Federated Interference Management for Industrial 6G Subnetworks

Master Thesis

Bjarke Bak Madsen

Aalborg University
Department of Electronic Systems
MSc. in Signal Processing and Acoustics
With specialization in Signal Processing and Computing

AALBORG UNIVERSITY
STUDENT REPORT

Department of Electronic Systems

Signal Processing and Computing

Aalborg University

http://www.aau.dk

Title:

Federated Interference Management for Industrial 6G Subnetworks

Theme:

Radio resource optimization with federated reinforcement learning

Project Period:

Spring semester 2023

Project Group:

1071

Participant(s):

Bjarke Bak Madsen - 20182368

Supervisor(s):

Ramoni Ojekunle Adeogun

Copies: 1

Page Numbers: 100

Date of Completion:

June 2, 2023

Abstract:

6G in-X subnetworks are short-range low-power cells envisioned to support extreme communication requirements for data rate, latency, and reliability. However, interference represents a major limiting factor to extreme communication in dense deployments of in-X subnetworks. Recent studies have proposed interference management solutions based on multi-agent reinforcement learning, where the radio resource optimization problem is modeled as a multi-Markov decision process. The studies have been based on centralized or distributed training. While centralized training benefits from the experiences of all subnetworks during the training, it may lead to compromised privacy and security issues since it requires sharing of measurements between the subnetworks and a centralized agent. In contrast, agents in distributed training rely solely on only local measurements of the environment for decision which often leads to convergence problems. To overcome these challenges, a client-to-server horizontal federated reinforcement learning framework is proposed, where knowledge is shared implicitly through locally trained model weights. Simulations in an industrial environment using 3GPP propagation models have shown promising results for quick convergence, marginal performance improvement, and robustness to non-stationary environments.

Contents

Preface

This Master's thesis was written by Bjarke Bak Madsen of the Department of Electronic Systems master program in signal processing and acoustics, in the time period 1st of February to the 2nd of June 2023.

This thesis is produced in Overleaf LaTeX and all figures are produced by the author with draw.io software or scripts made with Python 3.10.4. References to figures are numerated according to chapters. Citations are made according to guidelines from IEEE, where a detailed list of all citations can be found in the bibliography. All acronyms are specified on first occurrence, where a list with all used acronyms can be found in the glossary.

The author would like to thank Ramoni Ojekunle Adeogun for excellent supervision throughout the development of this thesis. Furthermore, the author would like to thank the AI for Communications at AAU research group, especially Daniel Abode, for ideas and feedback.

Aalborg University, June 2, 2023

Glossary

3GPP 3rd Generation Partnership Project.

AD Automatic Differentiation.

Adam Adaptive Moment Estimation.

AP Access Point.

AWGN Additive White Gaussian noise.

CDF Cumulative Distribution Function.

CGC Centralized Graph Coloring.

DDQN Double Deep Q-Network.

DL Down-link.

DNN Deep Neural Network.

DP Dynamic Programming.

DQN Deep Q-Network.

FL Federated Learning.

FRL Federated Reinforcement Learning.

GNN Graph Neural Network.

HFL Horizontal Federated Learning.

HFRL Horizontal Federated Reinforcement Learning.

IEEE Institute of Electrical and Electronics Engineers.

IID Independent and Identically Distributed random variables.

LoS Line-of-Sight.

MADDQN Multi-Agent Double Deep Q-Network.

MAE Mean Absolute Error.

MAPPO Multi-Agent Proximal Policy Optimization.

MARL Multi-Agent Reinforcement Learning.

MC Monte Carlo.

MDP Markov Decision Process.

MMDP Multi-Markov Decision Process.

MSE Mean Square Error.

POMMDP Partially Observable Multi-Markov Decision Process.

PPO Proximal Policy Optimization.

ReLu Rectified Linear Unit.

RL Reinforcement Learning.

SARSA State–action–reward–state–action.

SGD Stochastic Gradient Descent.

SINR Signal-to-Interference plus Noise Ratio.

TD Temporal-Difference.

UL Up-link.

VFL Vertical Federated Learning.

VFRL Vertical Federated Reinforcement Learning.

Chapter 1

Introduction

The evolution of wireless communication has transformed the way in which we communicate and connect with the world around us. The increasing demand for faster and more reliable communication has resulted in significant improvements in wireless communication networks, such as the development of 5th generation (5G) technology. However, as technology continues to expand, the need for even more efficient and reliable communication networks has become apparent [1]. This is where 6th generation (6G) technology comes into play. The development of 6G technology promises to revolutionize wireless communication networks by providing ultra-reliable communication, higher data rates, and lower latency [2].

With the advancements in technology, the opportunities for applications using wireless communication are limitless [2]. For example, the use of virtual and augmented reality technologies in fields such as entertainment, education, healthcare, and manufacturing, enables immersive experiences and enhances training, visualization, and design capabilities. Similarly, the development of personalized healthcare solutions are supported, such as wearable devices that monitor health indicators and transmit data to healthcare providers, improving the accuracy and efficiency of medical diagnoses and treatments. In addition, wireless networks can be used for real-time environmental monitoring, such as air and water quality, climate change, and natural disasters.

To achieve the promise of ultra-reliable communication, higher data rates, and lower latency in 6G networks, new communication technologies must be developed. Recently, a new concept referred to as 6G short-range low-power in-X (inside-everything) subnetworks [3, 4, 5, 6] is envisioned as a viable technology for supporting demanding requirements inside entities such as robots, production modules, vehicles, or even human-body. Since deployment of in-X subnetworks can become dense, e.g., in-X subnetwork inside vehicles at the intersection of a busy road, or inside bodies of persons in a crowded event, the development of intelligent approaches for managing interference via optimized resource allocation has been the focus of active research in the last few years. See e.g., the works in [7, 8, 9]. As wireless technologies continue to proliferate, ensuring data privacy and security is becoming more important and at the same time challenging [2, 10]. Many of the applications that rely on advanced wireless communication networks involve sensitive personal or environmental data, and protecting this data is crucial to ensuring the trust and reliability of these systems. This is particularly true for the in-X subnetworks, which are to be deployed inside entities such as cars or human-bodies as stated earlier. To this end, techniques for resource allocation must intelligently guarantee the privacy and security of the wireless devices in the network in addition to the being able to pro-actively adapt to changing wireless environments.

1.1 Thesis Objective

Radio resource optimization involves allocation of limited radio resources, e.g., bandwidth, transmit power, or spectrum occupancy for multiple co-existing radio devices, while maximizing a specific metric such as Signal-to-Interference plus Noise Ratio (SINR) or spectral efficiency [5, 9]. Traditional solutions based on hard-coded heuristics, game theory, and geometric programming face computational challenges in scenarios with a high number of co-existing devices. Recently, machine learning, particularly Reinforcement Learning (RL), has emerged as a potential solution where no pre-generated labeled data-sets are required and can be adapted to dynamic communication environments [5, 9].

An important question is to what extent RL-based solutions for resource allocation can achieve extreme communication requirements of several co-existing autonomous subnetworks, such as high availability, low latency, and low packet error rate. To address this question, distributed algorithms for resource optimization must be developed without compromising the privacy of any subnetworks. A well-known learning framework referred to as Multi-Agent Reinforcement Learning (MARL) attempts to learn the optimal policy for solving complex decision problems via interaction with complex and dynamic environments [9]. Emerging in research, Federated Learning (FL) is a privacy-preserving collaborative learning framework where co-existing devices train a common global model without explicit exchange of sensitive local data, but by periodically aggregating local model weights [10].

1.1.1 Project Scope

This thesis aims to explore the applications of a combination between MARL and FL referred to as Federated Reinforcement Learning (FRL) for solving the complex problem of radio resource optimization in terms of interference management for 6G ultra-reliable communication systems [10]. By investigating this hypothesis, this research aims to contribute to the development of efficient and reliable radio resource optimization solutions for 6G wireless communication systems concerning data privacy. The developed solution based on FRL will be evaluated through simulation and compared with state-of-art algorithms for interference management in 6G in-X subnetworks.

1.1.2 Structure

The first chapter of this master thesis has laid the foundation for a comprehensive exploration of the intersection between 6G communication and resource optimization with a focus on data privacy, utilizing FL methods. The remaining parts of this thesis are organized as follows:

Chapter 2 will provide literature review of an existing branch of research in the field of 6G communication technology and interference management. **Chapter 3** will explore the fundamentals of RL and its potential for resource optimization by providing an overview of the state-of-the-art concepts and methods in the field. **Chapter 4** will present the methodology employed in this study, outlining the system model and problem formulation. In **chapter 5**, an approach to a FRL framework will be proposed and described. In **chapter 6**, the results of this study will be presented and analyzed, providing insights into the potential benefits and challenges associated with this approach. Finally, **chapter 7** will present the conclusions of this study, highlighting the key findings and offering recommendations for future research.

Chapter 2

Literature Overview

In recent research, the concept of 6G in-X subnetworks has emerged as a promising paradigm that envisions the replacement of critical operations running over wired connection in entities such as industrial robots, production modules, and vehicles with wireless communication [5]. In-X subnetworks promises the support of more demanding requirements than what is possible with former 5G solutions. Since the deployment of in-X subnetworks can easily become dense, interference represent a major hindrance to achieving the target requirements. Interference management therefore represent an important component of 6G in-X subnetworks design. This chapter provides an overview of current literature and state-of-the-art solutions for 6G in-X subnetwork interference management.

2.1 6G in-X Subnetworks

With the rise of large scale 5G radio technology deployment, research on 6G radio technology was initiated. By satisfaction of extreme communication requirements for low latency, high throughput, and high reliability, 6G is expected to attain high practical standards that meet performance requirements of more demanding and critical networks of the future [5]. In this section, an overview of 6G in-X subnetworks, applications, and requirements is presented.

2.1.1 Extreme Communication

It is desired to extend the current concept of connectivity with user wearables, implants, vehicular components, and other intelligent machines, where demanding computations can be distributed over the network. The vision of future communication involves using mixed reality, where requirements for link throughput rise to the tens to hundreds of Gbps [5]. In early work [3], the vision of 6G in-X subnetworks for life-critical and mission-critical communication is presented with extreme short cycles less than 100 µs and stringent reliability of six nines, where the requirement of a multi-GHz centimeter-wave spectrum is identified. In [4], a wireless asynchronous real-time system with identification of extreme reliability and short cycle time is presented for industrial sensor-actuator or intra-vehicle communication. Access mechanisms to a multi-GHz centimeter-wave spectrum is identified to potentially easing services in dense scenarios with coexisting subnetworks. Extreme communication in terms of requirements can be summarized as the following three definitions [3, 4, 5].

- **Data rate:** A throughput of tens to hundreds of Gbps per link.
- **Latency:** A minimal response delay of < 100 µs.
- **Reliability:** Minimum annual communication service availability of six nines.

The extreme communication requirements for different use cases may vary, however at least one of the requirements must be satisfied. Life-critical systems require high reliability and low latency, whereas entertainment systems often require high data rates with low latency [5].

2.1.2 Definition of Subnetworks

As stated earlier, 6G in-X subnetworks are short-range low-power cells installed in entities such as production modules, vehicles, human bodies, or houses [5]. The goal is to enable wireless communication support for critical control operations in factories (e.g., control of robot arms), vehicles (e.g., for ignition or brake control), and human-bodies (e.g., for wireless pace-maker). The cells are referred to as 6G in-X subnetworks, where *X* denotes the domain of deployment such as in-robot, in-vehicle, in-body, or in-house. Figure 2.1 illustrates the 6G in-X subnetwork concept in different application domains. The 6G in-X subnetwork concept, features the following main characteristics [5].

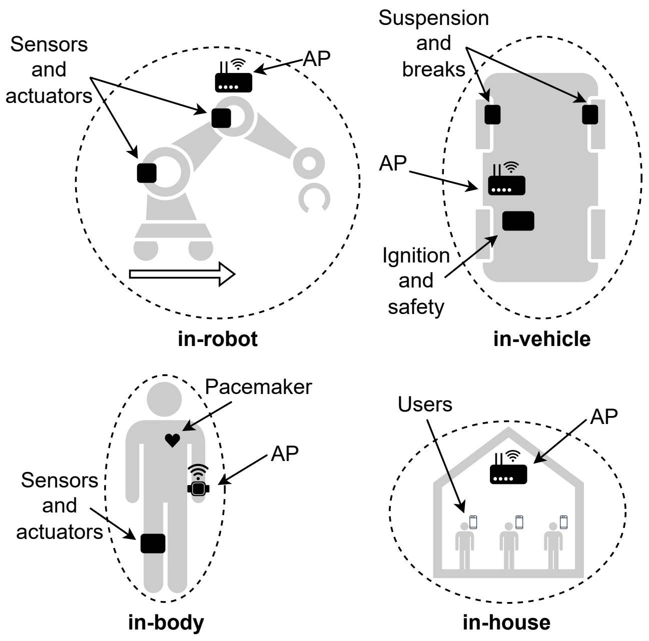

Figure 2.1: Illustrated examples for subnetworks: in-robot, in-vehicle, in-body, and in-house.

- At least one out of data rate, latency, or reliability requirements must be fulfilled in the terms of extreme communication.
- In both Up-link (UL) and Down-link (DL) scenarios, the range must not be greater than 10 m to achieve a very low transmit power of 0 dBm or below. The range can be extended by communication between subnetworks.

- The operations performed in each subnetwork is controlled by an Access Point (AP), in a tree or star topology hierarchical structure.
- Some of the subnetworks could be considered mobile in the environment, due to the nature in cases such as in-vehicle or in-body systems.

As each in-X subnetwork is a part of a larger network, traffic with different characteristics in terms of data flow must be handled. The general case of high critical flow with latency $\ll$ 1 ms and a reliability beyond five nines. Another case is medium critical flow with latency $>$ 1 ms and a maximum reliability of five nines. The final case where non-critical flows have non-strictly limited latency. The AP has integrated control processing capabilities, where measurements received from sensors can be processed and actions can be transmitted to the actuators. Requirements for high critical flows does not allow for external processing, however medium critical flows can eventually be processed in an external server [5].

Deployment of Subnetworks

Consider the exemplary in-X subnetwork scenarios presented in figure 2.1. The use of subnetworks in different everyday scenarios will undoubtedly intersect with each other at some point, such as in-body and in-vehicle. As with other wireless systems, deployment of subnetworks can be generalized into two subcategories, coordinated and uncoordinated networks as illustrated on figure 2.2 [5].

Coordination of a network involves consideration and planning of radio performance in terms of range when placing the radio units [5]. Coordinated deployment may be considered useful for stationary subnetwork scenarios, such as industrial in-production or assembly line modules with static placement.

Typically deployment of in-robot or in-vehicle subnetworks may be characterized by a dense mobile uncoordinated setting, where wireless technology is a necessity

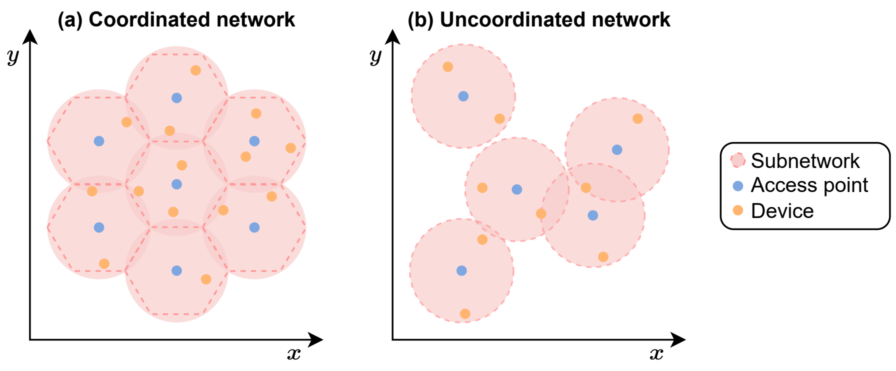

Figure 2.2: Exemplary illustration of (a) coordinated and (b) uncoordinated subnetworks.

for free movement [5]. The robots may carry out industrial tasks as efficiently as possible, with no regard to any other subnetwork. This is considered as uncoordinated radio deployment, where overlaps in subnetworks enables interference.

The approach to interference management depends on the deployment of subnetworks. In the coordinated scenario, controlling each subnetwork enables sharing information effectively, and scheduling ahead in time for an optimal use of radio resources [5]. However, the uncoordinated scenario deployment is random and comparable chaotic where outcomes and capabilities of subnetworks are unknown, and difficult to synchronize [5]. Subnetworks may be heterogeneous where radio and hardware resources vary, and be in uncoordinated motion enhances interference further. Hence, more adaptive approaches for interference management are required.

2.2 Interference Management

Interference management aims to allocate available resource such optimal communication is achieved. The objective is to optimize specific performance metrics, subject to practical resource constraints, by adjusting the available radio resources. Such allocation decision problems typically involve non-convex objective functions, known to be NP-hard where there is no universal optimal solution [11]. Interference management in networks can be performed in several ways depending on specific requirements, restrictions, or available technology. In this section the main resource management domains and frameworks will be presented followed by a reflection to the perspective of 6G in-X subnetworks and current literature.

2.2.1 Interference Management Domains

The technique of managing interfering components in a network is based on allocation of available resources. Radio resources may be categorized in spatial, time, frequency, and power domains [5, 12]. Following in this section, the resource management domains will be discussed.

Spatial Domain Interference Management

Interference management in the spatial domain refers to concepts of allocating resources in space, subject to minimizing interference. The power of a transmitted signal decays during propagation, where tolerable interference levels can be achieved by separating and spreading out antennas [13]. However, this is impractical in uncoordinated scenarios where spatial separation cannot be guaranteed.

Omnidirectional antennas provide uniform coverage in all directions surrounding an antenna, at the cost of emitting energy in directions where no recipients are located. By physically restricting beam-widths, a directional antenna beam is formed. Given that the beam is narrow relative to a omnidirectional pattern and no interfering sources obstruct the path, interference is reduced. However a Line-of-Sight (LoS) must be maintained between antennas, either by fixing or deploying steering mechanisms in antennas at the cost of restriction or complexity [14].

Another method to control directivity is beamforming [14]. The antenna size is inversely proportional to frequency, making it possible to arrange small arrays of antennas for 6G applications operating at relative high frequencies. Beamforming signal processing techniques adjusts signal phase and amplitude at each antenna element, generating a directional transmission or reception pattern. In [15], an adaptive beamforming interference control is proposed for a cognitive radio system using smart antenna arrays. Interference is reduced by directing the transmitter towards the desired receiver with beamforming, increasing strength of the received signal while minimizing interference and noise.

Time Domain Interference Management

Temporal interference management aims to separate communication in the time domain, effectively avoiding interference sourced from simultaneously active transmissions. Time separation of transmissions does however require queues, increasing the latency for participants. Instead, the concept may be useful for separation of UL and DL frames, repetitive short-term transmissions schemes, or short-term interference management. In [12, 16], radios dynamically adapts to changes in the wireless environment over time. Hence, the time domain is utilized to optimize the use of frequency resources and transmit levels over different time intervals using time-division multiple-access methods.

Frequency Domain Interference Management

Management in the spectral domain refers to selection of the scarce available frequency resources. Transmissions that are completely separated in operating frequencies does not interfere with each other. Every antenna pair could be granted a section of the frequency band to transmit on, however the resources are finite and properties vary. The propagating characteristics are mainly proportional to frequency. A large proportion of frequencies are licensed and regulated with a cost, and the remaining unlicensed ranges may suffer from overpopulation where interference is likely to occur [5].

Recall the effect of spatial separation, two transmitting antennas can share frequency if the spatial separation is great enough for tolerable interference. This suggests the use of an adapt allocation scheme to only switch frequency of transmissions that suffers from interference, referred to as interference avoidance [13]. A great increase in spectral efficiency is achieved if multiple transmissions can share frequency, given interference is avoided due to spatial separation. In [12], spectrum sensing and sharing is used for dynamic spectrum access to mitigate interference for cognitive radios. Frequency reuse methods for mobile cellular networks are discussed in [17], followed by a discussion on interference cancellation techniques with the use of more sophisticated algorithms. With focus on ultra-dense networks, [13] propose interference management with a frequency allocation strategy that divides the available frequency spectrum into different frequency reuse patterns. Joint interference management for ultra-dense small-cell networks are proposed in [16], involving frequency domain coordination techniques such as dynamic spectrum allocation and interference alignment.

Power Control

With less transmission power, the range of the signal decreases as well as the interference. Reducing transmission power is similar to the effect of separating antennas, assuming that the recipient is kept within tolerable range of the signal power. A power control system adjusts the signal powers to be constant at a recipient subject to communication requirements, where no more interference than necessary is generated [14]. In [12], the power control is based on a channel gain estimate sent by the receiver, where the transmit power level is minimized based on the desired signal strength at the recipient. In [16], interference management is jointly investigated from a multi-domain perspective using the water-filling algorithm for power control optimization.

2.2.2 Interference Management Frameworks

Allocating resource for any domain typically requires general knowledge about the network. To guarantee correct time alignment in the long term, a framework to gather such knowledge must be established. In this section three interference management frameworks for time coordination will be presented.

Centralized Framework

In the case where radios are within a wide area network, a centralized coordinator can be used to manage allocation and timing. It is expected that a coordinator can increase spectral efficiency, however participant must wait for instructions and the connection must be stable [5].

Implicit and Distributed Framework

Consider a congested group of radios out of any wide area network. In order for each participant to manage local resources correctly, implicit and distributed coordination is required. Unlicensed frequency bands are typically regulated with protocols such as listen-before-talk, where each participant must sense the channel and postpone transmission if occupied. However, this may not be a viable solution in the need of critical traffic. A disruptive protocol could be introduced to prioritize critical traffic which may result in decreasing spectral efficiency [5].

Hybrid Framework

A combination of distributed and centralized interference management can be proposed as a hybrid approach. A central coordinator manage subnetworks within wide area network coverage, and switches to distributed and implicit coordination when the connection is unstable. However to avoid interruptions for life-critical services, the switching must be performed seamless [5]. A hybrid radio resource management framework for 6G crowds in a umbrella network are introduced in [16], where a global decision agent exploits information from each subnetwork for deciding on policy and actions, while each subnetwork are equipped with a local decision agent.

