rational, and they are typically reluctant to disclose such private information.

This paper presents a two-stage resource allocation method designed for task-dependent scenarios. In the first stage, an incentive mechanism is proposed and modeled as a multivariate Stackelberg game. We then analyze the uniqueness of the Stackelberg equilibrium (SE) in an information sharing scenario, as well as design an iterative optimization algorithm that can approximate the SE solution. The research is then extended to non-information-sharing scenarios, where the game problem is formulated as a POMDP. A reinforcement learning algorithm based on a learning game is then proposed, which can learn from the participants' historical decisions while ensuring the privacy of game participants. The second stage aims to allocate the resources purchased by users under the incentive mechanism in a rational manner. A greedy-based deep reinforcement learning algorithm is designed to minimize task execution time. Specifically, edge server resources are allocated using a greedy method, and a sequence-to-sequence neural network-based reinforcement learning algorithm is employed to obtain optimal task allocation decisions. The S2S neural network is a deep learning model that is implemented using multiple layers of a recurrent neural network. It is capable of transforming an input sequence into a corresponding output sequence. Furthermore, these two approaches are then combined to minimize task execution time. The main contributions of this paper are as follows.

- We propose a two-stage resource allocation method in the context of dependent tasks.
- In the first stage, we model the problem of incentivizing users to request resources from edge servers as a multivariate Stackelberg game. We analyze the uniqueness of SE under the scenario of information sharing. Furthermore, we investigate the incentive problem in the absence of information sharing, and we transform it into a partially observable Markov decision process for multiple agents. To solve the SE in this situation, we design a learning-based game-theoretic reinforcement learning algorithm.
- In the second stage, to allocate resources effectively, we design a greedy-based deep reinforcement learning algorithm to minimize the task execution time.
- Through experimental simulation, it is demonstrated that the reinforcement learning algorithm proposed in this paper, which is based on learning games, can achieve SE in scenarios without information disclosure, and that it outperforms the conventional A2C algorithm. The reinforcement learning algorithm, grounded in the principle of greediness, can significantly reduce the execution time of tasks.

The rest of the paper is structured as follows. Section 2 describes related works. Section 3 presents our system model. Section 4 discusses the design of incentives in information sharing scenarios. Section 5 presents our game-theoretic learning algorithm in the context of non-information sharing. Section 6 details how greedy methods are used in

the design of reinforcement learning algorithms. Section 7 evaluates the performance of the proposed algorithm through simulations. Finally, in Section 8, we provide a comprehensive summary of the article.

2. Related Work and Preliminary Technology

2.1. Related Work

Existing research on the problem of task offloading and resource allocation falls into two main categories. The first assumes that ESs can provide resources at no cost [18–23]. The other assumes that ESs provide resources under some form of incentive mechanism [24–33]. Zhang et al. [18] performed a joint optimization of channel allocation and dependent task offloading, as well as designed an algorithm based on genetic algorithms and deep deterministic policy gradients. Liang et al. [19] studied the problem of optimal offloading and optimal allocation of computational resources under dependent task constraints and proposed a heuristic algorithm to solve the problem. Xiao et al. [20] optimized a task offload strategy, communication resources, and computing resources under the constraints of task processing delay and device energy consumption, and they went on to analyze the optimal solution from the perspectives of slow fading channels and fast fading channels. Jiang et al. [21] proposed an online framework for task offloading and resource allocation issues in edge computing environments. Chen et al. [22] transformed the optimal resource allocation problem into an integer linear programming problem, and they proposed a distributed algorithm based on Markov approximation to achieve an approximately optimal solution within polynomial-time complexity. Chen et al. [23] described the auxiliary caching-assisted computation offloading process, which is characterized as a problem of maximizing utility while considering the quality of experience (QoE). In addition, the problem was decomposed into two sub-problems for separate solutions.

The aforementioned literature implicitly assumes that ESs can provide resources freely. However, in reality, ESs are driven by profit. Therefore, it is necessary to consider how to incentivize ESs. Based on whether the edge servers have sufficient resources, the study of incentive mechanisms can be divided into two aspects: First, when ESs lack adequate resources, the energy consumption of ESs acts as a counter incentive for users [24,25]. Second, when the ESs have sufficient resources, the energy consumption of ESs becomes an incentive for users [26–29]. Avgeris et al. [24] studied the offloading problem involving multiple users and multiple edge nodes, where the energy consumption of the ESs was considered a constraining factor in finding the optimal solution. Chen et al. [25]. investigated the task offloading and collaborative computing problem in mobile edge computing (MEC) networks, and they proposed a two-level incentive mechanism based on bargaining games. Additionally, to address the issue of edge node overload, the energy consumption of edge nodes was treated as a disincentive in the second stage of problem analysis. The aforementioned paper considered the possibility of ES overload, but its research problem and scenario differ from the context of this paper.

The energy consumption of ESs serves as a positive incentive. Liu et al. [26] studied resource allocation and pricing problems in an EC system, then formulated them as mixed-integer linear programming and proved the problem to be NP-hard. To solve this problem, auction-based mechanisms and linear programming-based approximation mechanisms were proposed and developed. Tao et al. [27] studied the server resource pricing and task offloading problem by establishing a Stackelberg game model, as well as used a differential algorithm to solve for the optimal solution. Seo et al. [28] aimed to increase the utilization rate of ESs' computing resources, and they formulated the optimization problem as a Stackelberg game. Supervised learning was designed to obtain equilibrium strategies. However, the Stackelberg games used in [27,28] were based on a single variable and did not consider the dependencies between tasks. Kang et al. [29] introduced an auction mechanism to encourage ESs to offer services for tasks with dependencies, as well as designed an algorithm based on multiple rounds of truthful combinatorial reverse

auctions to solve the problem of maximizing social welfare. However, the multi-round auction process in this algorithm may lead to high waiting latency.

The algorithmic designs in the aforementioned literature are all centralized. However, in reality, much of the information from users and ESs is difficult to control globally. Over the recent years, certain literature has also explored the design of incentive mechanisms under distributed systems. Bahreini et al. [30] proposed a learning-based distributed resource coordination framework which transforms the computation offloading and resource allocation problems into dual timescale problems, and it then solves them using game theory and distributed reinforcement learning algorithms. Liu et al. [31] described the computational offloading mechanism with resource allocation in EC networks as a stochastic game, and they designed a Q-learning algorithm to achieve NE. Li et al. [32] jointly optimized offloading decisions and resource pricing, as well as designed a learning game method to obtain optimal decisions. Despite these methods being based on a distributed approach, the aforementioned studies did not consider the dependencies of tasks. Song et al. [33] proposed a multi-objective offloading optimization algorithm based on reinforcement learning, which was aimed at minimizing execution time, energy consumption, and the cost of dependent task offloading. However, that work did not dive into the cost aspect. This paper investigates the problem of incentive mechanisms in dependent task scenarios, as well as proposes two distributed reinforcement learning algorithms to find optimal solutions.

2.2. Preliminary Technology

Reinforcement learning (RL) is an approach used to tackle problems involving uncertainty and decision making. In RL, an agent generates corresponding actions by observing the state of the environment. After executing the action, the agent receives feedback from the environment in the form of reward signals. Subsequently, the agent continually adjusts its strategy based on this feedback so as to maximize cumulative rewards.

The Markov decision process (MDP) is a formal modeling of the interaction between an agent and the environment in RL. MDP assumes that the agent can fully observe the state of the environment, and that the current system state can be described by a state variable. The transitions of the system states satisfy the Markov property, thus meaning that future states depend only on the current state and are independent of past states. By defining the available actions, transition probabilities between states, and reward functions associated with each state–action pair, the optimal decision policy can be found for a given MDP. However, in real-world scenarios, we often encounter decision problems with incomplete observations, where the complete state of the system cannot be directly observed. To address this situation, the Markov decision process is extended to a partially observable Markov decision process (POMDP). In a POMDP, the agent can only infer the system's state based on partially observed information, and they utilize the inferred state to interact with the environment and obtain maximum rewards.

3. System Model

As shown in Figure 1, in this paper, we investigate an EC system composed of a single user and multiple ESs. In this system, the set of ESs is defined as $\mathcal{M} = \{1, 2, ..., m, ...M\}$, where each ES is equipped with an access point (AP). The user can select a designated AP to upload its dependent task to the corresponding ES. Moreover, these ESs are interconnected via wired connections. At a specific time slot, the user generates a dependent task. Given the limited resources and energy of the user's device, it is infeasible to complete the task execution within the desired time and energy consumption ranges. Consequently, the user offloads this dependent task to a proxy node. This proxy node then assigns the dependent task to other ESs that are incentivized to participate in task computation. Notably, this proxy node could be the ESs closest to the user.

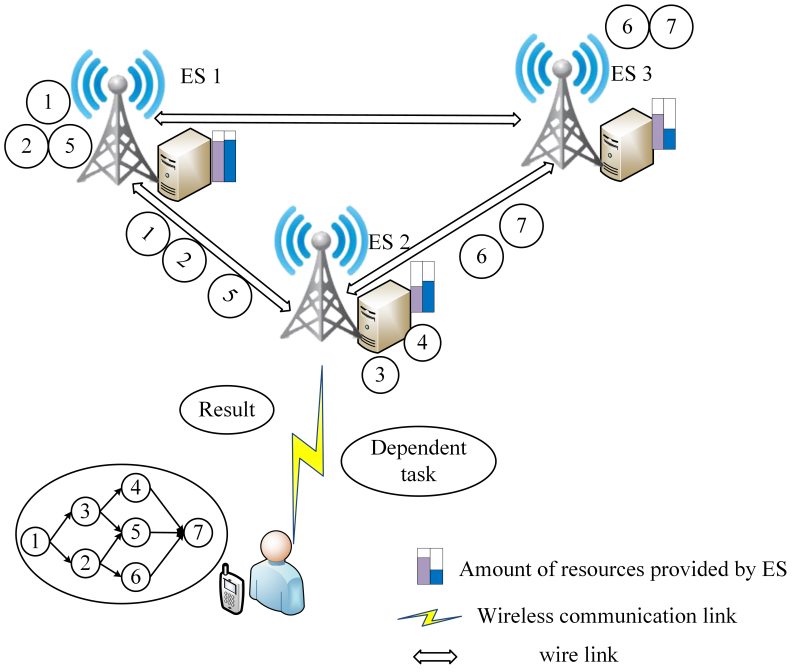

Figure 1. System architecture diagram.

