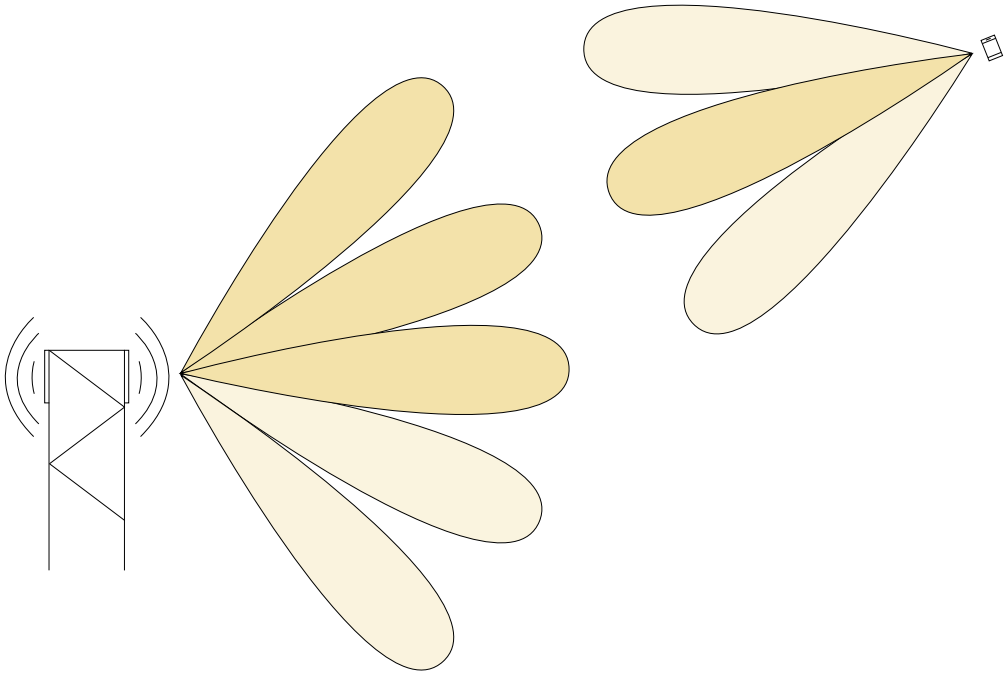
CHALMERS
UNIVERSITY OF TECHNOLOGY



Comparison of Machine Learning Techniques for Beam Management in 5G New Radio (NR)

Master's thesis in Master Programme of Data Science and AI

AXEL LUNDBERG
SIMON SVENSSON

DEPARTMENT OF ELECTRICAL ENGINEERING
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2023
www.chalmers.se

Master's thesis 2023

Comparison of Machine Learning Techniques for Beam Management in 5G NR

AXEL LUNDBERG
SIMON SVENSSON

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Electrical Engineering
Division of Communications, Antennas, and Optical Networks
Communication Systems Group
Chalmers University of Technology
Gothenburg, Sweden 2023

Comparison of Machine Learning Techniques for Beam Management in 5G NR
AXEL LUNDBERG
SIMON SVENSSON

Supervisor:	Bengt Hallinger	Tietoevry
	Azadeh Tabeshnezhad	Department of Electrical Engineering, Chalmers
Examiner:	Giuseppe Durisi	Department of Electrical Engineering, Chalmers

Master's Thesis 2023
Department of Electrical Engineering
Division of Communications, Antennas, and Optical Networks
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Base station and user equipment searching for the optimal beam pair.

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2023

Comparison of Machine Learning Techniques for Beam Management in 5G NR
AXEL LUNDBERG
SIMON SVENSSON
Department of Electrical Engineering
Chalmers University of Technology

Abstract

Aligning beams in the initial access of beam management is a challenging and time-consuming process. Especially, when the number of antenna elements grow large to compensate for high path loss of millimeter waves. Machine learning methods have successfully been applied to the problem of beam selection and perform much better than traditional methods like exhaustive search. In this thesis, some different machine learning approaches are investigated: decision tree, random forest, Adaptive Boosting (AdaBoost), Support Vector Machine (SVM), Multi-level Perceptron (MLP), Q-learning, Deep Q-Network (DQN) and Double Deep Q-Network (DDQN). Each model is adapted to specific scenarios with different preprocessing steps. A total of three scenarios are explored which have been defined by the 3rd Generation Partnership Project (3GPP): Urban Micro (UMi), Urban Macro (UMa) and Rural Macro (RMa). The UMi and UMa scenarios are both implemented with an explicit city layout containing static receivers. The RMa scenario is uniformly distributed and divided into two datasets: one for static receivers and one for dynamic receivers along tracks. Each scenario has been generated by the stochastic channel model called QuaDRiGa. The aim of the thesis is to provide a fair comparison of machine learning models by testing them on data from one simulator. Results show that random forest and AdaBoost perform best overall on all datasets with up to 90% accuracy when predicting the optimal beam pair, which suggests that the search space can be significantly reduced.

Keywords: 5G NR, Machine learning, Supervised learning, Reinforcement learning, Beam management, QuaDRiGa simulation, Beam alignment.

Acknowledgements

We would like to express our gratitude to Tietoevry for letting us write our thesis there and to our supervisor at Tietoevry, Bengt Hallinger, who has provided us with his expertise in telecommunication and helped us look at problems from a new perspective at times where we progressed slowly. We want to thank our supervisor at Chalmers, Azadeh Tabeshnezhad, who has guided us with administrative tasks, questions about how the report should be structured and proofreading our drafts during the semester. Also, we would like to thank Giuseppe Durisi for taking on the role of examiner for our project. Without him the project would not have happened.

