Asynchronous Multi-Agent Reinforcement Learning for Scheduling in Subnetworks

Ashvin Srinivasan*, Junshan Zhang†, Olav Tirkkonen*
*Department of Information and Communications Engineering, Aalto University, Finland
†College of Engineering, University of California, Davis, California, USA
{ashvin.1.srinivasan, olav.tirkkonen}@aalto.fi, jazh@ucdavis.edu

***Abstract*—We address radio resource scheduling in a network of multiple in-X subnetworks providing wireless Ultra-Reliable Low-Latency Communication (URLLC) service. Each subnetwork is controlled by an agent responsible for scheduling resources to its devices. Agents rely solely on interference measurements for information about other agents, with no explicit coordination. Subnetwork mobility and fast-fading effects create a non-stationary environment, adding to the complexity of the scheduling problem. This scenario is modeled as a multi-agent Markov Decision Process (MDP). To address the problem, we propose a Multi-Agent Deep Reinforcement Learning (MADRL) approach under URLLC constraints, which integrates Long Short-Term Memory (LSTM) with the Deep Deterministic Policy Gradient (DDPG) algorithm to manage non-stationarity and high-dimensional action spaces. We apply an asynchronous update strategy, where one agent is updating at a time. This reduces learning variability, resolves policy conflicts, and improves the interpretability of the MADRL approach. Simulation results demonstrate that the asynchronous update mechanism outperforms synchronous updates and baseline methods, achieving superior reliability, resource utilization, and explainability.**

***Index Terms*—6G, In-X subnetworks, URLLC, Dynamic Spectrum Allocation, Non-Stationary MDP, Multi-agent Reinforcement Learning, Explainable AI**

I. Introduction

Sixth-generation (6G) wireless technology aims to enable the Industry 4.0 vision of fully wireless factories [1]. In addition to increased mobility, wireless systems offer significant advantages over traditional cable networks, such as reduced deployment and maintenance costs [2]. Industry 4.0 requires seamless connectivity across diverse factory devices, ranging from basic sensors and actuators to complex industrial robots [3]. These networks, deployed within entities like machines, robots, or vehicles, and managed by an umbrella 6G network, are called *in-X subnetworks*. To ensure effective automation and control, in-X subnetworks must deliver reliability and latency comparable to traditional wired setups.

Ensuring Ultra-Reliable Low-Latency Communication (URLLC) in wireless settings is highly non-trivial. Subnetworks often need to share spectrum, which makes the wireless channel unpredictable and prone to interference from nearby systems [3]. The situation becomes even more complex when subnetworks are mobile, as is frequently the case in industrial environments. Studies on URLLC for industrial wireless systems have highlighted how dynamic channel conditions and random interference can undermine communication reliability. In Industry 4.0 scenarios, where dense deployments of subnetworks are common, fading channels, overlapping coverage areas, and the absence of coordination between subnetworks further elevate the risk of packet failures, making reliable communication even more difficult to achieve [4], [5].

Reinforcement Learning (RL), and specifically Multi-Agent Deep Reinforcement Learning (MADRL), has shown promise in managing resource allocation in Mobile in-X networks [6]–[9]. In [6], an MADRL approach is proposed for intelligent radio resource management, utilizing only received signal strength indicators. In [7], authors address the challenge of allocating a single channel to each agent using a Double Deep Q-Network (DDQN). Both works address non-stationarity by centralized training, where DRL agents learn the transition functions of the entire factory. In [8], a distributed MADRL algorithm is considered where agents update their Q-functions in a sequential manner. The updates prioritize devices with the most stringent latency requirements, aiming to mitigate the non-stationarity introduced by concurrent learning across multiple agents. In our previous work [9], an MADRL approach is explored where agents are trained independently without explicit communication but updated synchronously.

These works, however, have not adequately addressed several challenges, including those arising from the non-stationary environment. In [6], [7], non-stationarity is tackled by assuming centralized training, with agents learning the transition functions of the entire system, which leads to considerable communication overhead. In [8], sequential DQN is used, with an asynchronous update mechanism for scheduling. While this helps addressing non-stationarity, the approach becomes computationally impractical in high-dimensional action spaces. In [9], we explored an MADRL framework in high-dimensional action spaces where agents were updated simultaneously. Although this method is effective in handling dynamic environments, it increases variability and non-stationarity, which can result in learning conflicts.