The working architecture of the system in this article is shown in Figure 2, in which the user and the ESs construct an incentive mechanism through the Stackelberg game. Within this mechanism, the user takes the role of leader and proposes an incentive strategy, which is designed to encourage the ES to participate in task computation by offering a reward. The proxy node publicizes the user's incentive strategy, and, if the ES (follower) responds to the user's incentive strategy, it informs the proxy node of the amount of resources it can contribute. The proxy node then informs the user of the ES's response strategy. Next, the user offloads the dependent task to the proxy node, which, by assuming a greedy strategy, calculates that each ES needs to compute $i \in [\lceil \frac{C}{N} \rceil, C]$ sub-tasks, where C is the number of sub-tasks in the dependent task and N is the number of ESs encouraged to participate in task computation. The proxy node iterates through the values of i to distribute resources. Following this, based on each resource allocation strategy, a task distribution plan is obtained using reinforcement learning that is based on S2S neural networks. Upon completion of the task, the proxy node returns the results to the user. If the user verifies the results as correct, the reward is given to the proxy node, which then distributes the reward according to each ES's ratio of resource contribution. This process can be repeated if extended to a multi-user scenario.

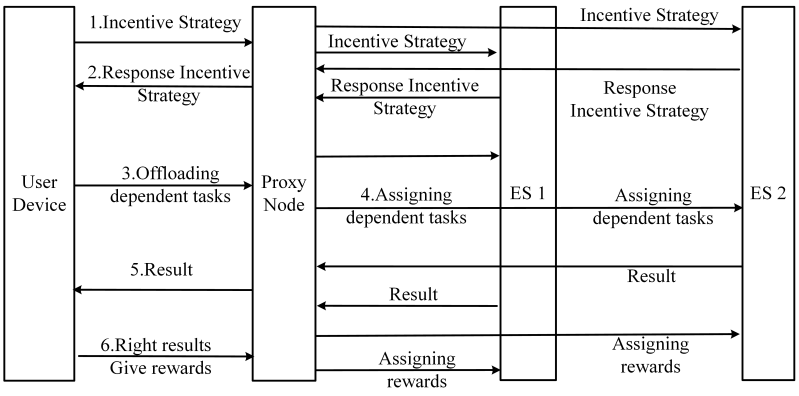

Figure 2. System work architecture diagram.

3.1. Local Computation

Although our study's task offloading strategy involves completely offloading the dependent task to the ES, the time it takes for the dependent task to execute locally still serves as a baseline for comparison, thus necessitating their modeling. In terms of the task model, we emulated most of the existing literature [18–21] by modeling a dependent task as a directed acyclic graph (DAG) $\mathcal{G} = \{\mathbf{V}, \mathbf{E}\}$, where $\mathbf{V} = \{\boldsymbol{v_n} | n = 1, 2, ..., N\}$ denotes the set of sub-tasks of the current dependent task. Each sub-task $\boldsymbol{v_n}$ is represented by a binary group, i.e., $\boldsymbol{v_n} = \{d_n, r_n\}$, where d_n denotes the size of the current task and r_n denotes the size of the computation result. $\mathbf{E} = \{e(\boldsymbol{v_i}, \boldsymbol{v_n}) | i, n \in 1, 2, ..., N\}$ denotes the dependency relationship between the sub-tasks, and when $\boldsymbol{v_i}$ depends on $\boldsymbol{v_n}$, $\boldsymbol{v_i}$ can start the computation only after $\boldsymbol{v_n}$ finishes and the result of the computation is transmitted to $\boldsymbol{v_i}$.

If f_l is used to represent the computing capacity of the local device, and η_u is the computing capacity required to process unit bit data, then the number of CPU cycles required to complete the task calculations locally is expressed as $c_{n,loc} = d_n \eta_u$. Therefore, the time T^n_{loc} required to execute $\boldsymbol{v_n}$ locally can be written as follows:

$$T^n_{loc} = \frac{c_{n,loc}}{f_l}. \tag{1}$$

Then, the local completion time of $\boldsymbol{v_n}$ is the following:

$$FT^n_{loc} = max\{AT^n_{loc}, \max_{i \in pre(v_n)} \{FT^i_{loc}\}\} + T^n_{loc}, \tag{2}$$

where $AT^n_{loc} = max\{AT^{n-1}_{loc}, FT^{n-1}_{loc}\}$ is the earliest available time of the local processor and $pre(\boldsymbol{v_n})$ is the set of direct predecessor tasks of $\boldsymbol{v_n}$.

3.2. Edge Computation

When a dependent task is offloaded to a proxy node, in order to fully utilize the resources purchased by the user, the proxy node uploads the sub-tasks of the dependent task to other ESs to achieve a collaborative execution of the task. When taking, as an example, a dependent task in ES j with the allocated sub-task $\boldsymbol{v_n}$, as well as when a target server is assigned to ES m, the execution of this task can be divided into three stages: the transmission phase, the execution phase, and the feedback phase of the result. When recording the completion time of sending as FT^n_{up}, the transmission time is T^n_{up}; then, we have

$$FT^n_{up} = max\{AT^n_{up}, \max_{i \in pre(\boldsymbol{v_n}) \wedge a_i \neq m} \{FT^i_{down}\}\} + T^n_{up}, \tag{3}$$

$$T^n_{up} = \frac{d_n}{r_j}, \tag{4}$$

where $AT^n_{up} = max\{AT^{n-1}_{up}, FT^{n-1}_{up}\}$ is the earliest available time of the uplink, $i \in pre(\boldsymbol{v_n}) \wedge a_i \neq m$ means that $\boldsymbol{v_i}$ is a direct predecessor of $\boldsymbol{v_n}$ and $\boldsymbol{v_n}$ when they are not executing on ES m, FT^n_{down} is the result of the backhaul time of the parent task of $\boldsymbol{v_n}$, and r_j is the size of the communication resources allocated for $\boldsymbol{v_n}$.

In the execution phase, we let FT^n_m, T^n_m , and f^n_m represent the completion time of $\boldsymbol{v_n}$ on ES m, the computation time, the computational resources allocated by ES m for $\boldsymbol{v_n}$, and the arithmetic power required by ES m to process the data per unit number of bits, respectively. Then, the computation completion time of the task is written as follows:

$$FT^n_m = max\{AT^n_m, \max_{i \in pre(\boldsymbol{v_n}) \wedge a_i = m} \{FT^i_m\}\} + T^n_m, \tag{5}$$

$$T^n_m = \frac{c_{n,m}}{f^n_m}, \tag{6}$$

where $AT_m^n = max\{AT_m^{n-1}, FT_m^{n-1}\}$ is the earliest available time of the ES m processor, $i \in pre(\boldsymbol{v_n}) \wedge a_i = m$ denotes that $\boldsymbol{v_i}$ is a direct precursor of $\boldsymbol{v_n}$ and executes on ES, m, $c_{n,m} = d_n\eta_m$ is the number of CPU cycles required by ES m to process $\boldsymbol{v_n}$, and η_m is the arithmetic power required by ES m to process one unit of bit data.

We let T_{down}^n represent the return time of the result after $\boldsymbol{v_n}$ computes the result on ES m. Then, the result return completion time FT_{down}^n can be written as follows:

$$FT_{down}^n = max\{AT_{down}^n, FT_m^n\} + T_{down}^n, \tag{7}$$

$$T_{down}^n = \frac{r_n}{r_m^n}, \tag{8}$$

where $AT_{down}^n = max\{AT_{down}^n, FT_{down}^n\}$ is the earliest available time for the downlink and r_m^n is the communication resource allocated to $\boldsymbol{v_n}$ by ES m for the return result.

4. Incentives under Information Sharing Conditions

In order to incentivize the participation of ESs in task computation, this section establishes an incentive mechanism based on the Stackelberg game, wherein the user is the leader and the ESs are the followers.

4.1. Participant Utility Functions

This subsection first discusses the utility function of ESs (the followers). We let f_m denote the resources contributed by ES m, and let R_1 represent the funds spent by the user to calculate the current dependent task. Given that ESs incur additional energy consumption when processing external tasks, this extra energy consumption is considered to be the inconvenience that is caused by handling these tasks. Therefore, for each ES, its utility is the reward gained from the contributing resources subtracted by its internal inconvenience [34]. Consequently, the utility of ES m by selling computational resources is as follows:

$$u_{1,m} = \frac{R_1 f_m}{\sum_{m=1}^{M} f_m} - \alpha_m D\eta_m \kappa_m f_m^2, \tag{9}$$

where α_m is the unit cost expense of computing energy consumption, D is the average data size for each sub-task, and κ_m is the effective switching capacitance.