Axel Lundberg, Simon Svensson, Gothenburg, June 2023

Contents

Glossary

QuaDRiGa	QuaDRiGa, short for QUAsi Deterministic RadIo channel GenerAtor, is a statistical ray-tracing model, that unlike a classical ray-tracing model, uses a stochastic geometric representation
RMa	Rural Macro. A scenario defined by 3GPP that focuses on larger and continuous wide area coverage in a rural environment with high speed vehicles
UMa	Urban Macro. A scenario defined by 3GPP that focuses on open areas in a street setting where the base stations are placed above rooftop levels of surrounding buildings
UMi	Urban Micro. A scenario defined by 3GPP that focuses on open areas in a street setting where the base stations are placed below rooftop levels of surrounding buildings

Acronyms

3GPP	3rd Generation Partnership Project.
4G	4th Generation.
5G	5th Generation.
AdaBoost	Adaptive Boosting.
AI	Artificial Intelligence.
AoA	Angle of Arrival.
AoD	Angle of Departure.
AP	Access Provider.
BS	Base Station.
CSI-RS	Channel State Information Reference Signal.
DDQN	Double Deep Q-Network.
DL	Deep Learning.
DMRS	Demodulation Reference Signal.
DNN	Deep Neural Network.
DOA	Direction of Arrival.
DQN	Deep Q-Network.
DRL	Deep Reinforcement Learning.
FDD	Frequency Division Duplex.
GPS	Global Positioning System.
IA	Initial Access.
IoT	Internet of Things.
ISD	Inter-Site Distance.
ISI	Inter-symbol interference.
LoS	Line of Sight.
LTE	Long Term Evolution.
MDP	Markov Decision Process.
MIMO	Multiple-input-multiple-output.

ML	Machine Learning.
MLP	Multi-level Perceptron.
mmWave	Millimeter wave.
NLoS	Non-line of Sight.
NR	New Radio.
O2I	Outdoor to Indoor.
OFDM	Orthogonal Frequency Division Multiplexing.
PBCH	Physical Broadcast Channel.
PDSCH	Physical Downlink Shared Channel.
PSS	Primary Synchronization Signal.
PTRS	Phase Tracking Reference Signal.
RL	Reinforcement Learning.
RMa	Rural Macro.
RT	Ray Tracing.
SINR	Signal-to-interference-plus-noise ratio.
SL	Supervised Learning.
SSB	Synchronization Signal Block.
SSS	Secondary Synchronization Signal.
SVM	Support Vector Machine.
TDD	Time Division Duplex.
UE	User Equipment.
UMa	Urban Macro.
UMi	Urban Micro.
UN	United Nations.
URA	Uniform Rectangular Array.

Nomenclature

Below is the nomenclature of symbols for communication theory, supervised learning and reinforcement learning that have been used throughout this thesis. If the same symbol is defined in multiple places, the meaning is explicitly stated in the context where the symbol occurs.

Communication Theory

Δ_f	Subcarrier spacing
λ	Wavelength of radio wave
μ	Numerology
ϕ	Elevation angle
θ	Azimuth angle
N_ϕ	Number of directions to cover in elevation
N_θ	Number of directions to cover in azimuth

Supervised Learning

$\mathcal{X}$	Input space
$\mathcal{Y}$	Output space
θ	Parameters of the network

Reinforcement Learning

α	Learning rate
ϵ	Probability of exploration over exploitation
ϵ_{decay}	Rate of exploration decay
ϵ_{min}	Minimal exploration probability allowed
γ	Discount factor
$\mathbb{E}_\pi$	Expectation given policy π
$\mathcal{A}$	Action space
$\mathcal{R}$	Reward space
$\mathcal{S}$	State space
π	Policy
π_*	Optimal policy
τ	Target network update factor
θ_t^-	Target network parameters
G_t	Return at time step t
$n_{\text{iterations}}$	Number of training iterations
$q_*(s, a)$	Action-value function when following optimal policy

$q_\pi(s, a)$	Action-value function when following policy π
T	Time steps
$v_*(s)$	State-value function when following the optimal policy
$v_\pi(s)$	State-value function when following policy π

Others

$\mathcal{U}[a, b]$	Uniform random variable in the interval $[a, b]$
k	Number of selected beams

List of Figures

List of Tables

1

Introduction

5th Generation (5G) New Radio (NR) is the next generation of telecommunication networking following the 4th Generation (4G). With the new telecommunication technology comes opportunities but also challenges. 5G NR is the fundamental structure on which high traffic and low latency wireless communication can be built upon in the near future. As society is trending toward higher technological integration in all domains, many of the advancements are easiest to handle wirelessly, and are therefore not enabled due to the limitations of current networks. Some use cases involve the Internet of Things (IoT), sensors, industry equipment, cars and high quality video streaming [2].

Another subject that is in the center of current technological progress is Artificial Intelligence (AI). One important field in AI that has had much expansion in recent years is Machine Learning (ML). Machine learning is a field for methods that leverage data to improve the performance of a system. Machine learning methods have proven to be surprisingly good on tasks that appear to require human intelligence, e.g., the game of Go [3]. The benefits of machine calculations together with human intuition and decision making can present solutions to many difficult problems.

AI has been under development for at least 50 years, but it has always been limited by the hardware capacity. Moore's Law [4] predicts that the number of transistors doubles about every two years, and in recent years, hardware has gotten to the point where end consumers are able to train complicated machine learning models on their devices [5]. The rise of massive datasets, often called big data [6], through various collection methods has further enabled machine learning model advancements [7]. The emergence of cloud computing [8] has accelerated machine learning training efforts as well.

1.1 Background

The specification of 5G NR by 3GPP has been designed to meet a number of key requirements: ultra-reliable low-latency communications, enhanced broadband for mobile users by exploitation of much higher frequencies, ultra-lean design to enhance the energy performance and reduce the interference, forward compatibility and a beam-centric antenna design [1, 2]. To support the ever-increasing demand for bandwidth due to the increasing number of devices, 5G NR aims to take advantage

of the currently underutilized Millimeter wave (mmWave) frequencies. However, operating at higher frequencies increases the channel impairments such as path loss because of the physical properties of mmWave frequencies. By directing signals using a technique called beamforming, a procedure of beam management, and placing the transmitter antenna at the Base Station (BS) more closely to the end-user devices, User Equipment (UE), channel impairments can be minimized [9].

A few prominent areas of recent ML breakthroughs are Deep Learning (DL) and Deep Reinforcement Learning (DRL) [10]. One popular type of DL are Deep Neural Networks (DNNs), which are structures of artificial neurons connected in multiple layers. They are trained using Supervised Learning (SL), where a loss function is used to measure how incorrect the model's prediction is on each sample. Other examples of SL models are decision tree, random forest, SVM, AdaBoost and MLP. In DRL a concept of agents and environments is employed, in which agents interact with the environment and receive feedback in form of rewards and punishments. The agent maximizes the cumulative reward over time to achieve the best performance. Examples of Reinforcement Learning (RL) methods are Q-learning, DQN and DDQN.

