Privacy-preserving Incentive Mechanisms for Resource Sharing in IoT Markets

Prasoon Patidar[1], Jinding Xing[2]
[1,2]Carnegie Mellon University
5000 Forbes Ave, Pittsburgh, PA 15213
prasoonpatidar@cmu.edu[1],jindingx@andrew.cmu.edu[2]

Abstract

Computation offloading is a promising solution for resource-limited IoT devices to accomplish computation-intensive tasks. In order to promote the service trading between edge computing service providers and IoT devices, a series of works have explored incentive mechanisms for IoT-edge computing. However, traditional incentive mechanisms (such as Stackelberg's game-based approaches) expose the privacy of participants. Moreover, the existing Reinforcement Learning (RL) based incentive mechanisms do not consider the competition among multiple providers, which is not in line with reality.

In this project, we model the trading between IoT devices and computation resource providers as a multi-leader and multi-follower Stackelberg game with consideration of market competition and privacy preservation. In the proposed model, providers have a limited set of resources and IoT devices need to learn the best demand strategies and compete for getting the resources. We utilized different kinds of reinforcement learning techniques to learn the best purchase and pricing strategies for buyers and sellers without knowing the private information of any market players. Lastly, we evaluated the learned policies in various market scenarios, as find best seller strategies in different settings.

1 Introduction

The amount of data generated, stored and processed is increasing rapidly with the increase in the number of IoT devices in the market. Various kinds of IoT devices, i.e drones, smart cameras, smart speakers, etc are generating various modalities of data which are further used for understanding their respective environments, modeling human behavior, providing inferences, and assisting in decision making. A wide variety of computationally intensive operations, training various deep learning models is required to enable these. It is difficult to accomplish with IoT devices due to their limited computational resources (Wang et al. 2019). One way to tackle this problem is to offload heavy computation to nearby devices(also called edge nodes), which have higher compute resources, which is emerged as a paradigm of edge computing in recent years.

These edge nodes are maintained by a service provider, who sells compute resources at these nodes to earn revenue. With the growth in popularity of edge computing, there are multiple edge nodes, maintained by different service providers. IoT devices can demand to compute resources from all these edge providers based on their requirements and prices set by these providers. In this trading framework, each provider and the IoT devices have their own interests and are influenced by the actions of all players in the market. The trading problem, i.e what strategies should edge providers use to set prices for their compute resources and what strategies should IoT devices follow for generating demand, can be formulated as a Stackelberg game and solved using game-theoretic approaches. In order to do that, every resource provider, and IoT devices have to reveal their personal preferences to everyone. In the real-world, it might be difficult as every player will have reservations to reveal their preferences and capabilities in order to protect their self-interest in this competitive setting.

In this paper, we formulate this trading problem between multiple devices and multiple providers as a series of Stackelberg games, and present reinforcement learning-based techniques to achieve optimal strategies by all players in the market, without revealing any of their private information. Our trading model is based on well-studied utility formulation in a similar setting, and also incorporates provider limitation in terms of resources capacity. In our formulation, each edge provider runs their private instance of RL-based techniques(discussed later) to decide pricing and resource allocation based on their experiences, and all buyers make independent decisions to raise demand from all sellers based on their individual experiences. The contribution of our work is as follows:

- We extend the state of the art model for solving trading incentive problems in multi-buyer, multi-seller scenarios to introduce provider limitations in terms of resources, and introduce penalty mechanisms to motivate trading
- We reform studied RL-based techniques to find optimal strategies in this extended model for providers to decide pricing and resource allocation based on computing resources demand.
- We introduce deep reinforcement learning methods, to make our solution scale to the large size of markets, and show that they learn optimal strategies which fare better than any of the current methods, and are scalable.

- We analyze the performance of different methods in various kinds of market scenarios and show how different pricing strategies can impact overall social welfare and incentives for trading.

2 Background Literature

Reinforcement learning has emerged as a powerful tool, which can solve NE of the Stackelberg game without the participants revealing their privacy. (Asheralieva and Niyato 2019) formulated the interaction between providers and users as a stochastic Stackelberg game with incomplete information and proposed two exact and approximate model-based RL algorithms to obtain NE without knowing privacy of participants. (Yao et al. 2019) proposed an RL-based algorithm to obtain NE under privacy protection. However, these works only consider the scenario of a single provider, which is not in line with reality where there are multiple providers (Xiong et al. 2019). When there are multiple providers in the market, the optimal pricing determination of provider becomes more complicated.