In this paper we specifically focus on the challenges caused by the mobility of in-X subnetworks managed by autonomous agents. Each agent functions independently, selecting channels and scheduling devices. The proposed method is designed to handle dynamic interference conditions while ensuring efficient and fair resource allocation under strict URLLC constraints. The main contributions of this work are as follows: Unlike our prior work [9], which focused on synchronous updates, this paper introduces an asynchronous update strategy for

MADRL, where agents learn independently by updating one at a time. This approach reduces variability in learning and avoids conflicts in policy updates. Furthermore, we introduce a framework where a Long Short-Term Memory (LSTM) network is pretrained to predict Signal-to-Interference-plus-Noise Ratios (SINRs) before being integrated into the MADRL framework. This explicit use of LSTM for SINR prediction enhances both the performance and the interpretability of the learning process, leaving the task to learn to operate in the multiagent environment to the RL agents.

The structure of the paper is as follows: Section II outlines the system model and the objectives of this study. Section III provides an overview of the MADRL framework employed. Section IV presents the simulation results, and Section V concludes the paper.

II. System Model

We focus on an indoor factory setting consisting of M mobile subnetworks, where each subnetwork is managed by an access point (AP) and serves J devices. Without loss of generality, we concentrate on downlink communication, following the framework introduced in [9]. The same principles can be readily extended to the uplink.

The total channel bandwidth is divided into N subchannels, and transmissions within each subnetwork are orthogonal to prevent intra-subnetwork interference. Each subnetwork operates in a fully scheduled multiuser communication regime, where the AP is assumed to have complete knowledge of the signal-to-interference-plus-noise ratio (SINR) for its devices across all subchannels. The AP schedules transmissions based on this information, and communicates scheduling decisions to its devices through control channels. Any combination of subchannels can be scheduled to each device.

The subnetworks share the radio spectrum without explicit coordination, meaning that multiple subnetworks can simultaneously use the same subchannels. This results in inter-subnetwork interference. To address this, an online learning algorithm is implemented at the APs to minimize the interference price caused by channel collisions. We assume a slotted channel access system, where all subnetworks adhere to a unified slotting scheme managed by an umbrella wide-area network. We assume a service model where in each time slot, the AP sends new data packets to its devices instead of retransmitting possibly failed packets [10].

The mobility of sub networks makes the channel gain vary over time. The channel gain from AP b to device j on subchannel n at time t is denoted as $h^n_{j,b}(t)$. The received signal at device j served by AP b is:

$$y^n_{j,b}(t) = h^n_{j,b}\, x_b + \sum_{m \neq b} h^n_{j,m}(t)\, x_m g_{n,m} + z_j, \qquad (1)$$

where x_b and x_m are transmitted symbols with power P_t, and $z_j \sim \mathcal{CN}(0, \sigma^2)$ is Additive White Gaussian Noise (AWGN).

The SINR for device j on subchannel n is:

$$\gamma_{n,j}(t) = \frac{|h^n_{j,b}|^2}{\sum_{m \neq b} |h^n_{j,m}(t)|^2 g_{n,m} + \frac{1}{\gamma_0}}, \qquad (2)$$

where $\gamma_0 = P_t / W\sigma^2$ represents the ratio between the transmit power and the noise power over subband width W, and $g_{n,m}$ indicates whether subchannel n is used by AP m.

In each time slot, a fixed number of information bits is sent to each device, but the number of subchannels used for transmission can vary. The AP agent solves a scheduling problem in each slot, determining which channels to allocate to each device. Let $\mathbf{a}^{(b)}_j$ denote an $N \times 1$ vector where entry $a^{(b)}_{n,j} \in \{0, 1\}$ indicates whether subchannel n is allocated to device j. The outcome of the scheduling problem for AP b is represented by the $N \times J$ matrix $\mathbf{A}^{(b)}$ consisting of the columns $\mathbf{a}^{(b)}_j$. Sub-channel occupancy by AP m is thus given by $g_{n,m} = \sum_j a^{(m)}_{n,j}$.

Error performance is modeled using a predefined modulation method and a fixed Forward Error Correction (FEC) scheme for each subchannel. Repetition coding is applied across multiple subchannels to enhance reliability. At the receiver, the signals from the allocated subchannels are Chase combined before FEC decoding. The effective SINR characterizing the communication channel between AP b and device j is then

$$\gamma_j = \sum_{n=1}^{N} a^{(b)}_{n,j} \gamma_{n,j}\,, \qquad (3)$$

and the finite block length error probability is [11]:

$$p_{j,b} = Q\left(\frac{2qC(\gamma_j) - 2k + \log_2(q)}{2q\sqrt{V(\gamma_j)}}\right), \qquad (4)$$

where $Q(.)$ is the Gaussian q-function, $C(\gamma) = \log_2(1+\gamma)$ is AWGN channel capacity, $V(\gamma) = \gamma \frac{2+\gamma}{(1+\gamma)^2} (\log_2 e)^2$ is channel dispersion, q is the code block length, k is the number of information bits, and $R = k/q$ is the code rate.

