Wireless Personal Communications
https://doi.org/10.1007/s11277-024-11093-4

Adaptive Methods for Revenue Model Learning of a Slice Broker in the Presence of Adversaries

Muhidul Islam Khan[1] · Gianfranco Nencioni[1]

Accepted: 8 April 2024
© The Author(s) 2024

Abstract
In the fifth-generation (5G) of mobile networks, Multi-Access Edge Computing (MEC) refers to the deployment of computing resources closer to the end-users for improved service delivery. In the context of 5G MEC, the slice broker plays a crucial role in managing the allocation of resources among the different network slices, which are logical networks on top of a shared infrastructure. The slice broker is a business entity that acts as an intermediary between the slice tenants and the infrastructure provider and is responsible for allocating resources (such as CPU, memory, and network bandwidth) required to set up the network. The slice broker must ensure that resources are allocated in a way that the revenue is maximized. In a dynamic environment, the slice broker must learn the revenue model adaptively and online. Adversaries can significantly reduce the revenue by misleading the system about the resources pretending to be selfish nodes, or creating noise. The slice broker should learn the revenue model in the presence of adversaries. We apply cooperative deep reinforcement learning with consensus mechanism and consensus deep learning to learn the revenue model adaptively. We also compare our proposed methods with the reference solution. Simulation results show that our proposed methods, especially the cooperative version, outperform the reference solution.

Keywords 5G MEC · Slice broker · Reinforcement learning · Consensus algorithm

1 Introduction

In the fifth-generation (5G) of mobile networks, Multi-Access Edge Computing (MEC) is a technology that enables the deployment of computing resources closer to the end-users, thereby improving the quality of service and user experience [1]. In the supply chain of 5G MEC, the three main components are the slice tenants, Infrastructure Providers (InPs), and the slice broker. InPs provide the physical infrastructure needed to support the MEC

✉ Muhidul Islam Khan
md.m.khan@uis.no

Gianfranco Nencioni
gianfrano.nencioni@uis.no

1 Department of Electrical Engineering and Computer Science, University of Stavanger, Stavanger, Norway

services, such as servers, network equipment, and storage devices. The slice tenants are users or organizations requesting different MEC services. The slice broker is instead a new business entity that buys the resources from InPs and creates and manages the slices, which are bought by the slice tenants. Slice brokers need to consider the economic aspects for dynamic resource allocation. One crucial aspect of this resource allocation is considering the economic aspects [2]. To learn the proper revenue model, it is necessary to dynamically allocate computing, storage, and network resources to different network slices to ensure efficient utilization of available resources and high-quality service delivery. However, in the learning process, adversaries such as selfish nodes and malicious users can threaten the system's security and efficiency [3]. Selfish nodes may try to exploit the system by consuming more resources than necessary, leading to inefficient resource utilization and degradation of the system's overall performance. On the other hand, malicious users may add noise to the system, causing interference and affecting the quality of service. So, we need a dynamic method that can adaptively learn about the revenue model in the presence of adversaries and noises.

In [4], the authors propose a heuristic approach to minimize the cost of the slice brokers in 5G MEC. They address the slice allocation problem. However, their method is not adaptive. There is no dynamic demand/request and no revenue model learning. In [5], the authors propose a novel framework for profit-driven network slicing and resource allocation in MEC systems. The proposed framework enables service providers to slice the network and allocate resources based on the profit generated by different services. However, their method is not adaptive, they do not have dynamic requests, and they do not learn the revenue model adaptively.

Security is a significant concern to consider for resource allocation. In recent years, some works have been based on resource allocation, considering security concerns. In [6], the authors propose a secure slice allocation in 5G. They consider device authentication using a password-based essential derivation function. In addition, they consider the massive traffic in their scenario. They propose a distributed Reinforcement Learning (RL) based method for Distributed Denial-of-Service (DDoS) attackers. However, their proposed method do not consider the revenue model learning adaptively. In [7], the authors propose a cooperative reinforcement learning method for revenue model learning for a slice broker in the 5G-MEC system in the presence of adversaries. This paper extends the work [7] by proposing another adaptive method and perform evaluation with the existing one and also with the reference solution.

In summary, the contributions of our paper are as follows:

- We consider a dynamic demand model based on the set prices by the agents. Demand varies based on the price. With the increment of the price, the demand decreases and vice versa.
- We consider two types of security issues: selfish nodes and added noises for resource allocation for slices. We also learn the revenue model considering these adversaries.
- We consider the consensus mechanism for adaptively learning the security threats for revenue model learning. We apply two variants of deep learning mechanisms with consensus. One with the cooperation among agents and another one to send information to the server and the server collaborates with the agents.
- We compare our proposed methods with the reference solution.

The paper is structured as follows. Section 2 provides some related works. Section 3 describes the problem of maximizing revenue in the presence of adversaries. Section 4 introduces the proposed method to solve the presented problem. Section 5 presents the results of comparing the proposed method with reference methods. Finally, Sect. 6 concludes the paper.

2 Related Works

Some current research works focus on resource allocation for slice brokers in 5G considering the profit level of the broker. Some works also consider adaptive methods, e.g., RL, Deep RL, and Artificial Intelligence (AI) based methods. In [8], the authors propose a distributed slice allocation using the adaptive machine learning method. Their proposed method helps to achieve the requirements for multiple information flows. However, they do not consider learning the revenue model for the broker, and also, there is no consideration of security issues for resource allocation. In [9], the slice allocation mechanism is proposed where dynamic slice admission is gained, and at the same time, the operator's profit is investigated. They apply Lyapunov optimization for the optimization. However, there is no revenue model learning and no consideration of security issues in this work. In [10], the authors propose an auction-based method for radio resource allocation in 5G. However, their proposed method is not adaptive, and also they do not consider the security threats to resource allocation. In [11], the authors propose a method named RL-NSB (RL-based Network Slice Broker), where the slice broker's profit is maximized adaptively by charging customers for the allocated resources while minimizing the cost of providing those resources. However, their proposed method is not learning any revenue model. They also do not consider the security issues for resource allocation. In [12], the authors propose RL and Deep RL for learning the admission policy that optimizes the profit of the slice broker. In [13], the authors propose an innovative approach to optimizing revenue in 5G Cloud Radio Access Networks (C-RANs). Using multi-agent deep reinforcement learning, the authors propose a dynamic system for network slicing and admission control. This method enables efficient resource allocation and service provisioning, ultimately maximizing profit-making opportunities for network operators. By applying advanced AI techniques, the paper demonstrates how 5G C-RANs can adapt and thrive in a competitive, revenue-driven environment, making it a valuable contribution to the telecommunications industry. In [14], the authors introduce a novel resource allocation strategy for 5G networks, focusing on maximizing profit for slice brokers. Leveraging reinforcement learning, the paper presents a bi-level optimization framework that dynamically allocates resources across multiple network slices. This approach ensures efficient resource utilization and adapts to changing network conditions, enhancing revenue generation potential for slice brokers. By optimizing both short-term gains and long-term profitability, this research offers a valuable strategy to support profit-making endeavors in the competitive 5G market, making it essential reading for slice brokers and network operators. In [15], the authors present a groundbreaking approach to profit optimization for slice brokers in 5G networks. By employing multi-agent deep reinforcement learning, the research introduces a coordinated system for dynamic resource slicing and admission control. This strategy maximizes the broker's profit potential by intelligently allocating resources, ensuring efficient service delivery, and adapting to evolving network conditions. With a focus on revenue generation and cost management, this paper offers a pivotal framework for slice brokers aiming to thrive in the competitive 5G landscape, making it essential reading for profit-making strategies in this domain.