(Xu et al. 2021) talks about taking privacy concerns and competition among providers into consideration; they utilize reinforcement learning (RL) techniques to design a privacy-preserving incentive mechanism for multiple providers and multiple IoT devices. The pricing and demand problem of providers and IoT devices is a multi-leader multi-follower Stackelberg game, in which the providers work as leaders to determine their prices first, and then the IoT devices determine their demands as followers, and finally, leaders make decisions based on assigned incentives. Due to privacy concerns, providers and IoT devices are unwilling to disclose their own parameters, which makes the derivation of NE becoming a great challenge. (Xu et al. 2021) assumes there are M IoT devices and N edge computing service providers in the IoT-edge computing system. Each provider $j \in N$ is equipped with rich computation capability, it can make profit through selling computing services to IoT devices. Each IoT device can purchase computing service from N providers and offload its computation-intensive tasks the corresponding provider.

The trading process of computing service Γ_{ij} between IoT device i and provider j can be divided into two stages: 1) In the first stage, each provider j determines the unit price of its computing service $_j$; 2) In the second stage, each IoT device i purchases x_{ij}amount of computing services from provider j with probability θ_{ij}. In the trading process, all providers and IoT devices aim at maximizing their own utilities. The formulation of the utilities of providers and IoT devices are as follow.

2.1 IoT Device Modeling

The task profile for IoT device i is (a_i, V_i), where a_i is resource occupation time for the task and V_i is the task completion utility. In the trading process, IoT device i first select a service provider j as its seller and demands x_{ij} amount of computing services from provider j.

$$r_{ij}^d = V_i \ln(x_{ij} - a_i + e) \tag{1}$$

Device i's cost for purchasing x_{ij} amount of computing services from provider j:

$$f_{ij}^d = x_{ij} p_j \tag{2}$$

The net profit ξ_{ij}^d of IoT device i from the trading deal Γ_{ij} is defined as the difference between revenue for task execution r_{ij}^d and the payment f_{ij}^d for service demand:

$$\xi_{ij}^d = V_i \ln(x_{ij} - a_i + e) - x_{ij} p_j \tag{3}$$

Each IoT device i chooses provider j with probability θ_{ij} which is proportional to the reciprocal of the service price of provider j:

$$\theta_{ij} = \frac{\frac{1}{p_j}}{\sum_{k \in N} \frac{1}{p_k}} \tag{4}$$

IoT device i can choose any provider $j(j \in N)$ as its seller, the expected utility ϕ_i of IoT device i in the trading model is:

$$\phi_i = \sum_{j \in N} (V_i \ln(x_{ij} - a_i + e) - x_{ij} p_j) \frac{\frac{1}{p_j}}{\sum_{k \in N} \frac{1}{p_k}} \tag{5}$$

2.2 Service Provider Modeling

For service providers, providing computation services to IoT devices incur a cost, which include electrical payments, hardware loss etc. Let c_j denote the unit cost of computing service for provider j.

Provider j's cost of providing x_{ij} amount of computing services for IoT device i is:

$$f_{ij}^e = x_{ij} c_j \tag{6}$$

Provider j can receive a revenue r_{ij}^e from trading Γ_{ij}:

$$r_{ij}^e = f_{ij}^d = x_{ij} p_j \tag{7}$$

The net profit for provider j from trading Γ_{ij}:

$$\phi_{ij}^e = r_{ij}^e - f_{ij}^e = x_{ij} p_j - x_{ij} c_j \tag{8}$$

Provider j can be chosen by any IoT device $i(i \in M)$ as its provider, the expected utility θ_j of provider j in the trading model is:

$$\phi_j = \sum_{i \in M} (x_{ij} p_j - x_{ij} c_j) \frac{\frac{1}{p_j}}{\sum_{k \in N} \frac{1}{p_k}} \tag{9}$$

In this scenario, each provider $j(j \in N)$ adjusts its price p_j to maximize its utility θ_j, each IoT device $i(i \in M)$ adjusts its service demand $x_i = \{x_{ij}\}_{i \in N}$ to maximize its utility ϕ_i.

However, in real life IoT markets, computation resource providers have limited resources that they can't guarantee to satisfy all the IoT device's resource demands. Further, (Xu et al. 2021) tested the learned policies in small market (2 buyers and 5 sellers). It's unknown how the size (e.g., large and small) and the type of competition (e.g., Monopsony market and monopoly market) impact the model performance.

3 The Extended Model

We introduced provider resource limitation to extend the current model. The current model assumes that providers have adequate computation resource, and each IoT device can purchase any amount of computation resources x_{ij} from provider j, which is not always true. The trading process of computing service Γ_{ij} between IoT device i and provider j in the extended model consists of the following stages: 1) In the first stage, each provider j determines the unit price of its computing service p_j; 2) In the second stage, each IoT device i submit order to provider j for purchasing x_{ij} amount of computing services with probability θ_{ij}; 3) In the third stage, each provider j determines the amount of computation services z_{ij} can be provided to IoT device i.