The resource scheduling problem is formulated as

$$P_1 : \min \sum_{j,b} p_{j,b}(t) \quad \text{s.t.} \quad p_{j,b}(t) \leq p_0, \qquad (5)$$

where p_0 is the reliability threshold, often taken to be 10^{-5} in URLLC scenarios. This optimization problem is solved using a reinforcement learning framework, where the asynchronous update mechanism ensures stability and adaptability in non-stationary environments.

III. Reinforcement Learning Framework

We model a non-stationary Markov Decision Process (MDP) by $(\mathcal{S}, \mathcal{A}, P_T(\cdot, t), r(\cdot), \eta)$. Here, $\mathcal{S}$ represents the state space, $\mathcal{A}$ represents the action space, $P_T(\cdot, t)$ denotes the time-varying state transition probability, $r(\cdot)$ is the reward function, and η is the discount factor.

At time t, the state of the system is represented as $\mathbf{S}(t) \in \mathcal{S}$, and the action taken at that time is $\mathbf{A}(t) \in \mathcal{A}$. For each of the N subchannels, the AP can take $J + 1$ actions—allocating it to any of the J devices, or leaving it unallocated. The dimensionality of the action space is thus N^{J+1}. The transition probability $P_T(\mathbf{S}(t+1)|\mathbf{S}(t), \mathbf{A}(t); t)$ represents the likelihood of transitioning to the next state $\mathbf{S}(t+1)$ given the current state $\mathbf{S}(t)$ and the action $\mathbf{A}(t)$. This formulation allows us to model

the dynamic and non-stationary nature of the environment where interference patterns evolve over time.

A. State Representation with LSTM

An LSTM network is used to improve state representation by predicting SINR values. The network is pretrained offline using historical SINR data in a supervised learning setup. By analyzing a sliding window of t_r past observations, the LSTM learns to predict future SINRs and capture temporal patterns in their dynamics. During reinforcement learning (RL) training, the LSTM's output becomes part of the input state, enabling a more accurate understanding of the environment. It is worth to note that the actions taken by an AP influence the SINRs of its devices indirectly, with these effects appearing after a one-slot delay rather than immediately [9]. In our framework, the true state corresponds to the SINRs of devices, which represent the underlying system dynamics of the network. To address the one-slot delayed impact of prior actions on interference and SINRs, the observable state is defined as a tuple comprising of the LSTM-predicted SINRs and the previous actions taken by the AP.

$$\mathbf{S}(t) = [\hat{\mathbf{\Gamma}}(t), \mathbf{A}(t-1)], \tag{6}$$

where $\hat{\mathbf{\Gamma}} \in \mathcal{R}^{N \times J}$ is the matrix of SINR estimates generated by the LSTM, and $\mathbf{A} \in \{0,1\}^{N \times J}$ represents the action matrix indicating channel allocations. The integration of LSTM ensures more accurate and temporally aware state representations, stabilizing the training process.

To adapt to environmental dynamics, the LSTM undergoes periodic hybrid retraining, combining new observations from exploration with the original training dataset. The specifics of this are discussed in Sec. IV.

B. Reward and Optimization Objective

As the aim in (5) is to minimize packet loss probability subject to the reliability constraint, we design a reward function to penalize high packet loss probabilities, thereby promoting reliable resource allocation:

$$r(t) = -\sum_{j=1}^{J} \log\left(\frac{p_{j,b}(t)}{p_0}\right). \tag{7}$$

Here p_0 is the reliability threshold. This function combines a barrier-function of interior-point methods with the objective function of (5).

Each AP controlling a subnetwork is modeled as an independent RL agent. The objective of each agent is to maximize its cumulative reward thereby ensuring that the packet loss probability remains below the predefined threshold. The cumulative discounted reward over a time horizon T is:

$$R_{ac}(t) = \sum_{k=t}^{T} \eta^{k-t} \mathbb{E}[r(k)], \tag{8}$$

and the optimization objective for each RL agent is to maximize the cumulative reward:

$$P_2 : \max_{\mathbf{A}(t)} R_{ac}(t), \quad 0 \le t \le T. \tag{9}$$

C. Learning Algorithm

To learn scheduling policies, we utilize a modified Deep Deterministic Policy Gradient (DDPG) algorithm [12], adapted for high-dimensional discrete action spaces under URLLC constraints. The DDPG algorithm is designed to find *deterministic continuous* actions, and it consists of the following key components:

- *Actor Network*, which outputs a deterministic policy:

$$\hat{\mathbf{A}} = f_\phi(\mathbf{S}), \tag{10}$$

  where ϕ represents the parameters of the actor network. The actor network generates the action $\hat{\mathbf{A}}$ based on the current state $\mathbf{S}$.
- *Critic Network*, which evaluates the quality of actions by estimating the Q-value function for a given policy:

$$Q_\pi(\mathbf{S}, \mathbf{A}) = \mathbb{E}_\pi\left[\sum_{k=t}^{T} \eta^{k-t} r(k) \,\middle|\, \mathbf{S}_t = \mathbf{S}, \mathbf{A}_t = \mathbf{A}\right], \tag{11}$$

  where π is the policy being followed, and η is the discount factor. The critic network, parameterized by θ, approximates this Q-value function for the given state-action pair $(\mathbf{S}, \mathbf{A})$.

The actor network generates continuous-valued actions which represent a proposed allocation of resources. To provide concrete scheduling decisions, these have to be discretized. To meet URLLC requirements, there is a minimum requirement that a packet has to be transmitted to each user. Accordingly, when discretizing the continuous actions, we impose an URLLC constraint, ensuring that each device is assigned at least one resource.

For action discretization, we adopt the binary tree search mechanism of [9], which operates in two steps. In the first step, we identify a set $\mathcal{Z}$ of K candidate initial points $\mathbf{A}_k \in \{0,1\}^{N \times J}$ such that exactly one resource is allocated to each user. This process begins by sorting the entries of the continuous action matrix $\hat{\mathbf{A}}$ and selecting the J largest values. If the chosen entries fail to create a valid schedule, ensuring each device is assigned a unique resource, the set is gradually expanded until K candidate initial points are generated. Once the initial points are generated, the second step involves completing each schedule. This is accomplished by assigning the remaining resources to users in a greedy manner, prioritizing their closeness to $\hat{\mathbf{A}}$, or leaving them unallocated—if $\hat{a}_{n,j} < 1/2$ for all devices j, resource n is left unassigned. This process results in a set $\mathcal{Z}$ containing up to K distinct matrices, with each matrix representing a potential resource allocation. Finally, the critic network evaluates the K candidate schedules and selects the one that maximizes the Q-value, choosing the schedule as: $\mathbf{A} = \arg\max_{\mathbf{A}_k \in \mathcal{Z}} Q_\theta(\mathbf{S}, \mathbf{A}_k; t)$.

D. Target Networks and Updates

The actor and critic networks work together to produce the discrete action $\mathbf{A}_t$ based on the state input $\mathbf{S}_t$. The agent interacts with the environment, receives a reward r_t, and moves to the next state $\mathbf{S}_{t+1}$. The transitions are stored in a replay

buffer $\mathcal{B}$, which is continuously updated over time. Random samples from the buffer are used to train and update the networks.

To ensure stable training, we maintain target networks $f_{\bar{\phi}}$ and $Q_{\bar{\theta}}$, which are updated more gradually than the main networks using soft updates. A batch of samples $\mathcal{S} \subset \mathcal{B}$ is drawn, and the discounted estimated Q-value is computed by the target networks as

$$y(j) = r(j) + \eta Q_{\bar{\theta}}(\mathbf{S}_{j+1}, f_{\bar{\phi}}(\mathbf{S}_{j+1}; t+1)), \tag{12}$$

where $\bar{\theta}$ and $\bar{\phi}$ are the parameters of the target critic and actor networks, respectively. The critic network parameters θ are then updated by minimizing the loss [12]:

$$L(\theta) = \frac{1}{|\mathcal{S}|} \sum_{j=1}^{|\mathcal{S}|} \left[y(j) - Q_\theta(\mathbf{S}_j, \mathbf{A}_j; t) \right]^2 , \tag{13}$$

while the actor network parameters ϕ are updated using the deterministic policy gradient [12]:

$$\nabla_\phi f_\phi = \frac{1}{|\mathcal{S}|} \sum_{j=1}^{|\mathcal{S}|} \nabla_{\mathbf{A}} Q_\theta(\mathbf{S}_j, \mathbf{A}; t)\big|_{\mathbf{A}=f_\phi(\mathbf{S}_j)} \nabla_\phi f_\phi(\mathbf{S}_j). \tag{14}$$

Finally, the target networks are updated as

$$\bar{\theta} \leftarrow \tau\theta + (1-\tau)\bar{\theta}, \quad \bar{\phi} \leftarrow \tau\phi + (1-\tau)\bar{\phi}, \tag{15}$$

where $\tau \in (0,1)$ is the soft update coefficient. The pseudo-code for the RL framework, presented in Algorithm 1, is adapted from our previous work [9], with modifications to incorporate asynchronous updates.