There are a few different simulators for channel propagation of radio waves. Although Ray Tracing (RT) models can provide very accurate results, its use is constrained to specific propagation environments and the computational cost increases exponentially with the number of reflections and diffractions allowed [11]. However, [12] states that ray tracing models are motivated by the difficulty of dealing with long-term spatial consistency as well as the problem of correlating small-scale parameters.

Moreover, [11, 12] argue that stochastic channel models, an alternative to RT models, have problems simulating massive Multiple-input-multiple-output (MIMO) systems, by overestimating the performance in dense Non-line of Sight (NLoS) scenarios while underestimating the performance in Line of Sight (LoS) scenarios. State-of-the-art stochastic models for 5G that try to improve spatial consistency include QuaDRiGa [13] and NYUSIM [14]. Spatial consistency refers to the principle that nearby UE should have similar channel properties. Since QuaDRiGa is only a channel model it does not by its own support procedures in the 5G protocol, like reference signals.

QuaDRiGa is a three dimensional geometry-based stochastic channel model, which can be understood as a *statistical ray-tracing channel model* that uses approximations for the geometric representation and distributes scatterers randomly with a concept of batching scattering into clusters. A scatterer is an object over which the electromagnetic wave bounce off in different ways. QuaDRiGa determines the channel parameters stochastically based on statistical distributions extracted from channel measurements. The parameter distributions are defined for, e.g., delay spread, delay values, angle spread, shadow fading, and cross-polarization ratio [13].

QuaDRiGa comes with predefined scenarios, that samples from different parameter distributions, e.g., realistic parameters for city environments. QuaDRiGa is very flexible in the way different antennas are supported. Beyond a set of default antennas

like omni-directional antennas and Uniform Rectangular Arrays (URAs), custom antennas can be built by placing and rotating existing antennas in different ways. When simulating, QuaDRiGa calculates the delay, power, Angle of Arrival (AoA) and Angle of Departure (AoD) of multiple rays that take different paths at certain time steps. The rays are propagated through clusters of scatterers before arriving at the receiver where some aggregation is required to calculate the received power [13].

1.2 Problem Description

To fully take advantage of the benefits that 5G NR offers concerning higher capacity and lower latency, the scalability issue regarding the Initial Access (IA) when bigger antenna arrays are introduced, must be mitigated. Directing the transmit power to a UE through beamforming is a rather big task. It requires special antenna configurations and protocols to manage the initial contact with the UE as well as keeping the connection alive.

In previous cellular systems such as 4G, the IA has been performed using omni-directional antennas. Only when the connection has been established, beamforming has been performed to utilize the beam gain. However, 5G needs to leverage beamforming in the IA to overcome the high path loss that comes with mmWaves. By using beamforming at both the BS and the UE, beam pairs are created for the established links. As the number of beams increases, with bigger antennas and larger potential directional gains, the search space for the optimal beam pair between the BS and the UE becomes bigger.

It is not feasible to achieve the performance requirements by exhaustively searching over all beam pairs to find the optimal beam pair as proposed in the initial release of 5G [1]. Using a hierarchical search as proposed in [15], in which exhaustive search is applied to narrower and narrower beams, reduces latency compared to exhaustive search but lack coverage to cell-edge users. Using context aware heuristics, as in [16], by steering directly towards the closest UE are prone to blockage, NLoS scenarios, between the BS and the UE. In these cases the optimal direction is not necessarily straight. By applying ML techniques to the problem of beam management, the search space can hopefully be substantially reduced to meet the performance requirements for the 5G technology.

During recent years, research into solving the beam management problem has been carried out in different places. Since 5G is a new technology, there is not much publicly available data to test ML solutions on. Instead, each research group uses separate data from different simulators and unique measurements. When research groups publish new ML methods, it is therefore difficult to make fair comparisons between them. Hence, implementing several ML solutions in one state-of-the-art publicly available simulator is helpful.

1.3 Purpose

The purpose of this thesis is to compare ML solutions for the beam management problem in 5G using the QuaDRiGa simulator. There has been a lot of activity in this area recently, and the direction of beam management has steered from traditional methods, like exhaustive and hierarchical searches towards utilizing some kind of machine learning algorithm. This thesis provides a comparison between the following models: decision tree, random forest, AdaBoost, SVM, MLP, Q-learning, DQN and DDQN. The objective is to determine which model performs the best and under what circumstances.

1.4 Delimitation

The scope of the thesis has to be restrained to meet expected deadlines. Therefore, many interesting questions and perspectives are not covered in this thesis. These are better suited for future work.

Handovers are essential when UE are moving between cell sites. However, handovers are not part of the simulation even though a UE might move outside the cell site and hence be a candidate for a handover procedure. Rather, only one BS is considered to keep track of all UE. The UE are able to move, but only along a fixed track for a finite set of time steps in the RMa scenario. It would be interesting to explore more complex paths, e.g., UE moving vertically inside buildings for the UMi or UMa scenario and behavior by integrating, e.g., traffic simulators.

Furthermore, the beam sweep is restricted to the azimuthal plane to simplify the simulation results and reduce the expected time for each generation. Moreover, an antenna with a small set of antenna elements will be considered as to reduce the time to generate data samples since three different scenarios need to be supported in different ways. As such, wide beams that become narrower once a connection between a BS and the UE has been established are not considered.

Another concept that is not considered in the simulation is traffic congestion. There is no consideration of the possible effects of a very high density of UEs in a small area, for example during an event, and how that affects the optimal beam pair in terms of load balancing. No packets are simulated and no reference signals. Generally, no higher layer features of 5G NR are considered.

2

Communication Theory

The following section presents some of the underlying communication theory for 5G NR.

2.1 OFDM

Separating uplink and downlink communication between a UE and a BS is necessary to avoid overlap. 5G NR achieve this by two different transmission modes, Frequency Division Duplex (FDD) and Time Division Duplex (TDD). When the transmission mode is set to FDD, the time dimension is constant and the UE as well as BS distinguish the communication by using different frequencies. Contrary, TDD keep the frequency dimension constant, and separate transmissions in time instead. Although, both of these schemes are used in 5G, they operate on different frequency ranges based on advantages and disadvantages with each approach.

A communication system like 5G transmits data by modulating a carrier signal with a technique called Orthogonal Frequency Division Multiplexing (OFDM). The modulation step encodes a stream of bits onto the carrier signal by a certain scheme. The sequence of bits are represented by a certain state, referred to as *symbol*, depending on the bit pattern and the modulation scheme. Multiple bits can be encoded per symbol. In OFDM, an encoded block of symbols is broken up and each symbol is sent with a different frequency called *subcarrier*.