For communication resource r_m, this paper uses communication speed as a measure. If the user purchases communication resources with amount R_2, then the utility of ES m by selling communication resources is as follows:

$$u_{2,m} = \frac{R_2 r_m}{\sum_{m=1}^{M} r_m} - \beta_m r_m, \tag{10}$$

where β_m is the unit transmission rate cost. From this, the total utility of ES m can be obtained as follows:

$$u_m = u_{1,m} + u_{2,m}. \tag{11}$$

When designing utility functions for the user, two factors need to be considered: price satisfaction and resource acquisition satisfaction. Since these two aspects are in conflict with each other, the satisfaction gained from resource acquisition follows a law of diminishing returns. This principle can be represented by a continuously differentiable, concave, strictly increasing function [35]. Therefore, a user's utility can be modeled as follows:

$$U = \delta ln(\sum_{m=1}^{M} f_m + 1) + (1-\delta) ln(\sum_{m=1}^{M} r_m + 1) - R_1 - R_2, \tag{12}$$

where δ is the compromise factor.

4.2. Problem Formulation

The goal of this paper is to incentivize ES participation in task computations under the premise of maximizing user utility, as well as to rationally allocate the resources purchased by users under the incentive mechanism to minimize the task execution time. Therefore, the objectives of this research can be defined as two sub-optimization problems.

Given user strategy $\mathbf{R} = \{R_1, R_2\}$, each ES competes for the rewards offered by the user. As a result, a non-cooperative game is formed among these ESs, with the goal of reaching a state that all ESs find satisfactory. Once this state is achieved, the user adjusts their strategy to maximize their utility. Thus, the optimization objective at this stage can be stated as follows:

$$\begin{aligned} P1: &\max_{R_1,R_2} U, \\ st.C1: &R_1 + R_2 \leq R_{max}, \\ C2: &\max_{(f_m, r_m, f_{-m}, r_{-m})} u_m, \end{aligned} \tag{13}$$

where $C1$ indicates the funds used for purchasing resources, which must not exceed the maximum funds available from the user; and $C2$ represents the state of satisfaction from the perspective of the party involved.

Under the designed incentive mechanism, the user purchases the resources needed to process the current dependent task. Thus, the objective of the second sub-problem is to effectively allocate the purchased computing resources and obtain the optimal task allocation strategy to minimize the task completion time. Therefore, the optimization objective at this stage can be written as

$$\begin{aligned} &\min(T(\mathcal{G})), \\ &st. \quad (13), \end{aligned} \tag{14}$$

where $T(\mathcal{G}) = \max_{v_n \in end(\mathcal{G})}\{FT_j^n, FT_{down}^n\}$, $end(\mathcal{G})$ is the set of exit tasks that depend on task $(\mathcal{G})$.

4.3. Stackelberg Equilibrium Analysis under Information Sharing Conditions

In this paper, we first analyze the existence and uniqueness of the SE under information sharing conditions, in which each participant in the game can obtain all the information from other participants, such as the preference factor δ reflecting the user's preference for communication and computing resources, and the inconvenience factors α_m , β_m reflecting the ES m's reception and processing tasks. The objective of the Stackelberg game model proposed by the user and ESs is to find a unique SE that maximizes user utility. In this equilibrium, neither the user nor the ESs have the motivation to unilaterally change their strategy. The SE is defined in this paper as follows:

Definition 1. *If the leader's strategy is denoted by* $\mathbf{R} = \{R_1, R_2\}$ *and the followers' strategy is denoted by* $\mathbf{Z_m} = \{f_m, r_m\}$*, then there exist optimal strategies* $\mathbf{R}^* = \{R_1^*, R_2^*\}$ *and* $\mathbf{Z}^* = \{\mathbf{Z_1^*}, \mathbf{Z_2^*}, ..., \mathbf{Z_m^*}\}$ *that are the respective Stacklberg equilibria for the leader and the follower if the following conditions are satisfied:*

$$\begin{aligned} &\forall \mathbf{R}, U(\mathbf{R}^*, \mathbf{Z}^*) \geq U(\mathbf{R}, \mathbf{Z}^*), \\ &\forall \mathbf{Z}, u_m(\mathbf{R}^*, \mathbf{Z}^*) \geq u_m(\mathbf{R}^*, \mathbf{Z}). \end{aligned}$$

Next, the Nash equilibrium (NE) of the non-cooperative game between the follower parties is analyzed, first to analyze the follower's best response for the computational resources, as well as to differentiate Equation (9) with respect to f_m to obtain the following:

$$\frac{\partial u_{1,m}}{\partial f_m} = \frac{\sum_{n=1, n\neq m}^{M} f_n}{(\sum_{m=1}^{M} f_m)^2} R_1 - 2\alpha_m D \eta_m \kappa_m f_m, \tag{15}$$

$$\frac{\partial^2 u_{1,m}}{\partial f_m^2} = -2\frac{(\sum_{n=1,n\neq m}^{M} f_n)(\sum_{m=1}^{M} f_m)}{(\sum_{m=1}^{M} f_m)^4} R_1 - 2\alpha_m D\eta_m \kappa_m < 0. \tag{16}$$

Clearly, utility function $u_{1,m}$ with respect to f_m is concave; as such, there exists a maximum value for $u_{1,m}$. This implies that a non-cooperative game concerning the allocation of computational resources has an NE. Therefore, by making $\frac{\partial u_{1,m}}{\partial f_m} = 0$, we can obtain the following:

$$\frac{\sum_{n=1,n\neq m}^{M} f_n}{(\sum_{m=1}^{M} f_m)^2} R_1 - 2\alpha_m D\eta_m \kappa_m f_m = 0. \tag{17}$$

By summing both sides of Equation (17), we obtain

$$\sum_{m=1}^{M} \left(\frac{\sum_{n=1,n\neq m}^{M} f_n}{(\sum_{m=1}^{M} f_m)^2} R_1\right) = 2\sum_{m=1}^{M} \gamma_m f_m, \tag{18}$$

where $\gamma_m = \alpha_m D\eta_m \kappa_m$, and, since the difference between any two γ_m is extremely small, the right side of Equation (18) can be approximated as $2\bar{\gamma}\sum_{m=1}^{M} f_m$. As such, Equation (18) is further computed as follows:

$$\frac{|M|-1}{\sum_{m=1}^{M} f_m} R_1 = 2\bar{\gamma}\sum_{m=1}^{M} f_m. \tag{19}$$

Thus, it can be derived that

$$\sqrt{\frac{|M|-1}{2\bar{\gamma}} R_1} = \sum_{m=1}^{M} f_m. \tag{20}$$

By substituting Equation (20) back into Equation (17), we obtain the closed-form solution for the optimal response of f_m:

$$f_m = \begin{cases} 0 & \text{if } |\mathrm{M}| < 2 \\ \sqrt{\frac{(|M|-1)R_1\bar{\gamma}}{2((|M|-1)\gamma_m+\bar{\gamma})^2}} & \text{otherwise.} \end{cases} \tag{21}$$

Given user's strategy R_1, each ES always has its own optimal response f_m. Due to the concavity of the utility function of the ES $u_{1,m}$, the optimal response is unique. Therefore, the NE of the non-cooperative game between the ESs regarding computational resources is also unique.

For communication resource Equation (10), the derivative with respect to r_m can be obtained as follows:

$$\frac{\partial u_{2,m}}{\partial r_m} = \frac{-R_2 r_m}{(\sum_{m=1}^{M} r_m)^2} + \frac{R_2}{\sum_{m=1}^{M} r_m} - \beta_m, \tag{22}$$

$$\frac{\partial^2 u_{2,m}}{\partial r_m^2} = -\frac{2R_2\sum_{n=1,n\neq m}^{M} r_n}{(\sum_{m=1}^{M} r_m)^3} < 0. \tag{23}$$

Since Equation (23) is a constant and less than zero, utility function $u_{2,m}$ is a concave function; thus, there is an NE in the non-cooperative game between the follower parties over communication resources.

From reference [36], if the game of communication resources satisfies Theorem 1, the uniqueness of the non-cooperative game's NE can be demonstrated.

Theorem 1. *Given strategy R_2 and set $\bar{\mathbf{S}} = \{m \in \mathcal{M} | \bar{r}_m > 0\}$ of follower parties, we let $\bar{r} = (\bar{r}_1, \bar{r}_2, ..., \bar{r}_n)$ be a Nash equilibrium (NE) strategy if the following four conditions are satisfied. If so, then the NE is proven to be a unique NE as follows:*

(1). $|\bar{\mathbf{S}}| \geq 2$,

(2). $\bar{r}_m = \begin{cases} 0 & if \quad m \notin \bar{\mathbf{S}} \\ \frac{(|M|-1)R_2}{\sum_{m=1}^{M}\beta_m}(1-\frac{(|M|-1)\beta_m}{\sum_{m=1}^{M}\beta_m}) & otherwise \end{cases}$,

(3). If $\beta_m \leq \max_{j\in\bar{\mathbf{S}}}\{\beta_j\}$, *then* $m \in \bar{\mathbf{S}}$,

(4). Suppose the unit rate transmission cost of the edge server satisfies $\beta_1 \leq \beta_2 \leq ... \leq \beta_n$, *and let h be the largest integer in [2,n] such that* $\beta_h < \frac{\sum_{j=1}^{h}\beta_j}{h-1}$, *then* $\bar{\mathbf{S}} = \{1, 2, ..., h\}$.

Clearly, the non-cooperative game between parties over communication resources satisfies these four conditions.

