Article

Migratory Perception in Edge-Assisted Internet of Vehicles

Chao Cai [1], Bin Chen [1], Jiahui Qiu [1], Yanan Xu [1], Mengfei Li [2] and Yujia Yang [2,*]

[1] China United Network Communications Co., Ltd., Intelligent Network Innovation Center, Beijing 100048, China; caichao2@chinaunicom.cn (C.C.); chenbin12@chinaunicom.cn (B.C.); qiujh21@chinaunicom.cn (J.Q.); xuyn39@chinaunicom.cn (Y.X.)

[2] State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, Beijing 100876, China; fiona_lee@bupt.edu.cn

* Correspondence: yangyj2022@bupt.edu.cn

Abstract: Autonomous driving technology heavily relies on the accurate perception of traffic environments, mainly through roadside cameras and LiDARs. Although several popular and robust 2D and 3D object detection methods exist, including R-CNN, YOLO, SSD, PointPillar, and VoxelNet, the perception range and accuracy of an individual vehicle can be limited by blocking from other vehicles or buildings. A solution is to harness roadside perception infrastructures for vehicle–infrastructure cooperative perception, using edge computing for real-time intermediate features extraction and V2X networks for transmitting these features to vehicles. This emerging migratory perception paradigm requires deploying exclusive cooperative perception services on edge servers and involves the migration of perception services to reduce response time. In such a setup, competition among multiple cooperative perception services exists due to limited edge resources. This study proposes a multi-agent reinforcement learning (MADRL)-based service scheduling method for migratory perception in vehicle–infrastructure cooperative perception, utilizing a discrete time-varying graph to model the relationship between service nodes and edge server nodes. This MADRL-based approach can efficiently address the challenges of service placement and migration in resource-limited environments, minimize latency, and maximize resource utilization for migratory perception services on edge servers.

Keywords: migratory perception; multi-agent deep reinforcement learning; edge computing

check for updates

Citation: Cai, C.; Chen, B.; Qiu, J.; Xu, Y.; Li, M.; Yang, Y. Migratory Perception in Edge-Assisted Internet of Vehicles. *Electronics* **2023**, *12*, 3662. https://doi.org/10.3390/electronics12173662

Academic Editors: Mehdi Sookhak and Francesco Moscato

Received: 8 August 2023
Revised: 24 August 2023
Accepted: 28 August 2023
Published: 30 August 2023

1. Introduction

With the rapid development of artificial intelligence, the automation level of autonomous vehicles has been continuously rising. One of the critical technologies of autonomous driving is the accurate perception of highly dynamic traffic environments using roadside cameras and LiDARs. Some popular methods such as R-CNN [1], YOLO [2], and SSD [3] for 2D object detection and PointPillar [4] and VoxelNet [5] for 3D object detection have demonstrated robust performance in detecting traffic entities such as vehicles and pedestrians. However, blocking from other vehicles or buildings limits the perception accuracy and range of a single vehicle, rendering it insufficient for driving safety. Given the advantages of roadside perception infrastructures in accuracy and angle of view, it will be vital to use these infrastructures to assist vehicles in performing cooperative perception. In the vehicle–infrastructure cooperative perception paradigm, roadside perception infrastructures can transmit raw data, intermediate features, or detection results to vehicles through the vehicle-to-everything (V2X) network, achieving corresponding early fusion, intermediate fusion, and late fusion. Due to the low latency and high accuracy requirements of vehicle–infrastructure cooperative perception, intermediate fusion methods such as Where2comm [6], V2X-ViT [7], and DiscoNet [8] are typically used to achieve a trade-off between data volume and feature granularity. Specifically, edge computing can be utilized for the real-time extraction of intermediate features on the roadside, while the V2X network can be used to transmit these features to nearby vehicles. However, this novel vehicle–infrastructure cooperative perception paradigm imposes new challenges on existing edge computing systems from the following aspects:

Exclusive Perception Service. Different types of autonomous vehicles use distinct neural networks for cooperative perception, necessitating the deployment of exclusive cooperative perception services for each vehicle on the edge server. These services work to extract intermediate features understood by a specific vehicle from perception data obtained from roadside infrastructures.

Migratory Perception Service. As vehicles are fast-moving, the associated roadside perception devices are constantly changing. To ensure the latency constraint of the service, the cooperative perception service needs to be migrated accordingly to reduce the service response time.

Competition Among Multiple Cooperative Perception Services. This indicates the limited availability of edge resources and the existence of competition among multiple services. If several autonomous vehicles simultaneously send requests for cooperative perception services to the edge server, the server needs to maximize resource utilization by optimizing service placement.

Therefore, service scheduling and migration for vehicle–infrastructure cooperative perception are crucial for ensuring the efficient and timely transmission of intermediate environment features. This yields a *migratory perception* paradigm. However, this paradigm faces the curse of dimensionality, which has two main aspects. On the one hand, a large number of services exist, each using a different perceptual neural network and requiring different computational resources, leading to a large state space for service scheduling. On the other hand, the action space for service scheduling is also extensive due to the large quantity of edge servers. As a solution, we propose a service scheduling method based on multi-agent reinforcement learning to guarantee the quality of services for migratory perception. The main contributions of this paper are three-fold.

- Due to the various neural network architectures used for cooperative perception in different types of autonomous vehicles, an edge-assisted migratory perception framework is proposed which leverages edge services to perceive data, extracts intermediate features, and fuses them in vehicles to achieve collaborative cognition.
- A discrete time-varying graph is designed to model the relationship between service nodes and edge server nodes. This transforms the service scheduling problem into a link prediction problem to better quantify the temporal variability of services.
- We propose a multi-agent reinforcement learning (MADRL)-based service scheduling method specifically designed to tackle the complex challenges of service placement and migration in a resource-limited environment. Migratory perception services on edge servers are modeled as multiple learning agents to minimize latency and maximize resource utilization for migratory perception.

This article initially presents the background and focus of the study in Section 1. Section 2 is dedicated to surveying previous research within the realms of perception and resource scheduling. The framework of the system under investigation is introduced in Section 3, complete with network and computing models, highlighting the issues the research aims to resolve. Moreover, it presents an intelligent service scheduling method, reliant on multi-agent deep reinforcement learning. This section also elaborates on the construction of the discrete-time graph and provides detailed information about the state space, action space, reward functions, and state transitions. In Section 4, a resource scheduling algorithm for edge computing based on QMIX is proposed to implement the method. Detailed information about the experimental environment, model definition, and training, as well as experimental data and results, is provided in Section 5. The final section concludes the paper, summarizing the key findings and achievements of this study.

2. Related Work

2.1. Cognition and Decision Making

Advancements in autonomous vehicle technology have spurred considerable research and development in single-vehicle 2D and 3D perception. Many of the key algorithms proposed involve the use of convolutional neural networks (CNNs) [9] like YOLO [2], SSD [3], and Faster R-CNN [10] for image-based object detection and recognition services. These algorithms excel in the rapid and accurate recognition and localization of objects from sensor streams, playing crucial roles in enabling autonomous navigation. Further, semantic segmentation algorithms like DeepLab [11] and U-Net [12] have considerably improved vehicles' ability to understand the environment by parsing complex scenes into intelligible segments like road, cars, and pedestrians.

In addition to image-based perception, 3D perception, mainly using LIDAR, has seen significant advancement. Algorithms like PointNet [13] and VoxelNet [5] excel at processing rich 3D point cloud data for object detection and pose estimation. Simultaneous localization and mapping (SLAM) [14], with popular algorithms such as ORB-SLAM [15], has also found widespread use in autonomous vehicles to map unknown environments and track a vehicle's location.