In contrast to all these approaches, our proposed methods adaptively learn the revenue model considering some selfish nodes and noisy nodes in the system. To our best knowledge, this is the first work which considers to learn the revenue model adaptively with security attacks.

3 Problem Description

We consider three business entities, e.g., InPs, a slice broker, and a slice tenant, in the 5G-MEC system.

The assumption is that each InP has one MEC host (MEH). The problem can be generalized to multiple MEHs belonging to the same InP. We keep the system simple with one MEH as multiple MEHs need orchestration by a Multi-access Edge Orchestrator (MEO).

A MEH is a computing platform with computational resources in the system, e.g., processing power in vCPU.

The business entity slice broker buys the computational resources from the InPs, which resells the resources to multiple slice tenants. For the sake of simplicity, we assume only one tenant but the problem and the solution can be generalized to having multiple tenants.

We also assume that the slice broker has already bought an amount of computational resources from each InP. Therefore, in each MEH, the broker has one *chunk* of computational resources. The set of chunks is denoted as $\mathcal{M}$. For each chunk $m \in \mathcal{M}$, the amount of bought computational resources is denoted as μ_m. The tenant dynamically requests to the broker an amount of computational resources that we call *slice demand* and indicate as d^t, where t identifies the *time interval*.

At each time interval t, the broker decides the amount of resources from each chunk, which will be allocated to fill the slice demand. The portion of chunk allocated to the tenant is denoted as *subchunk*. The subchunk's computational resources from the chunk $m \in \mathcal{M}$ at the time interval t is denoted as δ^t_m. The broker also decides the *subchunk price* (in €/vCPU), denoted as c^t_m.

In our system, we consider that the broker has one software agent for each chunk $m \in \mathcal{M}$ helping to decide the amount of computational resources, δ^t_m, and price, c^t_m, for the related subchunk at each time interval t. Figure 1 represents the investigated 5G-MEC system. The adversary and noise injections will be explained in the second part of this section.

We use a practical demand model to get the slice demand of the slice tenant, d^t. In the real world, demand changes over time and depends not only on the current price but can also be impacted by the magnitude of recent price changes. A price decrease can create a temporary demand increment, wherever a price increase can cause a fall in demand. The impact of price changes can also be asymmetric, so price increases have a much more significant or smaller impact than decreases.

In our case, at every time interval, the slice demand depends on the weighted average price, which is computed from the prices of each subchunk as follows.

$$p^t = \frac{1}{d^t} \sum_{m \in \mathcal{M}} \delta^t_m \cdot c^t_m \tag{1}$$

Based on the price-demand function in [16], we compute the slice demand for the next time interval $t + 1$ as follows.

$$d^{t+1} = d^0 - k \cdot p^t - a \cdot s((p^t - p^{t-1})^+) + b \cdot s((p^t - p^{t-1})^-), \tag{2}$$

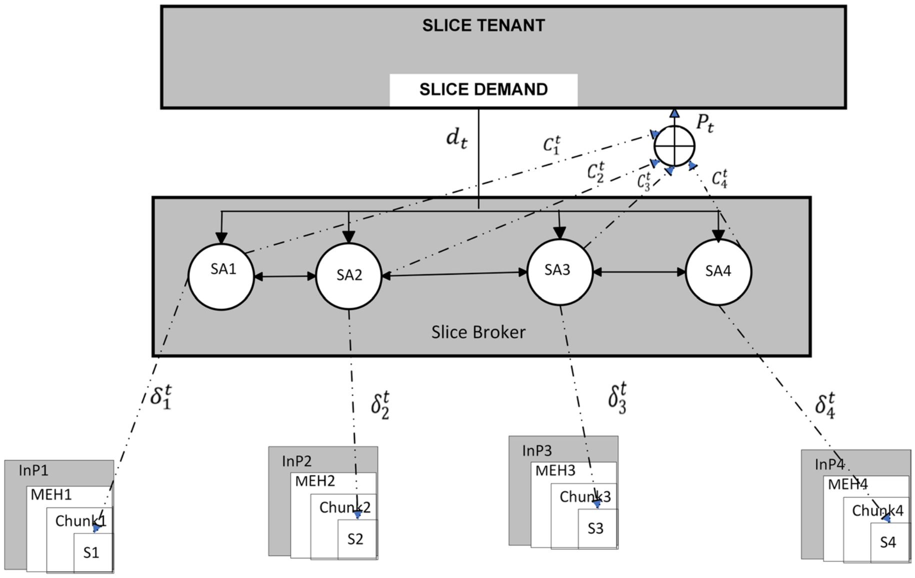

Fig. 1 5G-MEC system under consideration for deep learning with consensus mechanism

where

$$(p^t - p^{t-1})^+ = \begin{cases} p^t - p^{t-1}, & \text{if } p^t > p^{t-1} \\ 0, & \text{otherwise} \end{cases},$$

$$(p^t - p^{t-1})^- = \begin{cases} p^t - p^{t-1}, & \text{if } p^t < p^{t-1} \\ 0, & \text{otherwise} \end{cases},$$