μ	$\Delta f = 2^\mu \cdot 15$ (kHz)	Useful symbol time (μs)	Cyclic prefix (μs)
0	15	66.7	4.7
1	30	33.3	2.3
2	60	16.7	1.2
3	120	8.33	0.59
4	240	4.17	0.29

Table 2.1: Each numerology is denoted by the greek letter μ and each subcarrier spacing is derived by scaling the numerology by a power of 2. The supported types of subcarrier spacing Δf can be seen in the second column. Data is acquired from [1].

The frequency difference between each subcarrier is denoted as Δf, the subcarrier spacing. The subcarrier spacing in 5G NR may vary, unlike the subcarrier spacing in 4G, according to a specific numerology, see Table 2.1. Consecutive symbols are separated with a gap called *cyclic prefix* to overcome synchronization errors known as Inter-symbol interference (ISI) in multipath environments. The cyclic prefix also depend on the numerology, see Table 2.1 [1, 9].

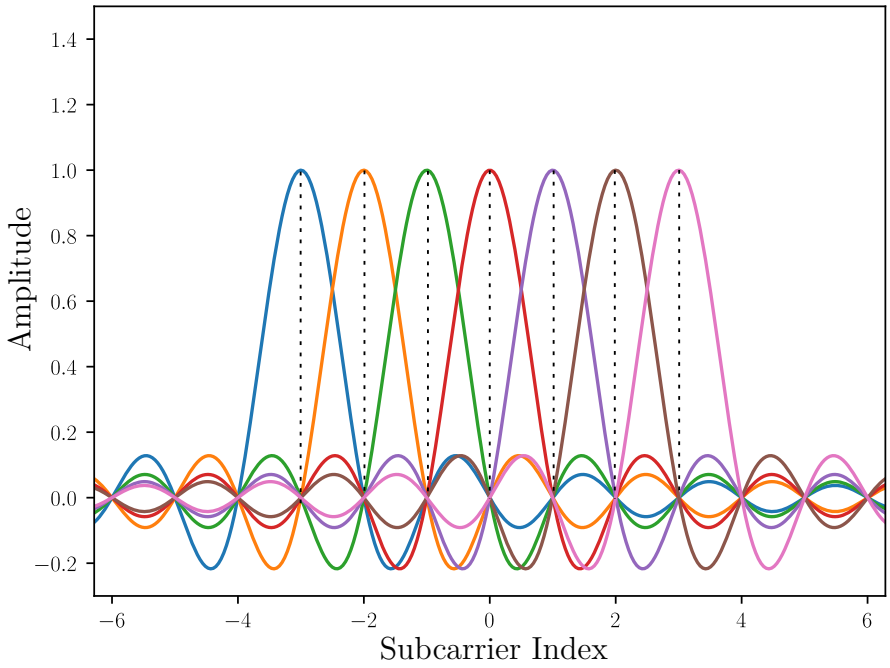


Figure 2.1: Orthogonal subcarriers as visualized by every other subcarrier being zero at a peak.

A frequency range is assigned to a device based on its limitations and needs. Within that frequency range lies several subcarriers. A subcarrier is a small portion of a frequency band around where the carrier lies. Each UE is assigned a few subcarriers depending on its transmission rate needs. The subcarriers are orthogonal to each other which prevents interference among them. A visualization of the orthogonal subcarriers is shown in Figure 2.1. The orthogonality here means that the peak of each signal lies in such a way that the amplitude of all other signals are zero, effectively canceling out the interference [9].

Transmission in 5G NR are organized into frames and subframes. Each frame is 10 ms and consists of 10 subframes. Each subframe is 1 ms and consists of a number of OFDM symbols depending on the numerology. The OFDM symbols in each subframe are divided into slots. There are always 14 OFDM symbols in each slot if normal cyclic prefix is used. If extended cyclic prefix is used instead, there are 12 OFDM symbols in each slot. The number of OFDM symbols for each subframe can

be written as $2^{\mu} \cdot 14$ by using the notation μ for numerology according to Table 2.1. In Figure 2.2, the frame structure for numerologies $\mu \in \{0, 1\}$ are shown. Numerology $\mu = 0$ that corresponds to the frequency 15 kHz has 14 OFDM symbols in total using only one slot for each subframe. Similarly, numerology $\mu = 1$ that corresponds to the frequency 30 kHz has 28 OFDM symbols in each subframe, divided into two slots [17].

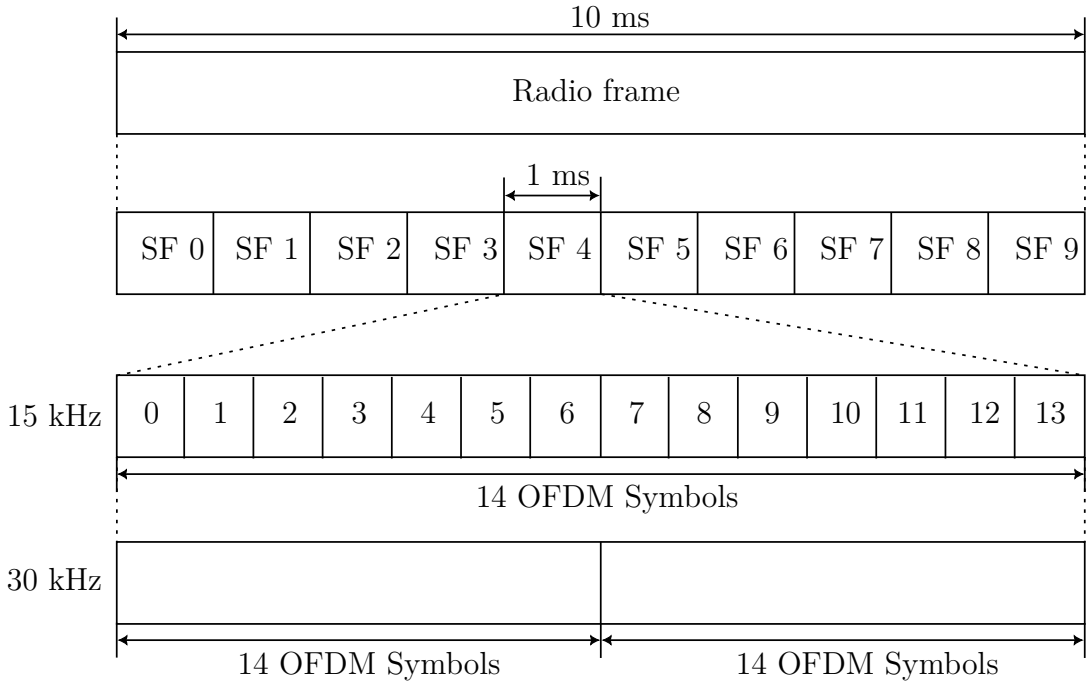


Figure 2.2: NR Frame Structure for the first and second numerology. SF stands for subframe. The figure is inspired by the figure in chapter 5 of [1].