While the above-mentioned algorithms primarily focus on enhancing single-vehicle perception, there is an emerging paradigm shift towards multi-vehicle and vehicle–infrastructure cooperative perception. This cooperative perception leverages inter-vehicle communication (using technologies like DSRC [16] and V2X [17]) to share perception data between vehicles, contributing to a broader perception scope beyond individual vehicles. Furthermore, cooperative SLAM algorithms enable multiple vehicles to work in unison to construct a comprehensive environmental map, significantly improving overall perception and positioning accuracy. Data fusion techniques have also gained prominence in this realm, dealing with disparate sources of perception data from multiple vehicles and infrastructure. To ensure the accuracy and consistency of the data transmitted to the central fusion node, various techniques have been developed for data pre-fusion. These techniques encompass data pre-processing [18], calibration [19], and feature extraction [20], which aim to standardize the data in terms of coordinate systems and timestamps. Moving on to the mid-fusion stage, different fusion techniques are employed at the central node to integrate the data from multiple sources effectively, which techniques include traditional filtering methods [21], deep neural networks [22], Bayesian inference [23], weighted averaging [24], and fuzzy logic [25]. The goal is to generate more accurate and consistent perception results. Finally, in the post-fusion phase, the fused data are applied to specific domains or decision-making services which can involve target tracking, scene analysis, and decision formulation. Ensemble learning [26], deep fusion neural networks [27], graph models [28], and non-negative matrix factorization [29] can be used to further improve the accuracy and robustness of the fusion results. By leveraging the outcomes of data fusion, applications like autonomous driving can utilize post-fusion results for services such as path planning and control. In addition, algorithms need to not only be effective but also consider cost reduction and energy efficiency. Promising approaches such as the one proposed by Huang et al. [30] focus on cost-aware collaborative task execution within energy-harvesting device-to-device (D2D) networks. This approach highlights the potential for intelligent decision making in resource-constrained settings.

Moreover, as automated vehicles become an integral part of city traffic, the need for improved security architectures becomes increasingly essential. Basilio et al. [31] proposes an innovative osmotic computing framework that accounts for the dynamic relationship involving vehicular-to-vehicular (V2V) and vehicular-to-edge-cloud (V2EC) interactions. These developments in autonomous vehicle technology, particularly in multi-agent reinforcement learning and cooperative perception, hold promising potential for transforming future urban mobility, enhancing both efficiency and safety.

2.2. Resource Scheduling

In this section, the methods of service placement and resource allocation in edge computing are reviewed. With the expansion of edge computing application scenarios, many scholars have conducted in-depth research on edge computing.

In the domain of delay reduction, several strategies minimize the total delay of all users. For example, Wang et al. [32] and Wu et al. [33] utilized neural network-based and distributed algorithms, respectively, specifically by optimizing offloading workloads from mobile users to edge servers. These approaches significantly enhance computational efficiency and improve user experience by reducing the delays associated with the processing of services. However, issues related to the trade-off between delay minimization and computational cost remain a challenge in this category of methods. Addressing these shortcomings, a new line of research aims to achieve efficient computation offloading, predominantly influenced by game theory. Here, migratory perception services are distributed among multiple network entities to optimize the use of resources dynamically. Methods proposed by Chen et al. [34] emphasized simultaneously minimizing energy consumption and completion time, with a major focus on achieving Nash equilibrium in multi-channel wireless competitive environments. Despite these achievements, challenges related to the unstable nature of the wireless environment and complexities related to maintaining Nash equilibrium persist. Furthermore, a subset of studies such as Tan et al. [35] and Chen et al. [36] extended the problem beyond the conventional offloading, channel allocation, and power control to include aspects like caching and virtual full-duplex communication. This extends the notion of resource allocation and offers compelling solutions to some of the challenges faced by the previous methods. Yet, there exists a further need for solutions that can dynamically adapt to changes in network conditions and user requirements.

In order to optimize the service placement strategy, He et al. [37] tackled the optimization problem of network, caching, and computational resources allocation in vehicular networks via a novel deep reinforcement learning method. On the other hand, Ren et al. [38] applied a deep deterministic policy gradient-based resource allocation scheme, aiming to maximize system performance by effectively allocating computing and communication resources in a multi-agent edge computing (MEC) environment. However, a common challenge for these studies is that they do not fully take into account service variability, especially when handling multiple services concurrently. Aiming at this problem, Wang et al. [39] proposed a deep Q-network-based strategic computation offloading algorithm capable of learning optimal policies without prior knowledge. Yet, these methods often overlook service migration issues raised due to service mobility in multi-service cases.

In response to the limitations observed in both categories, some recent studies have begun to take a more comprehensive approach. For instance, Chen et al. [40] proposed a strategic computation offloading algorithm based on deep networks that makes offloading decisions based on channel quality, energy queue states, and service queue states, aiming to minimize long-term costs. However, the issue of service time-variation still remains partially addressed in the current literature, signaling an imperative need for future explorations into this area.

2.3. Security and Intelligent Decision Making

As vehicular systems become increasingly interconnected, new challenges and complexities arise, pushing the boundaries of traditional safety and security models. The integration of advanced frameworks and technologies becomes vital not only to mitigate potential risks and vulnerabilities but also to leverage the potential of IoV for enhanced transportation efficiency.

Alina et al. [41] proposed an ingenious multi-agent autonomous intersection management (MA-AIM) system that significantly contributes to vehicular network safety. Their system uses vehicle-to-infrastructure (V2I) and infrastructure-to-vehicle (I2V) communications, reinforced by blockchain technology, demonstrating a forward-looking approach to reducing intersection-related collisions. Subsequently, Alina et al. [42] also emphasized secure vehicle-to-vehicle (V2V) and vehicle-to-intersection (V2I) interactions by proposing a secure and dependable multi-agent AIM system directed at minimizing traffic collisions caused by human mistakes at intersections. This system incorporates blockchain and smart contracts, providing an extra layer of security in vehicular communications.

Marrying these ideas with the revolutionary framework proposed by Basilio et al. [43] offers an even more comprehensive approach to improving vehicular network security. Their innovative osmotic computing framework is designed to adapt to fluctuating city traffic, underscoring the dynamic relationship between vehicular-to-vehicular (V2V) and vehicular-to-edge-cloud (V2EC) interactions. As a result, their framework presents prospects for carefully maintaining traffic flow and safety within intricate city environments that feature vehicles, pedestrians, and multiple physical infrastructures.

As vehicular networks become more sophisticated, they also become more vulnerable to various threats and attacks. Thus, it is imperative to continually evolve security dynamics in line with technological advancements. Furthermore, the scalability and adaptability of current solutions to accommodate dynamic and varying traffic scenarios presents an ongoing challenge. Future research needs to focus on developing more flexible, resilient, and scalable solutions that can readily adapt to the ever-changing nature of smart city environments. Thus, while current research has paved the way for innovative solutions to enhance vehicular network security, there remains an extensive scope for further exploration and development in this field. The quest for a truly secure, efficient, and intelligent transportation system within smart cities continues, representing a critical frontier for future research and development.

3. System Model and Problem Formulation

3.1. Framework of Migratory Perception

Our migratory perception framework is utilized for vehicle-to-infrastructure collaborative perception services, as illustrated in Figure 1. This framework consists of several essential components, including edge servers, roadside cameras, vehicles, and edge cloud. In this system, the service entities are deployed on edge servers to process real-time image data from roadside cameras. Roadside cameras are connected to the edge servers through wired connections, and they serve as vital components within the system. In addition, autonomous vehicles are equipped with advanced communication modules that enable the real-time reception of intermediate features transmitted by the edge servers through wireless communication technology. After acquiring these features, the vehicles achieve long-range cognition through intermediate fusion. In order to improve resource utilization and service quality, edge servers migrate services to other edge servers for optimal service placement. Services that exceed the edge servers should be migrated to the cloud for processing.

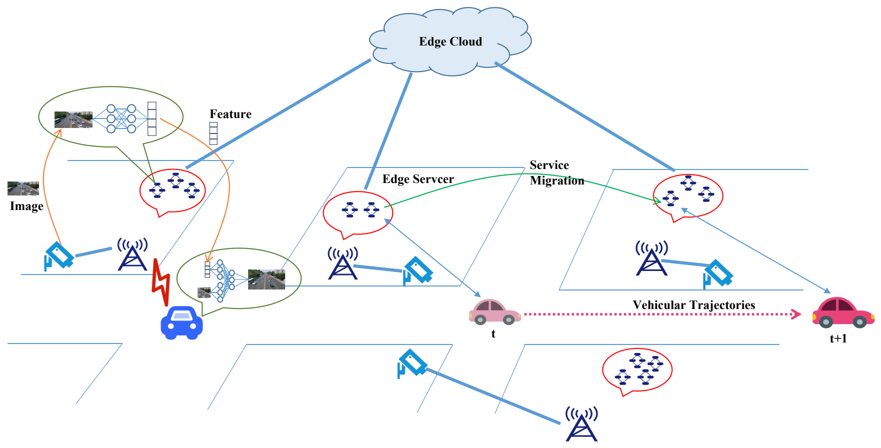

Figure 1. Framework of migratory perception.