In the extended trading model, as providers have limited resources, IoT device i prepare purchase order based on task needs and the estimated amount of computation service each provider j can provide to IoT device i. If the estimated amount of computation service x_{ij} is larger than z_{ij}, IoT device i is unable to finish the planned task, resulting a loss of task incomplete for IoT device i. Providers in the extended trading model have limited computation services, each provider decide the amount of computation services being sold to IoT device i based on its service capacity l_j and the received orders $x_j = \{x_{ij}\}_{i \in M}$from all the IoT devices.

3.1 Extended IoT Device Model

we added task incomplete loss h_i^d to the expected utility ϕ_i of IoT device i. Due to privacy concerns, providers disclose their service capacity. IoT device i estimate providers' service capacity based on its past trading experiences $f(z_{ij})$ with provider j. IoT devices only has access to their own trading experiences and will not share their trading experiences with each other. $f(z_{ij})$ is defined as the average computation service device i bought from provider j in the past n trading experiences:

$$f(z_{ij}) = \frac{\sum_{t=1}^{T-1} z_{ij}^t}{T-1} \tag{10}$$

Loss h_i resulting from task incomplete is defined as:

$$h_i^d = \sigma(f(z_{ij}) - x_{ij})^2 \tag{11}$$

The expected utility ϕ_i of IoT device i in the trading model is:

$$\phi_i = \sum_{j \in N} (V_i \ln(x_{ij} - a_i + e) - x_{ij} p_j) \frac{\frac{1}{p_j}}{\sum_{k \in N} \frac{1}{p_k}} - \sigma(f(z_{ij}) - x_{ij})^2 \tag{12}$$

3.2 Extended Provider Model

we added service waste loss h_i^e to the expected utility ϕ_j of provider j. If provider j failed to sell all its service to devices in one trading, a service waste loss will incur, such loss can be rental payment, electricity payment etc. We define service waste as the difference between provider j's capacity l_j and the amount of sold service $z = \{z_{ij}\}_{i \in M}$:

$$h_i^e = \lambda \left(\sum_{i=1}^{M} z_{ij} - l_j \right) \tag{13}$$

Provider j decide the amount of computation services z_{ij} being sold to IoT device i based on its service capacity l_j and the received orders $x_j = \{x_{ij}\}_{i \in M}$ from all the IoT devices:

$$z_{ij} = \begin{cases} x_{ij} & \text{if} \sum_{i=1}^{M} x_{ij} \le l_j \\ x_{ij} - \frac{x_{ij}}{\sum_{i=1}^{M} x_{ij}} \left(l_j - \sum_{i=1}^{M} x_{ij} \right) & \text{if} \sum_{i=1}^{M} x_{ij} > l_j \end{cases} \tag{14}$$

The expected utility ϕ_j of provider j in the trading model is:

$$\phi_j = \sum_{i \in M} (x_{ij} p_j - x_{ij} c_j) \frac{\frac{1}{p_j}}{\sum_{k \in N} \frac{1}{p_k}} + \lambda \left(\sum_{i=1}^{M} z_{ij} - l_j \right) \tag{15}$$

3.3 Game Analysis

The goals of this paper are to find a service purchase strategy for each IoT device and a pricing strategy for each provider so that all of them can benefit. Same as (Xu et al. 2021) , we formulate this problem as a two-stage Stackelberg game with multi-leader and multi-follower. In the first stage of Stackelberg game, providers work as leaders to decide their unit price for computing services; In the second stage, IoT device submit their order to the providers according to their task profiles and the estimated provider's service capabilities, and provider's unit price.

IoT device i sub-game: Given that all providers' price p are known, IoT device i would like to adjust its service demand x_i within the limitation of each provider's service capabilities for maximization its utility ϕ_i. Therefore, the sub-game for IoT devices is defined as follows:

$$\begin{aligned} &\max_{x_i} \phi_i(x_i, p) \\ &\text{subject to } x_i > 0, \quad \forall_{i \in M, j \in N} \end{aligned} \tag{16}$$

In the extended model, IoT devices can estimate provider's resource capabilities by learning from past trading experiences and make optimal purchase decisions. The challenge lies in finding the optimal purchase decision for each IoT device with consideration of each device's trading histories. In this project we adopted a heuristic approach by using the SLSQP (Sequential Least Squares Programming) optimizer to find the optimal purchase decision for each IoT device.