As the NR frame consists of both a frequency and time dimension, it is common to represent symbol boundaries in a time-frequency grid, see Figure 2.3. The smallest physical resource in NR is a *resource element* consisting of only one subcarrier during one OFDM symbol. Moreover, 12 consecutive subcarriers in the frequency domain make up a *resource block*. In Figure 2.3, some resource elements are grayed. These elements represents reference signals, that is, signals that both the BS and UE know beforehand used to derive the actual transmitted signals which may be distorted [1]. There are multiple different types of reference signals in 5G NR.

There are reference signals, like Physical Broadcast Channel (PBCH) Demodulation Reference Signal (DMRS) and Physical Downlink Shared Channel (PDSCH) DMRS, that are used to help the receiver to demodulate the incoming signal. Other reference signals, like the Channel State Information Reference Signal (CSI-RS) and Phase Tracking Reference Signal (PTRS), are used for beam management and compensation of rapid phase fluctuations respectively. The density of a reference signal is not the same across the time domain and the frequency domain or between reference signals because of the different properties of each reference signal [9].

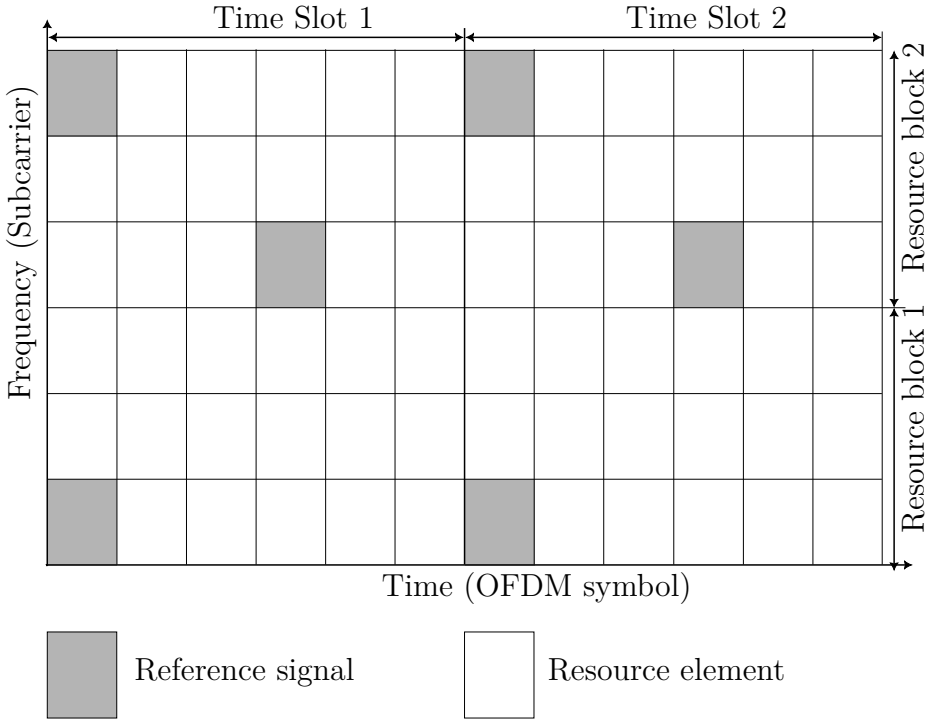
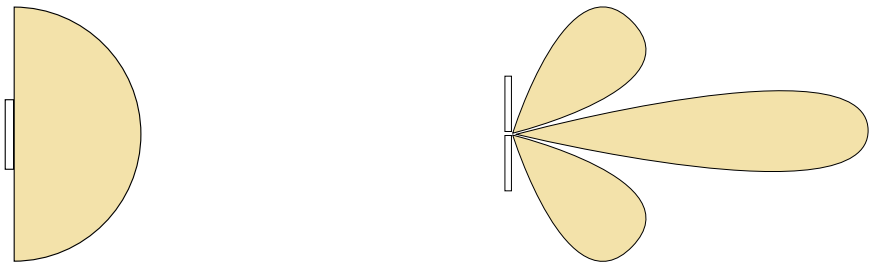


Figure 2.3: Time-frequency grid. Note that the time slots and resource blocks as visualized in the figure may not represent exactly how it works in a real setting, but rather illustrates that reference signals have predefined positions in the time-frequency grid.

2.2 Beamforming

Beamforming is a technique applied at the BS to aim a signal in a particular direction. When transmitting a signal omnidirectionally, the signal is transmitted in all directions from the BS, see Figure 2.4a. Hence, the power of a signal is equally divided in all directions. Although a signal is intended for UE at a specific location, other locations in the same radius will receive the same signal with the same power, wasting large portions of energy to unnecessary directions. Beamforming allows a BS to channel the power in one direction by constructing a certain antenna pattern, thus mitigating the high path loss at high frequencies used by mmWaves.



(a) Beam gain using an omnidirectional antenna.

(b) Beam gain using multiple omnidirectional antenna elements.

Figure 2.4: Two antennas with different properties.

Figure 2.4b shows an anteanna with directed beam gain. Each beam contains a main lobe, nulls and sidelobes. The main lobe exhibits the largest constructive interference. The sidelobes have smaller constructive interference and vary in number depending on the number of antenna elements. The nulls are the places where the interference cancels out the signal. In Figure 2.4b, no phase shift is applied which makes the beam directed perpendicular to the antenna array. UE that are located at the nulls of the radiation pattern will not receive any power, which is favorable if the transmission was not intended for them. To direct a beam in a certain direction, the notion of *beam steering* is used. Steering a beam is then reduced to requiring the following phase difference between the adjacent antenna elements

$$\phi = \frac{2\pi d \sin\theta_0}{\lambda} \tag{2.1}$$

where θ_0 is the direction the beam is steered towards, λ the wavelength of the radio waves and d the distance between the antenna elements [9]. There are three different types of beamforming: analog beamforming, digitial beamforming and hybrid beamforming.