Algorithm 1 Asynchronous Multi-Agent(MA) DDPG Algorithm

For each agent k, create the critic network Q_{θ_k} and actor network f_{ϕ_k}, initializing their parameters as θ_k and ϕ_k randomly.
Set up the target critic network $Q_{\bar{\theta}_k}$ and target actor network $f_{\bar{\phi}_k}$, initializing their parameters $\bar{\theta}_k$ and $\bar{\phi}_k$ identically to their corresponding networks.
Set $\bar{\theta}_k \leftarrow \theta_k$, $\bar{\phi}_k \leftarrow \phi_k$.
Initialize replay buffer $\mathcal{B}_k$ for each agent k.
Set the initial state $\mathbf{S}_k^{(0)}$ randomly for each agent k.
for epoch = 1 to $E_{\max}$ **do**
  if epoch $\neq$ 1 **then**
    Set state $\mathbf{S}_k^{(T)}$ as the final state from the previous epoch for all agents.
  end if
  for t = 1 to T **do**
    Select agent $k_t = (t \mod K) + 1$, where K is the total number of agents.
    Select candidate continuous action $\hat{\mathbf{A}}_{k_t} = f_{\phi_{k_t}}(\mathbf{S}_{k_t}^{(t)}) + \mathcal{N}_{k_t}(t)$, where $\mathcal{N}_k$ is a random process for action exploration.
    Select discrete action $\mathbf{A}_{k_t}^{(t)}$ from $\hat{\mathbf{A}}_{k_t}$ [9].
    Execute action $\mathbf{A}_{k_t}^{(t)}$, observe reward $r_{k_t}^{(t)}$ and new state $\mathbf{S}_{k_t}^{(t+1)}$ from the environment.
    Store transition $(\mathbf{S}_{k_t}^{(t)}, \mathbf{A}_{k_t}^{(t)}, r_{k_t}^{(t)}, \mathbf{S}_{k_t}^{(t+1)})$ in buffer $\mathcal{B}_{k_t}$.
    Sample a random mini-batch $\mathcal{S} \subset \mathcal{B}_{k_t}$.
    Compute target $y_{k_t}(j)$ using (12).
    Update the critic network $Q_{\theta_{k_t}}$ and actor network $f_{\phi_{k_t}}$ as per (13) and (14).
    Update the target networks as per (15).
  end for
end for

IV. Simulation Results

To test the performance of the proposed DRL framework, we simulate a factory environment of size 20 × 20 meters. The setup includes three subnetworks, each acting as an RL agent. Each subnetwork contains one access point (AP) located at its center and three devices within a radius of 1 meter. The subnetworks move inside the factory along straight paths, changing directions at random when meeting a wall. The simulation parameters are summarized in Table I. We use PyTorch as the deep learning framework.

The channel coefficient between AP m and device j on for subchannel n is modeled as:

$$h_{j,m}^n = \sqrt{\beta_{j,m}^n}\, \xi_{j,m}^n \,, \tag{16}$$

where $\xi_{j,m}^n$ represents small-scale fading. With $d_{j,m}$ the distance between the AP and device, distance-dependent path loss is $\beta_{j,m}^n = 10^{-\rho(d_{j,m})/10}$.

We apply the Indoor-Factory path loss model (InF:SL) of [13]. The path loss $\rho(d)$ in dBs is defined as the larger of the line-of-sight (LoS) and non-line-of-sight (NLoS) alternatives:

$$\mathrm{PL}_{\mathrm{LoS}}(d) = 31.84 + 21.50 \log_{10}(d) + 19 \log_{10}(f_c),$$
$$\mathrm{PL}_{\mathrm{NLoS}}(d) = 33 + 25.5 \log_{10}(d) + 20 \log_{10}(f_c),$$

where f_c is the carrier frequency, and d is the propagation distance in meters.

The intra-subnetwork communication channels are assumed to remain constant during the simulation, with a Rician small-scale fading coefficient. However, the inter-subnetwork channels are dynamic due to the movement of the subnetworks. The small-scale fading of these channels is modeled using the Jakes model [14], given by:

$$\xi(t) = \frac{1}{\sqrt{L}} \sum_{l=1}^{L} c_l e^{i(2\pi f_l t + \psi_l)} \,.$$

Here, L denotes the total number of multipath components, c_l specifies the gain of each path, f_l corresponds to the Doppler frequency, and $\psi_l \in (0, 2\pi)$ represents the phase shift associated with each component.