2.2.3 Interference Management for 6G in-X Subnetworks

A review of the current literature for 6G in-X subnetworks may enlighten current possibilities and challenges involved with interference management. However, first a reflective discussion of the interference management domains in respect to 6G in-X subnetworks is required to identify drawbacks that rule out entire domains.

Interference Management Domains for 6G in-X Subnetworks

Spatial separation of radios or directional antennas in the scenario of uncoordinated mobile subnetworks is considered to be an impractical. The nature of some 6G in-X subnetworks include a small form factor, referring to a more compact and portable device where small antennas can be integrated [5]. Beamforming may be ineffective given the limited form factor for APs and devices preventing a larger number of antennas, especially for lower carrier frequencies [14].

The time domain may not be an exclusively viable solution domain for extreme communication requirements in terms of low latency due to delays from a schedule. However, as mentioned earlier, interference management can be based on a multi-domain approach, where resources are optimized over intervals of time, and has proven to be very efficient [12, 16].

Battery capacity is critical for some subnetworks, where a power control may indeed be beneficial. Here power can be adjusted in terms of user requirements such as throughput, energy efficiency, and spectral efficiency while minimizing interference [12].

Resources in the frequency domain may be sparse, but suffers from less significant drawbacks in comparison to the other domains. The operational frequency domain may be divided into several chunks to be dynamically selected or assigned to subnetworks, such interference is minimized [12, 17].

Current Literature on Interference Management in 6G in-X Subnetworks

Several interference management solutions for different wireless systems has been proposed in literature. The new field of 6G in-X subnetworks has received increasing attention lately, as radio concepts with potential to support extreme communication requirements [3, 5, 6]. Previous work on 6G in-X subnetworks will be presented in this section as three categories: heuristics, machine learning, and RL.

Heuristic methods can be used to solve interference management problems by generating near-optimal solutions based on simple rules or algorithms. These methods can provide good results with low computational complexity. In [18], three heuristic based algorithm viz are presented. The first is ϵ-greedy channel selection, where each controller selects the best channel with a probability for choosing a random action instead. Second is nearest neighbor conflict avoidance, where each controller selects channels that is not occupied by subnetworks producing the highest interference power. Final is minimum SINR guarantee, where controllers attempt to select the worst channel satisfying a required threshold value. If no threshold are met, the channel with least interference is chosen. In [18, 19], Centralized Graph Coloring (CGC) is presented as a benchmark algorithm, where graph coloring techniques are used to select the best channel available. However,it is a impractical centralized approach based on a global view on the system. A distributed interference-aware dynamic channel selection is presented in [19], where the number of packet repetitions are based on channel conditions. A secondary exclusive group of channels are used when high interference levels are anticipated. In [20], channel selection is approached based on centralized selective graph constructions, inspired by the concept of fractional frequency reuse. Here graph identification and construction approaches for interference mitigation is discussed.

In some cases, machine learning may provide better quality of service in comparison to heuristics by learning from historical data and adapting to changing network conditions. Patterns can be identified in large data-sets and used to make predictions and optimize resource allocation. For example, network traffic can be predicted, network congestion identified, and resources can be allocated to meet demands of different applications. A hybrid optimization approach can also be defined where machine learning results can be used as input to heuristics algorithms or the other way around. In [21], a Deep Neural Network (DNN) is successfully trained in offline simulations using CGC with mobile subnetworks, which after training is deployed for real-time distributed channel selection. A novel solution for centralized power control is presented in [7], where instead of using channel state information as conventional approaches, the decision is based on positioning information using Graph Neural Network (GNN). Such information is usually known by a centralized coordinator. Benchmarks such as maximum transmit power for all links, weighted minimum mean square error, and purposely inputting errors to the GNN power control are used and discussed. It is shown that the limited information provides sufficient data for the proposed method, and achieves similar results to existing impractical solutions.

Another approach may be RL, which involves an agent learning to make decisions based on observations [22, 23]. Radio resources can be allocated in dynamic and uncertain environments. In [9] a joint allocation of channel and transmit powers are based on a distributed multi-objective optimization problem that maximize the capacity of subnetworks. An approach of Q-learning for multiple agents based on limited sensing information is proposed, along with a rule-based algorithm termed Q-heuristics. Both methods indicate good performance and robustness in comparison to heuristic and former machine learning methods. A distributed framework for Multi-Agent Reinforcement Learning (MARL) resource management based on GNN termed GA-Net is presented in [24], where the sum of interference relationships in channels is used for centralized training methods. Another MARL approach is proposed in [25], for a distributed channel allocation. A centralized training procedure is adapted for local training of Deep Q-Network (DQN) models, performed at a central location. It is shown that convergence and stability may be enhanced with the use of a Double Deep Q-Network (DDQN).

The choice between heuristics, machine learning, and RL for radio resource allocation depends on the specific problem being addressed, the available data and resources, and the trade-off between accuracy, computational complexity, and real-time decision-making. Additionally, sharing sensitive data between subnetworks may not be a viable practical solution due to user privacy concerns and must be considered. From other works, [10] presents a survey about the concept of FL, where MARL techniques allows agents to collaborate and jointly train a model without directly sharing any sensitive data. This is new to the field of 6G in-X subnetworks, and may be investigated for compatibility.

Chapter 3

Reinforcement Learning Methods

This chapter provides an overview of RL techniques, which are critical for developing efficient and reliable solutions for 6G in-X subnetworks. RL is explored in detail, where each in-X subnetwork is considered as an agent that interacts with the environment and learns optimal policies. Furthermore, the vision and framework of FL will be presented.

3.1 Reinforcement Learning

The goal of RL is to learn some desired behavior from interactions with a system. This section provides the fundamental knowledge, with concepts and solutions, needed for understanding and implementing RL systems.

3.1.1 Agent-Environment Interaction

The process of picking an action can be described as a Markov Decision Process (MDP), which is an underlying framework mathematically defined as the four-tuple $(\mathcal{S}, \mathcal{A}, P, R)$ [22]. A state $s_t \in \mathcal{S}$ is observed in the environment at time t. The agent interacts with the environment by taking action $a \in \mathcal{A}(s)$, yielding a reward $R(s, s', a)$, which depends on transitioning from state s to s', by taking action a. The interaction can also cause a state transition, where $P(s'|s, a)$ is the probability of transitioning from state s to s' when taking action a.

When modeling RL problems as MDPs, fundamentally two systems interact together. The first system known as an agent must learn an nearly-optimal policy that maximize the reward by interacting with the second system. The second system is the environment that presents a situation known as a state to the agent. A reward is granted to the agent if the action leads to a desired state. A conceptual illustration is shown in figure 3.1 [22].

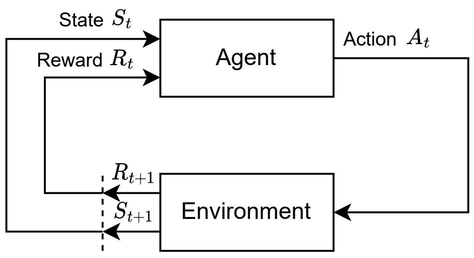

Figure 3.1: The agent-environment interaction model for MDP.

For simplicity, assume the interactions between agent and environment is defined by a sequence of discrete time steps, $t = 1, 2, \ldots$. The agent observes environment state S_t and take action A_t to receive the reward R_{t+1}. The environment change state to S_{t+1} where the agent will take new action A_{t+1} to continue the circular interaction loop. The probability that action a in state s at time t will lead to state s' at time $t + 1$ is given by (3.1) [22].

$$p(s', r|s, a) = Pr(S_{t+1} = s', R_{t+1} = r|S_t = s, A_t = a) \tag{3.1}$$

Which is known as the Markov property of MDPs. The state transition probability to a new state s' depends only on the current state s and action a, that is conditionally independent of all previous states an actions.

Reward and Return

The goal of the agent is to learn a policy, that maximizes the cumulative reward. It is thus critical that the rewards truly indicate what is wanted to be accomplished. The agent should know *what* to achieve, and not *how* to achieve it. In general, it is the expected return, $G_t = R_{t+1} + R_{t+2} + \cdots + R_T$, that is maximized, where the last time step T indicates the repeated interaction naturally breaks into sub-sequences known as episodes or trials. For $T = \infty$ it is known as a continuous task, where the reward might go towards infinity. To counter this, a discounted return is formed,

$$G_t = \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k \qquad \text{(Finite)} \tag{3.2}$$

$$G_t = \sum_{k=0}^{\infty} \gamma^{k} R_{t+k+1} \qquad \text{(Infinite)} \tag{3.3}$$

where $0 \leq \gamma \leq 1$ is the discount rate, used to control how farsighted the agent is regarding future rewards [22]. As γ approaches 1, future rewards are weighted more strongly.

Policies and Value Functions

A measure of how good in terms of future rewards it is for an agent to be in a given state is estimated with value functions. These are defined with respect to a particular way of acting, known as policies π, which formally maps states to probabilities of selecting each possible action [22].

$$\pi(a|s) = Pr(A_t = a|S_t = s) \tag{3.4}$$

The expected return when starting in state s and following π is defined with the state-value function $v_\pi(s)$,

$$\begin{aligned} v_\pi(s) &= \mathbb{E}_\pi \left[G_t | S_t = s, A_t = a \right], \quad \forall s \in \mathcal{S} \\ &= \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s \right] \end{aligned} \tag{3.5}$$

where $\mathbb{E}_\pi[\cdot]$ denotes the expected value of a random variable, given the agent follows policy π for any time step t [22]. Similarly, action value function denoted

$q_\pi(s,a)$ define the value of taking action a in state s under policy π [22].

$$\begin{aligned} q_\pi(s,a) &= \mathbb{E}_\pi\left[G_t | S_t = s, A_t = a\right], \quad \forall s \in \mathcal{S}, a \in \mathcal{A} \\ &= \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s, A_t = a\right] \end{aligned} \tag{3.6}$$

The goal is to learn the policy that maximize the value function by interacting with the environment. The goal is not to maximize reward from cycle to cycle, but in the long run [10].

$$\pi^* = \arg\max_\pi v_\pi(s), \quad \forall s \in \mathcal{S} \tag{3.7}$$

One policy is the ϵ-greedy, where in most instances the maximal estimated action value is chosen, but with probability $0 \le \epsilon \le 1$ an exploratory random action is selected instead [22]. The probability ϵ sets the order of exploration, which is beneficial in early stages of learning to obtain knowledge about the environment ($\epsilon \to 1$). When sufficient knowledge has been collected, the agent should start exploiting the policies instead of exploring the environment ($\epsilon = 0$). However, if the environment proves to be non-stationary, it would be more efficient to never stop exploring [22].

3.1.2 Value-Based Methods

Value-based learning also known as tabular methods refer to problems where the state an action spaces are small enough for approximate value functions to be presented as arrays and tables [22, 23]. The table associates states with actions based on previous experiences, and is recursively updated with new values to improve estimates and the policy. Three fundamental preliminary knowledge viz are required to solve MDPs with tabular methods, i.e., Dynamic Programming (DP), Monte Carlo (MC) methods and Temporal-Difference (TD) learning [22].

DP refers to a group of algorithms that can be used to compute the optimal policies, given a perfect model of the environment as a MDP [22]. MC methods is used for solving learning problems based on averaging sample returns. This only applies for episodic tasks, that is if the experience can be divided into episodes, that eventually terminate no matter what actions are selected. MC methods does not require a model of the environment, but are not suited for online computations [22]. TD learning is a combination of ideas from DP and MC. It is not constrained by prior knowledge and are able to perform online computation, at the cost of increased analytical complexity [22]. To enhance insight on these applications, two methods with model-free TD viz is be presented, State–action–reward–state–action (SARSA) and Q-learning.

State-action-reward-state-action

Assuming the agent will keep following the same policy that was used to generate the experience, the update rule for SARSA defined in the following [22].

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)\right] \tag{3.8}$$

SARSA is an on-policy TD control method, meaning the $Q(S, A)$ function is learned from actions using current policy $\pi(a, s)$. Thus by using this method, the action value estimate does not converge to the optimal action value but rather a sub-optimal action value function by exploration. SARSA is presented as pseudo-code in algorithm 1.

Q-learning

Q-learning differs from SARSA by following an optimal policy to generate experiences, and using another policy to update Q-values. The update rule for Q-learning

Algorithm 1 SARSA [22]

```
Input: Step size $\alpha \in (0,1]$ and small $\epsilon > 0$
Initialise: $Q(s,a)$ for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$
 1: loop for each episode
 2:     Initialize: $S$
 3:     Choose $A$ from $S$ using policy derived from $Q$ (e.g., $\epsilon$-greedy)
 4:     loop for each step in episode
 5:         Take action $A$, observe $R$, $S'$
 6:         Choose $A'$ from $S'$ using policy derived from $Q$ (e.g., $\epsilon$-greedy)
 7:         $Q(S,A) \leftarrow Q(S,A) + \alpha\,[R + \gamma Q(S',A') - Q(S,A)]$
 8:         $S \leftarrow S'$; $A \leftarrow A'$;
 9:     end loop when $S$ is terminal
10: end loop
```

is given as the following [22].

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)\right] \tag{3.9}$$

Q-leaning is an off-policy TD control method, meaning the behavior policy used to control the agent during learning is different from the estimation policy whose value is being learned. Q-learning is presented as pseudo-code in algorithm 2.

Algorithm 2 Q-learning [22]

```
Input: Step size $\alpha \in (0,1]$ and small $\epsilon > 0$
Initialise: $Q(s,a)$ for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$
1: loop for each episode
2:     Initialize: $S$
3:     loop for each step in episode
4:         Choose $A$ from $S$ using policy derived from $Q$ (e.g., $\epsilon$-greedy)
5:         Take action $A$, observe $R$, $S'$
6:         $Q(S,A) \leftarrow Q(S,A) + \alpha\,[R + \gamma \max_a Q(S',a) - Q(S,A)]$
7:         $S \leftarrow S'$; $A \leftarrow A'$;
8:     end loop when $S$ is terminal
9: end loop
```

3.1.3 Policy-Based Methods

If an environment generates new states on the run, the state space can become too large for tabular methods. Here the agent must try to generalize knowledge about known states to unknown states. The type of generalization required is formally known as policy-based methods or function approximation, where the idea is to parameterize a policy that can generate optimal actions based on historical or available observations from the domain [22, 23].

Linear Approximation Methods

In the case of function approximation, the value function is not a table but a parameterized function with weight vector $\boldsymbol{w} \in \mathbb{R}^d$. The approximation value of state s given weight vector $\boldsymbol{w}$ is denoted $\hat{v}(s,\boldsymbol{w}) \approx v_\pi(s)$. A special case of function approximation is when $\hat{v}(\cdot,\boldsymbol{w})$ is a linear function of the weight vector $\boldsymbol{w}$. Linear methods approximate state value function by the inner product of $\boldsymbol{w}$ and real-valued vector $\boldsymbol{x}(s) \doteq [x_1(s), x_2(s), \dots, x_d(s)]^T \in \mathbb{R}^d$ [22].

$$\hat{v}(s,\boldsymbol{w}) \doteq \boldsymbol{w}^T \boldsymbol{x}(s) \doteq \sum_{i=1}^{d} w_i x_i(s) \tag{3.10}$$

Where $\boldsymbol{x}(s)$ is the feature vector representing state s. One way to approximate function values is to use Stochastic Gradient Descent (SGD), where updates for $\boldsymbol{w}$ is computed in a sequence of discrete time steps $t = 1, 2, \dots$ [22].

$$\begin{aligned}\boldsymbol{w}_{t+1} &\doteq \boldsymbol{w}_t - \frac{1}{2}\alpha \nabla \left[v_\pi(S_t) - \hat{v}(S_t,\boldsymbol{w}_t)\right]^2 \\ &= \boldsymbol{w}_t + \alpha \left[v_\pi(S_t) - \hat{v}(S_t,\boldsymbol{w}_t)\right] \nabla \hat{v}(S_t,\boldsymbol{w}_t) \end{aligned} \tag{3.11}$$

Where $\alpha > 0$ is the step-size, and $\nabla f(\boldsymbol{w}) \in \mathbb{R}^d$ for any scalar expression $f(\boldsymbol{w})$ denotes the vector of partial derivatives with respect to the components of the weight vector. The step in $\boldsymbol{w}_t$ is proportional to the negative gradient of the squared error, which correspond to the direction where the error decreases most rapidly. The target output of the t^{th} training sample is denoted $U_t \in \mathbb{R}$. By substituting U_t in the place of $v_\pi(S_t)$ in SGD yields following method [22].

$$\boldsymbol{w}_{t+1} \doteq \boldsymbol{w}_t + \alpha \left[U_t - \hat{v}(S_t, \boldsymbol{w}_t) \right] \boldsymbol{x}(S_t) \tag{3.12}$$

If $\mathbb{E}\left[U_t | S_t = s\right] = v_\pi(S_t)$, that is if U_t is an unbiased estimate, then $\boldsymbol{w}_t$ is guaranteed to converge to a local optimum for decreasing α [22].

Policy Gradient Methods

Methods mentioned previously learns the values of actions, and select actions based on estimated action values. Policy gradient methods can select actions, without the use of a value function. Consider a policy parameter vector denoted as $\boldsymbol{\theta} \in \mathbb{R}^{d'}$. The objective function is defined as $\mathcal{J}(\boldsymbol{\theta}) \doteq v_{\pi_\theta}(s_0)$, where v_{π_θ} is the true value function for π_θ determined by $\boldsymbol{\theta}$. The policy parameter is updated in the opposite direction of the gradient as $\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t + \alpha \nabla \mathcal{J}(\boldsymbol{\theta}_t)$, which is the direction that maximizes the cumulative reward. The policy gradient theorem provides an analytical expression with equal proportionality of the objective function that does not involve the derivative of the state distribution [22].

$$\nabla \mathcal{J}(\boldsymbol{\theta}) = \mathbb{E}_\pi \left[\sum_{a \in \mathcal{A}} q_\pi(S_t, a) \nabla_\theta \pi_\theta(a | S_t, \boldsymbol{\theta}) \right] \tag{3.13}$$

Given $\pi_\theta(a|s, \boldsymbol{\theta})$ is differentiable, an approximation of the gradient update can be based on MC sampling. The point of interest is the one action taken at time t, replacing the action a with sample action $A_t \sim \pi$ and assuming $\mathbb{E}_\pi[G_t | S_t, A_t] =$

$q_\pi(S_t, A_t)$ an approximation can be derived from rearranging (3.13) [10, 22].

$$\nabla \mathcal{J}(\boldsymbol{\theta}) \approx \sum_i \left(G \sum_{t=1}^{T} \nabla_\theta \ln \pi_\theta(A_t^i | S_t^i) \right) \tag{3.14}$$

Above equation is formally known from the REINFORCE algorithm, where the policy gradient theorem is used to update the unbiased gradient through MC sampling. The cumulative reward G is approximated from a large number of samples, which may learn slow due to high variance [22].

3.1.4 Deep Reinforcement Learning

An additional field of theory is DNN, which introduces the concept of automating the process of designing task-relevant features for function approximation. In combination with RL the value or policy function can be approximated while solving the curse of dimensionality, that is applications with very large or continuous state or action spaces [22]. The concept of DNN will be presented in this section, as an introduction to combining Q-learning with neural networks. With knowledge about the network architectures, procedures used to train neural networks will be presented.

Deep Neural Network

DNN are widely used for non-linear function approximation, as a network of interconnected units in layers with properties inspired from neurons, the main component of nervous systems. There exists many types of neural networks, figure 3.2 presents a generic fully connected feed-forward DNN. The input vector $\boldsymbol{x}$ are not influenced by the output vector $\boldsymbol{y}$, as the hidden layers in between only feeds information forward in the network [22].

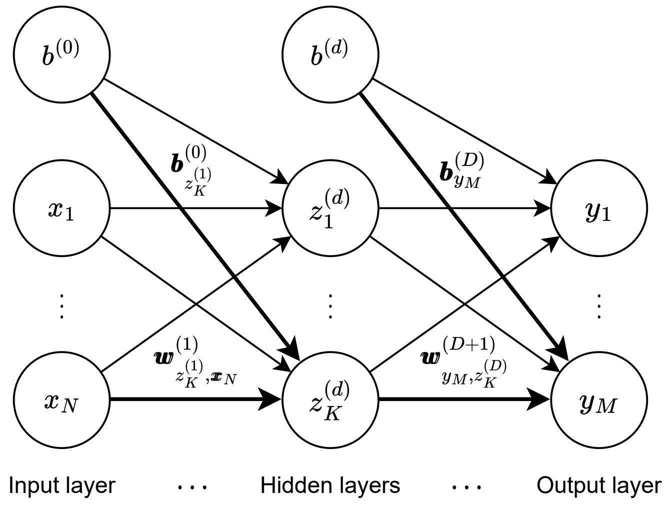
$b^{(0)}$
$b^{(d)}$
$\boldsymbol{b}^{(0)}_{z_K^{(1)}}$
$\boldsymbol{b}^{(D)}_{y_M}$
x_1
$z_1^{(d)}$
y_1
$\boldsymbol{w}^{(1)}_{z_K^{(1)},\boldsymbol{x}_N}$
$\boldsymbol{w}^{(D+1)}_{y_M,z_K^{(D)}}$
x_N
$z_K^{(d)}$
y_M
Input layer
Hidden layers
Output layer

Figure 3.2: Conceptual illustration of a simple feed-forward DNN with N inputs, M outputs, and D hidden layers each with K nodes for simplicity. Examples with bold arches indicate parameters between the first and last layers.