3.2. Network and Computation Model

The system includes a set of edge servers E, a set of roadside cameras U, and a set of vehicles V. At any given moment, there is a set of perception services M waiting to be placed. Each service, denoted as $a \in M$, is described via a five-tuple (v, u, y, r, d), where v represents the identifier for the vehicle, u represents the roadside camera that generated the service, y represents the data volume of the service, r represents the initiation time of the service, and d represents the deadline for the service.

Communication Cost. Roadside cameras are directly connected to edge servers through wired connections, transmitting image data to the edge servers. The edge servers are interconnected via local area network (LAN) or wide area network (WAN). The connections between edge servers and vehicles are established through wireless communication technologies such as cellular networks (e.g., 4G or 5G) or other dedicated communication technologies (e.g., vehicle-to-infrastructure communication). Considering the fast transmission speed of data collected by cameras propagated to edge servers through wired connections, the corresponding duration is negligible. Furthermore, when edge servers return the intermediate features to the vehicles, the data transmission latency is considered negligible in the context of our investigation due to the small data volume, which consists of only a few bytes.

Service Computation Cost. In this system, the service computation cost is determined by both the data volume of services and the computational resources required for services. Therefore, the computation cost T for each service is below:

$$\begin{cases} T = H(y, b) = \psi \times y + \omega \times \frac{1}{b} \\ b = \frac{B}{N} \end{cases} \tag{1}$$

Here, $T = H(y, b)$ represents the latency of the migratory perception service, where y denotes the data volume of the service and b denotes the computational resources required for the service. H is a function that describes the relationship between service latency, service data volume, and resource quantity. ψ and ω are coefficients that represent the relative impact of the data volume of the service and the computational resources required for the service, respectively. B represents the total resources of edge servers hosting the service, and N represents the total number of services on the edge server. Using the function $H(y, b)$, we can calculate and analyze the latency of a service.

Migration Costs. Migrating service entities between edge servers inevitably incurs migration costs. Within our system, migration costs are defined as the expenses incurred during the transmission of data from service entities on the current edge server to another edge server across a given time period.

$$c^{mgt} = \alpha \times g \times y + \beta \times l \tag{2}$$

In this equation, the variables α and β represent the relative weights assigned to data volume and distance, respectively. In our system, g indicates whether the service undergoes migration and y represents the data volume from migrated services, while l represents the distance between the edge server hosting the service and the edge server where the service is to be migrated.

3.3. Problem Formulation

In the context of a three-tier network architecture comprising a vehicle, an edge server, and an edge cloud, our objective is to optimize the service migration strategy for vehicular networks.

Assuming that each edge server has one or more migratory perception services, for the services that require migration, they will be migrated to other edge server nodes that meet the resource requirements. Given the limited number of edge server nodes, the system aims to minimize the migration cost and the service timeout rate.

The migration costs of services are constituted by the service data volume and migration distance. To minimize the migration cost and the service timeout rate, we incorporate them into the optimization function as follows:

$$\begin{cases} F = \sum(\mu * c_i^{mgt} + \gamma \times (1 - p_i) \times c_p) \\ \sum_j x_{i,j} = 1 \\ \sum_i (b_t^{i,j}) \leq f_t^j \\ x_{i,j} \in \{0, 1\} \end{cases} \tag{3}$$

Suppose the remaining computing resource quantity for edge server node j at time t is f_t^j. In order to ensure that the resource requirements of each service allocated to the edge server node are met, an edge computing resource upper limit constraint is designed, where $\sum_i (b_t^{i,j})$ represents the total allocated resources for service i on edge computing node j at time t.

In the equation provided above, it can be observed that each service i can only be assigned to one edge server j and the completion time of each service i should be within a given deadline. Here, μ and γ, respectively, represent the weights of service migration cost and service timeout, c_p represents the penalty due to service timeout, c_i^{mgt} represents the migration cost for service i, and the migration cost is defined in Equation (2). $x_{i,j}$ represents whether service i is assigned to server j. p_i represents whether the service has timed out, and it is defined as follows:

$$\begin{cases} p_i = 1, \sum(T_{i,j}) + r_i \leq d_i \\ p_i = 0, \sum(T_{i,j}) + r_i > d_i \end{cases} \tag{4}$$

In addition, $T_{i,j}$ represents the time of service i on server j.

4. Method

4.1. Intelligent Service Scheduling Discrete Time-Varying Graph Construction

A discrete three-dimensional time-varying graph comprises a set of graphs $G_1, G_2, ..., G_t$. Each graph $G_t = < K, D >$ represents the topological relationship of service migration between edge servers at time slot t, which can be described as follows: edge servers form a network topology, enabling the migration of services through interconnected links. This topological relationship is dynamic and depends on the requirements and strategies of service migration. During the service migration process, the source edge server transfers the execution state and data of the service to the target edge server. In the context of a time-varying graph G_t, the nodes K comprise two categories: service nodes M and edge server nodes E, denoted as $K = M \cup E$. Service node a is described via a four-tuple (b, x, r, d), where b represents the resource quantity, x represents the location of the current edge server node s where service node a is situated, r represents initiation time, and d represents deadline, while edge server node s is described via a two-tuple (B, f_t),where B represents total resource and f_t represents the available resource remaining at time slot t. The edges D represent the edges between service nodes and edge server nodes. The edge weight of service nodes and edge server nodes represents the migration cost, from the current edge server hosting the service at time t to the target edge server at time $t + 1$. This is further elaborated upon in Equation (2). The existence of corresponding edges between service nodes and edge server nodes is only valid when the constraint conditions are met. The nodes and edges of the time-varying graph change over time.

Figure 2 provides a simplified representation of the local state concerning the same service in two adjacent time slot graphs, G_t and G_{t+1}. The figure involves six edge server nodes. In the left graph corresponding to G_t, service a_1 is connected by edges to the edge server nodes s_1, s_4, s_5, and s_6, indicating that these four edge server nodes can satisfy service a_1 constraint requirements at that time. Therefore, service a_1 can be completed within the constraints on any of these computing nodes, while the remaining edge server nodes do not meet the constraints. In contrast, the right graph corresponding to G_{t+1} shows that, due to changes in service spatial positions or edge server node loads, s_1, s_5, and s_6 no longer satisfy a_1 constraints. Edge server nodes s_2, s_3, and s_4 now satisfy the service's constraints, so a_1 adds connecting edges to s_2, s_3, and s_4.

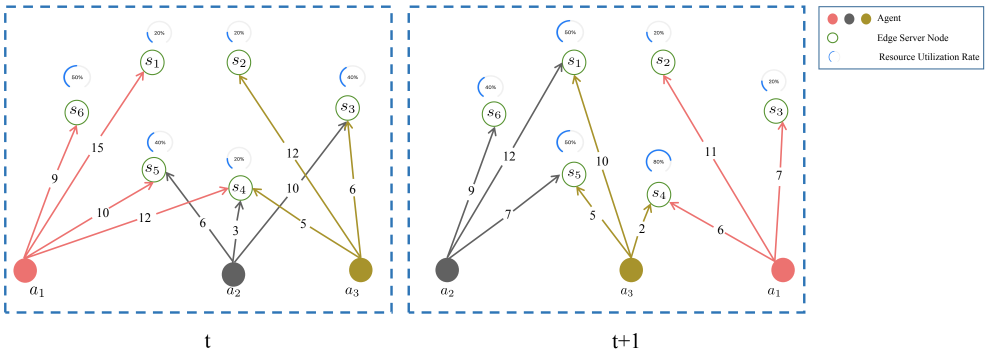

Figure 2. The left graph represents G_t, while the right graph represents G_{t+1}.

In Figure 2, by connecting service nodes in adjacent time slots with the direction pointing toward increasing timestamps, the model establishes the temporal relationship between the same services across time dimensions.

Using this graph, the migration cost incurred by traversing the edges between service nodes and edge server nodes can be computed, which is elaborated upon in Equation (2). In conjunction with the global state of each edge server node and service, we determine the optimal migration strategy for service placement.

4.2. Intelligent Service Scheduling Decisions Based on Multi-Agent Deep Reinforcement Learning

In the context of edge cooperative perception services for vehicular networks, there are multiple services that need to be scheduled and placed effectively. Traditional methods may struggle to handle the complex coupling relationships and migration trends among these services. MADRL, which combines concepts from reinforcement learning and multi-agent systems, allows multiple agents to learn and make decisions in a collaborative manner. By using MADRL, it is aimed to optimize the scheduling and placement of multiple services in a cooperative and intelligent way.

Therefore, we can model the problem as a multi-agent deep reinforcement learning (MADRL) framework, where each cooperative perception service deployed on an edge server is modeled as an intelligent agent. The objective is to minimize the function F. To model this problem, we define the state space, action space, reward function, and state transition as follows.