and where p^t is the price for the current time interval t and p^{t-1} is the price for the previous time interval. The first two terms of Eq. (2) correspond to a linear demand model with intercepting d^0 and slope k. The second two terms model the response to a price change between two intervals. Coefficients a and b define the sensitivity to positive and negative price changes, respectively, and s is a shock function that can be used to specify a non-linear dependency between the price change and demand. We assume $s(\cdot) = \sqrt{\cdot}$. The reason for choosing this type of demand function is to make it more realistic based on prices. The problem of maximizing the revenue of the slice brokers to serve the slice demand at the time interval t can be formulated by basing on Bertnard model [17]:

$$P : \max \Phi^t = \max \sum_{m \in \mathcal{M}} c_m^t \cdot \delta_m^t, \tag{3}$$

subject to

$$\begin{aligned} C1 &: \delta_m^t \le \eta_m \qquad \forall m \in \mathcal{M}, \\ C2 &: \sum_{m \in \mathcal{M}} \delta_m^t \le d^t, \end{aligned} \tag{4}$$

Where Φ^t is the defined revenue function and the objective function of the problem, $C1$ is the constraint that limits the size of the subchunk to the size of the related chunk, and $C2$ is the constraint that limits the cumulative size of all subchunks to the slice demand.

3.1 Environment

Our proposed method is based on RL. In RL, the agents learn over time intervals by performing a particular action, and it shifts from one state to another. After performing an action, the agents receive a reward for the performed action. RL does not need any knowledge about the environment before. The agent learns over the previous experiences and perform the action in future which provided better perfromance at a particular state. The environment of the RL determines the states, actions, and reward function [18]. In our case, the states are the slice demand at time t, d^t, the size of the subchunks at time interval $t-1$, δ_m^{t-1} and all the previous prices of the subchunk m, c_m^τ $\forall \tau \in [1, t-1]$. The actions are the size and the price of the subchunks m at t, δ_m^t, and c_m^t, respectively. The reward function denotes the revenue for allocating the subchunk m at the time interval t.

$$r_m^t = c_m^t \cdot \delta_m^t \tag{5}$$

3.2 Presence of Adversarial Agents

According to the European Network and Information Security Agency (ENISA) 5G threat landscape [19], one of the potential threats related to 5G MEC is the compromised supply chain (i.e., vendor and service providers). Since the tampering of network products (creating adversaries and added noises) can result in service unavailability, information destruction, or misinformation generation. For example, in Fig. 2, the attacker compromises the software agents SA1 and SA2, making them selfish or uncooperative. In this case, SA1 and SA2 will try to maximize their reward without cooperation, eventually causing less revenue for the broker.

Let $\mathcal{M}^+$ and $\mathcal{M}^-$ denote the set of cooperative agents and adversaries, respectively, where $\mathcal{M} = \mathcal{M}^+ \cup \mathcal{M}^-$. Since we have assumed that there is one agent per chunk (and subchunk), we use $\mathcal{M}$ to interchangeably indicate the set of agents or the set of chunks.

The objective of agents $m \in \mathcal{M}^+$ is to maximize a team-average objective function given as follows.

$$\max_{c_m^t, \delta_m^t} J^+ = \max_{c_m^t, \delta_m^t} \mathbb{E}\left[\sum_{\tau=0}^{\infty} \frac{1}{\mathcal{M}} \sum_{m \in \mathcal{M}} \gamma_m \cdot r_m^\tau \right] \tag{6}$$

where τ is the time under investigation, γ_m is a discounted factor, has a value between 0 and 1, and indicates how much the RL agents care about rewards in the distant future concerning those in the immediate future.

The cooperative agents are unaware of the presence of an adversarial agent that seeks to maximize a different objective function. We define the objective function for $m \in \mathcal{M}^-$ as follows.

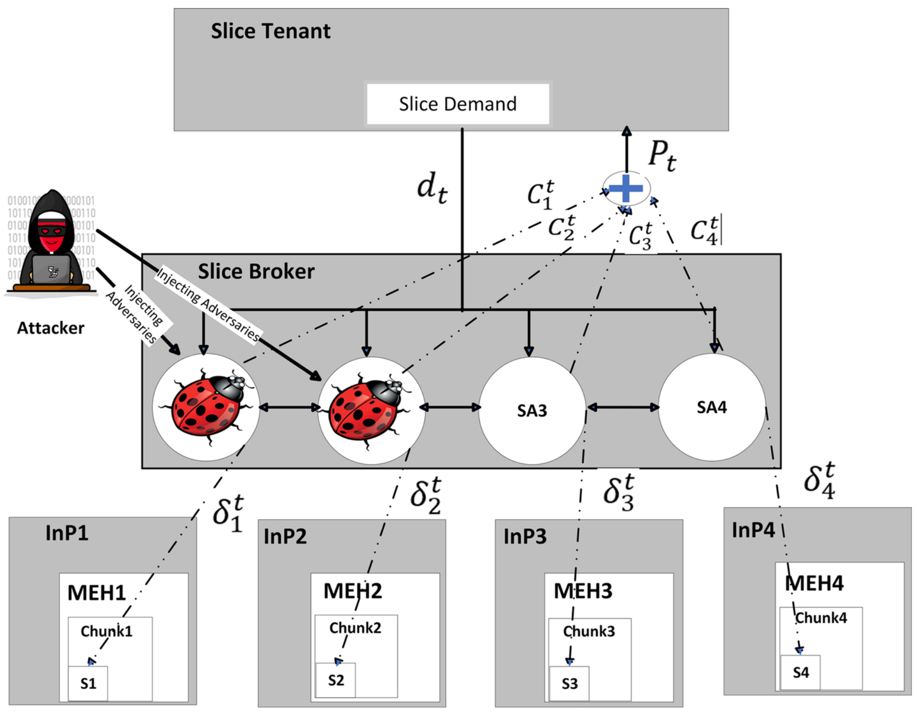

Fig. 2 5G-MEC system with the adversaries

$$\max_{c_m^t, \delta_m^t} J^- = \max_{c_m^t, \delta_m^t} \mathbb{E}\left[\sum_{\tau=0}^{\infty} \gamma_m \cdot r_m^\tau\right] \tag{7}$$

It is important to note that the adversarial agent can compromise the rewards $r_m^\tau, m \in \mathcal{M}^-$, to incentivize its malicious behavior [20]. Furthermore, once the agents establish communication, the adversary can spread false information about the entire network's performance embedded in the compromised rewards r_m^τ. This may eventually lead to incentivizing bad behavior in the cooperative agents.