IoT device i sub-game: Give the prices of the other providers p_{-j} and the responses of all IoT devices, provider j adjusts its price p_j to maximize its utility:

$$\begin{aligned} &\max_{p_j} \phi_j(p_j, p_{-j}, x^j(p)) \\ &\text{subject to } p_{\min} < p_j < p_{\max}, \quad \forall_{j \in N} \end{aligned} \tag{17}$$

4 Learning Optimal Policies

To address the challenge caused by privacy protection, we model the trading process as a Markov decision process and implemented different types of RL based algorithms to learn the optimal strategies of providers without knowing any privacy information. In the Markov decision process, the action is provider's price p_j and the state is some function of all the providers' prices at time $t-1$, the reward is the provider's utility at time t. Following subsections describe learning algorithms in detail.

4.1 Tabular RL Methods

Q Learning We start by implementing most basic version of Q learning (Watkins and Dayan 1992) for our extended model. Although,(Xu et al. 2021) has shown that Q-learning never converges even without any resource limitations to providers, it serves as a baseline for other algorithms.

WoLF-PHC (Xu et al. 2021) proposed a new RL based pricing mechanism(RLPM), in which they employ the value iteration of RL to derive the optimal pricing policy for each service provider. To start with, RLPM deploys independent agents to providers, each responsible for making pricing decisions. More precisely, each RLPM agent represents a stochastic pricing policy to meet the stochastic nature of IoT device demands. They used WoLF-PHC algorithm proposed in (Bowling and Veloso 2002) for policy updates. We used the similar formulation for our extended setting, to assess how does our model extension to incorporate resource limitations impacts performance of this state of the art algorithm.

4.2 Deep RL Methods

As count of service providers increase in market, state space grows exponentially. Thus, any tabular RL methods, which learns different policies for all states lead to performance degradation. Thus, we utilize three deep learning based RL methods to allow for policy learning in significantly large market settings.

Deep Q Learning Deep Q-Learning(DQN) is a variation of Q learning. Instead of tabular functions, DQN utilizes multi-layer neural networks to approximate Q functions. Further, Instead of using one Q function for calculating both target and estimation, uses two separate neural networks, a target neural network and an online neural network. At last, instead of using last obtained experience to update Q function, DQN stores the experiences in a replay memory and used mini-bathes by using stochastic gradient descent (Bottou 2010) to minimize loss. In vanilla version, samples are taken uniformly at random from experience replay memory. However, by treating all samples the same, we are ignoring a simple intuition from the real world, that is, we can potentially learn more from the experiences for which the outcomes differ more from our expectations. To leverage this fact, we utilize prioritized experience replay (PER) (Schaul et al. 2015) samples experiences with probability proportional to the absolute difference between the target and estimation values for this experience which is known as TD error.

Double Q Learning In Deep Q-learning, the target Q network is used both to select best action for next state and to calculate the target to evaluate the selected action. This makes it more likely to select overestimated values, resulting in overoptimistic value estimates. Double Q learning, (Van Hasselt, Guez, and Silver 2016) is a variation of deep Q Learning which tries to reduce overestimation of target values by proposing a simple trick to decompose the action selection from the action evaluation. More specifically, in the double DQN, the online network is used to select an action and the target network is used to generate the target value for that action. We utilize this to avoid sellers to overestimate their rewards in initial runs, leading to better learned policies.

Dueling Networks In the DQN network architecture, although there exist two networks, which are online network and target network, for each of these networks, a single neural network is used to estimate the action-value function Q that measures the value of choosing a particular action when in a particular state. The key insight behind the dueling network architecture (Wang et al. 2016) is that for many states, it is unnecessary to estimate the value of each action choice. To this end, in this network architecture, two separate functions are estimated, a value function V that measures how good it is to be in a particular state and an advantage function A that measures the relative importance of choosing a particular action when in a particular state. Then, these two functions are aggregated to estimate the action-value function Q.

The rationale behind using this kind of architecture in our problem setting is to reduce training efforts to optimize for provider pricing decisions which are incoherent and might never occur in real life scenarios.

5 Experiments

5.1 Experiments Setting

The training pipeline is shown in Figure 1. In the simulation, the IoT-edge computing market is composed of N service providers and M IoT devices. Table 1 shows the designed market scenarios. We designed two variations of markets: tight market and loose market, distributed market, and monopoly markets. Compared with a tight market, in a loose market, the providers have larger computation capabilities and are able to provide more resources to the IoT devices. In the distributed market, all the providers in the market have similar computation capabilities and are initialized with a similar amount of computation resources. In the monopoly market, a monopoly seller is initialized with significantly larger computation capabilities than the rest of the sellers. In order to make comparisons across various markets, we kept the count of buyers and sellers in different markets constant. Further, we also keep buyer configuration constant across all markets. We used similar setting Provider's resource waste penalty and IoT device's task incomplete penalty are set as $\lambda = 0.2$ and $\sigma = 0.5$ respectively. Providers unit resource cost c_j are set as $[12, 18, 15, 14, 12]$. IoT devices required resource occupied time a_i, and IoT devices task completion