4.2.1. State Space

In our system, each agent represents an edge cooperative perception service. Multiple services can coexist on an edge server, with each service functioning as an individual agent. At time slot t, agents obtain their partial observation states based on the time-varying graph during that time slot. These observation states serve as input for the agents, assisting them in decision-making processes.

In this context, the observation state space z for an agent is represented via a five-tuple $(a_i, s^{opt}, f, c^{mgt}, y)$, where a_i represents the agent's identifier, s^{opt} denotes the set of optional edge server nodes, which represent edge server nodes that satisfy the constraints for service migration, f represents the remaining resources corresponding to the optional edge server nodes, c^{mgt} represents the migration cost from the edge server node hosting the service to the optional edge server nodes, and y represents the total data volume from the optional edge server nodes within a specific range. Thus, the observation state at time slot t can be defined as:

$$z_t = \{a_i, s_t^{opt}, f_t, c_t^{mgt}, y_t\} \tag{5}$$

4.2.2. Action Space

In our edge computing resource scheduling system, each agent, representing a service on an edge server, is required to select a suitable edge server node from the available nodes. Assuming the number of edge server nodes is n, the dimensionality of the action space corresponds directly to the number of nodes, represented as:

$$A = \{s_t^1, s_t^2, ..., s_t^n\} \tag{6}$$

As shown in Figure 2, the agent a_i points to the edge server node s, indicating that agent a_i can choose from the set of nodes s. If the selected edge server node remains unchanged from the previous time slot, no additional actions are necessary. However, if the chosen edge server node differs from the previous one, a migration action must be executed.

4.2.3. Reward Function

In our system, a joint reward is employed to regulate the training of the multi-agent deep reinforcement learning network for edge computing resource scheduling. The agents' rewards are determined by two key performance indicators: the migration cost and the service timeout rate. The reward function o is defined as follows:

$$o = R(z,e) = \sum(\mu \times c^{mgt}(z,e) + \gamma \times (1 - p(z,e)) \times c_p) \tag{7}$$

In the given objective function, $o = R(z,e) = \sum(\mu \times c^{mgt}(z,e) + \gamma \times (1 - p(z,e)) \times c_p)$, where o represents the objective function F to be optimized, as described in Equation (3), z denotes the observation state space, and e signifies the intelligent agent's action. Here, $c^{mgt}(z,e)$ represents the migration cost based on the current observation state space z and the intelligent agent's action e. $p(z,e)$ represents whether the service has timed out based on the current observation state space z and the intelligent agent's action e.

The objective function aims to optimize the performance of the edge computing system by minimizing the migration cost, ensuring task completion within the specified time constraints, and managing the penalty cost associated with task timeouts. By optimizing this objective function, we can guide intelligent agents to make decisions that lead to improved performance and resource utilization in the edge computing system.

4.2.4. State Transition

In the migratory perception framework, all agents make their selections at each time slot, resulting in a joint action. After the joint action is executed, the system transitions to a new time slot with an updated state. In this update status, the residual resource amount and the edge server node where the service is located in the time-varying graph are adjusted according to the probability transition function. This transition denotes movement from G_t to G_{t+1}, as illustrated in Figure 2. Within the edge computing resource scheduling framework, at time step $t+1$, agent decisions from the previous time step are used to update information such as the remaining resource volume for edge server nodes, agent positions, and other relevant parameters. This leads to the generation of a renewed time-varying graph, which serves as the basis for the next round of decision making.

4.3. Edge Computing Resource Scheduling Algorithm Based on QMIX

The QMIX algorithm [44] is employed in this study to address the problem of edge server resource scheduling in scenarios where edge server node resources are scarce and service computation demands are high. QMIX considers the impact of each agent's behavior on the overall environment, optimizing resource allocation and ensuring an optimal service placement strategy. In contrast, Q-learning and DQN only consider the reward of a single agent, potentially neglecting the overall system performance optimization.

In our system, intelligent agents represent the migratory perception service on edge servers. The QMIX algorithm is utilized as the multi-agent reinforcement learning optimization algorithm for the system, achieving an optimized service placement strategy, which incorporates a mixing network to merge the local value functions of individual agents. By incorporating global state information during the training process, QMIX improves its performance.

The method of centralized training and distributed execution to train agent policies is employed. During the training phase, each agent produces a local action-value function C based on its local observations z. The migratory perception service on the edge servers processes the inputs from the network and generates intermediate features. Then, the mixing network combines the local value functions and utilizes the global state information S as auxiliary data for the merged local value function.

During the training iterations, agents continually explore the environment, collect actions and observations, and optimize rewards. Network backpropagation is performed, updating the network parameters using gradient descent. In the execution phase, the mixing network architecture is no longer utilized; each agent merely relies on its local network, outputting actions based solely on local observations.

As described in the previous section on multi-agent deep reinforcement learning modeling, in the edge computing resource scheduling framework, at time slot t, each service node agent's local observation is represented as $z_t = \{a_i, s_t^{opt}, f_t, c_t^{mgt}, y_t\}$. The global state is the union of all agents' local observations, represented as $S = \{S_t, S_{t+1}, F_t, C_t\}$, where S_t represents the states of all edge server nodes in the environment at time slot t, S_{t+1} represents the states of all edge server nodes in the environment at time slot $t+1$, F_t represents the total remaining resources of the edge servers in the current environment, and C_t represents the total migration cost of the service at time slot t.

Let $\tau = \{\tau_1, ..., \tau_n\}$ denote the joint action-observation history, where represents agent a's action-observation history. Q_{tot} represents the joint action-value function, while $Q_i(\tau_i, u_i; \theta_i)$, where θ_i represents the network parameters and u_i is the action of each agent.

QMIX employs a mixing network to merge the local action-value functions of individual agents, incorporating global state information during the training process to improve algorithm performance. As taking the *argmax* of the joint action-value function Q_{tot} is equivalent to taking the *argmax* of each local action-value function Q, their monotonicity values are the same, as demonstrated below:

$$argmax_u Q_{tot}(\tau, u) = (\begin{matrix} argmax_{u_1} Q_1(\tau_1, u_1) \\ argmax_{u_n} Q_n(\tau_n, u_n) \end{matrix}) \tag{8}$$

In this study, τ represents the joint action-observation history of all agents, τ_i represents the joint action observation history of agent i, u represents the joint action of all agents, u_i represents the action of agent i, and Q_i represents the local action-value function of agent i. Based on τ_i and u_i, we select the edge server node corresponding to the maximum action using the *argmax* of the local action value function Q.

The distributed strategy greedily acquires the optimal action through local Q_i. QMIX transforms the above equation into a monotonicity constraint: $\frac{\partial Q_{tot}}{\partial Q_i} \geq 0, \forall i \in \{1, 2, \ldots, n\}$. If the above monotonicity is satisfied, the equation holds. In order to achieve the aforementioned constraint, QMIX employs a mixing network, whose specific structure is shown below:

According to the QMIX mixing network diagram, each agent, i.e., the migratory perception service on edge server nodes, inputs the selectable actions along with the agents' information and passes through their GRU network to output the corresponding local action-value function. The output of the GRU network serves as the input to the mixing network, which also accepts the global observation functions as input, enabling it to achieve global optimality, as shown in Figure 3.

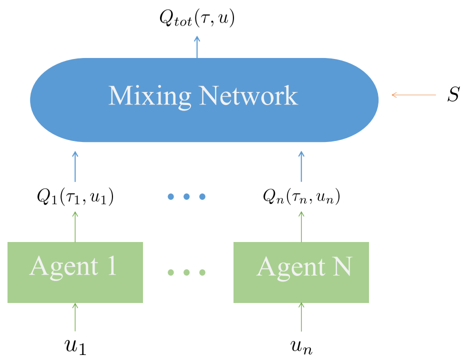

Figure 3. QMIX mixing network [44].

The QMIX model consists of two main components: an agent decision network that outputs the Q_i function of a single agent and a mixing network that takes Q_i as input and outputs a joint Q_{tot}.

The mixing network is a feedforward neural network that takes the output of the agent network as input and monotonically mixes the generated Q_{tot} values, as shown in Figure 4. To ensure monotonicity constraints, the mixing network's weights are restricted to non-negative values (biases can be negative). This allows the mixing network to approximate any monotonic function.