A real-valued weight vector $\boldsymbol{w}$ and bias vector $\boldsymbol{b}$ are associated to each layer, corresponding to synaptic connections in a real neural network. An expression for the output of the network, as function of the input passing through $d \in \{1, 2, \ldots, D\}$ hidden layers, can be derived as the following [26].

$$\boldsymbol{y} = f(\boldsymbol{x}) = f_D\left(\ldots f_2(f_1(\boldsymbol{x}))\right) \tag{3.15}$$

The output of each neuron is passed through a non-linear activation function such as the Sigmoid logistic function $g(x) = 1/(1+e^{-x})$, or the non-linear Rectified Linear Unit (ReLu) $g(x) = \max(0, x)$ among others, depending on the problem type. The output vector of the d^{th} hidden layer denoted $\boldsymbol{h}^{(d)}$ can be expressed based on the output from the previous layers [26].

$$\boldsymbol{h}^{(d)} = f_d\left(\boldsymbol{h}^{(d-1)}\right) = g^{(d)}\left(\boldsymbol{w}^{(d)}\boldsymbol{h}^{(d-1)} + \boldsymbol{b}^{(d-1)}\right) \tag{3.16}$$

When an input sample has been propagated through the network, the performance can be calculated in terms of loss based on the type of RL problem. The goal is to find an optimal weight vector $\boldsymbol{w}^*$, that minimizes the loss. In value based problems, the loss can be defined as the error between some target value y_n and a predicted value $\hat{y}_n$. Three common methods are Mean Absolute Error (MAE) weighting error values linearly, Mean Square Error (MSE) penalizing higher errors, and the Huber loss as a combination of both methods.

$$\mathcal{L}_{\text{MAE}} = \frac{1}{N}\sum_{n=1}^{N} |y_n - \hat{y}_n| \tag{3.17}$$

$$\mathcal{L}_{\text{MSE}} = \frac{1}{N}\sum_{n=1}^{N} (y_n - \hat{y}_n)^2 \tag{3.18}$$

$$\mathcal{L}_{\text{Huber}} = \begin{cases} \frac{1}{2}(y_n - \hat{y}_n)^2 & \text{for } |y_n - \hat{y}_n| \le \delta \\ \delta|y_n - \hat{y}_n| - \frac{1}{2}\delta^2 & \text{otherwise} \end{cases} \tag{3.19}$$

Where $n \in \{1, 2, \ldots, N\}$ denote the sample number, and δ is the discrimination parameter. In the case of RL problems, the loss is based on the reward signal.

Deep Q-Network

Q-learning being a value-based method is constrained to small state and action spaces, where there is no generalization ability for unseen states. To accommodate this, consider a DQN where a DNN is applied to approximate the values $\hat{Q}(s,a) = f(s,a,\boldsymbol{\theta})$, given vector $\boldsymbol{\theta}$ with the DQN learning parameters [22]. From the Q-learning term (3.9), loss is defined as the error between target value and predicted values.

$$\Gamma_Q = \underbrace{R_{t+1} + \gamma \max_a Q(S_{t+1}, a)}_{\text{Target } y_n} - \underbrace{Q(S_t, A_t)}_{\text{Prediction } \hat{y}_n} \tag{3.20}$$

The Q-value estimation if fitted to optimization of $\boldsymbol{\theta}$, where the DQN loss term typically is defined using one of (3.18), (3.17), or (3.19) based on the problem type, and substituting $\Gamma_Q \doteq y_n - \hat{y}_n$. It has also been shown that modifying the standard Q-learning loss Γ_G to be in the interval $[-1, 1]$ can improve stability. To improve the convergence the concept can be divided into a main network that approximates the value function, and a target network that estimates the expected returns from an action. This is known as a DDQN, where the loss function is divided equally defined by the following [22].

$$\Gamma_{\text{DDQN}} = R_{t+1} + \gamma \max_a Q(S_{t+1}, a, \boldsymbol{\theta}_{\text{target}}) - Q(S_t, A_t, \boldsymbol{\theta}_{\text{main}}) \tag{3.21}$$

The weights of the main network is updated continuously, whereas the weights of the target network is updated over limited periods of time. The used policy may be ϵ-greedy, where $\epsilon = \max\left(\epsilon_{\text{min}}, (\epsilon_{\text{max}} - \epsilon_{\text{min}})/\epsilon_{\text{step}}\right)$ decreases to a low fixed value ϵ_{min} over the first ϵ_{step} steps resulting in more exploitation over time [22].

To improve training in non-stationary environment, the experience replay method may be utilized to store the prior experience (S_t, A_t, R_t, S_{t+1}). The memory has a fixed length and accumulate experiences at every timestep. At each time step multiple Q-learning updates can be performed known as a mini-batch, based on samples drawn from the replay memory with an uniform distribution or an algorithm. This base updates on a batch of experience, where samples can be reused multiple times, typically accelerating convergence and reducing requirements for the total number of samples. Using an algorithm to draw relatively new samples in the replay memory allows the environment to change over time. However if the environment changes rapidly over short ranges of time, sample uncorrelation is proportional to the time separation resulting in convergence problems [22].

Proximal Policy Optimization

Policy-based methods can also be adapted to DNN, where one approach is the Proximal Policy Optimization (PPO). Two main concepts enable PPO, the clipped surrogate objective, and training over multiple epochs when performing a policy update. Consider the policy gradient, where the objective is to optimize the weights of a DNN [27].

$$\mathcal{L}(\theta) = \hat{\mathbb{E}}_t \left[\log \pi_\theta(a_t|s_t)\hat{A}_t\right] \tag{3.22}$$

Where π_θ is an stochastic policy and $\hat{A}_t$ is a estimation of the advantage function at timestep t. Multiple optimization steps within the same trajectory using (3.22) are not stable, due to accumulation of large gradients [27]. Instead, the surrogate objective $l_t(\theta)$, is introduced as a probability ratio between current and old policies. The objective function is rewritten to

$$\mathcal{L}(\theta) = \hat{\mathbb{E}}_t \left[l_t(\theta)\hat{A}_t \right] \tag{3.23}$$

$$l_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta,\text{old}}(a_t|s_t)} \tag{3.24}$$

where θ_{old} is the policy weights from before the update. To mitigate the possibility of excessively large gradient update, the objective is modified to penalize the policy [27]. The objective function is rewritten to

$$\mathcal{L}(\theta) = \hat{\mathbb{E}}_t \left[\min\left(l_t(\theta)\hat{A}_t,\ \text{clip}\left(l_t(\theta),\ 1-\epsilon,\ 1+\epsilon\right)\hat{A}_t \right) \right] \tag{3.25}$$

where the surrogate objective is modified by clipping the probability ratio to the interval $[1-\epsilon, 1+\epsilon]$. The final objective loss is the minimum between the clipped and unclipped objectives, to produce a lower bound on the unclipped value [27]. This approach only ignore the change in probability ratio when the objective improve, and is included when the objective becomes worse. The clipping is illustrated in figure 3.3.

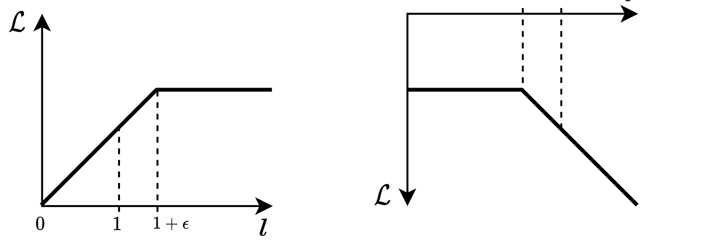

Figure 3.3: Plot illustrating a single timestep of the update function in (3.25), as a function of probability ratio l, for (a) positive advantages, and (b) negative advantages.

The PPO can be integrated as actor-critic style, where a policy function is used for decision making and a value function for evaluation during learning is trained simultaneously. A single actor approach is shown in algorithm 3 [27].

Algorithm 3 Vanilla PPO, actor-critic style [27]

```
1: Input: Number of epochs $K$, minibatch size $B \leq T$
2: for iterations $1, 2, \dots$ do
3:     Run policy $\pi_{\theta,\text{old}}$ for $T$ timesteps
4:     Compute advantage estimate $\hat{A}$
5:     Optimize (3.25) wrt. $\theta$
6:     $\theta_{\text{old}} \leftarrow \theta$
7: end for
```

Learning Procedure

The learning process for DNN consists of updating the network weights such that the loss is minimized. To automate the learning process, the gradient of the loss is estimated with Automatic Differentiation (AD) methods that exploits the differentiation chain rule. A special case known as reverse AD, referred to as backpropagation, is a popular and efficient method to achieve automatic learning [28].

Given the activation functions are differentiable, backpropagation is a efficient method to learn by alternating forward and backward passes of samples through the network. Passing samples through the network given current weights is known as a forward pass, where the loss can be calculated at the output for assessment. The loss can be propagated back through the network, known as a backward pass, where the gradient of the loss function with respect to the weights of the network can be efficiently estimated [22]. Gradient methods are feasible to calculate the updates, such as SGD and Adaptive Moment Estimation (Adam). Estimating the gradient with momentum from previous gradients and a decaying learning rate, Adam is able to achieve a quick convergence [28].

Backpropagation produces good results for a low number of hidden layers, but struggles with deeper networks. Over-fitting occurs when failing to generalize patterns in the data correctly caused by a large number of weights in the network.

The partial derivatives in backward passes either decay resulting in slow learning, or grow resulting in unstable learning [22].

3.1.5 Multi-Agent Reinforcement Learning

Multiple agents can coexist in a shared environment known as MARL, where agents receive rewards by interacting with the environment based on the task at hand [23]. The logical problem involving more than one agent is modeled as a Multi-Markov Decision Process (MMDP) and may be approached with stochastic game theory, a framework where agents must produce the optimal decision making in a strategic setting [23].

Agents may maximize their own rewards independently, in opposition and completely regardless to other agents. In another cooperative approach, agents collaborate to maximize the overall rewards, and execute collective decisions. A mixed cooperative-competitive approach allows agents to discover various coordination strategies, however this requires more intelligent agent behavior. Considering N independent agents with their individual action spaces and reward signals, a MARL approach modeled as MMDP is illustrated in figure 3.4 [9].

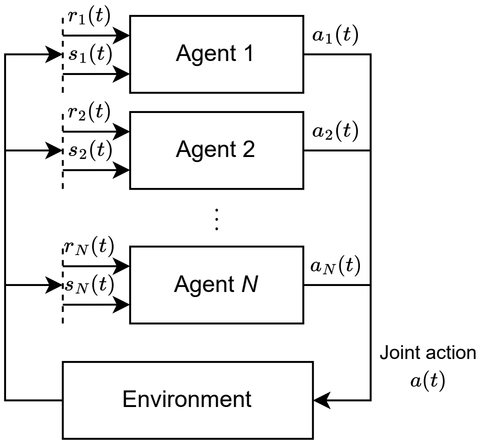

Figure 3.4: The multi-agent-environment interaction model for MMDP.

Each of the N agents tries to find an optimal policy π_n^*, based on local state and action information available. The optimal policy is typically obtained as the policy that maximizes the total reward function [9].

$$\pi_t^*(s) = \max_{\pi_t(s)\in\mathcal{A}} \left\{ r_t(s_t, \pi_t(s)) + \gamma \sum_{s'\in\mathcal{S}} p(s_t, s')\pi_{t+1}^*(s') \right\} \tag{3.26}$$

Where $0 \leq \gamma \leq 1$ is the discount factor. If an observation collects all information about the state in the environment, the MMDP is known as fully observable. This could be the case of chess, where all information about the entire game is easy to obtain. Practically, each agent usually only access a part of the information through an observation, resulting in a Partially Observable Multi-Markov Decision Process (POMMDP) [23]. Instead of directly observing the state, a set of observations with subliminal information about the current state is used.

Multi-Agent Reinforcement Learning Methods

MMDP is modeled and defined with the tuple $\left(N, \mathcal{S}, \{\mathcal{A}^n\}_{n=1}^N, \mathcal{P}, \{\mathcal{R}^n\}_{n=1}^N\right)$ [23]. The set of all possible states for all agents is defined as the state space $\mathcal{S} = \mathcal{S}_1 \times \cdots \times \mathcal{S}_N$, and the joint action space containing all possible actions for the nth agent $\mathcal{A}^n$ is denoted $\boldsymbol{\mathcal{A}} = \mathcal{A}^1 \times \cdots \times \mathcal{A}^N$. The reward signal for the nth agent is denoted $\mathcal{R}^n : \mathcal{S} \times \boldsymbol{\mathcal{A}} \times \mathcal{S}$ and the transition probability from state $s \in \mathcal{S}$ to state $s' \in \mathcal{S}$ by joint action $\boldsymbol{a} \in \boldsymbol{\mathcal{A}}$ is denoted $\mathcal{P} = \mathcal{S} \times \boldsymbol{\mathcal{A}}$.

The value-based Q-learning method for a single agent in (3.9) can be extended to the context of MARL. Consider an instantaneous timestep t, the transition data is sampled for each agent $\{s_t, \boldsymbol{a}_t, R^n, s_{t+1}\}_{t\geq 0}$ is gathered to update the Q-values [23].

$$Q^n(s_t, \boldsymbol{a}_t) \leftarrow Q^n(s_t, \boldsymbol{a}_t) + \alpha \left[R^n + \gamma \max_{a} Q^n(S_{t+1}, \boldsymbol{a}) - Q^n(s_t, \boldsymbol{a}_t) \right] \tag{3.27}$$

For MARL problems collaborative and adversarial behaviors can be configured by proper design of rewards [23]. To accommodate the curse of dimensionality in the value-based Q-learning method, a DQN is extended to MARL. Training may be distributed, where agents train models locally, or centralized where all agents train the same model [25]. The max operator in (3.27) utilize the same Q-values for selection and evaluation of an action. This makes it more likely to select over-estimated Q-values. To prevent this, the training stability and performance can be improved with a DDQN, which relies on two similar DQNs [29]. A main evaluation network is used to determine the ϵ-greedy policy, and the other to determine its target value. The target network is a periodic copy of the main network used for training [29].

The policy-based PPO method in (3.13) can similarly be extended in context to MARL. Each agent learns an optimal policy $\pi^n_{\theta^n}$ by updating the respective parameters $\{\theta^n\}^N_{n=1}$ from the joint policy $\boldsymbol{\pi}_\theta = \prod_{\forall n} \pi^n_{\theta^n}(a^n|s)$. The objective function is rewritten in terms of the n^{th} agent [23].

$$\nabla \mathcal{J}^n(\boldsymbol{\theta}) = \mathbb{E}_{\pi_\theta}\left[\nabla_{\theta^n} \ln \pi_{\theta^n}(a^n|s) Q^{n,\pi_\theta}(s,\boldsymbol{a})\right] \tag{3.28}$$

The expectation over the joint policy $\boldsymbol{\pi}_\theta$ implies that the agents have the ability to observe each others policies.

Multi-Agent Reinforcement Learning Frameworks

The two suggested approaches for MARL, are policies designed to learn how to solve a complex problem. The approach to training such methods depends on the system, and how information is relayed. Two general MARL framework viz for training are considered: centralized training with distributed execution, or distribution training and execution. In common for either approach, distributed execution refers to agents deployed in an environment and execute the learned policy.

Centralized training requires agents to share mutual experiences and cooperate to generate a common model for distributed execution. Here it is assumed a centralized unit performs training by collecting observations from agents. However, this is impractical for systems where sharing information is not possible due to data sensitivity or where communication with a central unit for training is unstable [25]. Furthermore, training complexity increase with the number of agents. Instead, the centralized unit can be disregarded if agents are able to perform training locally. Distributed training and execution without knowledge sharing is a local competitive training process in each agent. Although sharing of potential sensitive data is avoided in distributed training, the perspective of the environment is partial, where non-stationary scenarios tends to slow down learning.

3.2 Federated Learning

FL is an algorithmic framework for multiple parties in a network to contribute learning updates to a global model, under the requirement of keeping data private. The privacy between parties is kept by exchanging information about the models, and not the data itself, so that data privacy will not be compromised. The concept of FL is illustrated on figure 3.5.

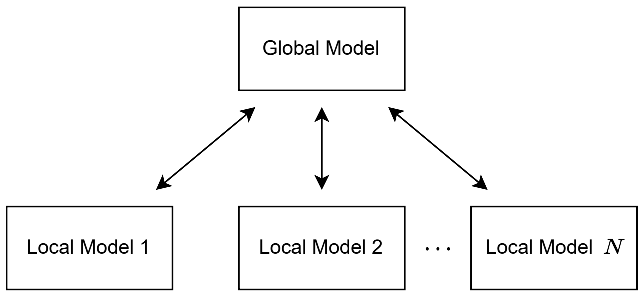

Figure 3.5: Conceptual illustration of FL.

Two or more parties are needed to jointly build a model, each holding independent data used for local model training. The learning profits are conveyed through model parameters that does not compromise privacy. Encryption schemes may be used to increase security, however this depends on the problem and how much security is required. Parties can be weighted equally, and are always able to use the global model for local improvement [10]. In this section the basics of FL will be presented followed by fundamental architectures, categorization by data partitioning, and an extension of the concept in combination with RL.

3.2.1 Basics of Federated Learning

Consider N parties $\{\mathcal{F}_i\}_{i=1}^N$, each storing their respective data-sets $\mathcal{D}_i$ from which a set of parameters w_i is learned. In traditional supervised machine learning, each training sample j is accompanied by a label with the expected output. A loss function $f_j(w)$ is defined as the error of sample j using parameters w, and is used to assess training. The loss of a data-set $\mathcal{D}_i$ can be defined as $F_i(w)$. Similarly, the common global model loss can be defined as $F_g(w)$ [10].

$$F_i(w) = \frac{1}{|\mathcal{D}_i|} \sum_{j \in \mathcal{D}_i} f_j(w) \tag{3.29}$$

$$F_g(w) = \sum_{i=1}^{N} \eta_i F_i(w) \tag{3.30}$$

Where $|\cdot|$ denotes the size of the set and $\eta_i > 0$ is the relative impact of each party. The term η can be configured constrained to $\sum_{i=1}^N \eta_i = 1$, where a natural selection would be the average $\eta = 1/N$. The goal is to find the optimal parameters w^*, which minimizes the global loss function [10].

$$w^* = \arg\min_{w} F_g(w) \tag{3.31}$$

One solution for (3.31) would be a gradient-descent approach, known as the federated averaging algorithm. Each party uses local data to perform a number of steps in gradient descent on current model parameters $\bar{w}(t)$ [10].

$$\forall i, w_i(t+1) = \bar{w}(t) - \gamma \nabla F_i(\bar{w}_i(t)) \tag{3.32}$$

Where $\gamma > 0$ is the learning rate, and $\nabla f(w)$ for any scalar expression $f(w)$ denotes the vector of partial derivatives with respect to the components of the parameters w. The global model is updated using a weighted average of the local parameters from the parties [10].

$$\bar{w}_g(t+1) = \sum_{i=1}^{N} \frac{n_i}{n} w_i(t+1) \tag{3.33}$$

Where n_i is the number of training samples in the i^{th} party, and n is the total number of samples in all data-sets. The global weight $\bar{w}_g$ can be aggregated back to the parties, which usually is done at scheduled intervals.

3.2.2 Architectures of Federated Learning

The architecture of FL can mainly be constructed in to ways, namely client-to-server model and a peer-to-peer model. The former model is a centralized learning method where a server is responsible for updating the global model, and the latter model is a distributed learning method with no need of hosting a server. In either setup, training of local models in respective parties is a decentralized learning framework.

Client-to-Server Model

In the client-to-server model a coordinator is considered as a centralized aggregation server, which aggregate local model updates from the parties. The client-to-server architecture is illustrated in figure 3.6 and the iterations steps are summarized in the following list [10]. Steps 2 to 5 are repeated until the model converges or other stopping criteria has been satisfied.

1. **Initialize:** The coordinator generates a model, and sends it to every party.

2. **Train:** Every party trains local models based on data-sets.
3. **Submit:** Local model weights are uploaded to the coordinator.
4. **Aggregation:** The coordinator combines the models with aggregation.
5. **Update:** The global model weights are send back to the parties.

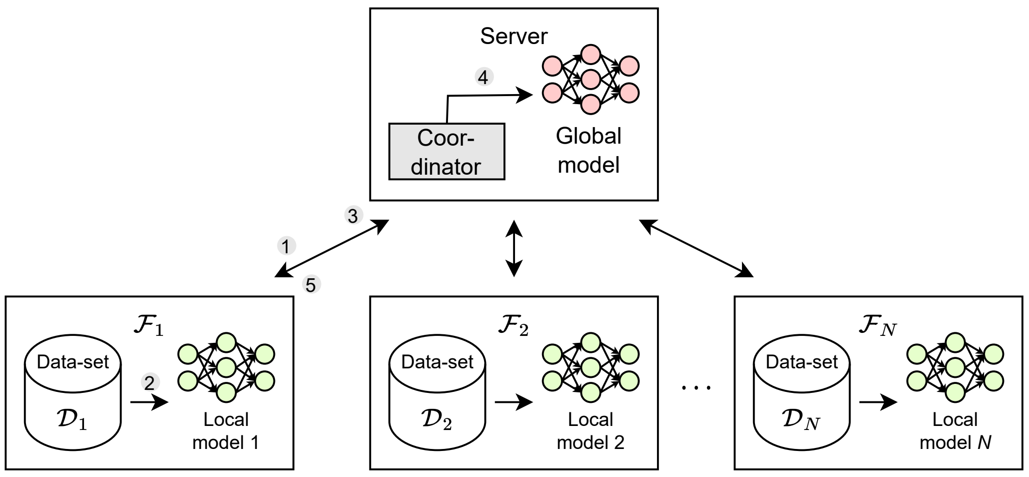

Figure 3.6: Conceptual illustration of the client-server architecture of FL systems. The numbers circled in grey indicates the steps in the process, with focus on actions in party $\mathcal{F}_1$.

The client-to-server model does rely on every party sending updates to a single entity, where the server can become a bottleneck of the system. The parties are assumed to be constrained by hardware performance, but given that the server is stable, a portion of the workload will be relieved.

Peer-to-Peer Model

The peer-to-peer model does not rely on a central coordinator, but instead can any of the parties be data owners and aggregate model updates. The peer-to-peer architecture is illustrated in figure 3.7 and the iterations steps are summarized in the following list [10]. Steps 2 to 4 are repeated until the model converges or satisfies other stopping criteria.

1. **Initialize:** Each participant generates a local model.

2. **Train:** Every party trains local models based on data-sets.
3. **Exchange:** Requests of model parameter exchange between parties.
4. **Aggregation:** Aggregate the model received from other parties.