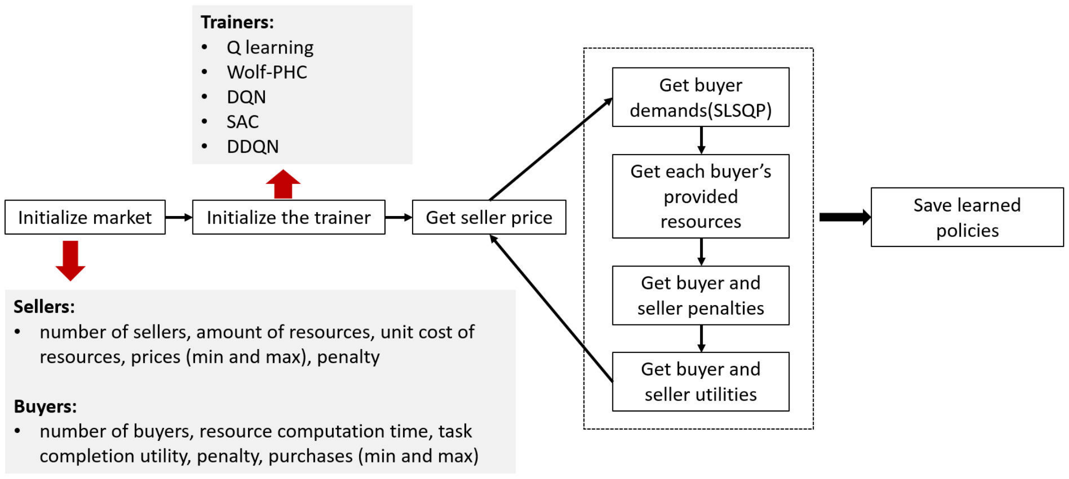
Trainers:
Q learning
Wolf-PHC
DQN
SAC
DDQN
Initialize market
Initialize the trainer
Get seller price
Get buyer demands(SLSQP)
Get each buyer's provided resources
Get buyer and seller penalties
Get buyer and seller utilities
Save learned policies
Sellers:
number of sellers, amount of resources, unit cost of resources, prices (min and max), penalty
Buyers:
number of buyers, resource computation time, task completion utility, penalty, purchases (min and max)

Figure 1: Training pipeline-the training pipeline includes three stages: in the first stage, we initialize the training environment (the market and the trainer). The market scenario is initialized by changing the configuration of sellers and buyers. In the second stage, we train the RL algorithm by running the trading simulation, namely get seller price, use the price to calculate each buyer's demand, get each buyer's provided resource by considering each sell's received demands from the market and its own resource limitations, then we can calculate buyer and seller penalties and utilities. The third stage will save the trained policies for evaluation.

utility V_i are set as random number in $[1, 2]$ and $[10, 100]$, respectively.

5.2 Evaluation Metrics

In our experiment setup, we used the following evaluation metrics:

- **Utility of IoT devices**: the utility value of the IoT devices in the market.
- **Utility of providers**: the utility value of the computation resource providers in the market.
- **provider prices (seller prices)**: each provider's unit price of computing service.
- **IoT device's demand**: total demand of all the IoT devices in the market.
- **Provided resources**: total resource provided by all the providers in the market.
- **Total penalties**: total loss of IoT device's task incomplete loss and providers' service waste loss. Low total loss indicates the gap between the demand resources and provided resources is small, and the market is operating in high efficiency.
- **Social welfare**: the total utilities of all IoT device and providers. High social welfare indicates a good trading market where both the providers and buyers have strong motivation to trade.

5.3 Training Results

Impact of training algorithms:

Figure 2 shows the changes in social welfare in the training. A steady increasing social welfare during the training indicates the algorithm is able to make the providers learn better pricing strategies that promote market trading and earn more utilities. DDQN and DQN only perform best in the monopoly market. DQN-Duel is relatively robust and performs well across all the market scenarios. In Q learning, the social welfare stops increasing at around 4000 iterations. Wolf-PHC learning only performs well in the distributed market, it fails to learn and shows a decreasing trend of social welfare in the loose market.