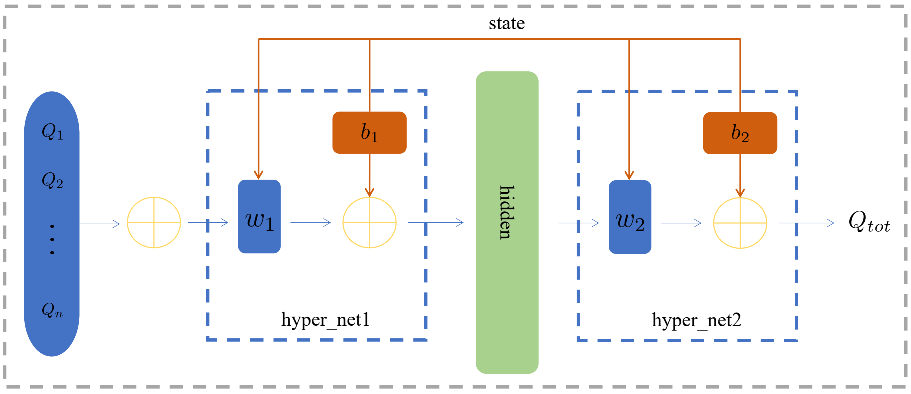

Figure 4. Mixing network.

The weights of the mixing network are generated by separate hypernetworks. Each hypernetwork takes the global state $S = \{S_t, S_{t+1}, F_t, C_t\}$ as input, and the decision network of each agent is implemented through the DRQN network. This network can be trained separately for different agents and can also have parameter sharing based on business requirements. In the context of resource scheduling, we adopt the approach of parameter sharing for training.

For each agent a, there exists a corresponding agent network which represents an individual value function $Q(\tau_a, u_a^{t-1})$. Here, u_a^{t-1} represents the action of agent a at time slot $t-1$, where τ_a represents the history of observation sequences. In this context, z_a^{t-1} represents the local observation input of agent a to the GRU at time slot $t-1$, and z_a^t represents the output local observation of agent a through the GRU at time slot t. We represent these agent networks as DRQNs, which receive the current individual action u and the joint action observation history of agent a τ_a as input at each time step, as depicted in Figure 5.

DRQN replaces the fully connected layer in DQN with a GRU network, and its recurrent layer consists of a 64-dimensional hidden state GRU. Recurrent networks exhibit stronger adaptability when the quality of observations changes. As depicted in Figure 5, the network contains three layers: input layer (MLP multi-layer neural network) → intermediate layer (GRU gated recurrent neural network) → output layer (MLP multi-layer neural network).

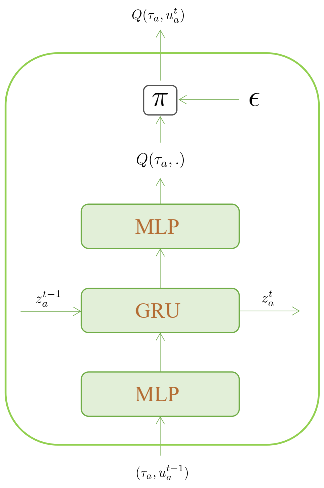

Figure 5. QMIX mixing network value calculation.

First, the global information of the environment is input, and actions are selected randomly with probability ϵ and greedily with probability $1 - \epsilon$. By employing this ϵ-greedy algorithm, a minimal regret value can be ensured. If it enters the evaluation network, the edge server node to which the service is migrated with the highest local action value is selected through the policy function as the action and input to the current environment. The updated environment observation S_t and the current reward o are input into the experience replay pool after the environment is updated according to the generated agent actions. The experience replay pool re-inputs the current reward and environment observation into the target network. The evaluate network and target network are updated using the TD-error algorithm, receiving the Q value of the action selected by the actor network under state Q as input and outputting $Q_{tot}(evaluate)$. The maximum Q value among all actions in state S_{t+1}, as input by the actor network, is received, and $Q_{tot}(target)$ is output. Edge Computing Resource Scheduling Algorithm based on QMIX is shown below (Algorithm 1):

Algorithm 1: QMIX Algorithm

```
Initialize experience replay buffer
Initialize Q networks for all agents and the mixer
Initialize target networks for all agents and the mixer
For each episode = 1, M do
    Collect joint observations from all agents
    For each agent $a$ do
        Select action $u_a$ using exploration strategy
        Execute joint action $\mathbf{u} = (u_1, \ldots, u_n)$
        Receive joint reward $R$ and next joint observations
        Store experience tuple in replay buffer
    End for
    Sample mini-batch of experiences from replay buffer
    For each agent $a$ do
        Update Q network $Q_a$ using gradient descent
        Calculate local Q-value $Q(\tau_a, u_a^{t-1})$
    End for
    Calculate global Q-value using mixer network
    Calculate TD target and TD error for each agent
    For each agent $i$ do
        Update Q network using TD error and gradient descent
        Update target network using a soft update rule
    End for
End for
```

5. Experiment

5.1. Experimental Environment

5.1.1. Training Environment

The simulation environment mainly simulates two parts:

Environment simulation: simulating vehicle movement, service generation, and distribution, as well as service completion and termination.

Edge computing platform simulation: simulating the location of edge server nodes and initializing and updating resource quantities. The simulation environment primarily simulates the agent's environment, while the service placement environment simulates the edge computing platform.

The simulation environment is mainly responsible for the initialization and attribute management of vehicles, the attribute management of service nodes on edge server nodes, service distribution, and destruction.

The service placement environment is primarily responsible for the initialization and attribute management of edge server nodes. The resource matrix of edge nodes records the total and remaining resource volumes, where the remaining volume will change according to the service schedule. The service record dictionary of edge server nodes records the services and their occupied resource volumes at the current time slot on that edge node.

Upon initializing the simulation environment, the environment generates a specified number of vehicles and randomly assigns their starting and destination locations. Vehicles move toward their destination locations at a specified speed. The environment dispatches services with random data sizes to edge server nodes at each time slot with a probability that is determined based on the current total number of services on the edge server nodes, and the MADRL model is responsible for the service schedule to complete services within a limited time and minimize function F. The service's time limit is proportional to its random data volume, and the dispatching probability is inversely proportional to the current total number of services on the edge server.

5.1.2. Training Process

In this experiment, the simulation environment is first initialized. Then, the relevant information is obtained from the environment and processed into the input of each agent in the multi-agent reinforcement learning QMIX model. The scheduling decision output of each agent is obtained. The output is input into the simulation environment, which updates the corresponding attributes and computes the relevant rewards.

Due to constraint limitations, only resource constraints are considered in the environment when selecting optional edge server nodes for service agents. In this environment, resource quantity is quantified as the number of CPUs, and the required number of CPUs is mapped from the current remaining data volume.

5.2. Model Definition and Training

5.2.1. Hardware Description

The experiments were conducted using a computer system equipped with an Intel(R) Core (TM) i7-7700, 3.6 GHz CPU, and 16 GB DDR4 RAM. The GPU used was an NVIDIA GeForce GTX 1050 Ti with 4 GB of VRAM. The operating system was Ubuntu 18.04 LTS, and the experiments were implemented using Python 3.7 and TensorFlow 1.15 for the reinforcement learning algorithm.

5.2.2. Experimental Data Description

The experimental data are obtained by simulating actual data. Services are placed on edge server nodes. When distributing services, the birth time r, deadline d, total data volume required y, and node information of the service are initialized. As the time slot changes, according to the actions of the previous time slot's services, the resource remaining volume and service remaining data volume of the edge server nodes are updated to construct the simulated data for the edge computing resource scheduling system.

5.2.3. Model Parameter Settings

In this experiment, the environment was set to a size of 32 cellular grids, the number of services ranged from a minimum of 1 to a maximum of 64, and there were 8 edge computing platforms, with each platform covering 4 cellular grid areas. The QMIX reinforcement learning model parameters are as shown in Table 1.

Table 1. QMIX algorithm service scheduling model definition.

Parameter Name	Value
GRU hidden layer dimension	64
Mixture network hidden layer dimension	32
Exploration factor	1.0–0.05
Reward discount factor	0.99
Buffer size	5000 rounds of simulated data
Sampling batch size	32 rounds of simulated data
Target network update frequency	Every 200 rounds of simulation
Learning rate	5×10^{-4}

5.2.4. Training Output

The training process uses the function F as the training objective. As the training process iterates, the function F converges to a stable value. A convergence curve is plotted to determine whether there are any issues in the training process. If convergence to a stable value is achieved, it indicates that the multi-agent reinforcement learning process is normal, as shown in Figure 6.