Proof of Theorem 1. When assuming $|\mathbf{S}| = 0$, no ES participates in the game and the game is not established. Therefore, it can be inferred that $\mathbf{S} \geq 1$. Now, we suppose $\mathbf{S} = 1$; this implies that $k \in \mathcal{M}$, $\bar{r}_k > 0$. According to Equation (10), the utility of ES k is $R_2 - \bar{r}_k\beta_k$, which means that ES k can unilaterally modify its strategy to increase its utility, thereby contradicting the NE. Thus, Condition 1 is proved. To next prove Condition 2 and to accumulate Equation (22), we obtain the following:

$$\sum_{m=1}^{M} r_m = \frac{(|M|-1)R_2}{\sum_{m=1}^{M}\beta_m}. \tag{24}$$

By substituting Equation (24) back into Equation (22), as well as by setting the result to zero while considering $\bar{r}_j = 0$ for any $j \in \mathcal{M} \setminus \bar{\mathbf{S}}$, we obtain the following:

$$\bar{r}_m = \begin{cases} 0 & if \quad m \notin \bar{\mathbf{S}} \\ \frac{(|M|-1)R_2}{\sum_{m=1}^{M}\beta_m}(1-\frac{(|M|-1)\beta_m}{\sum_{m=1}^{M}\beta_m}) & \text{otherwise} \end{cases}. \tag{25}$$

The proof of Condition 2 is thus complete.

For (3) given $|\bar{\mathbf{S}}|$ for any $i \in \bar{\mathbf{S}}$, we can deduce that $\bar{r}_i > 0$. According to Equation (25), $\bar{r}_i > 0$ implies $\frac{(|M|-1)\beta_m}{\sum_{m=1}^{M}\beta_m} < 1$, and hence we can obtain the following:

$$\beta_i < \frac{\sum_{j\in\bar{\mathbf{S}}}\beta_j}{|\bar{\mathbf{S}}|-1}, \forall i \in \bar{\mathbf{S}}. \tag{26}$$

This tells us

$$\max_{i\in\bar{\mathbf{S}}}\beta_i < \frac{\sum_{j\in\bar{\mathbf{S}}}\beta_j}{|\bar{\mathbf{S}}|-1}. \tag{27}$$

Assuming that there is $\beta_q < \max_{j\in\bar{\mathbf{S}}}\{\beta_j\}$, but $q \notin \bar{\mathbf{S}}$, then—according to Equation (25)—we have $\bar{r}_q = 0$. Hence, Equation (22) can be rewritten as follows:

$$\frac{R_2}{\sum_{j\in\bar{\mathbf{S}}}\bar{r}_j} - \beta_q = \frac{\sum_{j\in\bar{\mathbf{S}}}\bar{r}_j}{|\bar{\mathbf{S}}|-1} - \beta_q > \max_{i\in\bar{\mathbf{S}}}\{\beta_i\} - \beta_q. \tag{28}$$

This implies that ESs can unilaterally modify their strategy to increase their utility, which contradicts the NE definition. Thus, Condition 3 is proven.

Next, we prove Condition 4. From Condition 1 and 3, we have $\bar{\mathbf{S}} = 1, 2, ..., q$, where $q \in [2, n]$. According to Inequality (26), we can conclude that $q \leq h$. This implies that $\beta_{q+1} < \frac{\sum_{j}^{q+1}\beta_j}{q}$, which means that when $r = \bar{r}$, the derivative of utility function $u_{q+1,m}$ with respect to r_{q+1} is $\frac{\sum_{j}^{q+1}\beta_j}{q} - \beta_{q+1} > 0$. This suggests that $q = h$, and thus Condition 4 is proven. Therefore, the NE of the non-cooperative communication resources between the follower parties is unique. □

Theorem 2. *In the proposed multivariate Stackelberg game, there is a unique Stackelberg equilibrium between the user and the edge servers.*

Proof of Theorem 2. By substituting Equations (20) and (24) into Equation (12), we obtain the following:

$$U = \delta ln(\sigma_1\sqrt{R_1}+1) + (1-\delta)ln(\sigma_2 R_2 + 1) - R_1 - R_2, \tag{29}$$

where $\sigma_1 = \sqrt{\frac{|M|-1}{2\bar{\gamma}}}$,$\sigma_2 = \frac{(|M|-1)}{\sum_{m=1}^{M}\beta_m}$. The Hessian matrix of Equation (29) is obtained as follows:

$$\begin{bmatrix} \frac{\partial^2 U}{\partial R_1^2} & \frac{\partial^2 U}{\partial R_1 R_2} \\ \frac{\partial^2 U}{\partial R_2 R_1} & \frac{\partial^2 U}{\partial R_2^2} \end{bmatrix} = \begin{bmatrix} -\frac{\delta\sigma_1(\sigma_1+\frac{1}{2\sqrt{R_1}})}{2(\sigma_1 R_1+\sqrt{R_1})^2} & 0 \\ 0 & -\frac{(1-\delta)\sigma_2^2}{(\sigma_2 R_2+1)^2} \end{bmatrix}.$$

Since the eigenvalues of this matrix are all less than zero, the matrix is negative definite, implying that the original function is concave and hence possesses a maximum value. As the best response strategy of the square is unique, the value that maximizes U is also unique. Therefore, the equilibrium of this Stackelberg game is unique. □

According to the analysis above, this article develops a centralized algorithm that is capable of calculating the SE under information sharing conditions, the details of which are presented in Algorithm 1 . This algorithm achieves an approximate equilibrium for the multivariable Stackelberg game. The approximation precision depends on ε. When ε is smaller, the result is highly accurate but the algorithm converges slowly. When ε is larger, the accuracy is low but the algorithm converges quickly.

Algorithm 1 Coordinate Alternation Method

Input: initialization $\boldsymbol{D}, \boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\kappa}, I, \varepsilon, \boldsymbol{R}, \boldsymbol{x_i}$
Output: optimal strategy $\boldsymbol{R}^{[k]}, \boldsymbol{x_i}^{[k]}$

```
while ||R^[k] − R^[k−1]|| > ε do
    while ||x_i^[k] − x_i^[k−1]|| > ε do
        for i = 1, 2, ..., I do
            calculation of the followers' utility u_{1,m}, u_{2,m} by equations (9) and (10)
            save the strategy that maximizes u_{1,m} and u_{2,m} as x_i^[k]
        end for
        calculation of the leader's utility U by equation (12)
        save the strategy that maximizes U as R^[k]
    end while
end while
```

5. Study of the Incentives under Non-Information-Sharing Conditions

In the previous section, we focused on analyzing NE under the conditions of complete information transparency. However, each participant in the game is rational and may be unwilling to disclose their private parameters. This situation precludes the use of centralized algorithms to solve the NE. Deep reinforcement learning (DRL) achieves a balance between exploration and exploitation, thereby learning from the accumulated experience through environmental exploration to maximize rewards. Inspired by reference [37], we designed a reinforcement learning algorithm based on the learning game to solve the SE without sharing information. In the sections that follow, we first provide an overview of the entire framework for the incentive mechanism based on DRL. Subsequently, the NE solving problem is expressed as a DRL learning task.

5.1. Overview

To achieve the NE with respect to privacy preservation, each participant becomes an agent in the DRL, as shown in Figure 3.

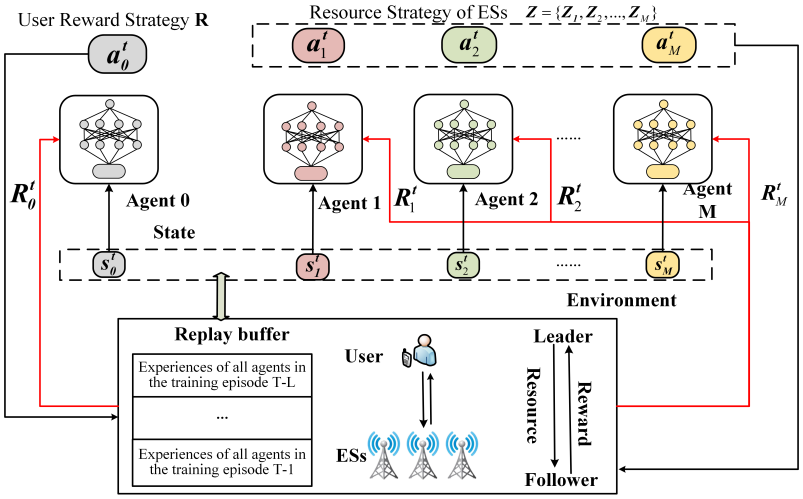

Figure 3. DRL-based incentive mechanism framework.

In the above figure, the user is viewed as Agent 0, ES m is viewed as Agent m, and the user cannot issue new incentives until the ES makes a new decision; this is because the learning state of each agent is updated according to each participant's decision. The agent training process is divided into T iterations, and in each iteration t, the user interacts with the environment as the leader of the game and determines action $\mathbf{a_0^t}$ through acquired state $\mathbf{S_0^t}$; furthermore, ES m determines action $\mathbf{a_m^t}$ by interacting with the environment as the follower of the game through the acquisition of state $\mathbf{S_m^t}$, as well as calculates incentives R_0^t and R_m^t for all participants by collecting, respectively, their strategies, which then creates the state of the next moment based on the collected strategies. Meanwhile, the agent saves the historical data of all actions in a buffer queue of size L. After D time slots, the experience in the buffer queue is re-played to compute the rewards, which are then utilized to update the network parameters. Since in each iteration of training the agent's state is generated based on the action decisions and no privacy information is acquired, this allows the finding of the NE without privacy leakage.

5.2. Design Details

In this paper, the interaction between the game participants is formulated as a multi-intelligence POMDP. The details of its state space, action space, and reward function are as follows.