Market	Resource range	Sellers	Buyers
Tight market	[22, 40]	5	50
Loose market	[152, 175]	5	50
Distributed market	[46, 62]	5	50
Monopoly market	[25, 250]	5	50

Table 1: IoT-edge computing market configurations

Impact of market scenarios:

Figure 3 shows the training performance of DQN-Duel in different market scenarios. In the loose market, the demanded resources and purchased resources are the same. As the total amount of available computation resources is much larger than the IoT devices' needs so that the providers can always satisfy buyers' demands. In the rest of the market scenarios, we can see an increasing trend both in demands and purchases, which indicates the price based on learned policies indeed promote trading between buyers and sellers. The monopoly market shows the providers' resource waste penalty in the extended model adds more incentives for providers to lower their prices and sell more computation resources. Seller 0 and seller 4 are initialized with the same unit resource cost, while seller 4 offers a lower price due to its high resources capability and resource waste penalty. Additionally, seller 2 is initialized with the highest unit resource

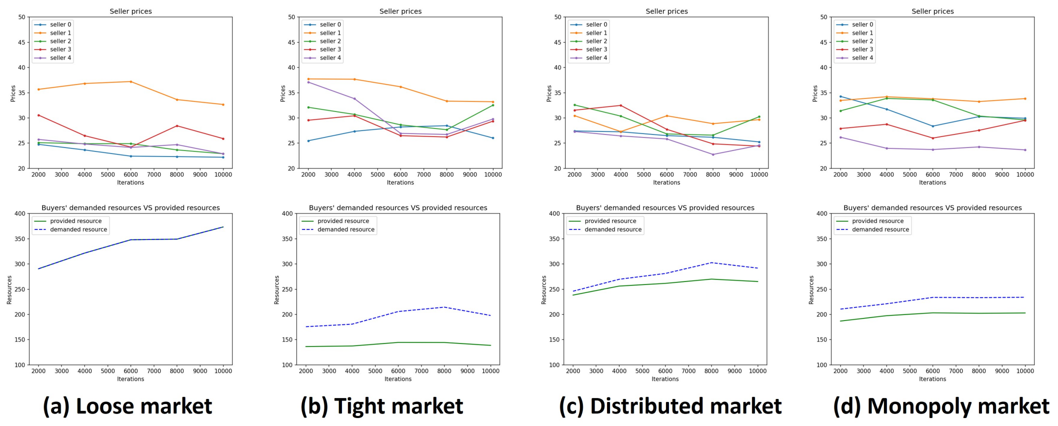

Figure 2: DQN-Duel training performance in different market scenarios.

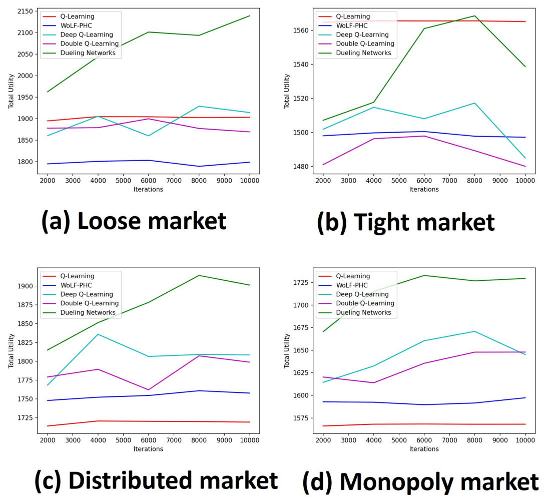

Figure 3: Training results-the five rows representing DDQN, DQN-Duel, DQN, Q learning and Wolf-PHC in different market scenarios, respectively.

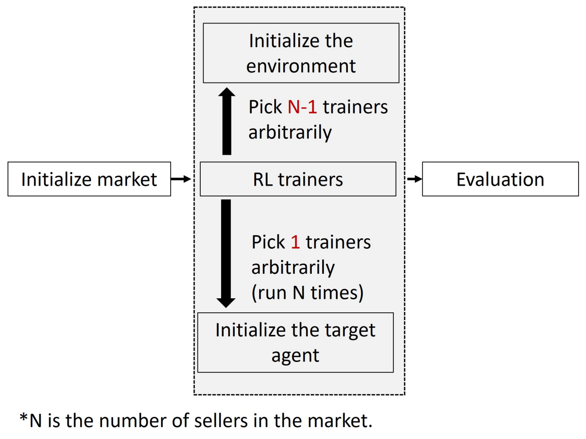
Initialize the environment
Pick N-1 trainers arbitrarily
Initialize market
RL trainers
Evaluation
Pick 1 trainers arbitrarily (run N times)
Initialize the target agent
*N is the number of sellers in the market.

Figure 4: The head-to-head policy comparison pipeline.

cost, so that seller 2 always has a higher price compared with other providers in the market.