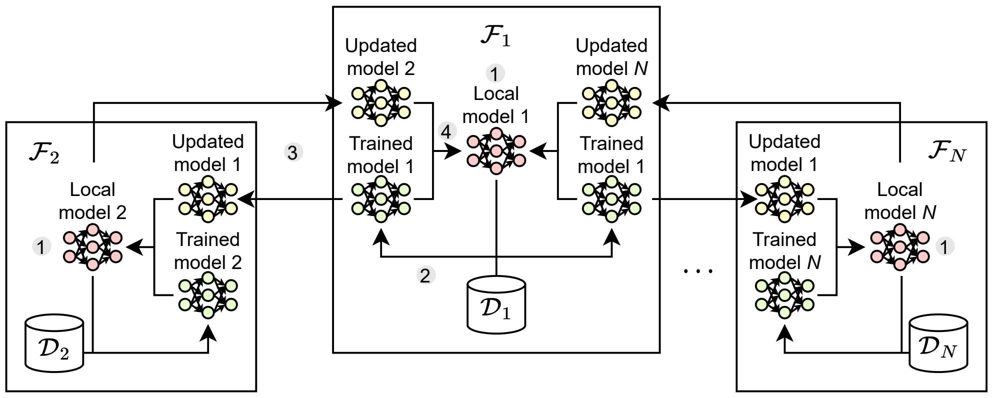

Figure 3.7: Conceptual illustration of the peer-to-peer architecture of FL systems. The numbers circled in gray indicates the steps in the process, with focus on actions in party $\mathcal{F}_1$.

The performance of individual parties are constrained by limited hardware resources, however distributing the workload between peers does delimit the need for central coordination or a stable host. Furthermore, two local parties are able to share knowledge directly.

3.2.3 Data Partitioning

How data is partitioned matters in terms of implementation, as well as the practical and technical challenges. Consider the i^{th} party with data-set $\mathcal{D}_i$, the data-set contains the feature space $\mathcal{X}_i$ as an abstraction of patterns in the data, the label space $\mathcal{Y}_i$ associates categories with input and task, and the sample ID space $\mathcal{I}_i$ used to discover possible connections among different features [10]. The methods of partitioning data-sets within feature, label, and sample space will be discussed in this section, and may be classified as Horizontal Federated Learning (HFL) and Vertical Federated Learning (VFL).

Horizontal Federated Learning

In the case of HFL, the data-sets of parties share the same feature space. The feature space and label space is assumed to be identical for data-sets i and j, however the sampling-ID space is different [10].

$$\mathcal{X}_i = \mathcal{X}_j, \quad \mathcal{Y}_i = \mathcal{Y}_j, \quad \mathcal{I}_i \neq \mathcal{I}_j, \quad \forall \mathcal{D}_i, \mathcal{D}_j, i \neq j \tag{3.34}$$

The goal is to increase the amount of data with similar features, without transmitting any of the original data. Regional hospitals has different patients but the symptoms for diseases are similar. HFL provides a secure method to jointly build a model for identifying diseases between hospitals, without sharing sensitive patient data [10].

Vertical Federated Learning

VFL is applicable in the case where multiple parties has different features. The feature space and label space of data-sets i and j is assumed to be different, as the sample ID space is identical [10].

$$\mathcal{X}_i \neq \mathcal{X}_j, \quad \mathcal{Y}_i \neq \mathcal{Y}_j, \quad \mathcal{I}_i = \mathcal{I}_j, \quad \forall \mathcal{D}_i, \mathcal{D}_j, i \neq j \tag{3.35}$$

It is desired to jointly build a model in a privacy-preserving manner by exploiting all features from different parties, which can even be used to generate new features. Banks with credit card-records and e-commerce companies with purchase-histories from the same clients can cooperate to safely create models used for trust evaluation of clients using VFL [10].

3.2.4 Federated Reinforcement Learning

The framework of FL can be used to enhance the security of RL, resulting in FRL where privacy of agents with the ability to handle non-Independent and Identically Distributed random variables (IID) data is protected. It is believed that performance of learning can be improved as the combination of frameworks introduce more stability to practical problems. FRL is divided into two categories based on environment partitioning, namely Horizontal Federated Reinforcement Learning (HFRL) and Vertical Federated Reinforcement Learning (VFRL) [10]. The concepts will be formally defined and described in this sections using practical examples.

Horizontal Federated Reinforcement Learning

HFRL is applicable in scenarios where agents with similar decision making tasks are distributed geographically, and the goal is to train a common model while keeping data private. When agents share experience, the privacy is protected by choosing a strategy to exchange encrypted models. Consider N agents $\{\mathcal{F}_i\}_{i=1}^N$ observing their respective independent environments $\{\mathcal{E}_i\}_{i=1}^N \in \mathcal{G}$, where state transition probabilities and reward functions are similar [10].

$$\mathcal{S}_i = \mathcal{S}_j, \quad \mathcal{A}_i = \mathcal{A}_j, \quad \mathcal{E}_i \neq \mathcal{E}_j, \quad \forall i,j \in \{1,2,\ldots,N\}, \mathcal{E}_i, \mathcal{E}_j \in \mathcal{G}, i \neq j \tag{3.36}$$

Imagine autonomous driving systems, where vehicles on the road explores various environments and train models locally. Properties of a single environment constraints the possibility of every situation occurring, hence vehicles in different environments can share their learned experience to improve the overall performance. Consider a client-to-server model where the agents may be composed of geographically distributed heterogeneous equipment, such as different types of vehicles in different regions. The following list summarize the HFRL procedure, where any user can join and leave at any given time [10].

1. **Initialize:** If the agent has no model, the global model weights must be downloaded from the coordinator. If the agent already have a model, the model type and parameters must be confirmed by the coordinator.
2. **Train:** Agents make independent observations on their environments. All agents evaluate selected actions by the next state and received reward. Local models is trained in state-action-reward-state cycles.
3. **Submit:** The local model weights achieved by training can be encrypted and transmitted to the coordinator at any time.
4. **Aggregate:** An aggregation algorithm is used by the coordinator to train the global model.
5. **Send:** The coordinator transmits the aggregated model weights to the agents
6. **Update:** Agents improve their respective local models using the global weights.

Devices that has just joined or does not meet the hardware requirements for training can obtain the shared model at any given time. Multiple agents sharing training experiences from different environments may accelerate the training process and improve model reliability, as the data correlation is generally reduced and discovery of rare states occurs more frequently [10].

Vertical Federated Reinforcement Learning

VFRL is applied where samples from multiple data-sets belonging to a common group of users have different feature spaces. The agents interacts with the same environment from an observation that is a part of the global state, making it suitable for POMMDP scenarios. The goal is to build a common model from heterogeneous feature spaces while keeping data private. Consider N agents $\{\mathcal{F}_i\}_{i=1}^{N}$ interacting with a global environment $\mathcal{E}$, where the i^{th} agent obtains the local partial observation $\mathcal{O}_i$ from the independent interaction $\mathcal{A}_i$ [10].

$$\mathcal{O}_i \neq \mathcal{O}_j, \quad \mathcal{A}_i \neq \mathcal{A}_j, \quad \mathcal{E}_i = \mathcal{E}_j,$$
$$\bigcup_{i=1}^{N} \mathcal{O}_i = \mathcal{S}, \quad \bigcup_{i=1}^{N} \mathcal{A}_i = \mathcal{A}, \quad \forall i,j \in \{1,2,\dots,N\}, \quad i \neq j \tag{3.37}$$

Consider a system with household users, the power company, and solar cell companies as agents observing the same environment but with different state spaces. Households manage electrical utilities by observing local power generation, consumption, and costs. The power company tries to optimally distribute and balance energy by observing power consumption for all households. Solar cell companies has information about the power generation. The agents need to communicate with each other to find to optimal policies, however all of the participant wants to keep data private. Consider two types of agents in a peer-to-peer model scenario. The first type is K decision-oriented agents $\{\mathcal{F}_i\}_{i=1}^{K}$ interacting with environment $\mathcal{E}$ based on respective local states $\{\mathcal{S}_i\}_{i=1}^{K}$ and actions $\{\mathcal{A}_i\}_{i=1}^{K}$. The other type is the remaining $N-K+1$ support-oriented agents $\{\mathcal{F}_i\}_{i=K+1}^{N}$, that only observes the local state $\{\mathcal{S}_i\}_{i=K+1}^{N}$ of the environment and takes no action. The following list summarize the VFRL procedure [10].

1. **Initialize:** Each participant generates a local model.
2. **Train:** Agents observes states from the environment. Decision-oriented agents perform an action, training local models in state-action-reward-state cycles.
3. **Submit:** The local model parameters achieved by training can be encrypted and transmitted to the global model.
4. **Aggregate:** An agent aggregate the global model.
5. **Send:** The global model is encrypted and transmitted back to other agents.
6. **Update:** Agents improve their respective local models using the global model.

The system model benefits from agents that cannot generate rewards by exploiting the partial observation information to improve the efficiency of overall learning. By inheriting ideas for privacy protection from FL, multiple agents can cooperate with the same information without worrying about leakage of data [10].

Chapter 4

System Model & Problem Formulation

This chapter presents a use case scenario for the 6G in-X subnetworks and describes the simulation system model, including the network traffic, mobility, and channel models. Based on this model, a set of optimization problems will be formulated to mitigate interference effects and improve the system's performance.

4.1 Use case

Characteristics of interference may rely on the specific environment in terms of radio technology, propagation media, terrain, and obstacles. Such characteristics may be defined with a use case for 6G in-X subnetwork, which outlines the framework of a simulation environment that can be used to prove the concept of interference management methods. Current literature suggests a variety of 6G in-X subnetwork use cases that individually can be generalized to a set of applications. Examples of use cases with key requirements are presented in table 4.1 [5].

Realization of next generation networks is expected to unleash wire-free, smart factories with revolutionizing ways to manufacture, improve, and distribute products. Integration of new technologies will transform the industry, with a plethora of new scenarios able to deliver real-time decision making, enhanced productivity, flexibility and agility [6].

Use case	**In-robot**	**In-vehicle**	**In-body**	**In-house**
Example of applications	Motion control, force/torque control, mobile position/proximity	Engine control, electric power steering, ABS, electric park brakes, suspension, ADAS sensors	Heartbeat control, vital signs monitoring, insulin pumping, muscle haptic control	Entertainment, gaming, training, education, healthcare (robotic-aided surgery)
Number of devices	$\sim$ 20 (control) $\sim 20-40$ (mobile)	$\sim 50-100$	< 20	~ 10
Max range	$\sim$ 5 m	$\sim$ 10 m	$\sim$ 2 m	$\sim$ 10 m
Data rate per link	$<$ 10 Mbps	$<$ 10 Mbps (Control) $<$ 10 Gbps (ADAS)	$<$ 20 Mbps	$<$ 7 Gbps
Traffic type	Periodic, event based	Periodic, event based, uncompressed video	Periodic, event based	Event based, compressed video
Min latency	$\sim$ 100 µs	$\sim$ 54 µs	$\sim$ 20 ms	$\sim$ 5 ms (VR) $\sim$ 2 ms (healthcare)
Communication service availability	Six to eight nines	Six to eight nines	Nine nines	Six nines (VR), eight nines (healthcare)
Max subnetwork density	$\sim 40000/\text{km}^2$	$\sim$ 150/lane-km (car) $\sim$ 15/aircraft	$\sim 2/\text{m}^2$	1/room
Life-critical	No	Yes	Yes	Yes (healthcare), no otherwise
Criticality of power consumption	Low	Low	High	Low/medium

Table 4.1: Examples of requirements for different in-X subnetwork use cases [5].

Consider a possible industrial scenario where robots distributed in a factory hall performs production operations. Table 4.1 present two types of applications for in-robot subnetworks viz: stationary control applications such as unit assembling, sorting, or packing lines, and mobile applications such as obstructed transportation, dynamic measuring, and sorting. In this project, subnetworks are deployed inside autonomous robots as an uncoordinated scenario. Each subnetwork in the factory consist of a single AP responsible for communication between deployed devices, i.e., sensors and actuators. It is assumed that multiple in-robot subnetworks may coexist within a factory building, with or without connection to a larger network for coordination. This type of network framework may cover a wide indoor area and is referred to as an in-factory network.

4.2 System Model

Following in this section the system model composed of network traffic, communication channel effects, and subnetwork mobility models will be defined. The goal is to establish a system framework inspired by the in-factory use case. Consider N subnetworks each with M devices and a single AP, individually deployed inside robots of the factory. The subnetworks are indexed $n \in \mathcal{N} = \{1, 2, \ldots, N\}$ and all the devices in each subnetwork as $m \in \mathcal{M} = \{1, 2, \ldots, M\}$. The notation $(\cdot)_{n,m}$ is used to reference a link between nodes n and m device, e.g., subnetwork to subnetwork or local devices.

4.2.1 Traffic Model

The 6G in-X subnetworks are expected to support varying applications in diverse scenarios, and may therefore feature different types of traffic. As with other wireless systems, traffic for subnetworks can be categorized as: synchronous versus asynchronous and periodic versus event based [3, 5]. This leads to the different types of traffic depicted in figure 4.1 to be supported by subnetworks.

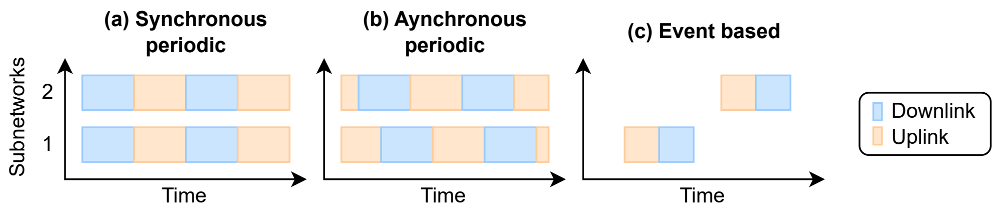

Figure 4.1: Illustration synchronous versus asynchronous and periodic versus event based traffic types.

In scenarios where data are communicated in fixed intervals, i.e., sensor measurements or actuator control commands from a mobile in-robot subnetwork, it is referred to as periodic traffic. Traffic among multiple subnetworks may be time-synchronized, meaning the UL and DL frames across subnetworks respectively are aligned. This however imposes the requirement of coordination in terms of time-synchronization accuracy across devices in subnetworks [5]. If there are no coordination among subnetworks, time-synchronization may be neglected resulting in asynchronous traffic. A lack of synchronization among subnetworks may result in a high probability for mutual interference over UL and DL frames [30].

Traffic can also be based on spontaneous events, e.g., either upon completion of a specific task or emergency-stop alarms, where low-latency may be a requirement. The time of activation cannot be predicted, and packets must suddenly be exchanged rapidly [5]. In terms of event based traffic, there is no coordination among subnetworks.

4.2.2 Mobility Model

The mobility model describes how the in-robot subnetworks move around in the factory environment. It is desired to achieve a degree of realism specific for the factory use case, which may be achieved by basing the mobility model on a real factory layout. The mobility model is based on an indoor industrial scenario, from existing production facilities of manufacturing companies, identified by the industry initiative, 5G alliance for connected industries and automation [31]. Such a layout is depicted on figure 4.2, as a 180×80 m hall containing several separate areas for production, assembly, storage, and human work zones.

Multiple in-robot subnetworks can be deployed with the task to transport materials or tools around in the facility. The alleys separating laboring areas are 5 m wide taking up $\sim$ 1600 m^2 of the factory area, and are outlined as two-lane roads in a right-handed traffic setting. An example random deployment of $N = 20$ subnetworks in the factory area is depicted in figure 4.2. It is worth mentioning that this translates to a density of 12,500 subnetworks/km^2. In the deployment the in-robot subnetworks are separated with a minimum distance of $d_{\text{min}} = 1$ m and move with

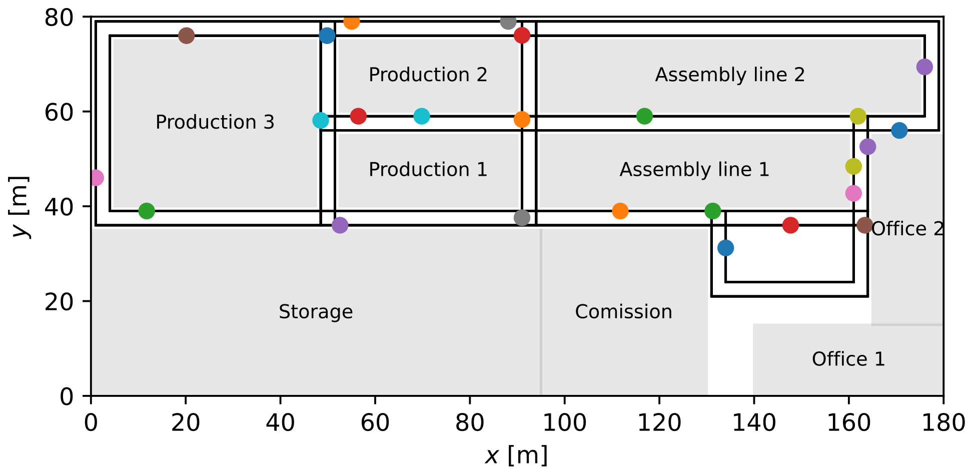

Figure 4.2: Example deployment of in-robots (scattered points) on lanes (black lines) in a indoor factory map.

a speed of 3 m/s. The in-robot subnetworks are modeled by a circular coverage area with a radius of $d_r = 1$ m, making it possible for robots to pass each other in the alleys. The minimum distance between two subnetworks is visualized on figure 4.3.

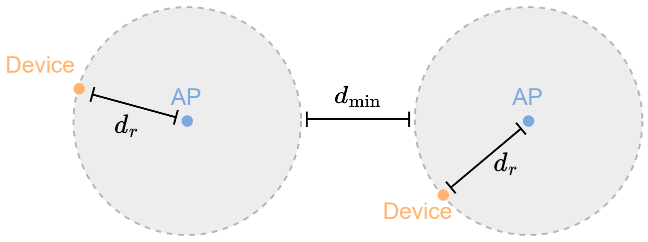

Figure 4.3: Visualization of two subnetworks with radius d_r and one device each, separated by the minimum accepted distance $d_{\min}$.

Collisions are avoided by prioritizing robots with shortest distances to a common intersection, slowing down any other robot that is within minimum separation distance. Visualization of road lanes and an initialization instance of subnetworks are illustrated in figure 4.2. Designated task destinations may be added to the map, and each robot could be given the purpose of determining an effective route with genetic algorithms. However modeling the mobility in such detail is considered out of scope for this project.

4.2.3 Channel Model

The channel model is based on three main propagation loss components, which will be defined with focus on the industrial scenario. The received signal power $g_{n,m}$ on the link between the nodes, n and m is expressed as

$$g_{n,m}(t) = p_n \cdot \Gamma_{n,m} \cdot \psi_{n,m} \cdot |h_{n,m}(t)|^2 \tag{4.1}$$

where p_n, $\Gamma_{n,m}$, $\psi_{n,m}$, and $h^k_{n,m}(t)$ are transmit power, path loss, correlated shadowing, and complex small-scale fading.

Path Loss

Signals naturally decay in power density as it propagates through space. The clutter in an active factory environment may obstruct the LoS between a transmitter and receiver, resulting in additional attenuation of the signal. A probability for a clear LoS from n to m can be expressed from the distance $d_{n,m}$, size of typical clutter elements d_{clutter}, and the clutter density r [32].

$$\Pr^{\text{LoS}}_{n,m} = \exp\left(\frac{d_{n,m}\ln(1-r)}{d_{\text{clutter}}}\right) \tag{4.2}$$

The clutter parameters are based on the surrounding environment, and may be generalized with scenario-based constants given in table 4.2. The probability for no clear LoS is denoted $\Pr^{\text{NLoS}}_{n,m} = (1 - \Pr^{\text{LoS}}_{n,m})$, where calculation of loss in a factory with varying clutter can be based on the following conditions [32].

$$\Gamma_{n,m} = \begin{cases} \Gamma^{\text{LoS}}_{\text{n,m}} & \text{for } \Pr^{\text{LoS}}_{n,m} > \Pr^{\text{NLoS}}_{n,m} \\ \Gamma^{\text{NLoS,S}}_{n,m} & \text{for } \Pr^{\text{LoS}}_{n,m} \leq \Pr^{\text{NLoS}}_{n,m} \text{ in sparse clutter} \\ \Gamma^{\text{NLoS,D}}_{n,m} & \text{for } \Pr^{\text{LoS}}_{n,m} \leq \Pr^{\text{NLoS}}_{n,m} \text{ in dense clutter} \end{cases} \tag{4.3}$$

where losses are calculated from the carrier frequency f_c and the following [32],

$$\Gamma^{\mathrm{LoS}}_{\mathrm{n,m}} = 31.84 + 21.5\log_{10}(d_{n,m}) + 19\log_{10}(f_c) \tag{4.4}$$

$$\Gamma^{\mathrm{NLoS,S}}_{n,m} = \max\left\{\Gamma^{\mathrm{LoS}}_{\mathrm{n,m}}, 33 + 25.5\log_{10}(d_{n,m}) + 20\log_{10}(f_c)\right\} \tag{4.5}$$

$$\Gamma^{\mathrm{NLoS,D}}_{n,m} = \max\left\{\Gamma^{\mathrm{NLoS,S}}_{n,m}, 18.6 + 35.7\log_{10}(d_{n,m}) + 20\log_{10}(f_c)\right\} \tag{4.6}$$

Correlated Shadowing

Obstacles between a transmitter and receiver may cause shadowing phenomena, where fluctuations in the received power mean can be observed over time. Subnetwork links are assumed to have correlated impairments, meaning a source of shadowing will affect several links simultaneously. Shadowing in the deployment area can be mapped in a grid with a stationary and isotropic Gaussian random field with zero-mean and exponentially decaying spatial correlation, with the covariance defined as

$$\operatorname{cov}(\boldsymbol{S}_n, \boldsymbol{S}_{n,m}) = \sigma_S^2 \exp\left(-\frac{d_{n,m}}{d_\delta}\right), \tag{4.7}$$

where $\boldsymbol{S}_{(\cdot)}$ is the value of a two-dimensional Gaussian at the location of the device or AP, σ_S^2 is the variance of the shadowing map defined in table 4.2, and d_δ is the decorrelation distance [33]. Using the random field $\boldsymbol{S}$, the link shadowing between nodes n and m is calculated using

$$\psi_{n,m} = \ln\left\{\frac{1-\exp\left(-\frac{d_{n,m}}{d_\delta}\right)}{\sqrt{2}\sqrt{1+\exp\left(-\frac{d_{n,m}}{d_\delta}\right)}}(\boldsymbol{S}_n + \boldsymbol{S}_{n,m})\right\} \tag{4.8}$$

Small-scale Fading

The environment and clutter repeatedly reflects propagating signals in different directions, resulting in receivers observing multiple faded replicas of the signal with a temporal and spectral difference. Due to the nature of propagating signals, relative velocities between transmitters and receivers may result in a frequency shift known as the Doppler shift [34]. Multi-path fading may be represented with $h \sim \text{Rayleigh}(\sigma_S^2)$ random variables. Temporal correlation can be modeled by utilizing Jake's Doppler model [34].

$$h_{n,m}(t) = \rho h_{n,m}(t-1) + \sqrt{1-\rho^2}\epsilon_{n,m} \tag{4.9}$$

where $\rho = J_0(2\pi f_d T_s)$ is the temporal autocorrelation coefficient and $\epsilon_{n,m}$ is an IID complex Gaussian variable. The temporal autocorrelation can be found with a zeroth order Bessel function J_0 based on maximum Doppler frequency f_d, and time slot duration T_s [34].