3.3 Presence of Noisy Agents

An attacker may try injecting noise into any component, e.g., a software agent, to compromise the system. For example, in Fig. 3, the attacker compromises the software agent SA1 and SA2. Noise may hamper the cooperation among agents in a way that the broker may have an idea that it has enough resources to allocate or sometimes get the knowledge of not allocating the resources, which is harmful to the revenue-making of a broker. We consider a few agents with noise. Here, we consider Additive White Gaussian Noise (AWGN). The noise can be normally distributed as follows:

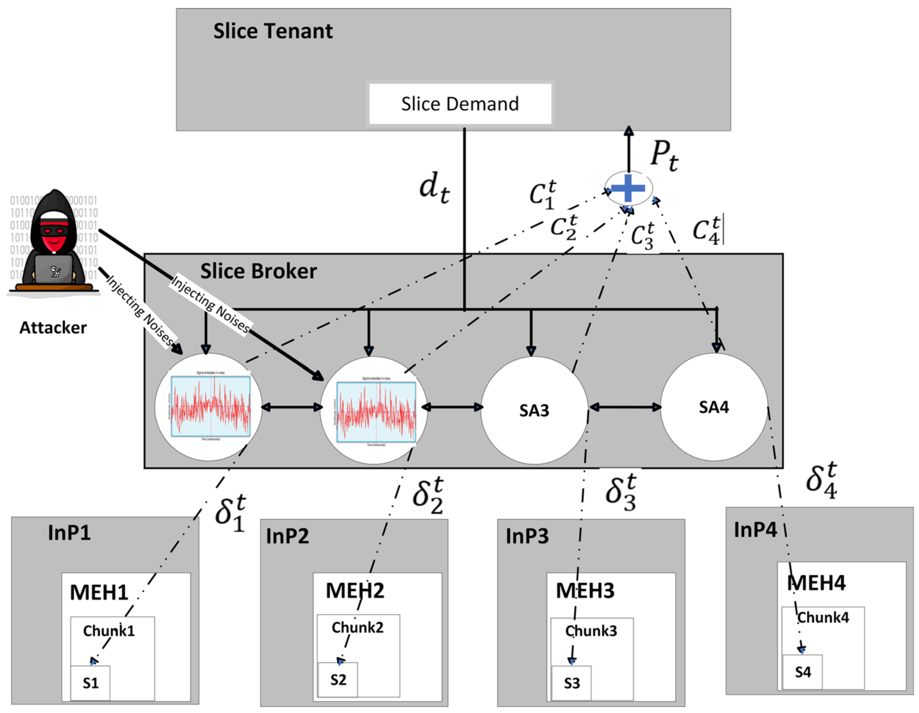

Fig. 3 5G-MEC system with the noises

$$N_m \sim \mathcal{N}\left(0, \left(\frac{A_m}{3}\right)^2\right) \tag{8}$$

where A_m is the magnitude of the noise.

4 Proposed Methods

4.1 Deep Learning with Consensus Mechanism

We propose a resource allocation method based on a generic Deep Q Network (DQN) algorithm [21] to learn the revenue model. We use the original DQN as it is simple considering other variants of the learning mechanism. We could not apply classical RL, e.g., Q-learning, State-Action-Reward-State-Action (SARSA) learning, as our states are continuous. A learning mechanism like Actor-Critic learning is not applied for its complexities.

RL considers the setup where an agent interacts with the environment in discrete time intervals to learn a reward-maximizing behavior policy. At each time interval t, with a given state s, the agent takes action a according to its policy $\pi(s) \rightarrow a$ and receives the reward r moving to the next state s'. We define our environment considering the RL terms as follows.

The goal of the algorithm is to learn an action policy π that maximizes the total discounted cumulative reward/return earned during the episode of T time intervals:

$$R = \sum_{t=0}^{T} \frac{1}{|\mathcal{M}|} \sum_{m \in \mathcal{M}} \gamma_m \cdot r_m^t \tag{9}$$

Such a policy can be defined if we know a function that estimates the expected return based on the current state and next action, under the assumption that all subsequent actions will also be taken according to the policy:

$$Q^{\pi}(s, a) = \mathbb{E}_{s,a}[R] \tag{10}$$

Assuming that this function (known as the Q-function) is known, the policy can be straightforwardly defined as follows to maximize the return:

$$\pi(s) = \arg\max_{a} Q(s, a) \tag{11}$$

We can combine the above definitions based on the Bellman equation as follows:

$$Q^{\pi}(s, a) = r + \gamma_m \max_{a'} Q(s', a') \tag{12}$$

where s' and a' are the next state and the action taken in that state, respectively. If we estimate the Q-function using an approximator, then the quality of the approximation can be measured using the difference as follows:

$$L(\phi) = \frac{1}{|\mathcal{M}|} \sum_{m \in \mathcal{M}} (y_m - Q_{\phi}(s_m, a_m))^2 \tag{13}$$

This value is called the temporal difference error, the loss function.

Algorithm 1 Proposed method based on deep Q learning

```
1: Parameters and Initialization:
2: $\phi$ - Parameters of the policy network $Q_\phi$
3: $\phi_{targ}$ - Parameters of the target network $Q_{\phi_{targ}}$
4: $\alpha$ - Learning rate
5: $B$ - Batch size
6: $t_u$ - Period of target updates
7: Initialization: $\phi_{targ} = \phi$
8: for t = 1 to $T$ do
9:     Choose the action based on $Q_\phi(s_t, a_t)$
10:    Execute the action and save transition $(s_t, a_t, r_t, s'_t)$ in the buffer
11:    Calculate the Q-value
12:    Consensus Step:
13:    Send the Q-value to the neighbors
14:    Update the Q-value of agent $m$ as follows considering the neighbor's
   impact:
15: $Q_m(t+1) = Q_m(t) + \Gamma \sum_{j \in \mathcal{M}_m^G} (Q_j(t) - Q_m(t))$
16:    End of Consensus Step
17:    Calculate target Q-values for each sample in the batch:
18: $y_m = r_m + \gamma_m \cdot \max_{a'} Q_{\phi_{targ}}(s', a')$
19: where $Q(s, a) = 0$ for last states of the episodes (initial condition)
20:    Calculate the loss:
21: $L(\phi) = \frac{1}{|\mathcal{M}|} \cdot \sum_{m \in \mathcal{M}} (y_m - Q_\phi(s_m, a_m))^2$
22:    Update the network's parameters:
23: $\phi = \phi - \alpha \cdot \gamma_m \cdot L(\phi)$
24:    if $t$ mod $t_u = 0$ then
25:        Update the target network:
26: $\phi_{targ} \leftarrow \phi$
27:    end if
28: end for
```

Algorithm 1 shows the step-by-step procedure about how the system works and training has been done.