2.2.1 Analog Beamforming

Analog beamformers consist of phase shifters that are integrated with the feed network of the antenna array. The phase shifters apply weights to the carrier wave's phase and amplitude in the different antennas to create constructive and destructive interference. A property of the analog phase shifters is that they must quantize the phase into low resolution values. Hence, analog beamforming can not be used to steer a beam with high precision [18].

2.2.2 Digital Beamforming

Digital beamformers process the signal in the baseband before it is modulated to a high frequency mmWave [18]. This is accomplished by finite impulse response filters that manage the weights of each antenna element adaptively. With digital beamforming it is theoretically possible to create perfect beam patterns. Advantages of digital beamforming include high-resolution Direction of Arrival (DOA) estimation, high spectral efficiency, high system security and enhanced energy efficiency [19].

2.2.3 Hybrid Beamforming

Analog beamforming is easier to implement but is less accurate than digital beamforming. Also, digital beamforming requires many converters between the analog and digital domain, increasing the cost, complexity and power consumption to levels that are too high for mmWave frequencies. To compromise between them, hybrid beamforming combines the two to optimize fidelity versus cost [18]. Technically, it is implemented using both digital baseband weights and analog phase shifters [18].

2.3 Beam Management

The process of *beam management* is to make practical use of the principle of beamforming. In theory, a BS applying the correct phase shifts to its antenna elements would be able to steer the main lobe in any direction to point exactly towards a UE of interest. However, in practice due to complications that arise when adapting a control loop with this behavior, phase differences are often limited to a set of fixed values instead. Hence, the antenna is capable of a set of fixed beams in fixed directions.

The direction of these beams are often chosen in a manner that align the main lobe of a certain beam with the nulls of the other beams. An antenna with m elements, can create m such beams. UEs usually also use beamforming. To determine which beams give the greatest power, or rather the best received Signal-to-interference-plus-noise ratio (SINR), the beams from the BS are paired with the beams from the UE, see Figure 2.5. Steering the beams is then equivalent to finding the best beam pair, a process called *beam selection*. In practise, reference signals are used to identify each beam and report the best incoming beam back to the transmitter [9].

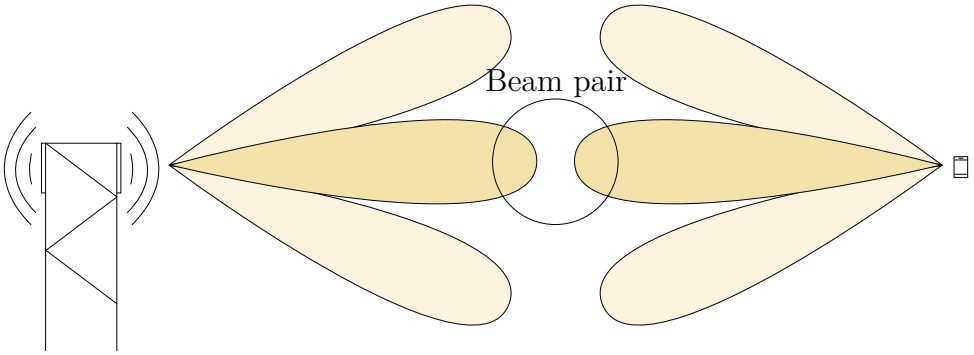


Figure 2.5: The beams from a BS are paired with the beams from a UE to form beam pairs. The illustration highlights a direct link between the BS and the UE suggesting a LoS scenario.

2.3.1 Initial Access

IA is the process of establishing a link, an initial beam pair, between UE and a BS. Both the BS and the UE sweep the area by steering beams in different directions. To establish the connection, two primary synchronization signals are used, the Primary Synchronization Signal (PSS) and the Secondary Synchronization Signal (SSS). These signals are composed into what is called a Synchronization Signal Block (SSB) which the BS transmits in all of its beams during the beam sweep. The UE measures the reception and reports the best SSB back to the BS to establish the initial beam pair [1, 9].

2.3.2 Beam Adjustment

Once a connection has been established by finding the best beam pair, there is a need to reevaluate all beam pairs at regular intervals to check if another beam pair has become better due to UE movement patterns or objects blocking or unblocking the connection [1]. Thus, beam adjustment is still important for stationary UE, as objects may come in the way of the connection. Beam adjustment also include changing the width of the beam to be e.g. more narrow once a connection has been established [1].

2.3.3 Beam Recovery

In some cases the movement pattern or object disrupting the connection may rapidly change without the necessary time for the beam adjustment to adapt resulting in the connection being lost. The NR specification includes procedures in case a beam failure occurs. In general the beam failure recovery process consists of detecting if a beam failure has occurred, identifying a new beam pair by which the connection can continue and reporting the recovery request to the network [1].

3

Machine Learning

There are three main paradigms of machine learning: SL, unsupervised learning and RL. The following section will briefly describe general concepts of SL and RL before going into more detail about different machine learning models within these paradigms.

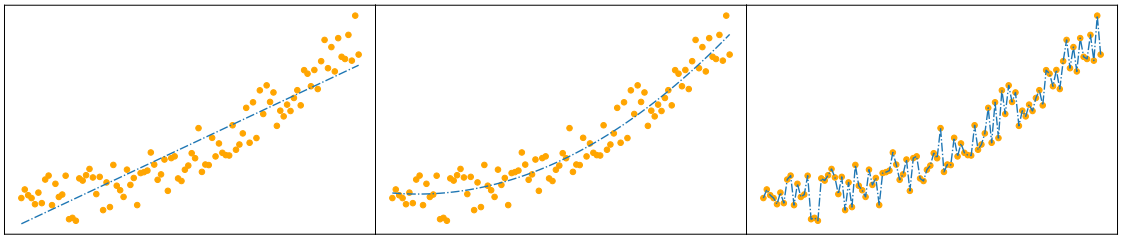
3.1 Supervised Learning

The goal of a SL algorithm is to create a function mapping between a set of input-output pairs, $\mathbf{y} = f(\mathbf{x})$. To create the function mapping, the algorithm is given a set of training samples with n input-output pairs, $(\mathbf{x}_i, \mathbf{y}_i)_{i=1}^{n}$. These samples are passed to the algorithm during the training phase, which alters a set of weights to adjust the model with the intention of converging to the function mapping. During the training phase, the model is given feedback on its prediction by a loss function, which scores how well the model predicted the target output.