State space: For the current iteration at time t, the state of each participant in the game is composed of its previous experiences and the experiences of all the other participants in the most recent training sets L. Specifically, $\mathbf{S_0^t} = \{\boldsymbol{\omega}^{t-L}, \boldsymbol{\omega}^{t-L+1}, ..., \boldsymbol{\omega}^{t-1}\}$ is expressed as the state of the user, and $\mathbf{S_m^t} = \{\mathbf{v_0^{t-L}}, \boldsymbol{\omega}_{-m}^{t-L}, \mathbf{v_0^{t-L+1}}, \boldsymbol{\omega}_{-m}^{t-L+1}, ..., \mathbf{v_0^{t-1}}, \boldsymbol{\omega}_{-m}^{t-1}\}$ is expressed as the state of ES m.

Action space: According to the game decision variables, the user's action at iteration time t is defined as $\mathbf{a_0^t} = \mathbf{v_0^t} = \{R_1, R_2\}$, and the action of ES m at time t is defined as $\mathbf{a_m^t} = \boldsymbol{\omega}_m^t = \{f_m, r_m\}$. And, in order to increase the learning efficiency, the action values of each agent are restricted to the range of 0–1 using the Min–Max normalization method.

Reward function: Based on the utility function and constraints, the agent reward function is designed since the maximum amount that the user can offer is R_{max}. Thus, a

penalty factor μ must be added to the reward function when the total price exceeds Rmax. Then, the reward function of the user at the moment of iteration t is set to be

$$R_0^t = \begin{cases} U & if \quad R_1 + R_2 \leq R_{max} \\ U - \mu(R_1 + R_2) & \text{otherwise} \end{cases}. \tag{30}$$

The reward function of ES m at time t is

$$R_m^t = u_m. \tag{31}$$

5.3. Optimization of Learning Objectives and Strategies

In this paper, an actor network π_θ and a critic network v_σ are designed for each agent. The agent learns to approximate the policy function with π_θ and the value function with v_σ, where θ and σ represent the parameters of the networks. Furthermore, for agent m, we express the state value function as $V(\mathbf{S_m^t}; \pi_{\theta_m})$ and the action value function as $Q(\mathbf{S_m^t}, \mathbf{a_m^t}; \pi_{\theta_m})$. The learning objective of the agent m is defined as $\mathcal{L}_m$. Therefore, its learning objective can be described as follows:

$$\begin{aligned} \theta_m^* &= arg \max_{\theta_m} \mathcal{L}_m(\pi_{\theta_m}) \\ &= arg \max_{\theta_m} E(V(\mathbf{S_m^t}; \pi_{\theta_m})) \\ &= arg \max_{\theta_m} E(Q(\mathbf{S_m^t}, \mathbf{a_m^t}; \pi_{\theta_m})). \end{aligned} \tag{32}$$

The training process in this study uses the proximal policy optimization (PPO) algorithm that was introduced in reference [38]. Specifically, the policy gradient and the policy gradient clipping term were defined as follows:

$$\begin{aligned} \nabla_{\theta_m} \mathcal{L}_m &= E_{\pi_\theta^m}[\nabla_{\theta_m} \log \pi_\theta^m(\mathbf{S_m}, \mathbf{a_m}) A_{\pi_\theta}^m(\mathbf{S_m}, \mathbf{a_m})] \\ &\approx E_{\pi_\theta^m}[\nabla_{\theta_m} \log \pi_\theta^m(\mathbf{S_m}, \mathbf{a_m}) C_{\pi_\theta}^m(\mathbf{S_m}, \mathbf{a_m})], \end{aligned} \tag{33}$$

$$C_{\pi_\theta}^m(\mathbf{S_m}, \mathbf{a_m}) = \min[P^m A_{\pi_\theta}^m(\mathbf{S_m}, \mathbf{a_m}), \mathcal{F}(P^m) A_{\pi_\theta}^m(\mathbf{S_m}, \mathbf{a_m})], \tag{34}$$

where $P^m = \frac{\pi_\theta^m(\mathbf{S_m}|\mathbf{a_m})}{\hat{\pi}_\theta^m(\mathbf{S_m}|\mathbf{a_m})}$ is the scale factor of the old and new policies, which is used to control the magnitude of the gradient update of the policy. $A_{\pi_\theta}^m(\mathbf{S_m}, \mathbf{a_m}) = Q_{\pi_\theta}^m(\mathbf{S_m}|\mathbf{a_m}) - V_{\pi_\theta}^m(\mathbf{S_m})$ represents the advantage function that adjusts the direction of the updates of the policy gradient.

The clipping function in Equation (34) is defined as follows:

$$\mathcal{F}(P^m) = \begin{cases} 1+\varepsilon & P^m > 1+\varepsilon \\ P^m & 1-\varepsilon < P^m < 1+\varepsilon \\ 1-\varepsilon & P^m < 1-\varepsilon \end{cases},$$

where ε is an adjustable parameter deployed to prevent policy updates from becoming excessively large (which could lead to unstable training). Following this, the actor–critic network model is updated using stochastic gradient ascent and gradient descent, respectively. As the training process progresses, the agent incrementally learns the optimal policy. Upon convergence of the training process, the agent determines the policy based on the output of the actor network.

6. Task Allocation for DRL Based on Greedy Thinking

Motivated by the user, edge servers participate in task computations. To efficiently allocate the resources purchased by user under the incentive mechanism, we design a deep reinforcement learning algorithm inspired by the work presented in reference [39]. This algorithm first employs the concept of greediness to distribute resources, and it then applies

sequence-to-sequence (S2S) neural network reinforcement learning to assign dependent tasks with the aim of minimizing task execution time. The specifics of S2S neural network reinforcement learning are detailed below.

6.1. Overview

During the task assignment and scheduling process, each edge server (ES) deploys an S2S neural network. The details of the network are shown in Figure 4.

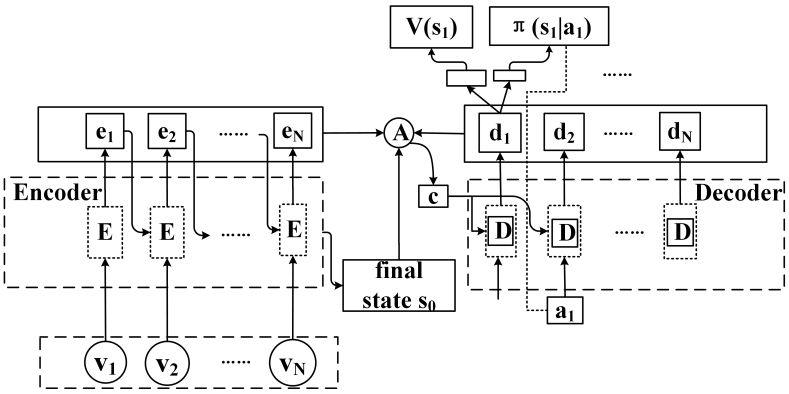

Figure 4. S2S neural network framework.

The vector embedding of the DAG is denoted as $\mathbf{V} = [\mathbf{v_1}, \mathbf{v_2}, \ldots, \mathbf{v_n}]$, the function of the encoder network is represented as f_{enc}, and by feeding the embedding vector into the encoder the hidden state of encoding step i is represented as

$$\mathbf{e_i} = f_{enc}(\mathbf{e_{i-1}}, \mathbf{v_i}; \theta_{enc}), \tag{35}$$

where θ_{enc} denotes the parameters of the encoder network. Upon completion of the encoding, the hidden-state representation of the original sequence is obtained. We let f_{dec} be the function of the decoder network, then output d_j of decoding step j is calculated as follows:

$$\mathbf{d_j} = f_{dec}(\mathbf{d_{j-1}}, a_{j-1}, \mathbf{c_{j-1}}; \theta_{dec}), \tag{36}$$

where a_{j-1} is the predicted value at the previous moment, θ_{dec} is the network parameter of the decoder network, and $\mathbf{c_j}$ is the context vector of the attention mechanism. According to reference [40], $\mathbf{c_j}$ is defined as follows:

$$\mathbf{c_j} = \sum_{i=1}^{n} \alpha_j(i)\mathbf{e_i}, \tag{37}$$

where α_j is defined as

$$\alpha_j(i) = \frac{v^T tanh(W^a[e_i; d_j])}{\sum_{k=1}^{n} v^T tanh(W^a[e_k; d_j])}, \tag{38}$$

where $[\mathbf{e_i}; \mathbf{e_j}]$ and $[\mathbf{e_k}; \mathbf{e_j}]$ are the concatenations of row vectors, and $\mathbf{W^a}$ and $\mathbf{v}$ are the learnable parameters. In addition, two fully connected layers are added to output $\mathbf{d_j}$ of the decoder, wherein one serves as the output distribution of the action network $\pi(\mathbf{a_j}|\mathbf{s_j})$ and the other serves as the output state value of value network $v(\mathbf{s_j})$. When the training of the S2S neural network is completed, the output of the network is the task assignment decision to be obtained.

6.2. Design Details

In order to solve the task offloading problem by using DRL, this paper modeled the task assignment problem as a Markov decision process (MDP), and the specific description of this process is as follows.

State space: When scheduling task $\mathbf{v_n}$, the current state of the system depends on the scheduling results of the tasks preceding $\mathbf{v_n}$; therefore, the state space is defined as a combination of the directed acyclic graph (DAG) information and a partial offloading plan, that is, $\mathbf{S} = \{\mathbf{G}, \mathbf{A_{1,i}}\}$, where G is the embedding vector of the DAG (including the index number of task $\mathbf{v_n}$), the estimated transmission and execution cost of the task, and the task numbers of the direct predecessor and direct successor. Furthermore, in this paper, we set the upper limit of the number of tasks of the direct predecessor and the direct successor to six, and $\mathbf{A_{1,i}}$ represents the sequence of decisions about task assignment from $\mathbf{v_1}$ to $\mathbf{v_i}$.