Consensus Procedure: An agent i communicates the Q-value to all its neighbors $j \in \mathcal{M}_m^G$. $\mathcal{M}_m^G$ denotes the set of neighbors of agent m. All agents update their Q-values through a linear combination of their own values and the information of neighbors received in the previous step. The procedure can be written as:

$$Q_m(t+1) = Q_m(t) + \Gamma \sum_{j \in \mathcal{M}_m^G} (Q_j(t) - Q_m(t)) \tag{14}$$

So, in other words, each agent updates its state by using the disagreement of states with all its neighbors, scaled by a factor of Γ. Thus the convergence rate of this algorithm depends on the scaling factor used. Convergence is guaranteed as long as the following constraint is met [22]:

$$0 < \Gamma < \frac{1}{d_{max}} \tag{15}$$

where d_{max} denotes the maximum degree among all nodes in the network. The constraint can be fulfilled by realizing an upper bound on the maximum possible neighbors for any node.

4.2 Consensus Deep Learning

We apply another variant of consensus deep learning. Our variant is based on Deep Q learning. Consensus deep learning is one kind of RL where Q values of multiple agents are combined to reach an optimal action in a given state [23]. This method is helpful in case of requirements where each agent will work independently and collaborate to reach a particular decision. In traditional Q learning, the agents learn about the action by calculating the Q values of the state-action pair by trial and error. However, to reach any sub-optimal solution, the agents need to collaborate, and at this point, a consensus is needed. For reaching a consensus, cooperation among the agents is important. For this cooperation, one possible option is cooperation among the neighboring agents, which needs lots of communication efforts and proper synchronization. The other option is to collaborate with a server in the learning mechanism. The server plays a critical role in deep consensus learning by enabling the networks to share their weights, and aggregate their updates, ultimately leading to improved performance and better generalization. Overall, the use of a server is important for achieving consensus and is a powerful tool for training complex and highly accurate models [24].

Algorithm 2 Proposed method based on deep consensus Q learning

```
1: Parameters and Initialization:
2: $E$ - Environment
3: $A$ - Set of Possible Actions
4: $\gamma_m$ - Discount Factor
5: $\alpha$ - Learning Rate
6: $N$ - Number of Consensus Iterations
7: $B$ - Batch Size
8: $T$ - Maximum Number of Episodes
9: for t = 1 to $T$ do
10:     for i= 1 to $B$ do
11:         Choose action based on the epsilon-greedy policy
12:         Execute the action and save transition $(s_t, a_t, r_t, s'_t)$ in the buffer
13:         Sample a minibatch of $B$ transitions from Replay buffer, $R$
14:         Compute Q-targets:
15:         For each transition $(s_t, a_t, r_t, s'_t)$ in minibatch, set target = $r + \gamma_m \cdot$ max$Q_j(s', a')$ where $j! = i$
16:         For each transition $(s_t, a_t, r_t, s'_t)$ not in minibatch, set target = $Q(s_t, a_t)$
17:         Update the $i$-th Q-network using the mean-squared error loss between $Q_i(s_t, a_t)$ and its target
18:         Compute the gradients of the loss with respect to the $Q_i$ network's weights
19:         Update the weights by using the learning rate $\alpha$ and the gradients
20:         for n=1 to $N$ do
21:             Compute the consensus Q-value for each state-action pair $(s_t, a_t)$ by averaging all collected Q-values from all $|\mathcal{M}|$ Q-networks:
22:             Set $Q_{consensus}(t+1) = (1/|\mathcal{M}|) \cdot \sum_{i \in \mathcal{M}}(Q_i(s_t, a_t))$
23:             Update each Q-network by using the mean-squared error loss between $Q_i(s_t, a_t)$ and $Q_{consensus}(s_t, a_t)$
24:             Compute the gradients of the loss with respect to the $i$-th Q-network's weights
25:             Update the weights by using the learning rate $\alpha$ and the gradient
26:             Update the state
27:         end for
28:     end for
29: end for
```

Here, in this proposed method, all software agents send the Q-values to another software agent/server in the broker to calculate the consensus of the mechanism. There is no neighboring factor and cooperation among agents for this algorithm.

Figure 4 shows the considered 5G-MEC system for deep consensus learning. The agents send the gradients of loss to the server. Server sends the updated weights to the

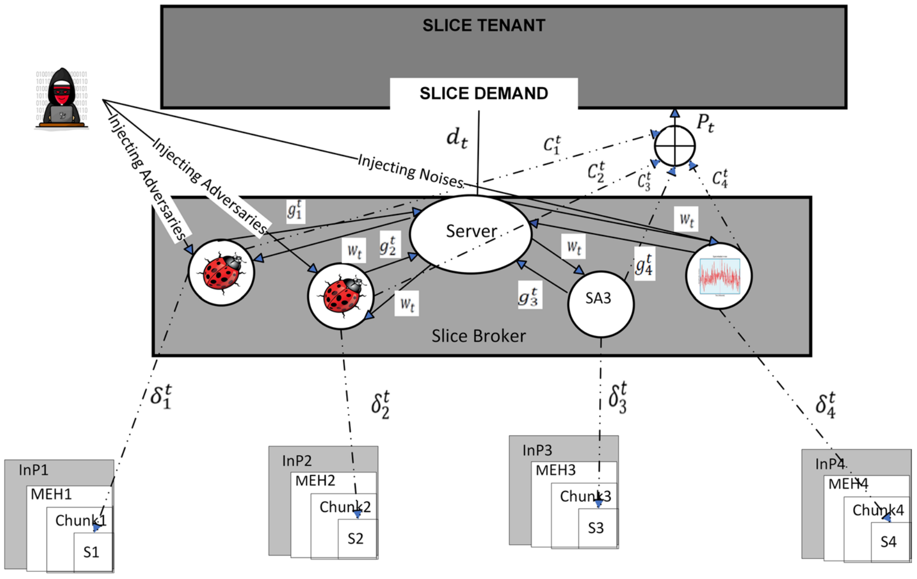

Fig. 4 5G-MEC system for consensus deep learning with adversaries and noises

agents. Instead of having cooperation among themselves like the other method, here agents send information to the server and server also update the agents with weights.