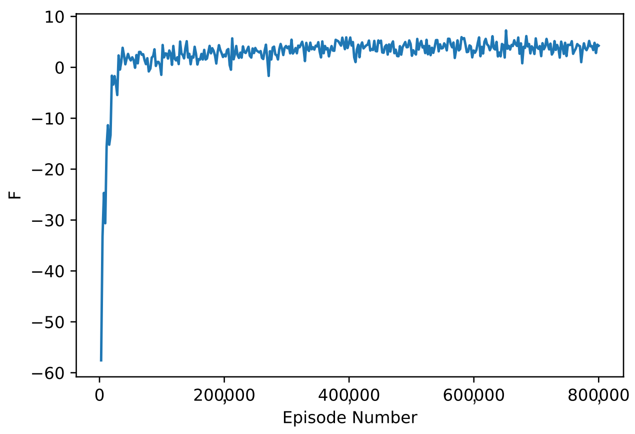

Figure 6. QMIX algorithm training process F situation.

5.3. Experiment Setup

Three different configurations are set up in the experiment to compare the training effects of reward, CPU resource utilization, and service completion rate in service scheduling.

(1) Service scheduling decision based on the QMIX algorithm.

(2) Service scheduling decision based on the greedy policy.

The greedy policy chooses the edge server node with the minimum of the function F among the nodes that satisfy the constraint conditions as the target migration node for the agent. It uses a strategy of choosing the locally optimal solution at each stage to achieve service scheduling and allocates services based on current information and resources, attempting to select the most optimal option for the current environment at each decision point. However, the greedy policy may encounter a high iteration count and low computational efficiency due to the computational complexity and overhead associated with considering global optimal solutions as the environment becomes more complex.

(3) Service scheduling decisions based on the random policy.

The random policy randomly selects an edge server node that satisfies constraint conditions as the migration target node for the agent. The random policy relies on random sampling rather than fixed rules or limits for allocating resources and services. Random policies may not provide an optimal solution and a completely random policy could result in unbalanced load distribution or service delays.

The hierarchical edge computing intelligent service scheduling model is compared with the classic greedy optimization algorithm and random policy.

When there are 5 computing nodes, 10 vehicles, 10 services, CPU resources for the computing nodes are within between 16 and 32, and service data volume is within between 100 and 1000, the reward values of the three algorithms are compared:

In Figure 7, the X-axis represents the number of training iterations and the Y-axis signifies the function F. Presented from left to right are the performances of the random policy, the QMIX-based algorithm, and the greedy policy.

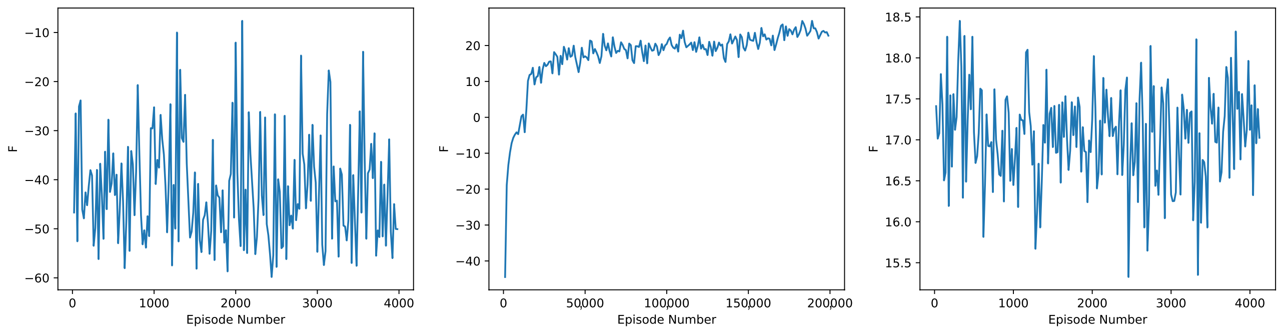

Figure 7. The corresponding F values from left to right are those of the random policy, QMIX policy, and greedy policy.

Upon observing the rewards returned by the random policy, it is evident not only that the performance is consistently poor, but the results also fluctuate significantly, suggesting an unstable policy. The greedy policy, on the other hand, converges quickly to a stable value of about 17. Despite this, the rapid early gain does not compensate for subsequent performance—which is outraced by the QMIX algorithm. The QMIX-based algorithm shows a gradual, consistent, and more significant increase in rewards as the training iterations increase. The value of F stabilizes to an approximate value of 20, which is better than that of the greedy policy. The results demonstrate the superior performance of the QMIX model in intelligent service scheduling in edge computing environments.

When there are 10 edge server nodes, 15 vehicles, 15 services, CPU resources for computing nodes are within between 16 and 32, and service data volume is within between 100 and 1000, the function F values of the three algorithms are compared in Figure 8:

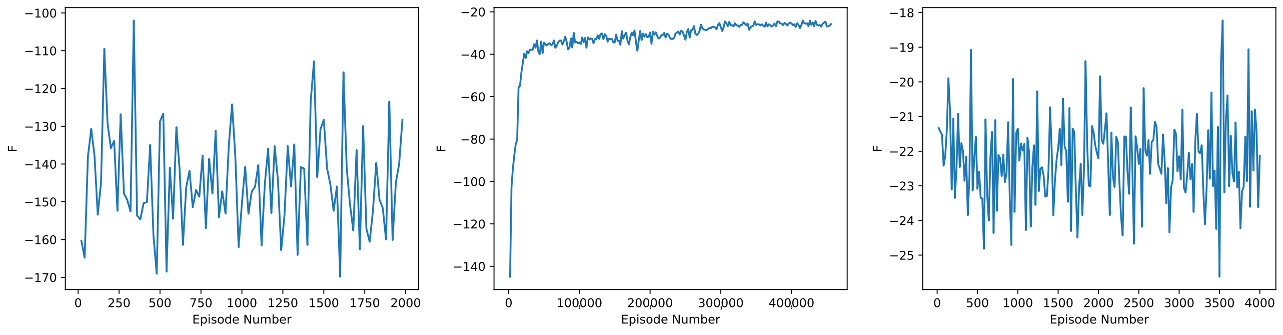

Figure 8. The corresponding F values from left to right are those of the random policy, QMIX policy, and greedy policy.

Expanding on the comparison made, an algorithm's performance is directly proportional to the complexity of the environment. The more complex the situations are, the more difficult it is for the algorithms to perform efficiently. An increase in edge server nodes, vehicles, and services presents a more complex environment for the three algorithms.

In the case of the greedy policy, it seems to adapt better to larger environments, showing superior performance when there are more nodes and vehicles. This could be attributed to its nature of algorithm design, which aims to maximize short-term rewards, therefore outperforming other policies in this aspect.

The QMIX policy, despite its sophisticated model design, seemed to struggle more than the greedy policy when dealing with large environments. It showed signs of difficulty in learning the service placement strategy well, leading to reduced performance. The decrease in function F value points to this. The reduced performance might be due to the QMIX policy's necessity for extensive training to learn complex service placement strategies.

On the other hand, the random policy demonstrated significant deficiency compared to the QMIX and greedy policies. This outcome was expected, given that the random policy lacks strategic selection methods to handle such complex situations efficiently.

In a situation where edge server node resources become scarce and the data volume of services increases, the service completion rates and resource utilization rates of the three algorithms across the five nodes are compared.

When there are 5 edge server nodes, 10 vehicles, 10 services, CPU resources for computing nodes are within between 5 and 10, and service data volume is within between 600 and 900, the resource utilization and service completion rates of the three algorithms are compared across all computing nodes in Figures 9 and 10:

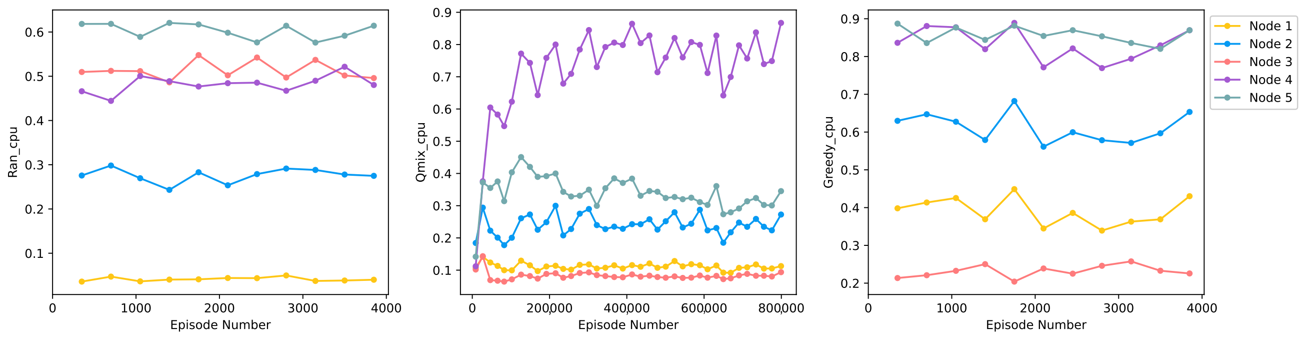

Figure 9. The corresponding resource utilization rates from left to right are those of the random policy, QMIX policy, and greedy policy.