Action space: After a user passes over the dependent task to the edge server (ES) j, the edge server cooperates to execute, and this is based on the resources purchased by the user. Therefore, the action space can be defined as $\mathbf{A} = \{1, 2, \ldots, j, \ldots, M\}$.

Reward function: Since the goal of this paper is to minimize the task completion time, in order to reach this goal, this paper defines the learned reward function as the time increment saved. Therefore, the offloading sub-task $\mathbf{v_n}$ reward function is defined as follows:

$$r_n = \frac{\bar{T}_j - \Delta T_m^n}{\bar{T}_j}, \tag{39}$$

where $\bar{T}_j$ represents the average execution time of the sub-task on ES j, and $\Delta T_m^n = T_{A_{1:n}}^m - T_{A_{1:n-1}}^m$ represents the actual time spent executing the task.

Since the purchased resources are limited, a penalty factor is set for situations where the usage exceeds the purchased resources. By taking the dependent task on ES j as an example, a counting function $C(j)$ is defined under the final decision on task assignment $A_{1:N}$. This then provides the count of action j in decision set $A_{1:N}$. This function can be represented as $C(j) = |\{\mathbf{A_{1:N}}\ in\ x : x = j\}|$. Then, reward R, for executing the current dependent task, can be written as follows:

$$R = \begin{cases} \sum_{n=1}^{N} r_n & if\ \mathrm{C(j)} f_j^n \le f_j\ and\ \mathrm{C(j)} r_j^n \le r_j \\ \sum_{n=1}^{N} (r_n - \phi) & otherwise \end{cases}, \tag{40}$$

where ϕ is the penalty factor.

6.3. Optimization of Learning Objectives and Strategies

Assuming the training objective is $\mathcal{L}$, the goal is to find an optimal policy that maximizes cumulative rewards; as such, the learning goal of this section is written as follows:

$$\max \mathcal{L}(\theta) = E[\max_{\theta} \pi_\theta(\mathbf{A_{1:N}}|\mathbf{G})R], \tag{41}$$

where θ is the parameters of the S2S neural network, N represents the sub-task number, and R represents a reward function with a penalty factor. The network is also trained using the PPO method in this section. During the training process, the agent uses a discount factor γ, and the S2S neural network is updated with the discounted cumulative rewards according to every T iterations. As the training process progresses, the network gradually converges. Subsequently, the ESs can obtain the optimal unloading decision and resource allocation scheme that minimizes the execution time of dependent tasks based on the predicted results of the S2S neural network.

7. Simulation Results

In this paper, we evaluate the performance of algorithms through numerical simulations, which were implemented in a Python 3.7 environment using TensorFlow. The efficacy of the two algorithms was verified under conditions where the number of the ESs was two, three, or four. Each nonproxy ES possesses the same computational capacity, but their communication capabilities differ. The dependent tasks used were generated using the DAG generator provided in reference [39]. The specific parameters are shown in Table 1 below:

Table 1. Main Simulation Parameters.

Parameters	Value
User device computational power f_l	1 GHz
Effective switching capacitance κ_m	10^{-27}
Transmit/receive task size d_i/r_i	5∼50 Kb
CPU cycles per bit of data processed η	500∼1500 cycles/bit
Unit cost of calculating energy consumption α	10^{-6}∼10^{-5}
Incentive mechanism penalty factor μ	20
Task assignment penalty factor ϕ	5
Unit cost of communication rate β	10^{-4}∼10^{-3}

7.1. Performance Analysis of the Incentive Mechanism Algorithm Based on Learning Games

This section presents the convergence analysis of the reinforcement learning algorithm based on learning games using two edge servers as an example. Additionally, a comparison is made between the proposed algorithm and the A2C algorithm proposed in reference [41]. A2C estimates the goodness of agent actions using an advantage function, and it updates network parameters using policy gradients to learn strategies that result in higher rewards. Furthermore, we introduce the greedy algorithm and random method as baselines through which to evaluate the performance of the proposed algorithm.

The convergence plots of the user and edge server utilities are shown in Figure 5a–c. From the graphs, it can be observed that the proposed algorithm in this paper achieves a complete convergence at approximately 600 rounds, whereas the A2C algorithm converges at around 800 rounds, thus indicating a performance improvement of approximately 22.4 percent by the proposed algorithm. In terms of stability, when the proposed algorithm converges, the utilities of the participants are particularly close to the theoretical maximum social welfare (SE), whereas the A2C algorithm still exhibits a significant gap from the theoretical SE at convergence. On the other hand, the greedy and random methods oscillate without converging during the solving process. These results highlight the significant advantages of the proposed algorithm in terms of utility convergence and stability.

Based on the analysis in Figure 5, it can be concluded that the greedy algorithm and random method are unable to solve for the maximum social welfare (SE). Therefore, in order to enhance the readability of the result graphs, the subsequent strategy and convergence plots do not include the results of these two methods. Figure 6a–c are the convergence graphs of the user price strategy, ES computing resource strategy, and the ES communication resource strategy, respectively. For the user, their price strategy is a reward offered to incentivize ESs to participate in task computation, and this corresponds to both computing resources and communication resources. In the algorithm designed in this paper, as the number of iterations increases, the participants' strategy gradually approaches the theoretical Nash equilibrium, which finally converges to a position that is exceedingly close to equilibrium. Although the A2C algorithm can also converge, its policy after convergence still has a certain distance from the theoretical Nash equilibrium strategy. Therefore, in terms of convergence speed or convergence accuracy, the method designed in this paper is generally superior to the traditional A2C method.

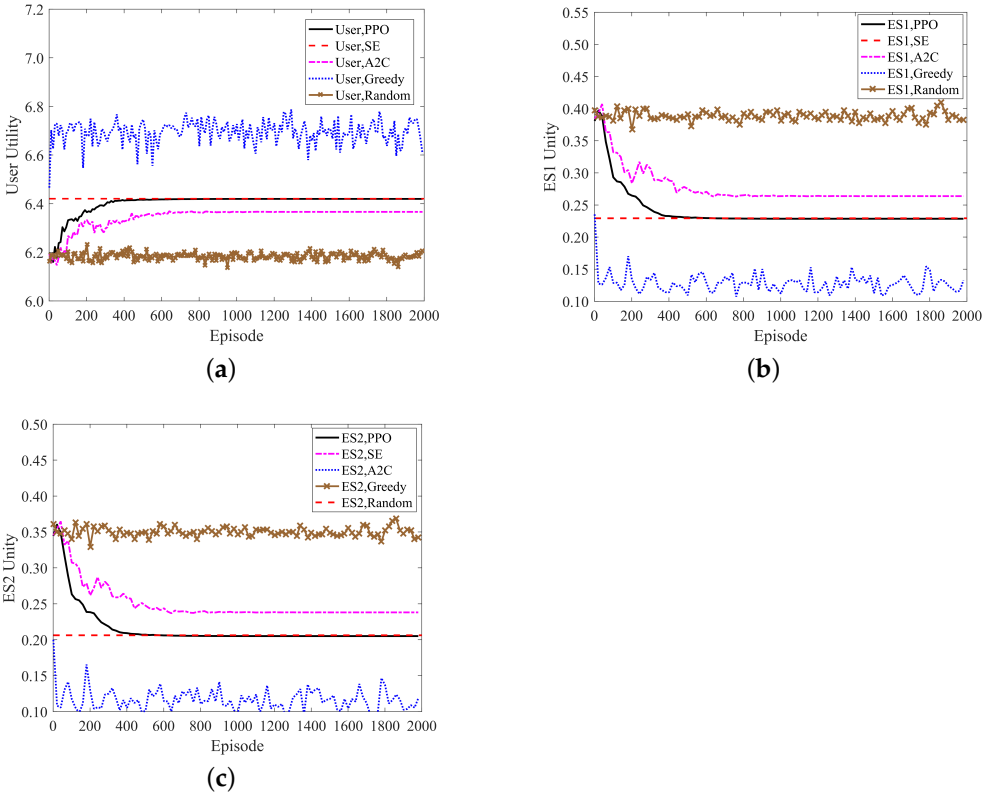

Figure 5. Utility convergence graph. (**a**) User unity convergence graph; (**b**) ES1 unity convergence graph; and (**c**) ES2 unity convergence graph.