Algorithm 2 defines the proposed method step-by-step.

Here action selection is done based on the epsilon-greedy policy.

The epsilon-greedy policy is one kind of mechanism to balance exploration and exploitation. Exploration means to try an action randomly, whereas exploitation means to exploit based on the learning algorithm. Here, epsilon, ϵ means to try an action randomly.

$$\text{Action at time } t = \begin{cases} \max Q_t(a) & \text{with probability } 1-\epsilon \\ \text{Any action } a(t) & \text{with probability } \epsilon \end{cases}$$

We need to sample a minibatch from the Replay buffer. For each transition, if it is in minibatch, set the Q values as

$$Q(s_t, a_t) = r + \gamma_m \cdot \max_{a'} Q(s', a') \tag{16}$$

We set the Q target as $Q(s_t, a_t)$. Then the loss function is calculated as the Eq. 13.

Consensus Steps: The server in slice broker collects all the Q values of the agents at each time interval and calculates the consensus Q value. The consensus Q values can be calculated as follows:

$$Q_{consensus}(t+1) = (1/|\mathcal{M}|) \cdot \sum_{i \in \mathcal{M}} (Q_i(s_t, a_t)) \tag{17}$$

Agents are informed about the $Q_{consensus}(t+1)$. After that, the gradients of loss is calculated. Agents send the gradients to the server and then the server calculates the weights using the learning rate and the gradients are as follows:

Table 1 Simulation parameters and their values

Parameter	Symbol	Value
Available chunks	$\lvert\mathcal{M}\rvert$	4
Number of agents	$\lvert\mathcal{M}\rvert$	4
Magnitude of the noise	A_m	5
Learning rate	α	0.5
Probability	ϵ	0.5
Number of episodes	T	5000
Batch size	B	512
Period of target updates	T_u	20
Discount factor	γ_m	0.5
Degree of a node	d_{max}	3
Size of the chunks	η_m	5000 vCPU
Neighboring factor	Γ	0.3
Intercept	d_0	5000 vCPU
Slope	k	20 vCPU/€
Response coefficient for price increase	a	300 vCPU/€$^{1/2}$
Response coefficient for price decrease	b	100 vCPU/€$^{1/2}$

$$w_{t+1} = w_t - \alpha + \sum_{t=1}^{T} g_t \tag{18}$$

Where α is the learning rate. g_t is the gradients calculated at time interval t.

Back propagation is used to train the neural network in the learning algorithm for making it more efficient. So, the selection of amount of resources, the calculated revenues, gradients will be in a way that the proper weight will be adjusted and the loss will be minimized [25].

5 Results and Discussion

We consider four software agents. The software agents are connected in cooperative deep learning with consensus mechanism. In consensus deep learning, these software agents communicate with another software agent/server. First, we consider revenue model learning without any adversaries. After that, we consider two software nodes as adversary nodes among the four agents, trying to maximize their revenue. Then we consider two noisy nodes that put noises to provide misinformation about the resources that hampers the learning mechanism. The prices are set by the RL for calculating revenues and the demand. We compare our proposed methods with independent distributed learning as well, which uses classical Q-learning [26].

Table 1 shows the simulation parameters that we consider. Here, the values are considered based on the empirical study.

We apply a deep learning mechanism where we consider 150 hidden nodes with three layers where 50 nodes are considered per layer. We consider Adaptive Moment Estimation

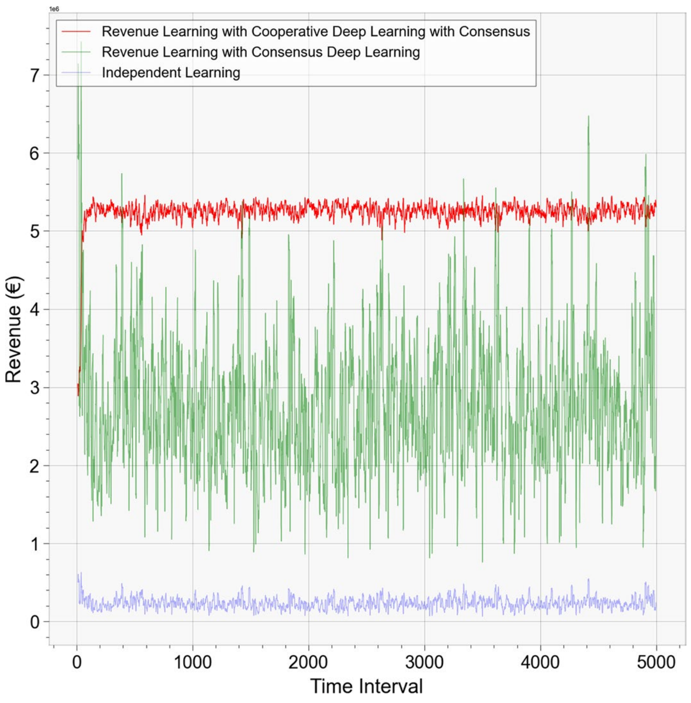

Fig. 5 Revenue over time interval with and without learning adversaries

(Adam) algorithm to optimize the Neural Network (NN) weights. We use the Rectified Linear Unit (ReLU) as an activation function to activate a particular input.

Figure 5 refers to the revenue over time intervals. We can observe that the cooperative deep learning with a consensus mechanism outperforms the other methods. Here we can see that initially, there is a rise, and then saturation comes over the time intervals. We can see there are some variations here for exploration and exploitation. We can observe that in consensus deep learning, where there is no cooperation and huge fluctuations over time intervals. This is for the exploration, exploitation variations, and training time requirements. We can find saturation here, but this method does not outperform cooperative deep learning with a consensus mechanism. For the independent Q learning, we can see the revenue is quite low compared with our proposed methods. This can happen for independent learning as some selfish nodes can maximize their own revenue and get a lower overall revenue.