Different settings in factories lead to varying types of clutter in terms of machinery, assembly lines, and storage shelves. For the factory scenario it is assumed that the clutter height is greater than the height of in-robot subnetworks. The clutter vary in density, size, surface, and structure, which may be generalized to a sparse clutter and a dense clutter scenario. The industrial environment is expected to have a characteristic influence on the channel model, where the examples shown in table 4.2 from the 3$^{\text{rd}}$ Generation Partnership Project (3GPP) may be utilized [32].

4.3 Problem Formulation

A system model for in-robot subnetworks in a indoor factory scenario has been established. It is intended to apply interference management methods discussed in

Parameter		Sparse clutter	Dense clutter
Clutter type		Big objects such as machinery with regular metallic surfaces	Small and medium object such as machinery with regular metallic surfaces
Clutter size d_{clutter}		10 m	2 m
Clutter density r		$< 40\%$	$\geq 40\%$
Shadowing σ_S^2	LoS	4	4
	NLoS	5.7	7.2

Table 4.2: Evaluation of in-factory parameters for sparse and dense clutter scenarios. It is assumed that the height of the AP is below the average clutter height [32].

section 2.2, to optimize the use of radio resources. A presentation and explanation of the problem formulation in terms of radio resource optimization follows in this section.

4.3.1 Optimization Problem

This project address the problem of mitigating the adverse effect of interference, with the goal of optimizing three key requirements for subnetworks: latency, reliability and data rate.

Links in a network serving a large number of users may become saturated with traffic, where storing data is required for later processing. Also, a complex network structure may require additional routing and switching for a signal to reach its destination. If a link suffers from significant interference, packet losses may be inevitable, but may however be accommodated with re-transmissions or error correction methods. Along with the former mentioned effects, the signal propagation path are contributing factors for a higher latency in the network. Shannon's formula provide fundamental insight for a channel subject to Additive White Gaussian noise (AWGN), which shows that the maximum achievable channel rate is proportional to the bandwidth penalized by the amount of interference. Hence, latency is a trade-off between network complexity, distance, and bandwidth.

It is assumed that the short-range in-robot factory scenario network density is controlled, and that the bandwidth and processing capabilities is sufficient, such minimum latency requirement of $\sim$ 100 μs has already been met. According to findings in section 2.2, power and channel allocation are promising interference management techniques. The problems formulated in this section therefore consider the transmit power and frequency channels as the optimization variables.

Performance Metrics

The received SINR, denoted $\gamma_{n,m}$, on a link between the n^{th} AP and m^{th} device is based on the aggregate interference as

$$\gamma_{n,m}(t) = \frac{g_{n,m}(t)}{\sum_{i \in \mathcal{I}_k} g_{n,i}(t) + \sigma^2} \tag{4.10}$$

where $\mathcal{I}_k$ denotes the APs or devices transmitting on channel k. The receiver noise power is expressed as $\sigma^2 = 10^{(-174+\mathrm{NF}+10\log_{10}(BW))/10}$, where NF denotes the noise figure and BW is the bandwidth of the channel. Assuming transmissions with large data packet such that the infinite block-length approximation holds [35], the channel capacity can be estimated using the Shannon's formula as

$$r_{n,m}(t) = B \cdot \log_2\left(1 + \gamma_{n,m}(t)\right) \tag{4.11}$$

However in practical 6G in-X subnetworks short packets (in the order of tens of bytes) are expected to be transmitted, making the infinite block-length and usage of Shannon approximation unrealistic. To compute the capacity, a penalty term is typically added to (4.11), to accommodate the finite block-length assumption [35].

The capacity is therefore expressed as

$$r_{n,m} = BW \left[\log_2 \left(1 + \gamma_{n,m}(t)\right) - \underbrace{\sqrt{\frac{V_{n,m}(t)}{l}} Q^{-1}(\epsilon) \log e}_{\text{Penalty term}} \right] \tag{4.12}$$

where Q is the complementary Gaussian cumulative distribution function based on the code-word decoding error probability ϵ, $\log e \approx 0.434$ is a constant constraint of the loss, and V is the channel dispersion defined by the following [35].

$$V_{n,m}(t) = 1 - \frac{1}{\left(1 + \gamma_{n,m}(t)\right)^2} \tag{4.13}$$

When the length of the code-word block l tends toward infinity, the achieved rate approach Shannon's approximation. If the code-word decoding error probability ϵ is constant, the channel rate loss is inversely proportional to $\sqrt{l}$ [35].

In this project, subnetworks with requirement of high data rates, with or without minimum SINR constraints based on desired common channel capacity is considered. The optimization problem must be approached as a joint problem, as it is desired to optimize radio resources for all N subnetworks simultaneously. As the optimization variable, the sum of throughput over devices can be weighted, but tends to favor channels with the best quality. Instead the minimum throughput of devices may introduce fairness, where the worst performance is considered [36].

Problem 1: Channel Allocation

The goal is to allocate channels for all subnetworks $\boldsymbol{c}$ to maximize the long-term averaged minimum throughput $r_{n,m}$ of devices in each subnetwork.

$$\text{P1:} \quad \left\{ \max_{\boldsymbol{c}} \lim_{T \to \infty} \frac{1}{T} \sum_{t \in \mathcal{T}} \min \left(\{ r_{n,m}(c_n; t) \}_{m=1}^{M} \right) \right\}_{n=1}^{N} \tag{4.14}$$

$$\text{s.t.} \quad \gamma_{n,m}(t) \geq \gamma_{n,m}^{\text{req}}, \quad \forall n \in \mathcal{N}, \forall m \in \mathcal{M}, \forall t \in \mathcal{T} \tag{4.15}$$

$$c_n(t) \in \mathcal{K}, \quad \forall n \in \mathcal{N}, \forall t \in \mathcal{T} \tag{4.16}$$

The constraint in (4.15) relates the minimum required SINR, where communication is considered reliable. (4.16) constraints the allocation of channels to the available set of channels $\{c_n \in \mathcal{K} | n = 1, \ldots, N\}$.

Problem 2: Joint Power and Channel Allocation

This problem is a modified version of the channel allocation problem, extended to the concept of transmit power optimization. The goal is to jointly allocate transmit powers $\boldsymbol{p}$ and channels $\boldsymbol{c}$ for all subnetworks to maximize the long-term averaged minimum throughput $r_{n,m}$ of devices in each subnetwork.

$$\text{P2:} \quad \left\{ \max_{\boldsymbol{p}, \boldsymbol{c}} \lim_{T \to \infty} \frac{1}{T} \sum_{t \in \mathcal{T}} \min \left(\{ r_{n,m}(p_n, c_n; t) \}_{m=1}^{M} \right) \right\}_{n=1}^{N} \tag{4.17}$$

$$\text{s.t.} \quad \text{(4.15) and (4.16)}$$

$$p_n(t) \geq 0, \quad \forall n \in \mathcal{N}, \forall t \in \mathcal{T} \tag{4.18}$$

$$\sum_{n=1}^{N} p_n(t) \leq P^{\max}, \quad \forall t \in \mathcal{T} \tag{4.19}$$

The constraints (4.18) and (4.19) respectively relates subnetworks to transmit with positive continuous powers, and not to exceed a maximum tolerated power level.

Chapter 5

Federated Learning for Radio Resource Allocation

This chapter presents the proposed MARL solutions for radio resource optimization of autonomous 6G subnetworks, densely deployed in a indoor-factory environment. A training approach using FRL will be presented, followed by a more in-depth description in combination with MARL methods.

5.1 Federated Learning Based Resource Allocation Method

Recall centralized and distributed MARL frameworks presented in section 3.1.5. Respectively, compromised data security and potential convergence problems were identified as challenges, limiting the potential for practical applications. The concept of FL, presented in section 3.2, is expected to overcome these drawbacks, by introducing new concepts for sharing knowledge implicitly. Hence, an adaptation of FRL may potentially make realization of such systems viable.

Consider agents distributed in the subnetworks, collecting and processing local sensitive data. Through local training, information featured in the data is embedded into the weights of the local model. Assuming training is based on a federated model with global weights containing knowledge from all participants, the embedded sensitive data becomes masked. Hence, the weights can convey information learned from sensitive data, without compromising security. The federated model is periodically updated with local weights from all agents with aggregation methods, and the updated global weights are then utilized by agents to continue training.

To handle aggregation, it is assumed a central coordinator is equipped with hardware resources to process weights from all agents in a client-to-server model. Communication is sufficient to relay weights perfectly without introducing additional latency. Furthermore, encryption of transmitted weights and decryption of received weights has already been performed correctly without any additional latency or error. Agents share the same problem and train similar models, that is radio resource allocation based on the same set of actions, and similar observations from a local perspective. Hence, data is partitioned horizontally where training is based on local samples with identical features. At a synchronized aggregation interval, all agents upload local weights to the central coordinator. When all the weights have been collected, the total average is used as the weights for the federated model. The global weights are then returned to each agent, overwriting local weights and distributed training continues. An illustration of the n^{th} agent in the proposed client-to-server HFRL framework is illustrated in figure 5.1.

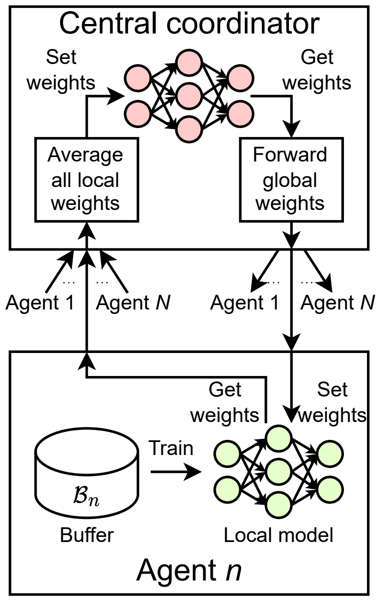

Figure 5.1: Illustration of the proposed HFRL framework for 6G in-robot subnetworks in a client-to-server model architecture.

With aggregation of distributed trained weights it is expected to share knowledge similar to the centralized approach. Here it is remarked, if the aggregation interval goes towards infinity, the framework will eventually become equivalent to the distributed learning scenario. However, if agents consistently receive averaged weights independent learning behaviors may be canceled out, resulting in slow or no convergence. Furthermore, performing aggregation temporally demands radio and processing resources from all participants, where the central coordinator potentially can become a bottleneck in dense scenarios with low intervals. This suggest a challenge on selecting a suitable aggregation interval.

5.2 Multi-Agent Reinforcement Learning Solutions

The optimization problems described in 4.3.1 indicate high complexity, where getting stuck on a local optimum imposes a challenge. In section 3.1.5, different MARL solution methods were discussed. A benefit of RL is the approach to exploration, which forces the optimizer to look for a global optimum. However the design of RL solutions requires correct definition of state observation and action spaces, the reward signal, and the policy. Different approaches for designing reward signals will be investigated, to motivate certain learning behaviors. Two policies from earlier RL discussions, DDQN and PPO, will be presented in the context of MARL.

5.2.1 Observation Space

For each in-robot subnetwork to perform an observation of the environment, is is assumed that each subnetwork is equipped with sensing capabilities to obtain measurements of the aggregate interference power on all used channels at time t.

$$\boldsymbol{o}_n^t = \left[I_{n,1}^t, \ldots, I_{n,K}^t\right]^T \in \mathbb{R}^{K\times 1} \tag{5.1}$$

Where interference power $I_{n,k}$ observed by the n^{th} agent, is based on other agents actively transmitting on channel k, given by the denominator in (4.10). Hence, the lowest observable interference level is the noise floor.

In order to achieve faster learning convergence, it is beneficial to ensure that the feature representations fall within a small range, to avoid certain features dominating the learning process. To achieve this, normalization techniques such as min-max or z-score may be used for feature scaling. The interference range can span from the noise floor to an unknown high value, which needs to be determined through analyzing simulation results.

5.2.2 Action Space

In the case of channel selection, subnetworks transmit with predefined power $P_{\max}$. The available frequency band is divided into a set of K equally sized channel bandwidths. This leads to a discrete action space, where the agent selects one of the K channels to use for communication.

$$\mathcal{A}_{\text{channel}} = \{c_1, \dots, c_K\} \tag{5.2}$$

For joint power and channel allocation, another aspect is to control the transmit powers of subnetworks in order to avoid unnecessary interference generation. The discrete action space can be obtained by dividing the continuous transmit power range into U equally spaced quantization levels.

$$\mathcal{A}_{\text{power}} = \{p_1, \dots, p_U\} \tag{5.3}$$

The action spaces $\mathcal{A}_{\text{channel}}$ and $\mathcal{A}_{\text{power}}$ can be merged to formulate the action space for the joint resource allocation problem. For every action taken, a channel and power level is selected. This does however increase the size of the action space, as the discrete case requires all possible permutations of channels and power levels. Thus, the action space for the joint allocation problem is formulated as:

$$\mathcal{A}_{\text{joint}} = \{\{c_1, p_1\}, \{c_1, p_2\}, \dots, \{c_K, p_U\}\} \tag{5.4}$$

5.2.3 Reward Signal

The reward signal must be based on a metric representing how good an action was in a given state. The optimization problems, (4.14) and (4.17), support applications in requirement of high data rates with or without minimum rate constraints. To capture the penalty of the SINR constraint in (4.15), a binary reward signal is introduced

$$R_n = \begin{cases} +\alpha & \text{if } \gamma_{n,m} \geq \gamma_{n,m}^{\text{req}}, \forall n, m \\ -\alpha & \text{otherwise} \end{cases} \tag{5.5}$$

where α is a constant reward value and $\gamma_{n,m}^{\text{req}}$ is the minimum required SINR. Furthermore, the agent can be motivated to achieve learning behavior towards higher data rates, by maximizing the achieved rate

$$R_n = \begin{cases} r_n & \text{if } \gamma_{n,m} \geq \gamma_{n,m}^{\text{req}}, \forall n, m \\ r_n - \lambda \Delta r_n & \text{otherwise} \end{cases} \tag{5.6}$$

where the tuning parameter λ can be adjusted for training, and the penalty can be based on $\Delta r_n = \min \left\{ r_{n,m}^{\text{req}} - r_{n,m} \right\}_{m=1}^{M}$.

5.2.4 Multi-Agent Double Deep Q-Network

In the case for value-based DDQN, the Q-networks are structured with DNNs. Observation of the state $\boldsymbol{o}_n^t \in \mathbb{R}^{K \times 1}$ is mapped to the radio resource action space $\mathcal{A} \in \mathbb{R}^{i \times 1}$ with a total of i actions [29]. The output is a probability distribution vector for decision making, which by taking the greatest probability is limited to discrete action spaces. However, the continuous power action space can be approximated as a discrete action space with (5.3) [29]. The method is extended to the context of Multi-Agent Double Deep Q-Network (MADDQN), where each subnetwork is deployed with an agent for distributed execution without changing the architecture of the Q-networks. Agents may be trained independent or joint with cooperative techniques, based on the used training framework.

A benefit with Q-learning, is the ability to control the degree of exploration, allowing to learn more complex behaviors in the communication system. The ϵ-greedy strategy tackles the trade-off between exploration and exploitation, where the exploration probability ϵ is decayed from ϵ_{max} to ϵ_{min} over ϵ_{steps} time steps.

$$\epsilon(t) = \max\left(\epsilon_{\text{min}}, \epsilon_{\text{max}} - t \cdot (\epsilon_{\text{max}} - \epsilon_{\text{min}})/\epsilon_{\text{steps}}\right) \tag{5.7}$$

Hence, the degree of exploration is decayed as training progress. The target network used for training evaluation is similar to the main Q-networks. Information collected for training is the prior observation that lead to a certain action, from which a new observation and reward is obtained.

5.2.5 Multi-Agent Proximal Policy optimization

The approach for PPO is categorized as a policy-based method [22], where radio resource optimization is explored and exploited with stochastic methods. The PPO relies on the policy to determine the quality of actions, which cannot be performed if the actions are sampled uniformly at random with ϵ-greedy. The method is extended to the context of Multi-Agent Proximal Policy Optimization (MAPPO), where each subnetwork is deployed with an agent for distributed execution.

To stabilize training, the policy network for each agent is structured in an actor-critic architecture. During a forward pass, the network generate action probabilities for radio resource management, and the critic value. The actor is a policy-based DNN in control of the agent behavior, where the output layer is a vector with same shape as the action space, used to generate a probability distribution for decision-making. For discrete action spaces, the observations are mapped to a categorical distribution over actions, where a discrete action can be sampled [37]. Here it is also remarked, that a benefit for MAPPO is the ability to use the mean and standard deviation of the observation vectors to produce a multivariate Gaussian distribution, from which an continuous action can be sampled [37]. The critic is a DNN similar to the actor, however the output layer is reduced to a single state-dependent value. Together, the actor action decision and log-probability, and the critic value-prediction, is collected in the replay-buffer during execution and later used for training.

5.2.6 Training Frameworks

Learned policies are distributed over agents in each subnetwork. How the agents learn, are determined by the deployed training framework. In this section, three approaches to training will be considered and presented in context of in-robot subnetworks: centralized, distributed and the proposed HFRL method. The former centralized and distributed methods are introduced to the discussion to provide a more in-depth understanding of the conventional training frameworks, and insight to the key differences that will be introduced in the presented approach.

The three training methods are illustrated in figure 5.2, where the agents are deployed in each subnetwork and are able to access the local sensing information. Training is divided into episodes, where the environment is randomly initialized at the beginning of every episode. During a training episode, each agent $\mathcal{N} = [1, \ldots, N]$ interacts with the environment for T timesteps and collects experiences for a replay buffer.

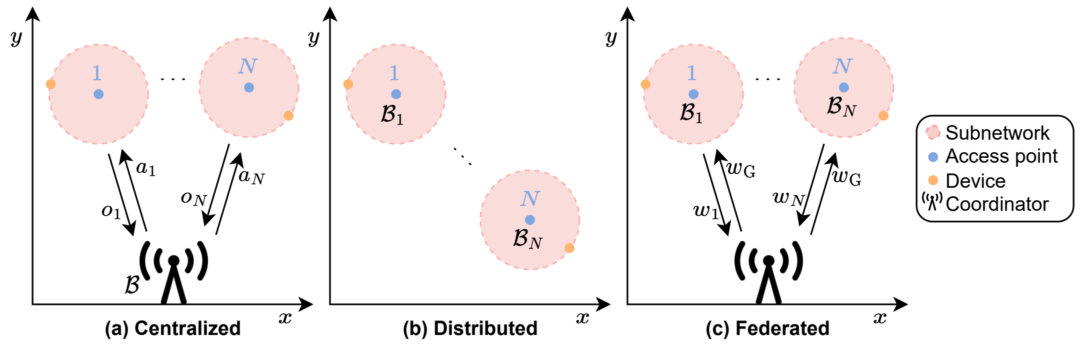

Figure 5.2: Illustration of training frameworks, (a) centralized where the model is trained in a central coordinator, (b) distributed models are trained locally, and (c) federated where local weights are updated with a central coordinator.

In centralized training, agents are cooperating to train a shared model, where every experience is stored in a shared buffer $\mathcal{B}$. A central coordinator collects observations from all agents, executes training, and provide the agents with new actions. For distributed training, each agent is training a local model from a private buffer $\mathcal{B}_n$, where no knowledge is being shared across subnetworks. The federated approach is based on distributed training, however a global agent to handle aggregation is introduced. The global agent is periodically updated every T^{Agg} steps, where all local model weights are collected, and the average of weights are aggregated to the federated model. Every subnetwork then receives the updated global weights, and continues training.

Distributed execution may result in greedy behavior. If a channel is observed to be very noisy it is likely that no subnetwork will use it for communication. In the next time step, the unused channel will have close to no interference resulting in all subnetworks jumping back to that specific channel. As long as subnetworks are allowed to update actions simultaneously this pattern will repeat, where all subnetworks jumps between the same two channels, resulting in very poor performance. Inspired by the work in [18, 21], simultaneously action updates can be prevented by introducing a random action transition step delay for each subnetwork τ_n, where subnetworks only are allowed to update actions and store experiences on specific random time intervals. The random delay is based on the value for maximum allowed delay $\tau_{\max}$, and all delays are generated in the beginning of every episode.

The two policies MADDQN and MAPPO were earlier introduced as different approaches to the same problem. In combination with the training approaches presented in this section, a total of six algorithms will be presented. The centralized, distributed, and federated learning approaches for MADDQN are respectively denoted as C-MADDQN, D-MADDQN, and F-MADDQN in algorithms 4, 5, and 6. Similarly, the approaches with MAPPO are respectively denoted and presented as C-MAPPO, D-MAPPO, and F-MAPPO in algorithms 7, 8, and 9. The centralized and distributed frameworks may also provide useful benchmarks for evaluation of the training performance.