The wireless channels (16) thus change due to mobility in two ways. First, the distances change, thus slowly changing the distance-dependent path loss. Second, the small scale fading coefficients change rapidly, with considerable change when the subnetworks move a distance corresponding to a carrier wavelength.

During the pretraining phase, the LSTM network takes ten SINR observations, $\mathbf{\Gamma(t-10)},\ldots,\mathbf{\Gamma(t-1)}$, as inputs and predicts the next SINR value, $\hat{\mathbf{\Gamma}}(t)$, as its output for a given time slot t. The LSTM network consists of a hidden layer with 256 neurons. In the RL online learning phase, the state input to these networks at time t follows the structure defined in (6). The pretrained LSTM network is periodically updated after every 10 epochs to incorporate the latest environmental dynamics. Each actor and critic network consists of two hidden layers with 512 and 256 neurons, respectively. The activation function used in the hidden layers is *ReLU*$(\cdot)$, while the output layer of the actor network employs a *Sigmoid*$(\cdot)$ activation function. The networks are trained using the *Adam* optimizer, with learning rates set to $\alpha_\phi = 10^{-4}$ for the actor and $\alpha_\theta = 10^{-4}$ for the critic. A mini-batch size of 128 is utilized for training. The target networks are updated with a soft update parameter $\tau = 0.005$. The discount factor is defined as $\eta = 0.985$, and the training process runs for $E_{\max} = 1000$ epochs. A custom-built Python program is used to simulate the network elements, including their mobility and communication channels. The reinforcement learning framework is implemented using PyTorch to train the agents and evaluate their performance.

Figure 1 demonstrates the convergence behavior of the LSTM network during the offline training, with the mean squared error loss plotted on a logarithmic scale. The significant initial reduction in loss reflects the network's rapid adaptation to the data, with subsequent stabilization indicating effective learning. Figure 2 compares the predicted and actual SINR values on unseen data, illustrating the LSTM network's ability to accurately track the temporal patterns of SINR caused by small-scale fading. This highlights the model's robustness and predictive capabilities, showcasing its utility for SINR prediction in the reinforcement learning framework.

Figure 3 illustrates the reward dynamics during the online learning phase for the asynchronous update mechanism under the URLLC constraint. The plot demonstrates steady reward improvement across the subnetworks, with convergence to a similar performance level. This indicates the effectiveness of the proposed multi-agent DRL framework in achieving robust learning in the non-stationary environment. While asynchronous updates may incur increased training time due to sequential agent updates, this is offset by improved stability, reduced learning conflicts, and better convergence—crucial for URLLC scenarios where reliability is prioritized over training speed.

Figure 4 shows the CDFs of error probabilities for various update strategies, highlighting the performance of asynchronous and synchronous updates, both with and without URLLC constraints. The asynchronous update with URLLC constraints achieves the best performance, significantly outperforming all other strategies, including the synchronous

TABLE I: Simulation parameters

Parameter	Value
Carrier frequency f_c [GHz]	1.3
Subnetwork velocity [km/h]	40
Deployment area [m^2]	20 × 20
Number of subnetworks, M	3
Bandwidth	18 MHz
Minimum distance between subnetwork centers [m]	2
Transmit power P_t	20 dBm
Number of devices per subnetwork, J	3
Data symbols per subchannel	500
Rx noise figure [dB]	10
Number of subchannels, N	10
Modulation	QPSK
Code rate, R	0.4
Target network update τ	0.005
Packet loss probability target, p_0	10^{-5}
Radius of subnetworks [m]	1
Minimum distance between devices in sub-network [m]	0.2
Information packet size [bits]	400
Hidden layer neurons (actor, critic)	[512, 256]
Discount factor η	0.985
Batch Size	128
Activation functions	ReLU, Sigmoid
Learning rate (actor, critic)	10^{-4}

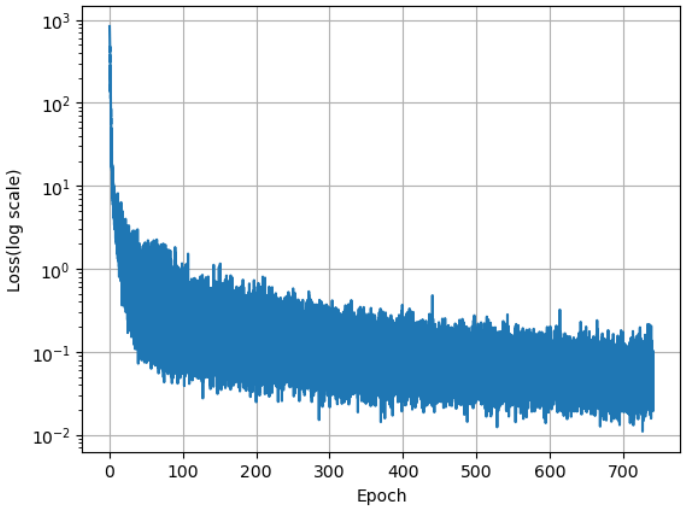

Fig. 1: Training performance: MSE loss on a log scale vs. epoch showing convergence.