Figure 6 shows the revenue over time intervals comparing with our learning methods and without learning adversaries. Here, there are selfish nodes in the simulation. We can observe that in our cooperative deep learning method, there is an increment at first, then decrements, and finally, it goes for saturation after 4000 intervals. The massive decrement happens due to the selfish nodes. When our consensus steps help us learn about the adversaries, we can see that learning converges to a level. We can see that the cooperative method outperforms the other method and obviously without learning adversaries. Learning adversaries with cooperative deep learning provides increment at first and then goes down, provides lower performances compared with the cooperative one, and finally

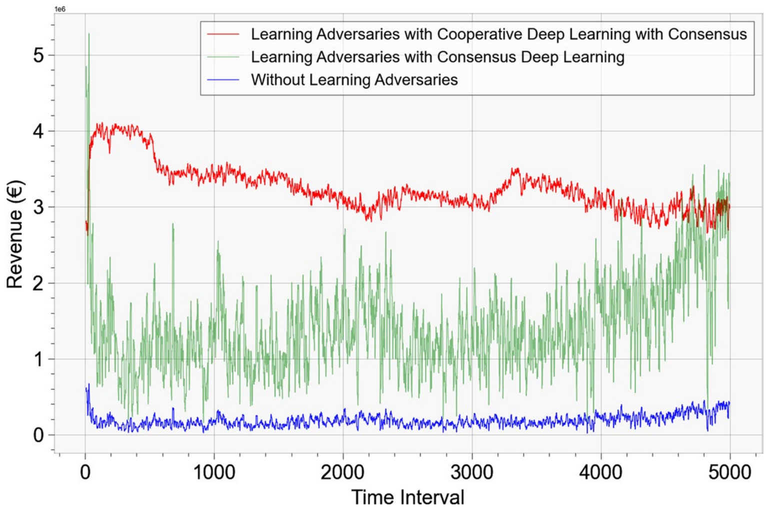

Fig. 6 Revenue over time interval with and without learning adversaries

reaches the cooperative one after having an increment. This increment collects the Q values to a particular agent and then collaboration for learning. On the other hand, without learning adversaries, the revenue goes very low over time intervals.

Figure 7 refers to the revenue over time intervals after adding noisy nodes. The cooperative one for dealing with the noises outperforms the consensus deep learning and the reference solution. We can observe that for noises, the revenue goes up and then massive decrement at the 3000-time interval and again increment after the consensus has been reached then, and adequate convergence happens. Whereas, the consensus deep learning starts with the higher revenue and then again goes down after the 3000 time intervals. So, dealing

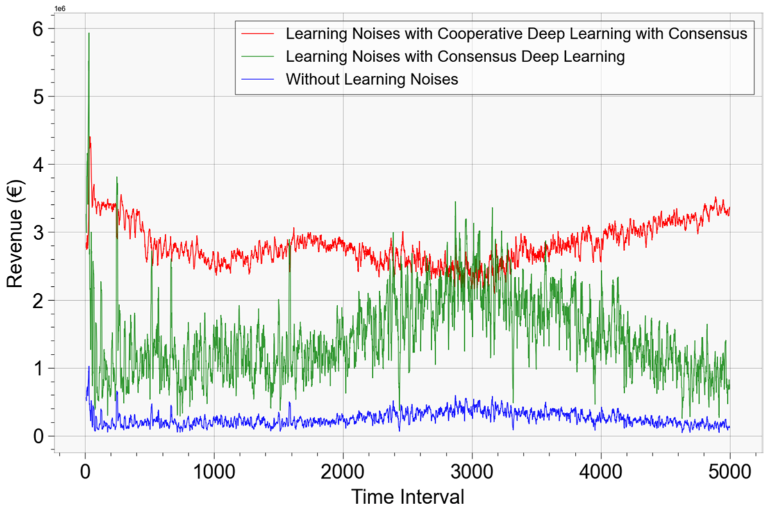

Fig. 7 Revenue over time interval with and without learning noises

with the noises cooperation is helping rather than the consensus deep learning. The overall fluctuations at every time interval are due to the exploration and exploitation and also for training periods. The decrement in the consensus deep learning at the later time intervals is because, in server-based consensus, there may be the presence of some gradients that further mislead the system, and the revenue may go down. Without learning the noises means, without the consensus steps, the learning algorithm provides the lowest performance in terms of revenue.

We can observe that for both types of adversaries, our proposed cooperative deep learning with consensus outperforms consensus deep learning and deep Q learning. We can find that without adversaries, the revenue is more compared with the revenue with adversaries. Furthermore, the revenue with selfish nodes is higher than with noisy nodes.

6 Conclusion

The slice broker plays a vital role as a business entity for the supply chain of 5G MEC. We propose two methods based on a consensus mechanism to adaptively learn a slice broker's revenue model. One method is cooperative, whereas the other sends information to another agent for consensus. The cooperative one outperforms the other variant and the reference solution. Our proposed method can track the adversaries through revenue model learning and get suitable results.

In the future, we plan to consider the faults in the system for learning the revenue model.

Acknowledgements This work was funded by Norwegian Research Council through the 5G-MODaNeI Project (No. 308909).

Funding Open access funding provided by University of Stavanger & Stavanger University Hospital. This work is funded by Norwegian Research Council.

Data Availability There is no dataset available for this project.

Code Availability Custom code implemented by Python.

Declarations

Conflict of interest The authors declare that they have no conflict of interest.

References

1. Liyanage, M., Porambage, P., Ding, A. Y., & Kalla, A. (2021). Driving forces for multi-access edge computing (MEC) IoT integration in 5G. *ICT Express, 7*(2), 127–137.