To measure the performance of the trained model, it is evaluated on a distinct sample set called the test set. If the model is able to perform well on the test set, the model generalizes well. That is, the model is able to correctly make inductions about new data samples that it has not seen before. The function f does not need to be a strict function of $\mathbf{x}$, but can be stochastic. In this case it is of interest to learn the conditional probability $\Pr(\mathbf{y} \mid \mathbf{x})$. The output space can be either finite set or a scalar value. In the case of a finite output space the learning problem is called a *classification* problem.

To solve a classification problem the learning algorithm has to create a mapping from the input space to the categorical output space. With the assumption that the input is a real valued vector of length n, the algorithm has to learn $f : \mathbb{R}^n \to \{1, 2, \ldots, m\}$ which is a function with n inputs and m classes. There are multiple encodings for each class in the output space. Two common ways of converting categorical values into numerical values are *label encoding* and *one-hot encoding*. Using label encoding will assign a numerical value for each class and the conversion function is simply to map the categorical value to the numerical value that the class belongs to. One-hot encoding follows the same concept, but rather uses a vector of n length. The correct class is represented as a one while all other classes are represented as zeros.

An important concept in SL is called overfitting. Figure 3.1 shows three different polynomials that try to describe the general trend of the scattered data points. The first polynomial of degree 1, see Figure 3.1a, does a poor job of describing the points since many of the points lie far away from the curve, resulting in underfitting the dataset. Contrary, the third polynomial, see Figure 3.1c, describes the set of points perfectly as every point lies directly on the curve. However, this results in overfitting the dataset as new data points are unlikely to lie directly on the curve. The desired fit is shown in Figure 3.1b, where the model has learned the general shape of the data in order to make good inferences about unseen samples.



(a) (b) (c)

Figure 3.1: (a) shows an estimator that is underfitting. (b) shows an estimator that generalizes well. (c) shows an estimator that is overfitting.

The problem with overfitting is that the model will generalize poorly. When presented with new data points, it will make predictions that are biased towards the kind of data it trained on. Some bias towards its training data is necessary, otherwise it will have no knowledge to base its predictions on, in which case it is underfitted. To mitigate overfitting, a popular technique is to use regularization. Regularization consists of different approaches of simplifying the pattern a model learns. For example, explicit terms in form of penalties or constraints may be added to the problem to change the characteristic of the optimization problem. Different models may also implicitly introduce regularization, e.g., ensemble models like random forest as described further down.

3.1.1 Training and Tuning

When attempting to solve a problem using SL, a first step is to understand the data. After understanding the data, an appropriate model can be chosen. The data should be preprocessed so that bad samples are not negatively impacting the performance of the model. It is also important that the data have the correct format. A dataset is often split up into three different datasets, the training set, the validation set and the test set. The ratio between these sets may differ, but usually follows something similar to 70 %, 15 % and 15 % for each set respectively. During the training phase, the model uses the validation set to measure how well the model generalizes.

The point of tuning is to find the optimal parameters and configurations for the model on the specific dataset. It is necessary to use a separate validation set and test set, as tuning the model on the test set will make it very biased towards the

test set. One common strategy for tuning is grid search. Grid search is a method that searches all desired combinations of parameters specified, and results in the combination which gave the highest score. The search is usually cross validated between multiple possible data splits to reduce random bias.

A common plot used during the tuning phase is the loss for each epoch on the training and validation sets. The plot can indicate whether the model is underfitting or overfitting. Using the plot as guidance, the model can be made larger and more complex until it overfits, then gradually introduce regularization or reducing its complexity to get a good fit. A good fit is found when the validation loss is slightly more than the training loss, so that the model generalizes well, and has trained enough epochs for the loss to stabilize.

3.1.2 Evaluation

Evaluation of the model is performed on the test set which has been held out. By using the model to make predictions based on the test set, the model simulates the performance of the model on unseen data. A score significantly lower than during the tuning phase indicates overfitting, but it should be unlikely if a validation set was correctly used. A score significantly higher is lucky and might be due to using a very small test set, which is unreliable. The score on the test set is the model's actual advertised score. Depending on the domain and problem, scoring can be based on different metrics.

For a classification problem, let $\mathcal{Y} = \{1, \ldots, m\}$ be the output space containing all possible beam pairs. All models in a classification setting in this thesis can be written as the function $f : \mathcal{X} \to \mathbb{R}^m$. That is, a function that maps an input vector to a probability vector $p \in [0, 1]^m$ that gives the probability for each class.

The primary metric used to evaluate the models in this thesis is top-k accuracy, which can be defined as the probability of including the optimal beam pair when searching among the top k beam pairs. Formally, let

$$\beta(p) = \{i \in \mathcal{Y} \mid p_i \geq p_{\bar{k}}\} \tag{3.1}$$

where $\beta(p)$ is a function that maps the probability vector p to a set of candidate beam pairs and $p_{\bar{k}}$ is the k largest element in p. Moreover, $\beta(p)$ has to be applied to the probability vector for every test sample, and the top-k accuracy is achieved by calculating all samples that include the optimal beam pair divided by all beam pairs in the test set. By using the definition of top-k accuracy, we can see that accuracy is equivalent to top-1 accuracy [20].

3.2 Reinforcement Learning

A SL algorithm needs to know the correct answer beforehand in a learning scenario. The algorithm is supervised through the training process. However, information about the best decision in many cases are seldom available. RL shifts focus. In the

absence of a supervisor, an agent teaches itself by utilizing feedback from a defined environment. The core concepts of RL are agents, environments and rewards. The agent is the decision maker and learner in a predefined environment.

In the environment, the agent occupies a certain state. The state contains information that is specific to the environment, e.g., in a grid environment the state may contain the (i, j) indices that represent the row and the column. Decisions made by the agent are called *actions*, and the environment will reward the agent based on different actions in certain states. To learn, the agent will try to maximize the rewards over time with regards to its choice of actions.

The strength of RL is that it may be the only feasible learning model in many complex domains with acceptable performance. In, e.g., chess or Go it is very hard to accurately provide evaluations of a large number of positions. The search space is too large. A supervised model will have a hard time to learn, as the provided examples are few in number. However, a reinforcement algorithm can evaluate positions while exploring the search space when it has won or lost to approximate the winning probability of a certain position [21].