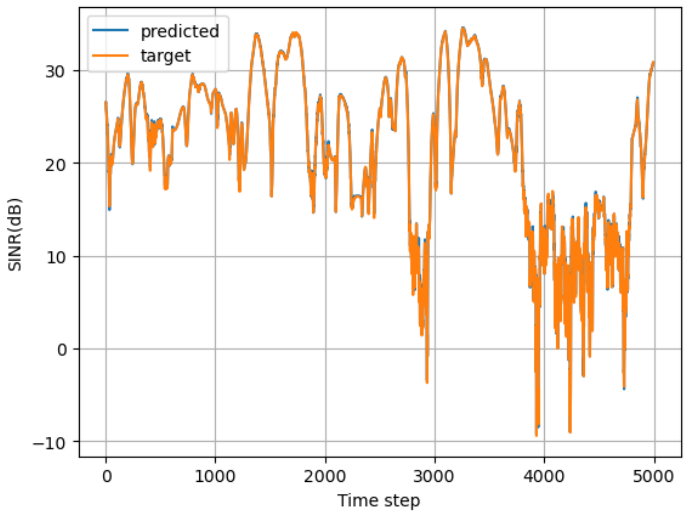

Fig. 2: Predicted vs. ground truth SINR on unseen test data, showing the LSTM model's generalization.

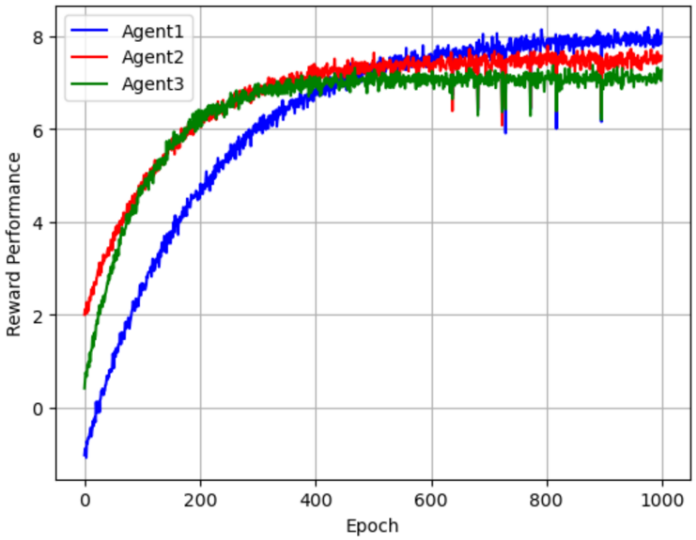

Fig. 3: Comparison of reward behaviors for the three sub-networks in one scenario.

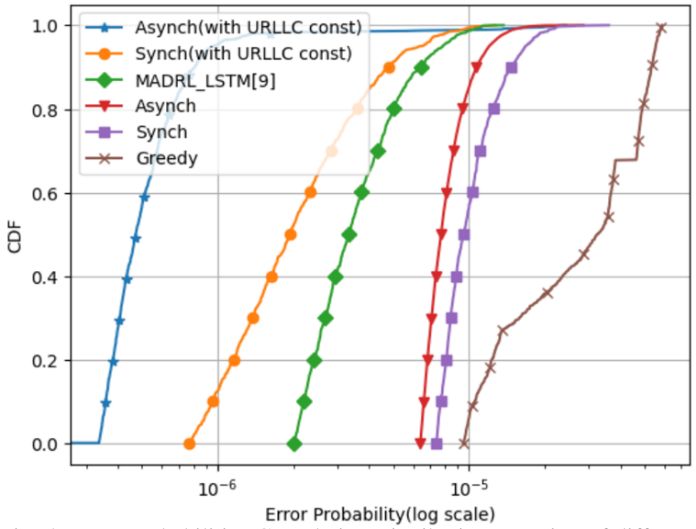

Fig. 4: Error probabilities Cumulative Distribution Function of different strategies.