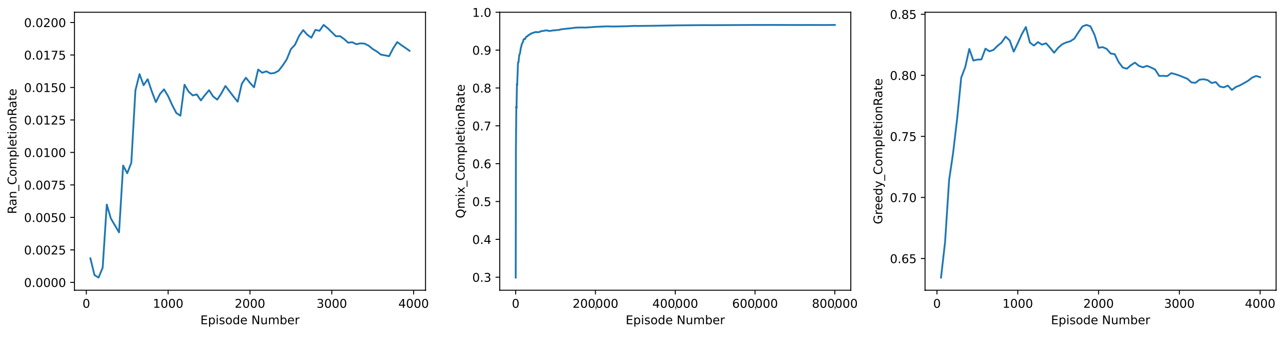

Figure 10. The corresponding service completion rates from left to right are those of the random policy, QMIX policy, and greedy policy.

The random policy proves to be the least effective of the three. It is notable for its relatively low service completion rate, which could be attributed to its lack of strategic resource allocation. The resources are randomly allocated, which runs the risk of inefficient utilization and lower service completion rates.

Contrarily, the QMIX policy, a value-based multi-agent reinforcement learning algorithm, delivers a noteworthy performance. The high service completion rate can be attributed to its learning ability to coordinate among multiple agents. In this case, the vehicles can be seen as agents and they learn to make optimal decisions regarding resource allocation and service deployment. This leads to a higher service completion rate, although the resource utilization rate is not uniformly distributed among the nodes. The reasons for nodes 1 and 3 having close-to-zero resource utilization under the QMIX policy needs further investigation. One possible reason could be that the QMIX policy tends to concentrate resources on a few nodes (like node 4 in this case), leading to higher overall service completion rate, but at the cost of lower utilization of other nodes.

The greedy policy strikes a balance between resource utilization and service completion rate. The significantly high resource utilization of nodes 4 and 5 indicates a tendency to assign tasks to those nodes that currently have the highest available resources, thus abiding by its 'greedy' nature. This likely explains its high service completion rate at the initial stage (episode number = 2000). However, the decline in the service completion rate over the course of the experiment could suggest a need for more sophisticated resource allocation and service deployment strategies that account for future variations in resource availability and service demands.

In summary, while all three policies have their own advantages and pain points, the QMIX algorithm appears to provide the highest service completion rate, and considering the increasing data volume and resource constraints, a trade-off must be made. Further investigations may be conducted for enhancing the performance of these algorithms, especially in terms of achieving a balanced resource utilization across the nodes while maintaining high service completion rates.

6. Conclusions

The emergence of edge computing has made it possible to realize time-sensitive, resource-intensive scenarios, such as target detection in edge intelligence. Service placement within edge computing plays a significant role in determining the overall quality of such systems. Considering the complex coupling relationships between service placement decisions and the hierarchical edge computing load, especially in multi-service situations, this study employs the multi-agent reinforcement learning algorithm QMIX to optimize intelligent service scheduling and placement.

In scenarios featuring service mobility, facilitating stable and continuous edge computing services inevitably results in service migration. We propose the use of discrete time-varying graphs to capture the temporality of services and combining these graphs with reinforcement learning algorithms to better leverage the coupling relationships between services and migration tendencies. This approach leads to a more optimized service scheduling and placement strategy, enabling a more efficient and reliable edge computing system.

Looking to the future, we realize the scope of the extension of our research in varied dimensions. One area could involve improving the QMIX algorithm or proposing a new multi-agent reinforcement learning algorithm that offers better performance in particular settings. In addition, we could consider different migration policies and their impact on service scheduling and placement. Moreover, we anticipate investigating other service placement strategies that entail the consideration of additional parameters like energy consumption, latency, and cost. The inclusion of more dynamic and complex scenarios, like fluctuating network conditions or user mobility, can also provide a valuable direction for future work.

Author Contributions: The research idea was conceived collaboratively by C.C., B.C., J.Q. and Y.X., who envisioned the scope, significance, and direction of the study. The conducting of experiments, involving data collection, the running of simulations, and the testing of methods, was handled by M.L. and Y.Y., while the analysis of the results was undertaken collectively by C.C., B.C., J.Q. and Y.X. The original manuscript was written by M.L., who documented the rationale, methodology, analysis, and conclusions of the study. The manuscript was subsequently checked and revised by C.C., B.C., J.Q. and Y.X., who provided critical feedback and contributed to refining the content and enhancing the academic writing. All authors have read and approved the final version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: This study did not require ethical approval.

Data Availability Statement: Not applicable.