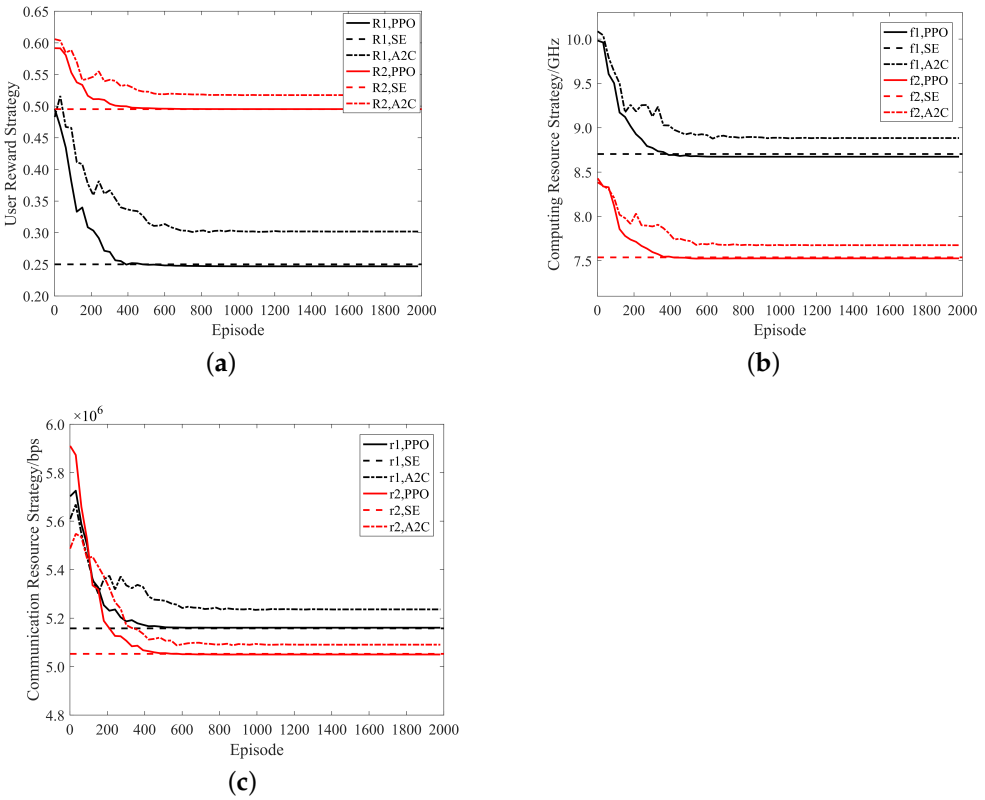

Figure 6. Strategy convergence graph. (**a**) Convergence graph of user reward strategy; (**b**) convergence graph of ES computing resource strategy; and (**c**) convergence graph of the user communication resource strategy.

7.2. Analysis of the Effectiveness of the Greedy-Based DRL Algorithm

In this subsection, we validate the effectiveness of the greedy-based DRL algorithm using two different dependency structures with identical task sizes and quantities. One dependency structure primarily consists of two parallel structures, while the other primarily consists of three parallel structures, and the depth of the dependency tasks in the two parallel structures is greater than that in the three parallel structures.

Figure 7a reflects the usage of computing resources under different numbers of edge servers. As shown in the figure, when the number of ESs is fixed, the dependency structures may vary, but the amount of computing resources consumed remains the same. This is because, regardless of the dependency structure in this case, the number of sub-tasks processed by the proxy ES in the optimal scenario is the same, the number of further distributed sub-tasks is the same, and the computing capability of each non-proxy ES is the same. Therefore, the amount of computing resources consumed is the same. Figure 7b shows the usage of communication resources in different situations. When the number of edge servers is two, regardless of the main parallel structure of the dependent task, the proxy ES distributes five sub-tasks externally; as such, the communication resources used in this case are the same. However, as the number of edge servers increases, the number of tasks assigned to each edge server under different dependent task structures varies, thus resulting in different usages of communication resources. Furthermore, the changes in total resources as shown in Figures 7a,b indicate that the more edge servers that are motivated to participate in the computation, the less resources each edge server contributes to, on average.

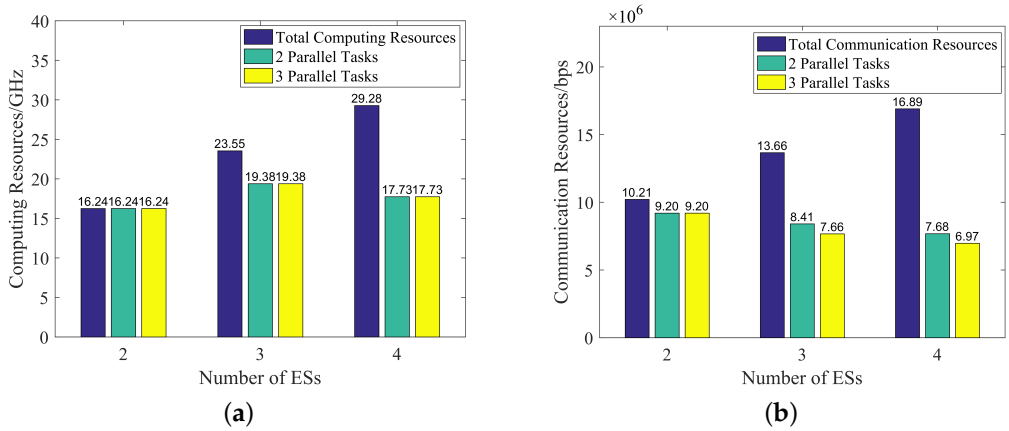

Figure 7. Resource usage graph. (**a**) Computing resource usage graph and (**b**) communication resource usage graph.

Figure 8 presents the impact of dependency structures and the number of ESs on the execution time of dependent tasks. When dependent tasks are fully executed locally, the dependency structure has no impact on the task, and thus its local execution time remains constant. For tasks primarily based on two parallel dependency structures, the execution time with three ESs is slightly less than with two edge servers. This is due to the waiting time for sub-tasks in the dependent tasks being greater than the communication time between edge servers. When the number of ESs is four, since the computing resources of each non-proxy ES are the same compared to when there are three ESs, the offloading optimal decision remains the same regardless of the parallel structure. However, as each edge server's average resource contribution decreases and less resources are allocated to each task, the execution time increases. But, overall, the execution time of the dependent task under the incentive mechanism is lesser than the total local execution time.

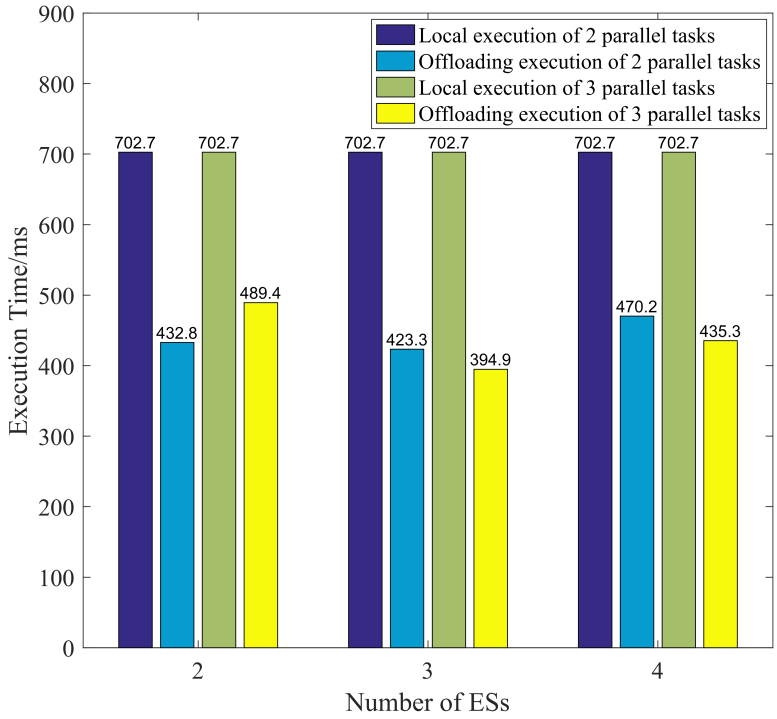

Figure 8. Task execution time.

8. Conclusions

This paper investigated a game-based incentive mechanism that is based on multiple Stackelberg variables. In this mechanism, the users act as leaders proposing incentive strategies, while the ESs respond as followers by providing available resources in response to user incentives. Subsequently, we analyzed the uniqueness of SE under information sharing conditions. Considering that participants are unwilling to disclose privacy parameters, we proposed a reinforcement learning method based on learning games to solve the Nash equilibrium under non-information sharing conditions. Upon obtaining the optimal decision, we used a greedy approach to allocate the resources provided by the ES. We employed a reinforcement learning method based on S2S neural networks to obtain the optimal decision on task allocation to minimize task execution time. The effectiveness of the model was finally demonstrated through empirical validation. Future work will consider more efficient resource allocation methods and will aim to further optimize the task allocation process with the goal of maximizing the utility of task execution.

Author Contributions: Conceptualization, Z.L. and H.J.; writing—original draft, H.J.; writing—review and editing Z.L.; methodology, H.J. and Z.L.; formal analysis H.J. and Z.R.; validation Z.L. and H.J.; software, H.J.; funding acquisition, Z.L. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the National Natural Science Foundation of China (grant number 62101174). This research was funded by the Hebei Natural Science Foundation (grant number F2021402005).

Data Availability Statement: Data are contained within the article.