update with URLLC constraints, the asynchronous update without URLLC constraints, and the synchronous update without URLLC constraints. This underscores the importance of URLLC constraints in guaranteeing resource allocation for every user, as strategies without these constraints exhibit relatively poor performance. Furthermore, the asynchronous update is inherently more stable than the synchronous update, as it avoids the uncertainty introduced by simultaneous agent updates. The proposed asynchronous mechanism with URLLC constraints outperforms the strategy from [9], where LSTM is integrated into the DRL agents in a synchronous MADRL setting—an approach that itself outperformed the Centralized Training with Distributed Execution(CTDE) framework—as well as the myopic greedy baseline. In the greedy approach, each AP chooses the scheduling strategy that provides the largest SINR to its worst user, given the current SINRs of its devices, i.e., solves $\max_{\mathbf{A}} \min_{j} \boldsymbol{\gamma}_j^{\mathrm{T}} \boldsymbol{a}_j$, where $\boldsymbol{\gamma}_j$ is the vector of SINRs of device j, and $\boldsymbol{a}_j$ represents its allocation vector.

V. Conclusion

This paper proposes an asynchronous multi-agent deep reinforcement learning framework for radio resource scheduling in mobile in-X subnetworks requiring URLLC services. By updating one agent at a time and predicting SINRs using a pretrained LSTM, the framework reduces learning variability, avoids policy conflicts, and adapts to dynamic interference without inter-agent coordination. Simulation results show that the asynchronous approach outperforms synchronous updates in reliability, resource efficiency, and interpretability. While current experiments use three subnetworks for tractability, the decentralized design supports scalability, and future work could assess performance under varying URLLC constraints and larger multi-agent settings.

Acknowledgement

This work was funded in part by Business Finland under the project "Extreme Machine Type Communications for 6G" and by the Research Council of Finland (grant 345109). We also acknowledge the Aalto Scientific Computing for the resources.

References

[1] M. A. Uusitalo & al., "6g vision, value, use cases and technologies from european 6g flagship project hexa-x," *IEEE Access*, vol. 9, pp. 160 004–160 020, 2021.

[2] A. M. Ramly, N. F. Abdullah, and R. Nordin, "Cross-layer design and performance analysis for ultra-reliable factory of the future based on 5g mobile networks," *IEEE Access*, vol. 9, pp. 68 161–68 175, 2021.

[3] G. Berardinelli & al., "Extreme communication in 6G: Vision and challenges for 'in-x' subnetworks," *IEEE Open Journal of the Communications Society*, vol. 2, pp. 2516–2535, 2021.

[4] Hamidi-Sepehr & al., "5G URLLC: Evolution of high-performance wireless networking for industrial automation," *IEEE Communications Standards Magazine*, vol. 5, no. 2, pp. 132–140, 2021.

[5] Nasrallah, Ahmed & al., "Ultra-low latency (ULL) networks: The IEEE TSN and IETF detnet standards and related 5G ULL research," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 1, pp. 88–145, 2019.

[6] Du, Xiao & al., "Multi-agent reinforcement learning for dynamic resource management in 6G in-X subnetworks," *IEEE Transactions on Wireless Communications*, vol. 22, no. 3, pp. 1900–1914, 2023.

[7] R. Adeogun and G. Berardinelli, "Distributed channel allocation for mobile 6G subnetworks via multi-agent deep Q-learning," in *IEEE Wireless Communications and Networking Conference (WCNC)*, 2023, pp. 1–6.

[8] Robaglia, Benoît-Marie & al., "Seqdqn: Multi-agent deep reinforcement learning for uplink urllc with strict deadlines," in *2023 Joint European Conference on Networks and Communications & 6G Summit (EuCNC/6G Summit)*, 2023, pp. 623–628.

[9] A. Srinivasan, U. Singh, and O. Tirkkonen, "Multi-agent reinforcement learning approach scheduling for in-x subnetworks," in *2024 IEEE 100th Vehicular Technology Conference (VTC2024-Fall)*, 2024, pp. 1–7.

[10] Khosravirad, Saeed R. & al., "Communications survival strategies for industrial wireless control," *IEEE Network*, vol. 36, no. 2, pp. 66–72, 2022.

[11] X. Zhang, Q. Zhu, and H. V. Poor, "Non-asymptotic performance for finite blocklength coding over Nakagami-m channels," in *IEEE International Conference on Communications (ICC)*, pp. 1–6.

[12] T.P. Lillicrap & al., "Continuous control with deep reinforcement learning." in *ICLR*, 2016.

[13] 3GPP, "Study on channel model for frequencies from 0.5 to 100 GHz," Tech. Rep. TR 38.901, V16.1.0, Dec. 2019.

[14] William C. Jakes, Ed., *Microwave Mobile Communications*. John Wiley & Sons, 1975.