Algorithm 4 C-MADDQN (Centralized training)

```
1: Initialize: $\mathcal{N}, Q^{\text{main}}, Q^{\text{target}}, \mathcal{B}$, factory environment
2: for each episode do
3:     Deploy all subnetworks with random delay index $\{\tau_n\}_{n=1}^{N}$ from $\tau_{\text{max}}$
4:     for each timestep, $t, \ldots, T$ do
5:         if $t$ modulo $\tau_n == 0, n \in \mathcal{N}$ then
6:             Subnetwork $n$ update action $a_n^t$ with $\epsilon$-greedy and $Q^{\text{main}}$
7:         end if
8:         Execute actions $\boldsymbol{a}$ in environment and decay of $\epsilon$ with (5.7)
9:         if $t$ modulo $\tau_n == 0, n \in \mathcal{N}$ then
10:            Store transition $\left(\boldsymbol{o}_n^t, a_n^t, r_n^t, \boldsymbol{o}_n^{t+1}\right)$ in replay buffer $\mathcal{B}$
11:        end if
12:        if $t$ modulo $T^{\text{main}} == 0$ then
13:            Train network $Q^{\text{main}}$ with minibatch from $\mathcal{B}$
14:        end if
15:        if $t$ modulo $T^{\text{target}} == 0$ then
16:            Update network $Q^{\text{target}}$ with weights from $Q^{\text{main}}$
17:        end if
18:    end for
19: end for
```

Algorithm 5 D-MADDQN (Distributed training)

```
1: Initialize: $\mathcal{N}, Q_n^{\text{main}}, Q_n^{\text{target}}, \mathcal{B}_n, \forall n$. Factory environment
2: for each episode do
3:     Deploy all subnetworks with random delay index $\{\tau_n\}_{n=1}^{N}$ from $\tau_{\text{max}}$
4:     for each timestep, $t, \ldots, T$ do
5:         if $t$ modulo $\tau_n == 0, n \in \mathcal{N}$ then
6:             Subnetwork $n$ update action $a_n^t$ with $\epsilon$-greedy and $Q_n^{\text{main}}$
7:         end if
8:         Execute actions $\boldsymbol{a}$ in environment and decay of $\epsilon$ with (5.7)
9:         if $t$ modulo $\tau_n == 0, n \in \mathcal{N}$ then
10:            Store transition $\left(\boldsymbol{o}_n^t, a_n^t, r_n^t, \boldsymbol{o}_n^{t+1}\right)$ in replay buffer $\mathcal{B}_n$
11:        end if
12:        if $t$ modulo $T^{\text{main}} == 0$ then
13:            Train networks $Q_n^{\text{main}}$ with minibatch from $\mathcal{B}_n$, $n \in \mathcal{N}$
14:        end if
15:        if $t$ modulo $T^{\text{target}} == 0$ then
16:            Update networks $Q_n^{\text{target}}$ with weights from $Q_n^{\text{main}}$, $n \in \mathcal{N}$
17:        end if
18:    end for
19: end for
```

Algorithm 6 F-MADDQN (FRL approach)

```
1: **Initialize:** $\mathcal{N}, Q_n^{\text{main}}, Q_n^{\text{target}}, \mathcal{B}_n, \forall n$. Factory environment and $Q^{\text{Agg}}$
2: **for** each episode **do**
3:     Deploy all subnetworks with random delay index $\{\tau_n\}_{n=1}^N$ from $\tau_{\max}$
4:     **for** each timestep, $t, \ldots, T$ **do**
5:         **if** $t$ modulo $\tau_n == 0$ **then**
6:             Subnetwork $n$ update action $a_n^t$ with $\epsilon$-greedy and $Q_n^{\text{main}}$
7:         **end if**
8:         Execute actions $\boldsymbol{a}$ in environment and decay of $\epsilon$ with (5.7)
9:         **if** $t$ modulo $\tau_n == 0, n \in \mathcal{N}$ **then**
10:            Store transition $\left(\boldsymbol{o}_n^t, a_n^t, r_n^t, \boldsymbol{o}_n^{t+1}\right)$ in replay buffer $\mathcal{B}_n$
11:        **end if**
12:        **if** $t$ modulo $T^{\text{main}} == 0, n \in \mathcal{N}$ **then**
13:            Train networks $Q_n^{\text{main}}$ with minibatch from $\mathcal{B}_n$, $n \in \mathcal{N}$
14:        **end if**
15:        **if** $t$ modulo $T^{\text{target}} == 0$ **then**
16:            Update networks $Q_n^{\text{target}}$ with weights from $Q_n^{\text{main}}$, $n \in \mathcal{N}$
17:        **end if**
18:        **if** t modulo $T^{\text{Agg}} == 0$ **then**
19:            Aggregate averaged weights $Q_n^{\text{main}}$, $n \in \mathcal{N}$ to network $Q^{\text{Agg}}$
20:            Update networks $Q_n^{\text{main}}$ with weights from $Q^{\text{Agg}}$, $n \in \mathcal{N}$
21:            Update networks $Q_n^{\text{target}}$ with weights from $Q_n^{\text{main}}$, $n \in \mathcal{N}$
22:        **end if**
23:    **end for**
24: **end for**
```

Algorithm 7 C-MAPPO (Centralized training)

```
1: **Initialize:** $\mathcal{N}$, $\pi^\theta$, $\pi^{\theta,\text{old}}$, $\mathcal{B}$, factory environment
2: **for** each episode **do**
3:     Deploy all subnetworks with random delay index $\{\tau_n\}_{n=1}^N$ from $\tau_{\max}$
4:     **for** each timestep, $t, \ldots, T$ **do**
5:         **if** $t$ modulo $\tau_n == 0, n \in \mathcal{N}$ **then**
6:             Subnetwork $n$ update action $a_n^t$ with $\pi^{\theta,\text{old}}$
7:         **end if**
8:         Execute actions $\boldsymbol{a}$ in environment
9:         **if** $t$ modulo $\tau_n == 0, n \in \mathcal{N}$ **then**
10:            Store transition $\left(\boldsymbol{o}_n^t, a_n^t, r_n^t, \boldsymbol{o}_n^{t+1}\right)$ in replay buffer $\mathcal{B}$
11:        **end if**
12:        **if** $t$ modulo $T^{\text{main}} == 0$ **then**
13:            Train network $\pi^\theta$ with minibatch from $\mathcal{B}$ for $K$ epochs
14:            Update network $\pi^{\theta,\text{old}}$ with weights from $\pi^\theta$
15:        **end if**
16:    **end for**
17: **end for**
```

Algorithm 8 D-MAPPO (Distributed training)

```
1: Initialize: $\mathcal{N}$, $\pi_n^\theta$, $\pi_n^{\theta,\mathrm{old}}$, $\mathcal{B}_n$, $\forall n \in \mathcal{N}$. Factory environment
2: for each episode do
3:     Deploy all subnetworks with random delay index $\{\tau_n\}_{n=1}^N$ from $\tau_{\max}$
4:     for each timestep, $t, \ldots, T$ do
5:         if $t$ modulo $\tau_n == 0, n \in \mathcal{N}$ then
6:             Subnetwork $n$ update action $a_n^t$ with $\pi_n^{\theta,\mathrm{old}}$
7:         end if
8:         Execute actions $\boldsymbol{a}$ in environment
9:         if $t$ modulo $\tau_n == 0, n \in \mathcal{N}$ then
10:            Store transition $\left(\boldsymbol{o}_n^t, a_n^t, r_n^t, \boldsymbol{o}_n^{t+1}\right)$ in replay buffer $\mathcal{B}_n$
11:        end if
12:        if $t$ modulo $T^{\mathrm{main}} == 0$ then
13:            Train network $\pi_n^\theta$ with minibatch from $\mathcal{B}_n$ for $K$ epochs, $n \in \mathcal{N}$
14:            Update network $\pi_n^{\theta,\mathrm{old}}$ with weights from $\pi_n^\theta$, $n \in \mathcal{N}$
15:        end if
16:    end for
17: end for
```

Algorithm 9 F-MAPPO (FRL approach)

```
1: Initialize: $\mathcal{N}$, $\pi_n^\theta$, $\pi_n^{\theta,\mathrm{old}}$, $\mathcal{B}_n$, $\forall n \in \mathcal{N}$. Factory environment and $\pi^{\theta,\mathrm{Agg}}$
2: for each episode do
3:     Deploy all subnetworks with random delay index $\{\tau_n\}_{n=1}^N$ from $\tau_{\max}$
4:     for each timestep, $t, \ldots, T$ do
5:         if $t$ modulo $\tau_n == 0, n \in \mathcal{N}$ then
6:             Subnetwork $n$ update action $a_n^t$ with $\pi_n^{\theta,\mathrm{old}}$
7:         end if
8:         Execute actions $\boldsymbol{a}$ in environment
9:         if $t$ modulo $\tau_n == 0, n \in \mathcal{N}$ then
10:            Store transition $\left(\boldsymbol{o}_n^t, a_n^t, r_n^t, \boldsymbol{o}_n^{t+1}\right)$ in replay buffer $\mathcal{B}_n$
11:        end if
12:        if $t$ modulo $T^{\mathrm{main}} == 0$ then
13:            Train network $\pi_n^\theta$ with minibatch from $\mathcal{B}_n$ for $K$ epochs, $n \in \mathcal{N}$
14:            Update network $\pi_n^{\theta,\mathrm{old}}$ with weights from $\pi_n^\theta$, $n \in \mathcal{N}$
15:        end if
16:        if t modulo $T^{\mathrm{Agg}} == 0$ then
17:            Aggregate averaged weights $\pi_n^\theta$, $n \in \mathcal{N}$ to network $\pi^{\theta,\mathrm{Agg}}$
18:            Update networks $\pi_n^\theta$ with weights from $\pi^{\theta,\mathrm{Agg}}$, $n \in \mathcal{N}$
19:        end if
20:    end for
21: end for
```

Chapter 6

Performance Evaluation

An overview of the simulation procedure and used parameters will be presented, followed by an introduction of the benchmark algorithms used for testing. This leads to a presentation and discussion of the simulation results.

6.1 Simulation Overview

The simulation is based on the system described in chapter 5, implemented with object oriented programming in Python 3.10.4. This section provides an overview of the simulation procedure and the used simulation parameters.

6.1.1 Procedure

Simulations are executed on two different machines. The computationally demanding training simulations are executed on a Ucloud-server with 16 vCPU (Intel Xeon Gold 6130) and 96 GB RAM. Further testing of models and any other development, has been performed on a quad-core laptop with 12 GB RAM.

It was ealier mentioned in section 5.2.6, training is executed in episodes with T timesteps. In conventional episodic tasks, e.g., a sports game or a puzzle, the terminal state is determined by the environment when a goal is scored or the puzzle is solved. The subnetworks however, do not have a terminal state, hence every action transition generated with the maximum delay mechanism is perceived as a terminal state. Additionally, the same states are being stored in the replay buffer and used for training. Algorithms used for resource allocation are MADDQN and MAPPO. These methods share the discrete resource allocation objective, but approaches the problem differently which was discussed in section 3.1.4. Results focus primarily on the channel allocation problem, with the action space in (5.2).

6.1.2 Simulation Parameters

The subnetworks are deployed in the indoor factory, earlier described with the mobility model in section 4.2.2, where each subnetwork contains an AP and a single device, $M = 1$. During training, $N = 20$ subnetworks are deployed in a factory with sparse clutter. Investigations of model robustness may be evaluated with various number of subnetwork in sparse and dense clutter. Radio resources are assumed to be sparse, hence the number of frequency channels is set to $K = 4$, forcing subnetworks to share the very limited resources. Radio propagation parameters are inspired by the works [25, 30]: the lowest frequency $f_c = 6$ GHz, channel bandwidth $BW = 10$ MHz, transmit powers $p_n(t) = -10$ dBm, noise figure $NF = 10$ dB, and a decorrelation distance $d_\delta = 10$ m. Insight to the impact of the maximum action transition delay can be found in appendix A.1, where the final value for simulations is $\tau_{\text{max}} = 10$. The process of observation normalization are discussed in appendix A.2. All parameters for the environment are fixed for all simulation results, and presented in table 6.1.

A similar MARL problem with MADDQN was solved in [25] with distributed learning, which is used as a inspirational source for parameters, i.e., ϵ-greedy, discount factor $\gamma = 0.99$, learning rate $\alpha = 0.001$. However, for deployment in the factory environment described in table 6.2, a batch size of $B = 256$ indicated best initial signs of convergence. Vanilla and modified approaches of the PPO is discussed in the works [27, 38], where the default clipping parameter $\epsilon_{\text{clip}} = 0.2$ has been suggested. The MAPPO presented in this project is a low-effort vanilla

Parameter		**Value**	
Total factory area	$\mathcal{R}$	[180, 80]	[m]
Clutter type table 4.2		Sparse	
Number of subnetworks	N	20	[·]
Timestep	t	0.005	[s]
Number of episode		2000	[·]
Number of steps per episode	T	200	[·]
Subnetwork separation distance	d_{min}	1	[m]
Subnetwork radius	d_r	1	[m]
Subnetwork velocity		3	[m/s]
Transmit power	$p_n(t)$	-10	[dBm]
Number of frequency channels	K	4	[·]
Carrier frequency	f_c	6	[GHz]
Bandwidth per subnetwork	BW	10	[MHz]
Noise figure	NF	10	[dB]
Shadowing decorrelation distance	d_δ	10	[m]
Max action switch delay	τ_{max}	10	[·]

Table 6.1: Simulation parameters for the environment and in-robot subnetworks.

implementation, where the number of epochs $\kappa = 2$ and model update interval $T^{\text{main}} = 8$ was chosen as convergence was indicated in early episodes. MADDQN and MAPPO parameters are presented in table 6.2. Both methods share the same network structure $|\mathcal{S}| - 100 - 100 - |\mathcal{A}|$, however MADDQN use ReLu for hidden-layer activation whereas PPO use hyperbolic tangent with softmax as output. In both cases, Adam is utilized, respectively with Huber and MSE loss for MADDQN and MAPPO.

No value for the optimal aggregation interval is known for this problem, hence a good interval is to be determined through analyzing training results. Episodes are rather short, consisting of $T = 200$ steps equivalent to 1 second, where models learn from experiences on every timestep. Hence, based on the discussion in section 5.1, it is deemed reasonable to span the aggregation intervals over multiple episodes, in order to avoid cancellation of independent features captured within successive episodes.

Parameter		**Value**
Multi-Agent Double Deep Q-Network (MADDQN)		
Learning rate	α	0.001
Batch size	B	256
Reward discount rate	γ	0.99
Initial epsilon greedy	ϵ	1.0
Epsilon greedy max	ϵ_{max}	1.0
Epsilon greedy min	ϵ_{min}	0.001
Number of epsilon greedy episodes	$\epsilon_{\text{episodes}}$	1500
Main model update interval	T^{main}	1
Target model update interval	T^{target}	10
Multi-Agent Proximal Policy Optimization (MAPPO)		
Learning rate	α	0.001
Batch size	B	256
PPO epochs	κ	2
Clipping parameter	ϵ	0.2
Model update interval	T^{main}	8

Table 6.2: Training parameters for MADDQN, and MAPPO.

6.2 Benchmark Algorithms

To establish a fair baseline for a training comparison, it is reasonable to benchmark the convergence of the proposed FRL method with centralized and distributed learning frameworks, both with distributed execution. A simple random resource selection algorithm is presented as baseline for the worst-case performance for selecting actions, the distributed heuristic greedy selection and Centralized Graph Coloring (CGC) as benchmark algorithms.

6.2.1 Centralized Training with Distributed Execution

Centralized training of a single model, where all observations from subnetworks are shared. The model learned is executed distributed over all subnetworks. By sharing experience, it is expected that the agent will learn more complex behaviors at a faster rate, and provide a good baseline for insight to the benefits of sharing.

The methods presented with centralized learning, is C-MADDQN in algorithm 4 and C-MAPPO in algorithm 7.

6.2.2 Distributed Training and Execution

Distributed training performed locally at each subnetwork, learning only from independent experiences. This approach will provide a good baseline for training, with more insight to scenarios where no experience is shared. The methods presented with distributed learning, is D-MADDQN in algorithm 5 and D-MAPPO in algorithm 8.

6.2.3 Random Action Selection

Selection of actions may simply be based on random allocation. At the beginning of every episode, a random action is allocated to each subnetwork based on a uniformly distributed random variable that spans over the action space. Actions across subnetworks are IID, which corresponds to a completely uncoordinated and distributed scenario.

6.2.4 Greedy Action Selection

The greedy allocation algorithm observes hidden SINR values based on each available action. For each subnetwork, the action with the greatest corresponding SINR value from previous time step is then chosen. The greedy channel allocation algorithm can then be formulated as the following.

$$c_n = \max \left\{ \mathrm{SINR}_n^k \right\}_{k=1}^{K} \tag{6.1}$$

The greedy allocation method does not consider other subnetworks and is uncoordinated and distributed. However as a greedy method, a random transition delay must be introduced to constraint simultaneously action decisions across subnetworks.

6.2.5 Centralized Graph Coloring Algorithm

The CGC algorithm utilize improper coloring to assign colors equivalent to channels for all subnetworks [30]. At each timestep t, the pair-wise interference power relationships among subnetworks, $\boldsymbol{I}(t) \in \mathbb{R}^{N \times N}$, are collected, and mapped to a mutual coupling graph G_t. Each vertex corresponds to a subnetwork, and edges are created by connecting each vertex to its $K-1$ nearest neighbors, where the weights of edges are equivalent to the interference power between the interfering subnetworks. Greedy coloring yields a reasonable channel assignment, and K-colorability of G_t is guaranteed with successive elimination of edges with lowest weight until the graph is colored with up to K colors [19].

As a result of the uncoordinated deployment of subnetworks, a centralized algorithm based on a global view of the network cannot be realized in practice. However, this will provide insights to how much improvement can be achieved over the greedy selection baseline.

6.3 Simulation Results

Simulations are presented in three result stages viz: Analysis of the training convergence, performance comparison with benchmarks, and sensitivity analysis based on the deployment and the factory clutter type.

6.3.1 Training Convergence

The channel allocation problem is approached with MADDQN and MAPPO, where different aggregation intervals $T^{\mathrm{Agg}} = [128, 256, 512, 1024]$ are investigated. The aggregation intervals spans from two updates every episode, to an update every fifth episode. Furthermore, aggregation intervals are not performed on the same timestep every episode, but shifts as the episodes progresses. Results for training are presented as the change in average reward over successive training episodes. The average is obtained from all in-robot subnetworks, for all steps within each episode. Two reward signals will be investigated, i.e., the binary reward in (5.5) with $\alpha = 10$, and the rate reward in (5.6) with $\lambda = 0$. The SINR threshold is set to $\gamma_{n,m}^{\mathrm{req}} \approx 33$ dB for all subnetworks, equivalent to a data rate of 11 bps/Hz.

Figure 6.1 shows the average data rate reward signal for the MADDQN. At convergence approximately after 1500 episodes, the MARL methods approach the distributed greedy selection baseline. However, a marginal advantage is not obtained due to ϵ-greedy has decayed to the minimum value, suggesting more exploration would likely result in surpassing the greedy baseline. At convergence, the FRL methods with greater aggregation intervals approach a marginally better performance than the centralized and distributed baselines. It is also shown that aggregation intervals can become too low, where features are overwritten before the agent has learned everything from local and shared information. However, this does in fact result in a discount on the problem, where performance is increased by using fewer resources in terms of update frequency.

Figure 6.2 shows the average binary reward signal for the MADDQN. Similar to the rate reward signal, convergence is achieved approximately after 1500 episodes where ϵ-greedy has decayed completely. A greater difference is observed between the centralized and distributed baselines, showing that the best features associated with binary rewards can be learned quicker through sharing knowledge. All

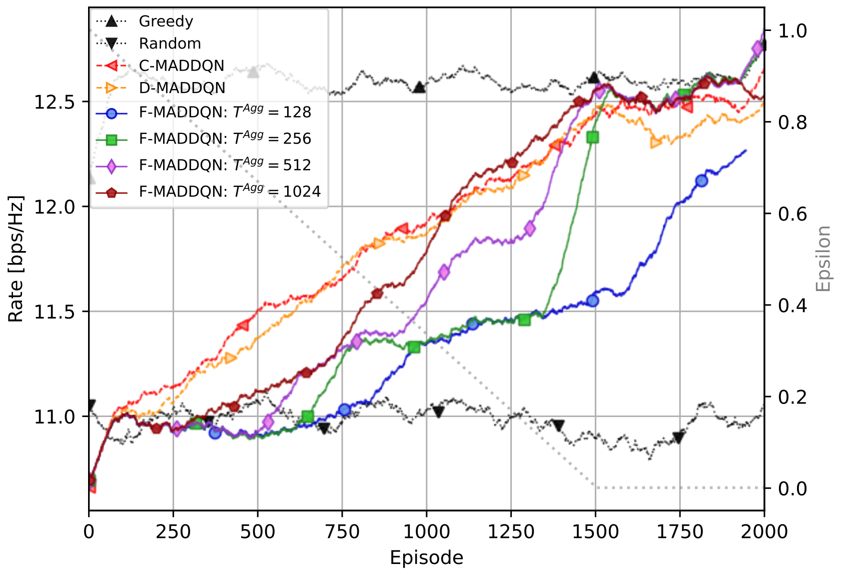

Figure 6.1: Training evaluation of agents, trained with MADDQN and data rate as the reward signal.

aggregation intervals for the FRL have marginally faster convergence than the centralized baseline, where the change in intervals does not have a significant impact on the training performance. In opposition to the rate reward signal, all features embedded in the binary reward signal can be learned over a low number of timesteps. Hence, any of the used aggregation intervals allows individual agents to learn everything from the shared knowledge, before a new aggregation update is performed. This suggest a potential for designing reward signals, for very high aggregation intervals to achieve a similar performance as to lower intervals in demand of more resources.