6 Policy Comparison

6.1 Comparison Setting

Figure 4 shows the head-to-head comparison pipeline. We first initialize the sellers with policies learned by different RL algorithms, then put all the sellers in the same market scenarios and run the simulation to find which RL algorithms learn the best price strategies. In the comparison process, there are two types of agents in the market. The first type of agent is the environmental agent, the second type of agent is compare agent. The agents in each comparison round are constituted of a group of environmental agents and one compare agent. The environmental agents are fixed, which are randomly initialized $N-1$ trainers. The compare agent is initialized as the learning agent that we aim to compare. For example, if we want to compare Q learning and wolf-PHC learning, we will initialize the compare agent with policies learned by Q learning trainer and wolf-PHC trainer, respectively. Then will run the simulation twice, the first run will be environmental agents and Q learning trainer, the second run will be the environmental agents and wolf-PHC trainer.

6.2 Comparison Results

Impact of seller computation capability on learned policies

Figure 5 shows different sellers' trading performance under the same learned policies. The same learned policy has different performances on sellers with different computation capabilities. Such differences can be significant under certain market scenarios. For example, in the loose market and distributed market, policy learned by dueling networks is the best policy for the small sellers (seller with fewer computation resources but the same unit resource cost), the policy

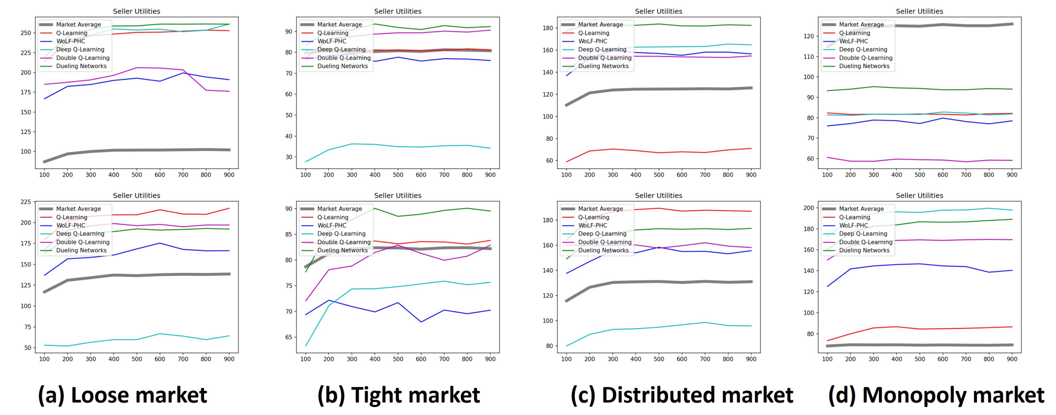

Figure 5: Comparison of learned policies in different market scenarios-the compare agent in the first row has less computation resources compared with the compare agent in the second row. In the monopoly market, the compare agent in the second row is the monopoly seller.

learned by Q learning is the best policy for large sellers. In the tight market, dueling networks perform best on all the sellers.

Impact of market scenarios on learned policies

Seller utilities. As shown in Figure 5, dueling networks is the most robust learning algorithm across different market scenarios. Especially, for small sellers, policy learned by the dueling networks almost always gets the largest seller utilities. For large sellers, policy learned by the dueling networks can also guarantee seller utility that is higher than the market average. To note that, in a monopoly market, the market average utility is much lower than the utility of the large seller. As the large seller is the monopoly seller, it can sell more resources and get a utility that is higher than the market average.

Social welfare. Figure 6 shows the social welfare of the large seller in different market scenarios. In different market scenarios, the best policy that gives the highest social welfare varies. Q learning, wolf-PHC learning, double Q learning, and deep Q learning have the highest social welfare in the loose market, tight market, distributed market, and monopoly market, respectively. Deep Q learning has the lowest social welfare in the loose market, tight market, and distributed market. Another interesting observation is that, in the tight market and distributed market, the policy that gives the highest seller utilities does not lead to the highest social welfare. Dueling networks and Q learning gives the highest seller utilities in the tight market and distributed market. However, wolf-PHC and double Q learning have the highest social welfare in these two market scenarios.

Seller price. Figure 7 shows the compare agent's seller price in different market scenarios. Deep Q learning has the highest price in the loose market, tight market, and distributed market, and lowest price in the monopoly market. The price strategies learned by other agents (except for deep Q learning) are well concentrated and did not show many fluctuations in the comparison.

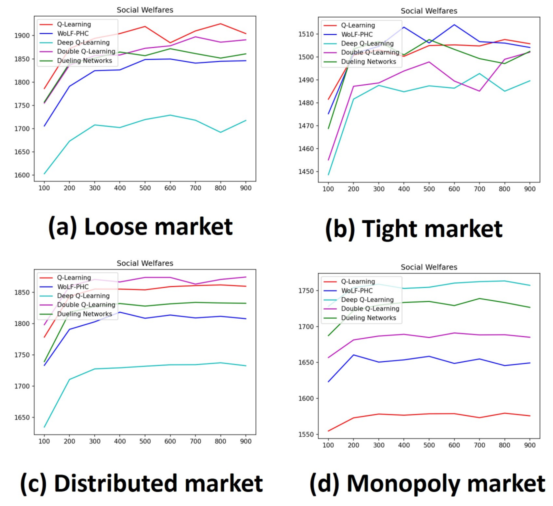

Figure 6: Social welfare of the large seller in different market scenarios.