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

References

1. 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]
2. Islam, A.; Debnath, A.; Ghose, M.; Chakraborty, S. A survey on task offloading in multi-access edge computing. *J. Syst. Archit.* **2021**, *118*, 102225. [CrossRef]
3. Patsias, V.; Amanatidis, P.; Karampatzakis, D.; Lagkas, T.; Michalakopoulou, K.; Nikitas, A. Task Allocation Methods and Optimization Techniques in Edge Computing: A Systematic Review of the Literature. *Future Internet* **2023**, *15*, 254. [CrossRef]
4. Liu, B.; Xu, X.; Qi, L.; Ni, Q.; Dou, W. Task scheduling with precedence and placement constraints for resource utilization improvement in multi-user MEC environment. *J. Syst. Archit.* **2021**, *114*, 101970. [CrossRef]
5. Zhang, Y.; Chen, J.; Zhou, Y.; Yang, L.; He, B.; Yang, Y. Dependent task offloading with energy-latency tradeoff in mobile edge computing. *IET Commun.* **2022**, *16*, 1993–2001. [CrossRef]
6. An, X.; Fan, R.; Hu, H.; Zhang, N.; Atapattu, S.; Tsiftsis, T.A. Joint task offloading and resource allocation for IoT edge computing with sequential task dependency. *IEEE Internet Things J.* **2022**, *9*, 16546–16561. [CrossRef]
7. Deng, X.; Li, J.; Liu, E.; Zhang, H. Task allocation algorithm and optimization model on edge collaboration. *J. Syst. Archit.* **2020**, *110*, 101778. [CrossRef]
8. Ma, L.; Wang, X.; Wang, X.; Wang, L.; Shi, Y.; Huang, M. TCDA: Truthful combinatorial double auctions for mobile edge computing in industrial Internet of Things. *IEEE Trans. Mob. Comput.* **2021**, *21*, 4125–4138. [CrossRef]
9. Huang, X.; Zhang, B.; Li, C. Incentive Mechanisms for Mobile Edge Computing: Present and Future Directions. *IEEE Netw.* **2022**, *36*, 199–205. [CrossRef]
10. Chen, J.; Yang, Y.; Wang, C.; Zhang, H.; Qiu, C.; Wang, X. Multitask offloading strategy optimization based on directed acyclic graphs for edge computing. *IEEE Internet Things J.* **2021**, *9*, 9367–9378. [CrossRef]
11. Jia, R.; Zhao, K.; Wei, X.; Zhang, G.; Wang, Y.; Tu, G. Joint Trajectory Planning, Service Function Deploying, and DAG Task Scheduling in UAV-Empowered Edge Computing. *Drones* **2023**, *7*, 443. [CrossRef]
12. Zhang, X.; Debroy, S. Resource Management in Mobile Edge Computing: A Comprehensive Survey. *Acm Comput. Surv.* **2023**, *55*, 1–37. [CrossRef]
13. Mitsis, G.; Apostolopoulos, P.A.; Tsiropoulou, E.E.; Papavassiliou, S. Intelligent dynamic data offloading in a competitive mobile edge computing market. *Future Internet* **2019**, *11*, 118. [CrossRef]
14. Zhang, K.; Yang, J.; Lin, Z. Computation Offloading and Resource Allocation Based on Game Theory in Symmetric MEC-Enabled Vehicular Networks. *Symmetry* **2023**, *15*, 1241. [CrossRef]
15. Roostaei, R.; Dabiri, Z.; Movahedi, Z. A game-theoretic joint optimal pricing and resource allocation for mobile edge computing in NOMA-based 5G networks and beyond. *Comput. Netw.* **2021**, *198*, 108352. [CrossRef]
16. Chen, Y.; Li, Z.; Yang, B.; Nai, K.; Li, K. A Stackelberg game approach to multiple resources allocation and pricing in mobile edge computing. *Future Gener. Comput. Syst.* **2020**, *108*, 273–287. [CrossRef]
17. Kumar, S.; Gupta, R.; Lakshmanan, K.; Maurya, V. A game-theoretic approach for increasing resource utilization in edge computing enabled internet of things. *IEEE Access* **2022**, *10*, 57974–57989. [CrossRef]
18. Zhang, H.; Yang, Y.; Shang, B.; Zhang, P. Joint resource allocation and multi-part collaborative task offloading in MEC systems. *IEEE Trans. Veh. Technol.* **2022**, *71*, 8877–8890. [CrossRef]
19. Liang, J.; Li, K.; Liu, C.; Li, K. Joint offloading and scheduling decisions for DAG applications in mobile edge computing. *Neurocomputing* **2021**, *424*, 160–171. [CrossRef]
20. Xiao, H.; Xu, C.; Ma, Y.; Yang, S.; Zhong, L.; Muntean, G.M. Edge intelligence: A computational task offloading scheme for dependent IoT application. *IEEE Trans. Wirel. Commun.* **2022**, *21*, 7222–7237. [CrossRef]
21. Jiang, H.; Dai, X.; Xiao, Z.; Iyengar, A.K. Joint task offloading and resource allocation for energy-constrained mobile edge computing. *IEEE Trans. Mob. Comput.* **2022**, *22*, 4000–4015. [CrossRef]
22. Chen, H.; Deng, S.; Zhu, H.; Zhao, H.; Jiang, R.; Dustdar, S.; Zomaya, A.Y. Mobility-aware offloading and resource allocation for distributed services collaboration. *IEEE Trans. Parallel Distrib. Syst.* **2022**, *33*, 2428–2443. [CrossRef]
23. Chen, S.; Rui, L.; Gao, Z.; Li, W.; Qiu, X. Cache-Assisted Collaborative Task Offloading and Resource Allocation Strategy: A Metareinforcement Learning Approach. *IEEE Internet Things J.* **2022**, *9*, 19823–19842. [CrossRef]
24. Avgeris, M.; Mechennef, M.; Leivadeas, A.; Lambadaris, I. A Two-Stage Cooperative Reinforcement Learning Scheme for Energy-Aware Computational Offloading. In Proceedings of the 2023 IEEE 24th International Conference on High Performance Switching and Routing (HPSR), Albuquerque, NM, USA, 5–7 June 2023; pp. 179–184.
25. Chen, G.; Chen, Y.; Mai, Z.; Hao, C.; Yang, M.; Du, L. Incentive-Based Distributed Resource Allocation for Task Offloading and Collaborative Computing in MEC-Enabled Networks. *IEEE Internet Things J.* **2022**, *10*, 9077–9091. [CrossRef]
26. Liu, Z.; Zhao, Y.; Song, J.; Qiu, C.; Chen, X.; Wang, X. Learn to coordinate for computation offloading and resource allocation in edge computing: A rational-based distributed approach. *IEEE Trans. Netw. Sci. Eng.* **2021**, *9*, 3136–3151. [CrossRef]
27. Tao, M.; Ota, K.; Dong, M.; Yuan, H. Stackelberg game-based pricing and offloading in mobile edge computing. *IEEE Wirel. Commun. Lett.* **2021**, *11*, 883–887. [CrossRef]
28. Seo, H.; Oh, H.; Choi, J.K.; Park, S. Differential Pricing-Based Task Offloading for Delay-Sensitive IoT Applications in Mobile Edge Computing System. *IEEE Internet Things J.* **2022**, *9*, 19116–19131. [CrossRef]

29. Kang, H.; Li, M.; Fan, S.; Cai, W. Combinatorial Auction-enabled Dependency-Aware Offloading Strategy in Mobile Edge Computing. In Proceedings of the 2023 IEEE Wireless Communications and Networking Conference (WCNC), Scotland, UK, 26–29 March 2023; pp. 1–6.
30. Bahreini, T.; Badri, H.; Grosu, D. Mechanisms for resource allocation and pricing in mobile edge computing systems. *IEEE Trans. Parallel Distrib. Syst.* **2021**, *33*, 667–682. [CrossRef]
31. Liu, X.; Yu, J.; Feng, Z.; Gao, Y. Multi-agent reinforcement learning for resource allocation in IoT networks with edge computing. *China Commun.* **2020**, *17*, 220–236. [CrossRef]
32. Li, S.; Hu, X.; Du, Y. Deep reinforcement learning and game theory for computation offloading in dynamic edge computing markets. *IEEE Access* **2021**, *9*, 121456–121466. [CrossRef]
33. Song, F.; Xing, H.; Wang, X.; Luo, S.; Dai, P.; Li, K. Offloading dependent tasks in multi-access edge computing: A multi-objective reinforcement learning approach. *Future Gener. Comput. Syst.* **2022**, *128*, 333–348. [CrossRef]
34. Huang, X.; Yu, R.; Pan, M.; Shu, L. Secure roadside unit hotspot against eavesdropping based traffic analysis in edge computing based internet of vehicles. *IEEE Access* **2018**, *6*, 62371–62383. [CrossRef]
35. Zhou, H.; Wang, Z.; Cheng, N.; Zeng, D.; Fan, P. Stackelberg-Game-Based Computation Offloading Method in Cloud–Edge Computing Networks. *IEEE Internet Things J.* **2022**, *9*, 16510–16520. [CrossRef]
36. Yang, D.; Xue, G.; Fang, X.; Tang, J. Incentive mechanisms for crowdsensing: Crowdsourcing with smartphones. *IEEE/ACM Trans. Netw.* **2015**, *24*, 1732–1744. [CrossRef]
37. Huang, X.; Zhong, Y.; Wu, Y.; Li, P.; Yu, R. Privacy-preserving incentive mechanism for platoon assisted vehicular edge computing with deep reinforcement learning. *China Commun.* **2022**, *19*, 294–309. [CrossRef]
38. Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; Klimov, O. Proximal policy optimization algorithms. *arXiv* **2017**, arXiv:1707.06347.
39. Wang, J.; Hu, J.; Min, G.; Zhan, W.; Zomaya, A.Y.; Georgalas, N. Dependent task offloading for edge computing based on deep reinforcement learning. *IEEE Trans. Comput.* **2021**, *71*, 2449–2461. [CrossRef]
40. Li, Z.; Cai, J.; He, S.; Zhao, H. Seq2seq dependency parsing. In Proceedings of the 27th International Conference on Computational Linguistics, Santa Fe, NM, USA, 20–26 August 2018; pp. 3203–3214.
41. Chen, Y.; Zhang, S.; Xiao, M.; Qian, Z.; Wu, J.; Lu, S. Multi-user edge-assisted video analytics task offloading game based on deep reinforcement learning. In Proceedings of the 2020 IEEE 26th International Conference on Parallel and Distributed Systems (ICPADS), Hong Kong, China, 2–4 December 2020; pp. 266–273.