2. Barakabitze, A. A., Ahmad, A., Mijumbi, R., & Hines, A. (2020). 5G network slicing using SDN and NFV: A survey of taxonomy, architectures and future challenges. *Computer Networks, 167*, 106984.
3. Eziama, E., Ahmed, S., Ahmed, S., Awin, F., & Tepe, K. (2019). Detection of adversary nodes in machine-to-machine communication using machine learning based trust model. In *2019 IEEE international symposium on signal processing and information technology (ISSPIT)* (pp. 1–6). IEEE.
4. Gohar, A., & Nencioni, G. (2021). Minimizing the cost of 5G network slice broker. In *IEEE INFOCOM 2021-IEEE conference on computer communications workshops (INFOCOM WKSHPS)* (pp. 1–6). IEEE.
5. Zanzi, L. (2022). Machine learning-based orchestration solutions for future slicing-enabled mobile networks. PhD thesis, Technische Universität Kaiserslautern.
6. Liu, X. (2020). Spectrum resource allocation in cognitive radio networks. In *Encyclopedia of wireless networks* (pp. 1353–1357). Springer.
7. Khan, M. I., & Nencioni, G. (2023). Revenue-model learning for a slice broker in the presence of adversaries. In *IEEE international conference on advanced networks and telecommunications systems*. IEEE.
8. Venturini, F., Mason, F., Pase, F., Chiariotti, F., Testolin, A., Zanella, A., & Zorzi, M. (2021). Distributed reinforcement learning for flexible and efficient UAV swarm control. *IEEE Transactions on Cognitive Communications and Networking, 7*(3), 955–969.
9. Perveen, A., Patwary, M., & Aneiba, A. (2019). Dynamically reconfigurable slice allocation and admission control within 5G wireless networks. In *2019 IEEE 89th vehicular technology conference (VTC2019-Spring)* (pp. 1–7). IEEE.
10. Benedetto, F., Mastroeni, L., & Quaresima, G. (2021). Auction-based theory for dynamic spectrum access: A review. In *2021 44th international conference on telecommunications and signal processing (TSP)* (pp. 146–151). IEEE.
11. Sciancalepore, V., Costa-Perez, X., & Banchs, A. (2019). RL-NSB: Reinforcement learning-based 5G network slice broker. *IEEE/ACM Transactions on Networking, 27*(4), 1543–1557.
12. Villota-Jacome, W. F., Rendon, O. M. C., & da Fonseca, N. L. (2022). Admission control for 5G core network slicing based on deep reinforcement learning. *IEEE Systems Journal, 16*(3), 4686–4697.
13. Sulaiman, M., Moayyedi, A., Salahuddin, M. A., Boutaba, R., & Saleh, A. (2022). Multi-agent deep reinforcement learning for slicing and admission control in 5G C-RAN. In *NOMS 2022-2022 IEEE/IFIP network operations and management symposium* (pp. 1–9). IEEE.
14. Yu, Z., Gu, F., Liu, H., & Lai, Y. (2023). 5G multi-slices bi-level resource allocation by reinforcement learning. *Mathematics, 11*(3), 760.
15. Sulaiman, M., Moayyedi, A., Ahmadi, M., Salahuddin, M. A., Boutaba, R., & Saleh, A. (2022). Coordinated slicing and admission control using multi-agent deep reinforcement learning. *IEEE Transactions on Network and Service Management*. https://doi.org/10.1109/TNSM.2022.3222589
16. Vishnoi, S. K., Bagga, T., Sharma, A., & Wani, S. N. (2018). Artificial intelligence enabled marketing solutions: A review. *Indian Journal of Economics & Business, 17*(4), 167–177.
17. Sharkey, W. W., & Sibley, D. S. (1993). A Bertrand model of pricing and entry. *Economics Letters, 41*(2), 199–206.
18. Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT Press.
19. Sabella, D., Reznik, A., Nayak, K. R., Lopez, D., Li, F., Kleber, U., Leadbeater, A., Maloor, K., Baskaran, S. B. M., Cominardi, L., et al. (2021). MEC security: Status of standards support and future evolutions. *ETSI White Paper, 46*, 1–26.
20. Figura, M., Kosaraju, K. C., & Gupta, V. (2021). Adversarial attacks in consensus-based multi-agent reinforcement learning. In *2021 American control conference (ACC)* (pp. 3050–3055). IEEE.
21. Fan, J., Wang, Z., Xie, Y., & Yang, Z. (2020). A theoretical analysis of deep Q-learning. In *Learning for dynamics and control* (pp. 486–489). PMLR.
22. Siami, M., Bolouki, S., Bamieh, B., & Motee, N. (2017). Centrality measures in linear consensus networks with structured network uncertainties. *IEEE Transactions on Control of Network Systems, 5*(3), 924–934.
23. Sun, W., & Wu, T. (2021). Deep consensus learning. arXiv preprint arXiv:2103.08475
24. Savazzi, S., Nicoli, M., & Rampa, V. (2020). Federated learning with cooperating devices: A consensus approach for massive IoT networks. *IEEE Internet of Things Journal, 7*(5), 4641–4654.
25. Feng, H., Pang, T., Du, C., Chen, W., Yan, S., & Lin, M. (2023). Does federated learning really need backpropagation? arXiv preprint arXiv:2301.12195
26. Watkins, C. J., & Dayan, P. (1992). Q-learning. *Machine Learning, 8*, 279–292.

Md Muhidul Islam Khan is a researcher with the University of Stavanger, Norway, from 2021. In 2009 he received his Masters degree from Bangladesh University of Engineering and Technology (BUET). He has participated in the “eLINK”-project at Corvinus University of Budapest, Hungary, from September 2009 until July 2010 (funded by the European Union). His specialization lies in the fields of Wireless Sensor Networks, Networked Embedded Systems, and Pervasive Computing. He completed his PhD studies in Interactive and Cognitive Environment under the Erasmus Mundus Grant from European Commission working at Klagenfurt University, Austria and University of Genova, Italy from January 2011 to September 2014. He obtained his joint doctorate degree in September 2014. He joined as an Assistant Professor in BRAC University, Bangladesh and served there for one year. After that, he completed his one-year postdoc from the Hebei University of Technology, Tianjin, China. He worked as a Research Scientist in the Electronics Department at Tallinn University of Technology, Estonia. He worked as a senior lecturer in School of Information Technologies, Tallinn University of Technology, Tallinn, Estonia.

Gianfranco Nencioni is Associate Professor with the University of Stavanger, Norway, from 2018. He is received the M.Sc. degree in telecommunication engineering and the Ph.D. degree in information engineering from the University of Pisa, Italy, in 2008 and 2012, respectively. In 2011, he was a visiting Ph.D. student with the Computer Laboratory, University of Cambridge, U.K. He was a Post-Doctoral Fellow with the University of Pisa from 2012 to 2015 and the Norwegian University of Science and Technology, Norway, from 2015 to 2018. He is currently the head of the Computer Networks (ComNet) research group and leader of the project 5G-MODaNeI funded by the Norwegian Research Council. His research activity regards modelling and optimization in emerging networking technologies (e.g., SDN, NFV, 5G, Network Slicing, Multi-access Edge Computing). His past research activity has been focused on energy-aware routing and design in both wired and wireless networks and on dependability of SDN and NFV.