Figure 6.3 shows the average data rate reward signal for the MAPPO approach. At convergence, achieved approximately after 750 episodes, all of the used aggregation intervals for FRL approach the greedy selection baseline. Furthermore, greater aggregation intervals in the tested sequence tends to show marginal better performance. A benefit of the MAPPO, is to base learning over multiple epochs for the same minibatch. The number of epochs however, is a hyper-parameter specific

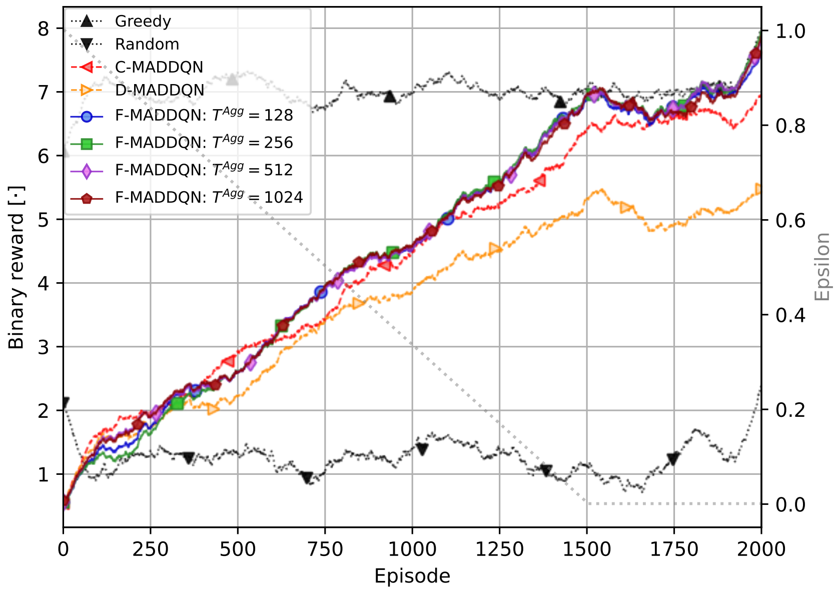

Figure 6.2: Training evaluation of agents, trained with MADDQN and the binary reward signal.

for the problem. The centralized baseline tends to converge slower than the distributed baseline, indicating that the number of epochs is sufficient to learn simple greedy behavior, but complex cooperative behavior may require even more epochs.

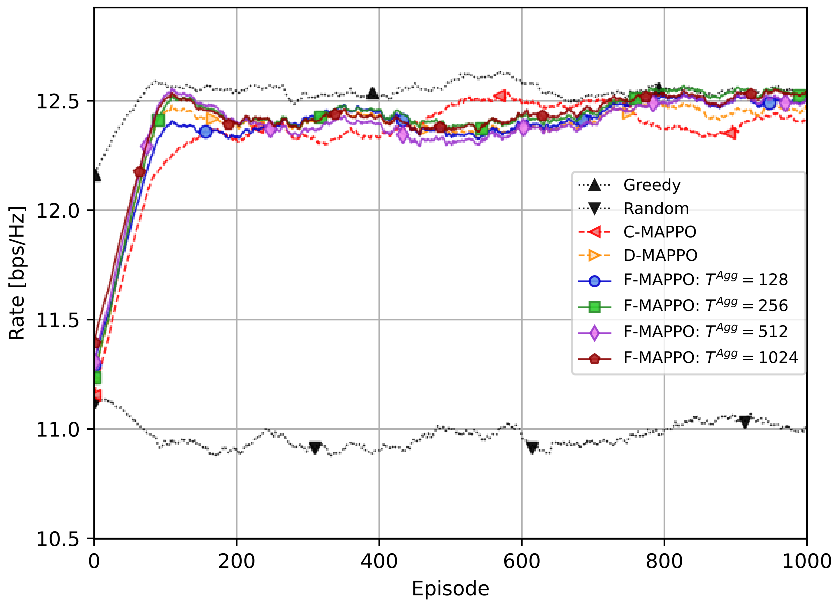

Figure 6.3: Training evaluation of agents, trained with MAPPO and data rate as the reward signal.

Figure 6.4 shows the average binary reward signal for the MAPPO. Similar to the rate reward signal, convergence is achieved approximately after 750 episodes, where performance does not exceed the greedy selection baseline. The centralized baseline indicates slow convergence in the start, but in opposition, it manages to approach the greedy selection baseline. At convergence, the FRL approach maintain a significant improvement over the distributed baseline.

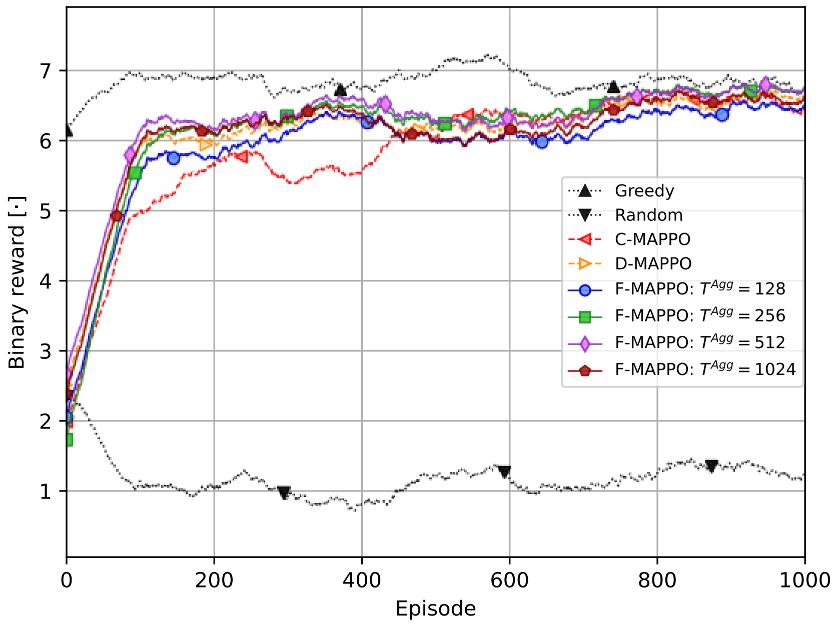

Figure 6.4: Training evaluation of agents, trained with MAPPO and the binary reward signal.

Training time required for convergence, that is 1500 episodes for MADDQN and 750 episodes for MAPPO, are illustrated in figure 6.5. For the MAPPO, a simplified sequential syntax for the actor-critic structure was used for experimentation. The MADDQN was implemented in an objective programming syntax, where the generation of unnecessary overhead is avoided. The MADDQN does however require longer training time, which possibly may be reduced by optimizing execution in the implementation. The time required for training may vary with the configuration of hyper parameters.

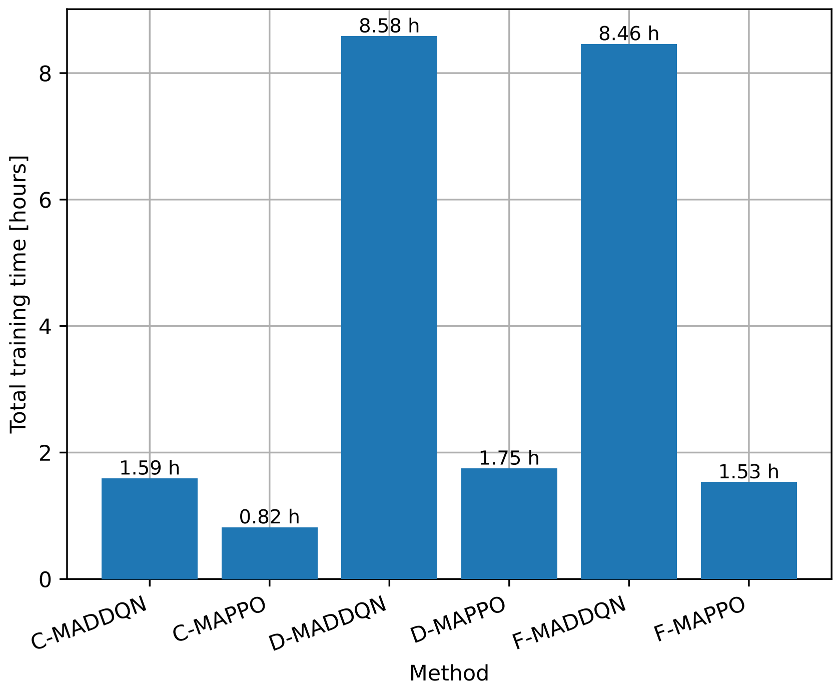

Figure 6.5: Measured convergence time for all methods.

6.3.2 Comparison with Benchmarks

The trained models are deployed in each in-robot subnetwork for distributed channel allocation, and the performance is compared with random, greedy, CGC, and the centralized and distributed training frameworks. Remark that no new models are trained in this section. A comparison of all methods are based on running the pre-trained models for 1000 episodes, and keeping record of all achieved channel rates. The training results indicated best convergence for larger aggregation intervals, however this observation was not consistent. To evaluate the trained models, and to keep fairness across methods, the aggregation interval is set to $T^{\text{Agg}} = 512$ during the performance evaluation.

Figure 6.6 shows the Cumulative Distribution Function (CDF) of achieved channel rate per in-robot subnetwork, for models trained with data rate as the reward signal. The proposed F-MADDQN performs marginally better than the distributed baseline, and significantly better than the centralized baseline below the 30th percentile. Furthermore, this performance is significantly better than the distributed greedy baseline. F-MAPPO similarly achieve marginally greater performance than the distributed baseline, and approaches the centralized baseline below the 30th percentile. Results for CGC illustrates the superior performance achieved by impractical centralized approaches in contrast to the remaining benchmarks.

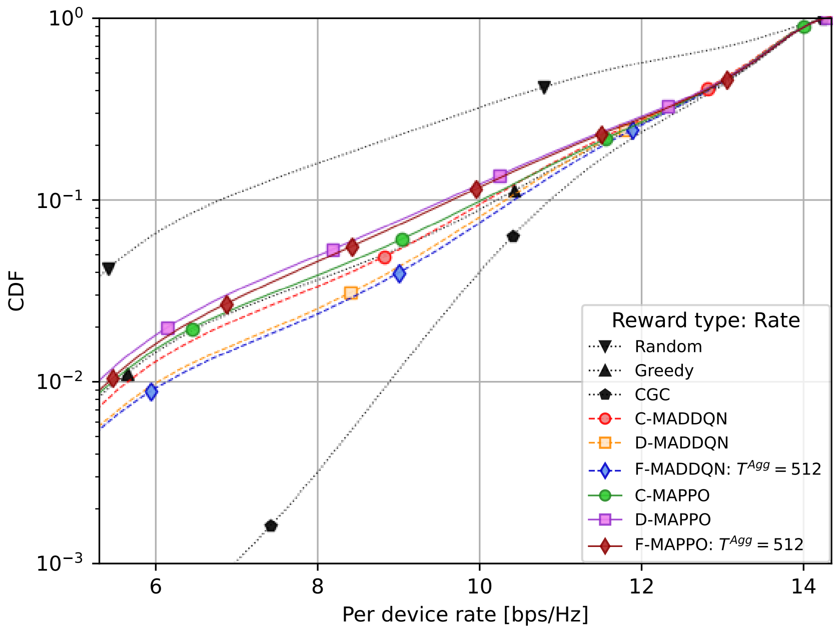

Figure 6.6: Performance evaluation of agents trained with data rate as the reward signal.

Figure 6.7 shows the CDF of achieved channel rate per in-robot subnetwork, for models trained with a binary reward signal. For F-MADDQN, a significant increased performance is observed below the 30th percentile, achieving better performance than the centralized, distributed, and greedy selection baselines. The performance of MAPPO however, differs where the distributed baseline manage to perform better than the FRL approach and the centralized baseline. This may have been caused by instability during training, where a great decrease in the centralized baseline was observed in comparison to the distributed baseline. F-MAPPO does however perform marginally better than the centralized baseline, above the 6th percentile.

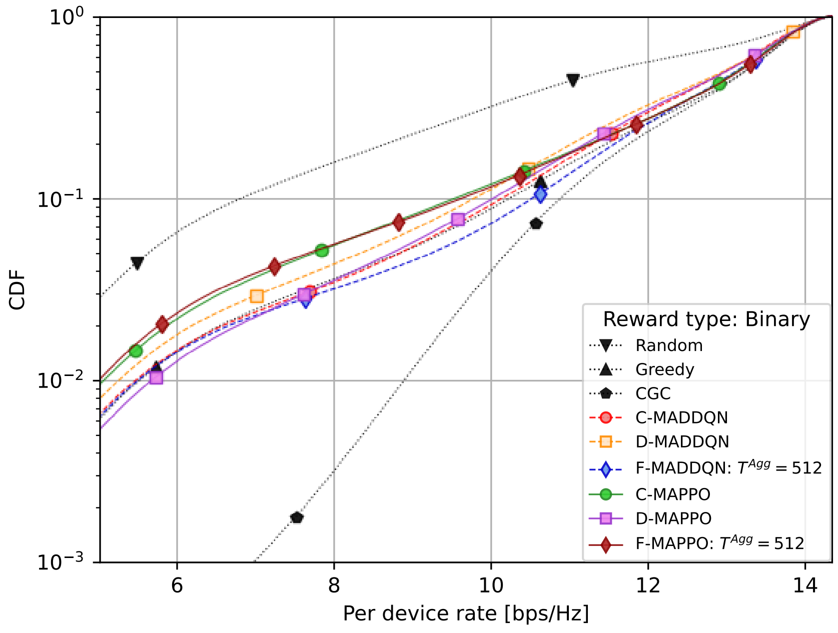

Figure 6.7: Performance evaluation of agents trained with the binary reward signal.

During the performance evaluation the average execution time was measured. A comparison of the execution times can be found in figure 6.8. No significant change in execution time is observed for different training frameworks, as to be expected since the output of each framework is identical models, where only the model weights are the key difference. The significant difference in MADDQN and MAPPO, is caused by variations of the implemented architectures.

6.3.3 Sensitivity Analyzes

Results for the sensitivity analysis with previous trained models are presented in this section, remark that no new models are trained for this investigation. The models trained in the factory with $N = 20$ subnetworks, are now deployed in scenarios with different number of subnetworks. In figure 6.9, the models trained with

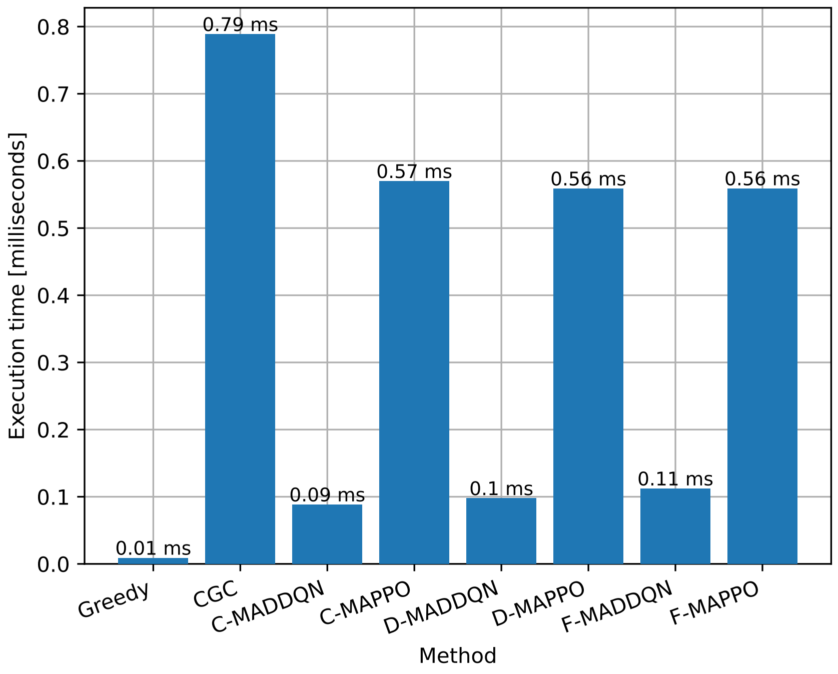

Figure 6.8: Average model execution time comparison of benchmarks and trained models.

the data rate reward signal are deployed in scenarios with $N \in [10, 20, 30, 40, 50]$ subnetworks. When the number of subnetworks increase, an overall decrease in performance with similar proportions across all methods are observed. This outcome was expected, as the optimal solution becomes constrained to more sparse radio resources. The MAPPO solution does not show great signs of robustness against a increasing number of subnetworks, as it converge towards the random baseline. In opposition, it is observed that the MADDQN solution maintains performance close to the greedy selection baseline and approaches the CGC baseline. For both MARL solution methods, the FRL approach maintains performance with a marginal improvement over their relatives.

In figure 6.9, the models trained with the binary reward signal are similarly deployed in scenarios with different number of subnetworks. The benchmarks algorithms maintain a performance decrease with signs of an exponential change, similar to the scenario with data rate as reward signal. The approaches with FRL

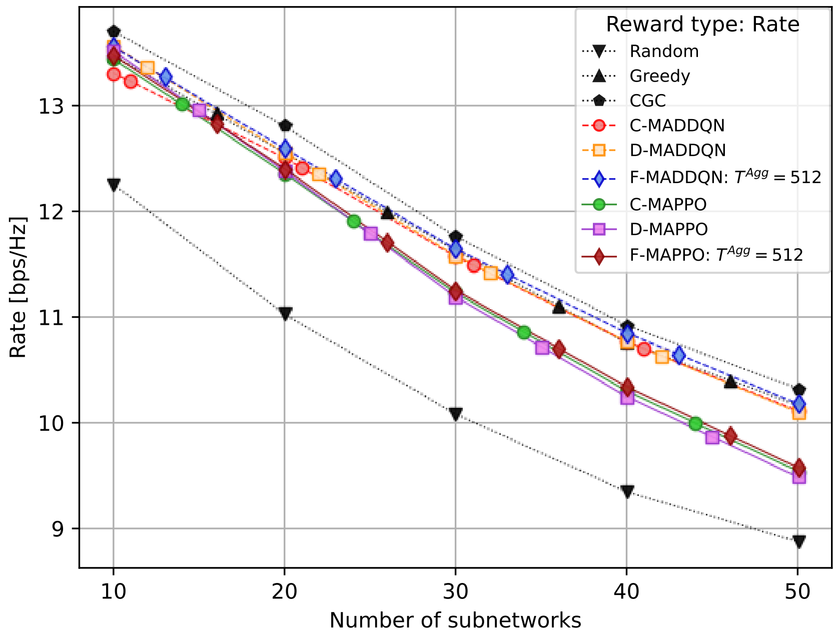

Figure 6.9: Sensitivity analysis of agents trained with the data rate reward signal ($N = 20$), versus the number of subnetworks.

however, does have a decrease in performance which is closer to linear. When the number of subnetworks exceeds $N = 20$, which was used for training, all solutions converge towards the random baseline. Careful consideration should then be taken when designing the reward signal in an environment with various numbers of subnetworks. Benefits may depend on the reward type and configuration of the trained model, however, the FRL approach does indicate the potential to obtain a marginal performance increase.

The models was trained in a factory environment with sparse clutter, that is an average clutter element size of $d_{\text{clutter}} = 10$ m with a density of $r_{\text{clutter}} = 20\%$. The models will now be introduced to environments with different types of clutter, presented with six different cases in table 4.2.

Figure 6.11 shows results for models trained with the data rate reward signal, tested in different types of clutter in the factory scenario presented in table 6.3.

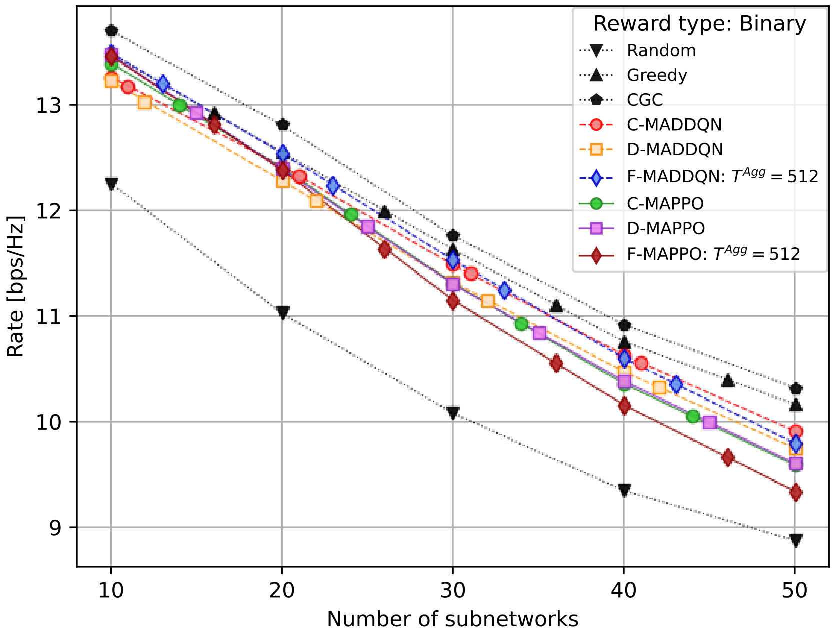

Figure 6.10: Sensitivity analysis of agents trained with the binary reward signal ($N = 20$), versus the number of subnetworks.

Parameter	**Sparse clutter**			**Dense clutter**		
Clutter size d_{clutter}	10 m			2 m		
Clutter density r_{clutter}	10%	20%	35%	45%	60%	80%

Table 6.3: Evaluation parameters for sparse and dense clutter scenarios, based on table 4.2.

The results are presented as error-bars, where the minimum, average, and maximum of all values obtained through simulation are presented. This shows that the FRL approach for both MARL methods are able to maintain its performance in comparison to the centralized and distributed baselines. Furthermore, similar to the sensitivity test against the number of subnetworks, the MADDQN is more robust changes in the environment, and maintains performance close to the greedy selection baseline. Remark that normalization of observations, were respectively based on sparse clutter with $r_{\text{clutter}} = 20\%$ and dense clutter with $r_{\text{clutter}} = 60\%$.

Figure 6.12 shows results for models trained with the binary reward signal, tested in different types of clutter in the factory scenario, earlier presented in table 6.3. In

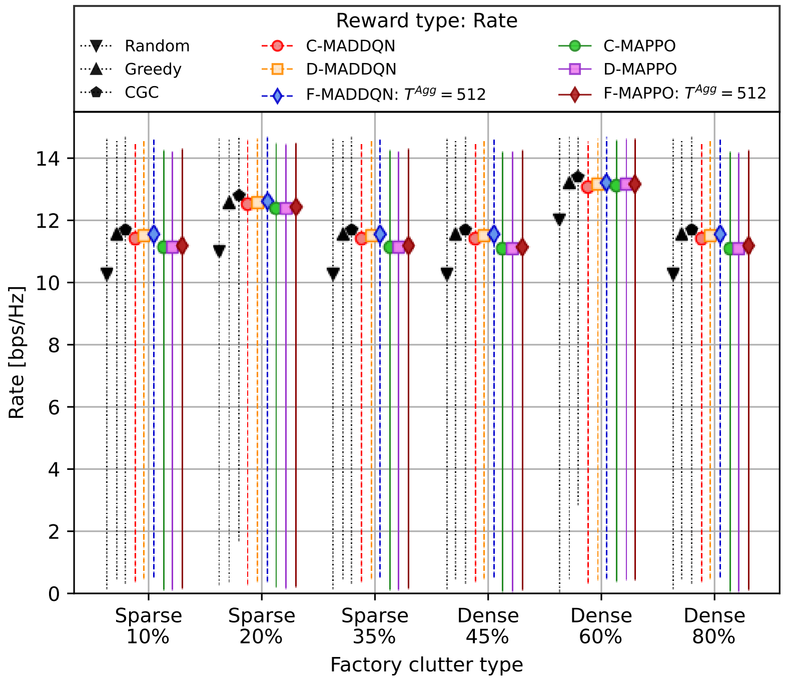

Figure 6.11: Sensitivity analysis of agents trained with the data rate reward signal in a sparse clutter environment with density $r_{\text{clutter}} = 20\%$, versus various scenarios of clutter.

opposition to the models trained with data rates as reward, a better performance for MAPPO is observed. The F-MAPPO does however suffer from different types of clutter, except for the dense scenario with $r_{\text{clutter}} = 60\%$. This indicates most resemblance between the sparse and dense scenarios respectively with $r_{\text{clutter}} = 20\%$ and $r_{\text{clutter}} = 60\%$. The F-MADDQN does however indicate similar signs of robustness to the scenario with data rate as reward type.