7 Conclusion

The number of IoT devices is growing rapidly and these devices usually produce or collect a large amount of raw data. It is challenging for these devices to accomplish data processing due to their limited computation resources. This paper, based on previous work, designed an incentive mechanism that aims to promote the trading between service providers and IoT devices with the consideration of privacy-preserving and providers' resource limitations.

In this paper, we formulated the trading between IoT devices and computation service providers as a series of Stackelberg Game and adopted reinforcement learning techniques to learn the optimal strategies for market players. Our experiment results show that dueling networks are robust across all market scenarios which almost always

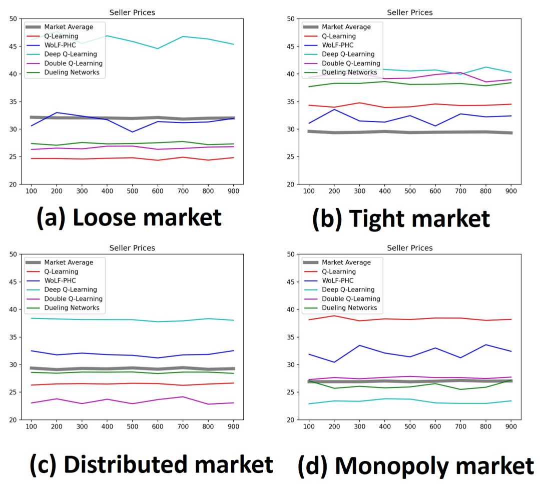

Figure 7: Seller price of the large seller in different market scenarios.

.

gives the highest seller utilities. Dueling networks also have higher than market average social welfare in the head-to-head comparison. Moreover, we also found that the seller computation capability and market scenario impact the choice of best policies. We have open-sourced our training, evaluation, and policy comparison frameworks (*https://github.com/prasoonpatidar/multiagentRL-resource-sharing*) to promote more exploration and studies across various other kinds of markets and allow the development of other training algorithms for our problem setting. In future work, we can extend the model by introducing random explorations in buyers' decisions of choosing the provider.

References

Asheralieva, A.; and Niyato, D. 2019. Learning-based mobile edge computing resource management to support public blockchain networks. *IEEE Transactions on Mobile Computing* .

Bottou, L. 2010. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*, 177–186. Springer.

Bowling, M.; and Veloso, M. 2002. Multiagent learning using a variable learning rate. *Artificial Intelligence* 136(2): 215–250.

Schaul, T.; Quan, J.; Antonoglou, I.; and Silver, D. 2015. Prioritized experience replay. *arXiv preprint arXiv:1511.05952* .

Van Hasselt, H.; Guez, A.; and Silver, D. 2016. Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 30.

Wang, J.; Ni, M.; Wu, F.; Liu, S.; Qin, J.; and Zhu, R. 2019. Electromagnetic radiation based continuous authentication in edge computing enabled internet of things. *Journal of Systems Architecture* 96: 53–61.

Wang, Z.; Schaul, T.; Hessel, M.; Hasselt, H.; Lanctot, M.; and Freitas, N. 2016. Dueling network architectures for deep reinforcement learning. In *International conference on machine learning*, 1995–2003. PMLR.

Watkins, C. J.; and Dayan, P. 1992. Q-learning. *Machine learning* 8(3-4): 279–292.

Xiong, Z.; Kang, J.; Niyato, D.; Wang, P.; and Poor, H. V. 2019. Cloud/edge computing service management in blockchain networks: Multi-leader multi-follower game-based ADMM for pricing. *IEEE Transactions on Services computing* 13(2): 356–367.

Xu, H.; Qiu, X.; Zhang, W.; Liu, K.; Liu, S.; and Chen, W. 2021. Privacy-preserving incentive mechanism for multi-leader multi-follower IoT-edge computing market: A reinforcement learning approach. *Journal of Systems Architecture* 114: 101932. ISSN 1383-7621. doi:https://doi.org/10.1016/j.sysarc.2020.101932. URL https://www.sciencedirect.com/science/article/pii/S1383762120301910.

Yao, H.; Mai, T.; Wang, J.; Ji, Z.; Jiang, C.; and Qian, Y. 2019. Resource trading in blockchain-based industrial Internet of Things. *IEEE Transactions on Industrial Informatics* 15(6): 3602–3609.