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

References

1. He, K.; Gkioxari, G.; Dollár, P.; Girshick, R. Mask r-cnn. In Proceedings of the IEEE International Conference on Computer Vision, Venice, Italy, 22–29 October 2017; pp. 2961–2969.
2. Huang, R.; Pedoeem, J.; Chen, C. YOLO-LITE: A real-time object detection algorithm optimized for non-GPU computers. In Proceedings of the 2018 IEEE International Conference on Big Data (Big Data), Seattle, WA, USA, 10–13 December 2018; pp. 2503–2510.
3. Liu, W.; Anguelov, D.; Erhan, D.; Szegedy, C.; Reed, S.; Fu, C.Y.; Berg, A.C. SSD: Single Shot Multibox Detector. In Proceedings of the Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, 11–14 October 2016; pp. 21–37.
4. Lang, A.H.; Vora, S.; Caesar, H.; Zhou, L.; Yang, J.; Beijbom, O. PointPillars: Fast Encoders for Object Detection from Point Clouds. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, Long Beach, CA, USA, 15–20 June 2019; pp. 12697–12705.
5. Zhou, Y.; Tuzel, O. VoxelNet: End-to-End Learning for Point Cloud Based 3D Object Detection. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Salt Lake City, UT, USA, 18–23 June 2018; pp. 4490–4499.
6. Hu, Y.; Fang, S.; Lei, Z.; Zhong, Y.; Chen, S. Where2comm: Communication-Efficient Collaborative Perception via Spatial Confidence Maps. *Adv. Neural Inf. Process. Syst.* **2022**, *35*, 4874–4886.
7. Xu, R.; Xiang, H.; Tu, Z.; Xia, X.; Yang, M.; Ma, J. V2X-ViT: Vehicle-to-Everything Cooperative Perception with Vision Transformer. In *European Conference on Computer Vision*; Springer: Berlin/Heidelberg, Germany, 2022; pp. 107–124.
8. Mehr, E.; Jourdan, A.; Thome, N.; Cord, M.; Guitteny, V. DiscoNet: Shapes Learning on Disconnected Manifolds for 3D Editing. In Proceedings of the IEEE/CVF International Conference on Computer Vision, Seoul, Republic of Korea, 27 October–2 November 2019; pp. 3474–3483.
9. Cohen, T.S.; Geiger, M.; Köhler, J.; Welling, M. Spherical CNNs. *arXiv* **2018**, arXiv:1801.10130.
10. Jiang, H.; Learned-Miller, E. Face Detection with the Faster R-CNN. In Proceedings of the 2017 12th IEEE International Conference on Automatic Face & Gesture Recognition (FG 2017), Washington, DC, USA, 30 May–3 June 2017; pp. 650–657.
11. Chen, L.-C.; Papandreou, G.; Kokkinos, I.; Murphy, K.; Yuille, A.L. Deeplab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs. *IEEE Trans. Pattern Anal. Mach. Intell.* **2017**, *40*, 834–848.
12. Siddique, N.; Paheding, S.; Elkin, C.P.; Devabhaktuni, V. U-net and its Variants for Medical Image Segmentation: A Review of Theory and Applications. *IEEE Access* **2021**, *9*, 82031–82057.
13. Qi, C.R.; Su, H.; Mo, K.; Guibas, L.J. PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Honolulu, HI, USA, 21–26 July 2017; pp. 652–660.
14. Bailey, T.; Durrant-Whyte, H. Simultaneous localization and mapping (SLAM): Part II. *IEEE Robot. Autom. Mag.* **2006**, *13*, 108–117.
15. Mur-Artal, R.; Montiel, J.M.M.; Tardos, J.D. ORB-SLAM: A Versatile and Accurate Monocular SLAM System. *IEEE Trans. Robot.* **2015**, *31*, 1147–1163.
16. Kenney, J.B. Dedicated Short-Range Communications (DSRC) Standards in the United States. *Proc. IEEE* **2011**, *99*, 1162–1182. [CrossRef]
17. Abboud, K.; Omar, H.A.; Zhuang, W. Interworking of DSRC and Cellular Network Technologies for V2X Communications: A Survey. *IEEE Trans. Veh. Technol.* **2016**, *65*, 9457–9470.
18. Famili, A.; Shen, W.-M.; Weber, R.; Simoudis, E. Data Preprocessing and Intelligent Data Analysis. *Intell. Data Anal.* **1997**, *1*, 3–23. [CrossRef]
19. Dawkins, C.; Srinivasan, T.N.; Whalley, J. Calibration. In *Handbook of Econometrics*; Elsevier: Amsterdam, The Netherlands, 2001; Volume 5, pp. 3653–3703.
20. Guyon, I.; Elisseeff, A. An Introduction to Feature Extraction. In *Feature Extraction: Foundations and Applications*; Springer: Berlin/Heidelberg, Germany, 2006; pp. 1–25.
21. Chen, R.; Hua, Q.; Chang, Y.-S.; Wang, B.; Zhang, L.; Kong, X. A Survey of Collaborative Filtering-Based Recommender Systems: From Traditional Methods to Hybrid Methods Based on Social Networks. *IEEE Access* **2018**, *6*, 64301–64320.

22. Montavon, G.; Samek, W.; Müller, K.-R. Methods for Interpreting and Understanding Deep Neural Networks. *Digit. Signal Process.* **2018**, *73*, 1–15. [CrossRef]
23. Ellison, A.M. Bayesian Inference in Ecology. *Ecol. Lett.* **2004**, *7*, 509–520. [CrossRef]
24. Yager, R.R.; Filev, D.P. Induced Ordered Weighted Averaging Operators. *IEEE Trans. Syst. Man Cybern. Part B Cybern.* **1999**, *29*, 141–150. [CrossRef] [PubMed]
25. Zadeh, L.A. Fuzzy Logic. *Computer* **1988**, *21*, 83–93. [CrossRef]
26. Sagi, O.; Rokach, L. Ensemble Learning: A Survey. *Wiley Interdiscip. Rev. Data Min. Knowl. Discov.* **2018**, *8*, e1249. [CrossRef]
27. Wagner, J.; Fischer, V.; Herman, M.; Behnke, S. Multispectral Pedestrian Detection using Deep Fusion Convolutional Neural Networks. *ESANN* **2016**, *587*, 509–514.
28. Harary, F.; Gupta, G. Dynamic Graph Models. *Math. Comput. Model.* **1997**, *25*, 79–87. [CrossRef]
29. Lee, D.; Seung, H.S. Algorithms for non-negative matrix factorization. *Adv. Neural Inf. Process. Syst.* **2000**, *13*, 1–7.
30. Huang, B.; Liu, X.; Wang, S.; Pan, L.; Chang, V. Multi-agent reinforcement learning for cost-aware collaborative task execution in energy-harvesting D2D networks. *Comput. Netw.* **2021**, *195*, 108176. [CrossRef]
31. Chen, C.; Xiang, H.; Qiu, T.; Wang, C.; Zhou, Y.; Chang, V. A rear-end collision prediction scheme based on deep learning in the Internet of Vehicles. *J. Parallel Distrib. Comput.* **2018**, *117*, 192–204. [CrossRef]
32. Wang, S.; Xiang, J.; Zhong, Y.; Zhou, Y. Convolutional neural network-based hidden Markov models for rolling element bearing fault identification. *Knowl.-Based Syst.* **2018**, *144*, 65–76. [CrossRef]
33. Wu, X.; Chen, J.; Xie, L.; Chan, L.L.T.; Chen, C.I. Development of convolutional neural network based Gaussian process regression to construct a novel probabilistic virtual metrology in multi-stage semiconductor processes. *Control Eng. Pract.* **2020**, *96*, 104262. [CrossRef]
34. Chen, Z.-X.; Zhao, M.; Hou, L.-P.; Zhang, X.-Q.; Li, B.-Q.; Huang, J.-Q. Toward Practical High-Energy-Density Lithium–Sulfur Pouch Cells: A Review. *Adv. Mater.* **2022**, *34*, 2201555. [CrossRef] [PubMed]
35. Tan, Y.; Wang, K.; Yang, Y.; Zhou, M.-T. Delay-optimal task offloading for dynamic fog networks. In Proceedings of the ICC 2019–2019 IEEE International Conference on Communications (ICC), Shanghai, China, 20–24 May 2019; pp. 1–6.
36. Chen, X.; Wu, C.; Liu, Z.; Zhang, N.; Ji, Y. Computation offloading in beyond 5G networks: A distributed learning framework and applications. *IEEE Wirel. Commun.* **2021**, *28*, 56–62. [CrossRef]
37. He, Y.; Zhao, N.; Yin, H. Integrated networking, caching, and computing for connected vehicles: A deep reinforcement learning approach. *IEEE Trans. Veh. Technol.* **2017**, *67*, 44–55. [CrossRef]
38. Ren, Y.; Guo, A.; Song, C.; Xing, Y. Dynamic resource allocation scheme and deep deterministic policy gradient-based mobile edge computing slices system. *IEEE Access* **2021**, *9*, 86062–86073. [CrossRef]
39. Wang, Y.; Liu, H.; Zheng, W.; Xia, Y.; Li, Y.; Chen, P.; Guo, K.; Xie, H. Multi-objective workflow scheduling with deep-Q-network-based multi-agent reinforcement learning. *IEEE Access* **2019**, *7*, 39974–39982. [CrossRef]
40. Chen, X.; Zhang, H.; Wu, C.; Mao, S.; Ji, Y.; Bennis, M. Optimized computation offloading performance in virtual edge computing systems via deep reinforcement learning. *IEEE Internet Things J.* **2018**, *6*, 4005–4018. [CrossRef]
41. Buzachis, A.; Celesti, A.; Galletta, A.; Fazio, M.; Fortino, G.; Villari, M. A multi-agent autonomous intersection management (MA-AIM) system for smart cities leveraging edge-of-things and Blockchain. *Inf. Sci.* **2020**, *522*, 148–163. [CrossRef]
42. Buzachis, A.; Celesti, A.; Galletta, A.; Fazio, M.; Villari, M. A secure and dependable multi-agent autonomous intersection management (MA-AIM) system leveraging blockchain facilities. In Proceedings of the 2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion (UCC Companion), Zurich, Switzerland, 17–20 December 2018; pp. 226–231.
43. Filocamo, B.; Galletta, A.; Fazio, M.; Ruiz, J.A.; Sotelo, M.Á; Villari, M. An innovative osmotic computing framework for self adapting city traffic in autonomous vehicle environment. In Proceedings of the 2018 IEEE Symposium on Computers and Communications (ISCC), Natal, Brazil, 25–28 June 2018; pp. 01267–01270.
44. Rashid, T.; Samvelyan, M.; De Witt, C.S.; Farquhar, G.; Foerster, J.; Whiteson, S. Monotonic value function factorisation for deep multi-agent reinforcement learning. *J. Mach. Learn. Res.* **2020**, *21*, 7234–7284.