Note that although the results presented in this chapter is based on 3GPP models for industrial environments with different types of clutter [32], the FRL approach may be useful in different scenarios or other wireless systems.

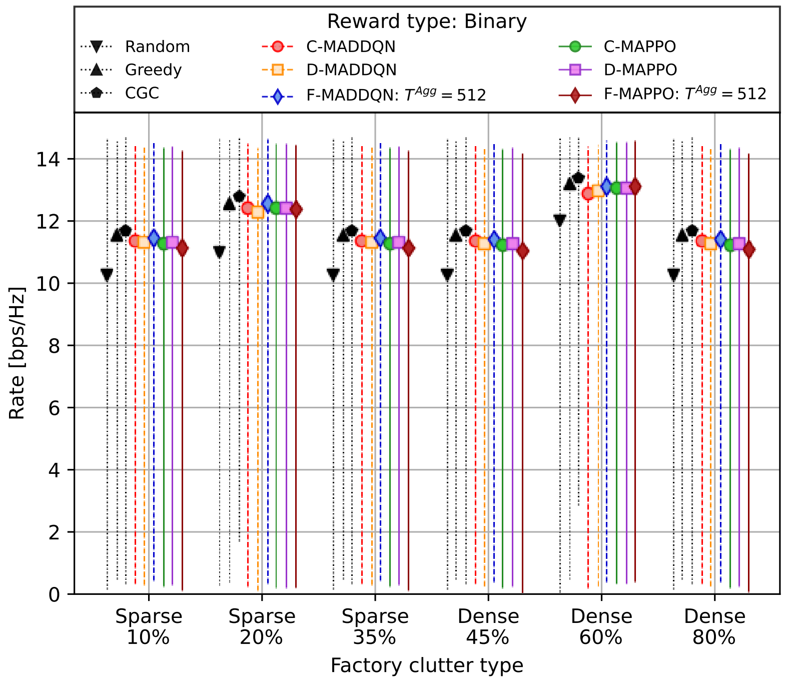

Figure 6.12: Sensitivity analysis of agents trained with the data rate reward signal in a dense clutter environment with density $r_{\text{clutter}} = 60\%$, versus various scenarios of clutter.

Chapter 7

Conclusion & Future Work

This thesis has explored the applications of combining MARL and FL for radio resource optimization in densely deployed 6G in-X industrial subnetworks. The focus of this research was on addressing concerns related to viable interference management and data privacy. The subnetworks were deployed within autonomous robots performing uncoordinated logistical tasks in a real production factory setting. The channel model utilized in this study was inspired by 3GPP standards and included different types of clutter, such as sparse and dense. Centralized and distributed training frameworks were evaluated in this context, revealing challenges including compromised data security in the centralized approach and potential convergence problems in the distributed approach. To overcome these challenges, a client-to-server HFRL framework was proposed, leveraging implicit data sharing through model weights on aggregation intervals.

Simulation results demonstrated that the proposed HFRL framework improves convergence and achieves performance that surpasses the centralized and distributed frameworks. Importantly, this improvement is achieved without compromising data security and without significantly increasing computational complexity for the agents. Longer aggregation intervals discounts the problem, enabling agents to fully capture both local and global features, leading to enhanced performance while utilizing fewer resources. Additionally, by carefully designing the reward signal to capture more abstract features, the update frequency can be reduced. Moreover, robustness against drastic changes in the environment, such as variations in the number of subnetworks and clutter types, allows to maintain a marginal advantage over prior frameworks.

In conclusion, by addressing issues of data security and convergence, the proposed client-to-server HFRL framework demonstrates potential for improving the performance and efficiency of radio resource management for densely deployed 6G in-X subnetworks in an industrial setting.

7.1 Future Work

Simulation of the federated framework was based on two assumptions, i.e., perfect time synchronized aggregation updates and flawless processing and communication. Scenarios with asynchronous aggregation updates, and robustness against errors in the system is proposed for future work. The method used to average weights proved to be effective for aggregation, however the same approach would require modification for asynchronous intervals. Benefits of other aggregation methods may be explored, i.g., weighted average [39] and soft average [40], or algorithms allowing only to submit important parts of the local weights. This would include investigations of the frequency of aggregation.

The distributed peer-to-peer model may improve learning for non-stationary environments, by letting two neighboring agents exchange model weights directly. That is, two adjacent in-robot subnetworks sharing experience about the local environment. This research should include an agent selection method for the federated model and mechanisms for triggering exchange.

FRL shows potential for privacy protection, however vulnerability concerns associated with attacks has not been addressed. An inside attacker may tamper with agent rewards, causing errors in the federated model. Outside attackers, i.e., intruders and eavesdroppers, may manipulate data or infer sensitive information from differences in gradient updates. Therefore, investigation of methods to protect against attacks are needed.

Bibliography

[1] Dinh C. Nguyen et al. "6G Internet of Things: A Comprehensive Survey". eng. In: *IEEE internet of things journal* 9.1 (2022), pp. 359–383. ISSN: 2327-4662.

[2] Harsh Tataria et al. "6G Wireless Systems: Vision, Requirements, Challenges, Insights, and Opportunities". eng. In: *Proceedings of the IEEE* 109.7 (2021), pp. 1166–1199. ISSN: 0018-9219.

[3] Ramoni Adeogun et al. "Towards 6G in-X subnetworks with sub-millisecond communication cycles and extreme reliability". eng. In: (2020). ISSN: 21693536.

[4] Gilberto Berardinelli, Preben Mogensen, and Ramoni O. Adeogun. "6G in-X subnetworks for life-critical communication". In: *2020 2nd 6G Wireless Summit (6G SUMMIT)*. IEEE. 2020.

[5] Gilberto Berardinelli et al. "Extreme Communication in 6G: Vision and Challenges for 'in-X' Subnetworks". eng. In: *IEEE open journal of the Communications Society* 2 (2021), pp. 2516–2535. ISSN: 2644-125X.

[6] Gilberto Berardinelli et al. "Beyond 5G Wireless IRT for Industry 4.0: Design Principles and Spectrum Aspects". eng. In: *2018 IEEE Globecom Workshops (GC Wkshps)*. IEEE, 2018, pp. 1–6. ISBN: 9781538649206.

[7] Daniel Ohizimede Abode, Ramoni Ojekunle Adeogun, and Gilberto Berardinelli. "Power Control for 6G Industrial Wireless Subnetworks: A Graph Neural Network Approach". eng. In: (2023). ISSN: 15253511.

[8] Gilberto Berardinelli and Ramoni Ojekunle Adeogun. "Hybrid radio resource management for 6G subnetwork crowds". English. In: *I E E E Communications Magazine* (2023). ISSN: 0163-6804.

[9] Ramoni Adeogun and Gilberto Berardinelli. "Multi-Agent Dynamic Resource Allocation in 6G in-X Subnetworks with Limited Sensing Information". eng. In: (2022). ISSN: 14248220.

[10] Jiaju Qi et al. "Federated Reinforcement Learning: Techniques, Applications, and Open Challenges". eng. In: (2021).

[11] Fatima Hussain et al. "Machine Learning for Resource Management in Cellular and IoT Networks: Potentials, Current Solutions, and Open Challenges". eng. In: (2019).

[12] M. Hoyhtya, Tao Chen, and A. Mammela. "Interference management in frequency, time, and space domains for cognitive radios". eng. In: *2009 Wireless Telecommunications Symposium*. IEEE, 2009, pp. 1–7. ISBN: 1424425883.

[13] Junyu Liu et al. "Interference Management in Ultra-Dense Networks: Challenges and Approaches". eng. In: *IEEE network* 31.6 (2017), pp. 70–77. ISSN: 0890-8044.

[14] Andreas F Molisch. "Spread Spectrum Systems". eng. In: *Wireless Communications*. 2nd ed. Wiley - IEEE. John Wiley & Sons, 2011. ISBN: 9780470741870.

[15] S. Haykin. "Cognitive radio: brain-empowered wireless communications". eng. In: *IEEE journal on selected areas in communications* 23.2 (2005), pp. 201–220. ISSN: 0733-8716.

[16] Jia Xiao et al. "Joint Interference Management in Ultra-Dense Small-Cell Networks: A Multi-Domain Coordination Perspective". eng. In: *IEEE transactions on communications* 66.11 (2018), pp. 5470–5481. ISSN: 0090-6778.

[17] Sheng Zhou. "An overview on intercell interference management in mobile cellular networks: From 2G to 5G". In: *IEEE Communications Surveys & Tutorials* 16.4 (2014), pp. 1803–1824.

[18] Ramoni Adeogun et al. "Distributed Dynamic Channel Allocation in 6G in-X Subnetworks for Industrial Automation". In: *2020 IEEE Globecom Workshops (GC Wkshps*. 2020, pp. 1–6. DOI: `10.1109/GCWkshps50303.2020.9367532`.

[19] Ramoni Ojekunle Adeogun, Gilberto Berardinelli, and Preben E. Mogensen. "Enhanced interference management for 6G in-X subnetworks". eng. In: (2022). ISSN: 21693536.

[20] Gilberto Berardinelli and Ramoni Adeogun. "Spectrum assignment for industrial radio cells based on selective subgraph constructions". In: *2021 IEEE 94th Vehicular Technology Conference (VTC2021-Fall)*. 2021, pp. 01–05. DOI: `10.1109/VTC2021-Fall52928.2021.9625183`.

[21] Ramoni Ojekunle Adeogun, Gilberto Berardinelli, and Preben E. Mogensen. *Learning to Dynamically Allocate Radio Resources in Mobile 6G in-X Subnetworks*. eng. 2021.

[22] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. Second edition. The MIT Press, 2018.

[23] Yaodong Yang and Jun Wang. "An Overview of Multi-Agent Reinforcement Learning from Game Theoretical Perspective". eng. In: (2020).

[24] Xiao Du et al. "Multi-agent Reinforcement Learning for Dynamic Resource Management in 6G in-X Subnetworks". eng. In: *IEEE transactions on wireless communications* 22.3 (2023), pp. 1–1. ISSN: 1536-1276.

[25] Ramoni Ojekunle Adeogun and Gilberto Berardinelli. *Distributed Channel Allocation for Mobile 6G Subnetworks via Multi-Agent Deep Q-Learning*. eng. 2023.

[26] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. First edition. Cambridge CB3 0FB, U.K.: Springer, 2006. ISBN: 978-0387-31073-2.

[27] John Schulman et al. *Proximal Policy Optimization Algorithms*. 2017. arXiv: `1707.06347 [cs.LG]`.

[28] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. `http://www.deeplearningbook.org`. MIT Press, 2016.

[29] Hado Van Hasselt, Arthur Guez, and David Silver. “Deep Reinforcement Learning with Double Q-Learning”. eng. In: *Proceedings of the ... AAAI Conference on Artificial Intelligence*. Vol. 30. 1. 2016.

[30] Ramoni Ojekunle Adeogun, Gilberto Berardinelli, and Preben E. Mogensen. *Learning to Dynamically Allocate Radio Resources in Mobile 6G in-X Subnetworks*. eng. 2021.

[31] *5G-ACIA - 5G Alliance for Connected Industries and Automation — 5g-acia.org*. `http://www.5g-acia.org/`. [Accessed 16-May-2023].

[32] 3GPP. *Study on channel model for frequencies from 0.5 to 100 GHz*. Technical Report (TR) 38.901. Version 16.1.0 Release 16. 3rd Generation Partnership Project (3GPP), Jan. 2020.

[33] Shani Lu, John May, and Russell J. Haines. “Effects of correlated shadowing modeling on performance evaluation of wireless sensor networks”. eng. In: (2015). ISSN: 15731659.

[34] W. C. Jakes and D. C. Cox. *Microwave Mobile Communications*. Hoboken, NJ, USA: Wiley, 1994.

[35] Binbin Lu et al. “Deep Reinforcement Learning-Based Power Allocation for Ultra Reliable Low Latency Communications in Vehicular Networks”. In: *2021 IEEE/CIC International Conference on Communications in China (ICCC)*. 2021, pp. 1149–1154. DOI: `10.1109/ICCC52777.2021.9580341`.

[36] K.T. Phan et al. “Power Allocation in Wireless Relay Networks: A Geometric Programming-Based Approach”. eng. In: *IEEE GLOBECOM 2008 - 2008 IEEE Global Telecommunications Conference*. IEEE, 2008, pp. 1–5. ISBN: 9781424423248.

[37] Chao Yu et al. “The Surprising Effectiveness of PPO in Cooperative, Multi-Agent Games”. eng. In: (2021).

[38] Muning Wen et al. “Multi-Agent Reinforcement Learning is a Sequence Modeling Problem”. eng. In: (2022).

[39] Liam Collins et al. *FedAvg with Fine Tuning: Local Updates Lead to Representation Learning*. 2022. arXiv: `2205.13692 [cs.LG]`.

[40] Adnan Ben Mansour et al. *Federated Learning Aggregation: New Robust Algorithms with Guarantees*. 2022. arXiv: `2205.10864 [stat.ML]`.

Appendix A

Additional Simulation Results

This appendix presents additional simulation results for the problem addresses throughout the master thesis. This includes insight to deciding on a maximum action transition delay, collecting statistics for normalization through an interference analysis, and an example for the joint channel and power allocation problem.

A.1 Maximum Action Transition Delay

In the beginning of every episode, each subnetwork is granted a random action transition delay based on a maximum tolerable delay τ_{max}. It is remarked that in the case of $\tau_{\text{max}} = 1$, all subnetworks update actions simultaneously, resulting in poor performance. Insight to the effect of using the maximum delay is illustrated for a distributed greedy selection algorithm in figure A.1. Parameters used are summarized in table 6.1. It is observed that the maximum delay indeed improve performance significantly.

It is remarked that for greater τ_{max} values, no significant improvement is observed. It is expected to beneficial to set $\tau_{\text{max}} = 10$, allowing agents to interact with the environment as frequent as possible.

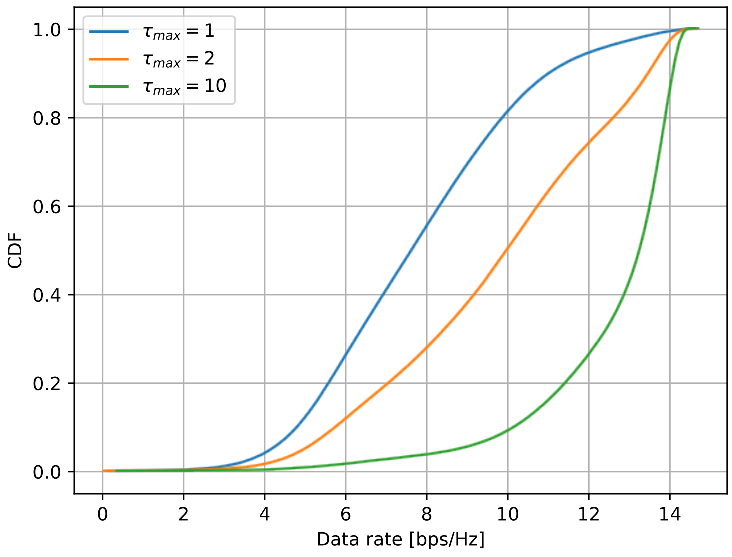

Figure A.1: Illustration of introducing an action transition delay from τ_{max} to subnetworks, based on the distributed greedy selection method.

A.2 Interference Statistics Analyzes

The numerical range of the observation space may be controlled for more stable learning. For range control, min-max normalization is used to scale the aggregated interference power I, to the scale $[0, 1]$ with

$$I_{\text{norm}} = \frac{I - I_{\text{min}}}{I_{\text{max}} - I_{\text{min}}} \tag{A.1}$$

where I_{max} and I_{min}, respectively are the statistical maximum and minimum observable interference powers. To obtain the parameters for min-max normalization and more insight of the environment behavior, an analysis of the possible observable interference powers is performed. Results for a sparse clutter factory scenario with clutter density $r_{\text{clutter}} = 20\%$ are illustrated in figure A.3. Parameters used are summarized in table 6.1.

A similar analysis is performed for a dense clutter factory scenario with clutter density $r_{\text{clutter}} = 60\%$. Results are illustrated in figure A.3

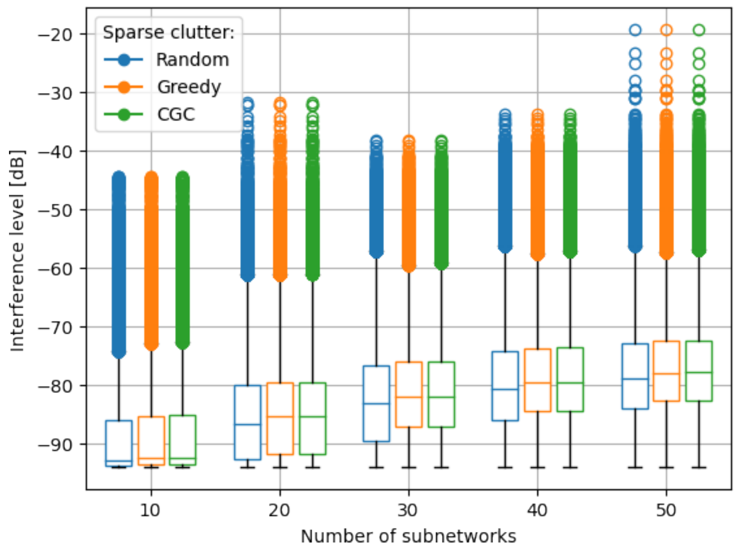

Figure A.2: Simulation results of observed aggregate interference powers in a sparse clutter scenario.

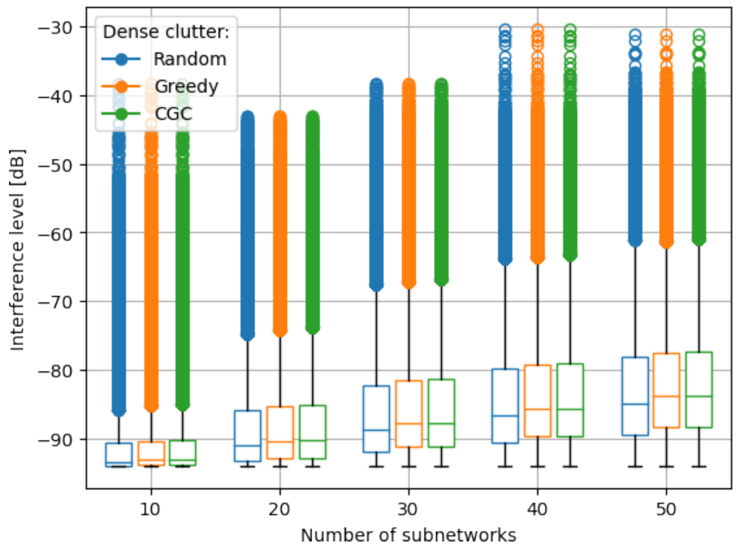

Figure A.3: Simulation results of observed aggregate interference powers in a dense clutter scenario.

The results are use to perform min-max normalization for the implementation. Statistics respectively from sparse and dense clutter are based all the represented results, and are not divided into number of subnetworks.

A.3 Joint Channel and Power Allocation

The joint channel and power allocation problem was presented in section 4.3.1, however this problem was not evaluated further. Here, an example of the joint allocation problem is presented. The power levels are represented with (5.3), using $U = 4$ power levels from $P_{\text{min}} = -10$ dBm to $P_{\text{min}} = 0$ dBm. Hence the joint action space, represented by (5.4), has a dimension of $K \cdot U = 16$ actions. The observation space remains unchanged, any other parameters used are presented in tables 6.1 and 6.2. An example training scenario with C-MAPPO based on the data rate reward signal is presented on figure A.4. The figure illustrates that the average reward are increased relative to the scenarios presented in figures 6.1 and 6.3, showing that a power control can improve radio resource optimization.

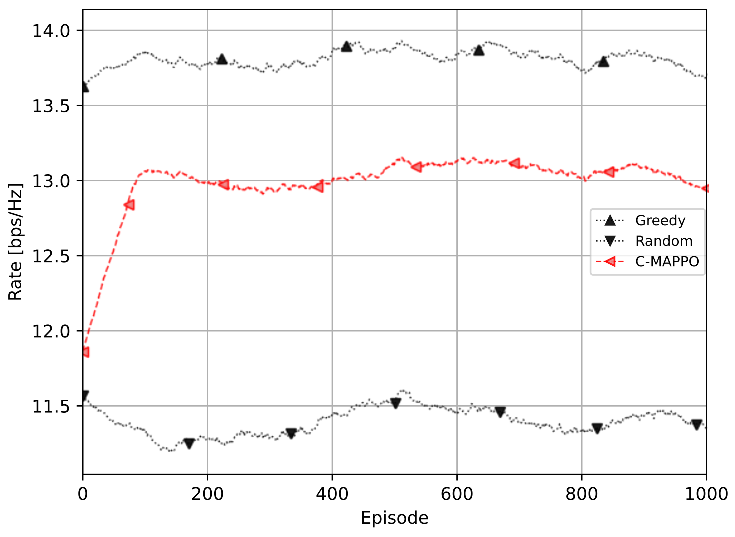

Figure A.4: Example of training joint channel and power allocation with C-MAPPO.