3.2.1 Markov Decision Process

The following section formalizes the agent-environment interaction using something called Markov Decision Process (MDP). An illustration of the agent-environment interaction can be seen in Figure 3.2.

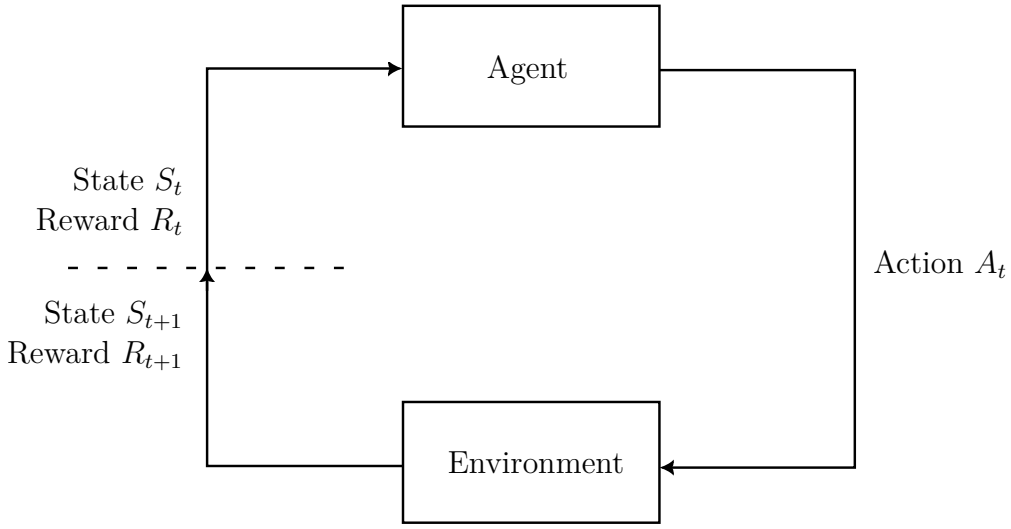


Figure 3.2: Agent-environment interaction.

MDPs are discrete-time stochastic control processes. Actions taken by RL agents not only influence immediate rewards, but will through subsequent states affect future rewards as well. MDPs provide a rigorous mathematical framework for the RL problem, enabling precise theoretical statements. The formalization of MDP involves

trading off immediate and delayed rewards. Key elements of the mathematical structure are introduced, such as returns, policies and value functions.

The environment and its dynamics are formally described by a MDP. In a reinforcement learning algorithm the agent and environment interact continually. The agent selects actions whereby the environment responds with feedback in form of a measurement of how good the selected action was, the reward, and a new state derived from the selected action. As the reward is a measure of the performance in a given environment, the agent seeks to maximize the reward over time by selecting better and better actions. MDPs can be described by the tuple $(\mathcal{S}, \mathcal{A}, \mathcal{R}, p, \gamma)$:

1. $\mathcal{S}$ is the set of all possible states, the state space. An agent will occupy a state $s \in \mathcal{S}$ at a given time.

2. $\mathcal{A}$ is the set of all possible actions, the action space. An agent will select an action $a \in \mathcal{A}$ at a given time. More specifically, it will select an action dependent on the current state $a \in \mathcal{A}(s)$.

3. $\mathcal{R}$ is the set of all possible rewards, the reward space. An agent will gain a certain reward $r \in \mathcal{R}$ based on its selected action.

4. $p(s', r \mid s, a) = \Pr(S_{t+1} = s', R_{t+1} = r \mid S_t = s, A_t = a)$ is the probability of the next state $s' \in \mathcal{S}$ and reward $r \in \mathcal{R}$, given the current state s and action a.

5. γ is the discount factor for future rewards.

The memoryless property of stochastic processes is often referred to as the Markov property. It states that for every stochastic process that fulfills the memoryless property, the conditional probability distribution of future states depend only upon the present state. The implication being that the past does not impact future states [22]. The Markov property is formally defined as:

$$\begin{aligned} &\Pr(S_t = s, R_t = r \mid S_{t-1}, A_{t-1}, R_{t-1}, \ldots, R_1, S_0, A_0) \\ &\qquad = \Pr(S_t = s, R_t = r \mid S_{t-1}, A_{t-1}) \end{aligned} \tag{3.2}$$

Finite Markov Decision Process

A MDP is finite if the state, action and reward space are finite, that is, the tuple $(\mathcal{S}, \mathcal{A}, \mathcal{R})$ have a finite number of elements. If this is the case, the random variables of p, S_{t+1} and R_{t+1}, will have well defined discrete probability distributions which are dependent only on the preceding state $S_t = s$ and the selected action $A_t = a$. At a particular time t, the random variables are represented by $s' \in \mathcal{S}$ and $r \in \mathcal{R}$ which are given by the probability function p. It is the function p that defines the dynamics of the environment and thus the MDP.

Episodic and Continuous Tasks

The RL problem is often divided into identifiable *episodes* when the agent-environment interaction breaks down into subsequences. A subsequence of this sort has one *ter-*

minal state, which is an end state with no further actions available for the agent. All other states are called *nonterminal*, meaning that the agent still has choices left to choose. A RL problem with episodes are therefore called *episodic tasks*. Each episode is independent of each other. Thus, the ending of one episode does not affect the start of the next episode.

Starting states are often drawn from a probability distribution of starting states or some predefined state as in, e.g., chess. However, not all RL problems can be divided into episodes. These problems continue indefinitely with no terminal states. They are because of this called *continuing tasks*. To unite the two ways of differentiating between RL formulations, a concept called *discounting* is useful [22]. This concept is explored further in the section about Returns further down.

Rewards

The reward is a fundamental concept in RL. It can be thought of as a formalized goal that the agent will try to optimize in the long run. Thus, it is not only the immediate reward, but rather the cumulative reward over time that is of interest. The reward is a scalar value $R_t \in \mathcal{R}$ which is usually modelled as -1 for negative feedback, as in losing, $+1$ for positive feedback, as in winning, and 0 for nonterminal states.

The reward must be provided to the agent in a such a way that the agent is likely to learn from the feedback and improve its behavior and achieve the goal of the RL task. A rule of thumb is to model the reward in terms of endgoals, and not in subgoals. If the agent is rewarded for achieving a subgoal, it might learn the wrong objective. E.g. if an agent is rewarded for taking pieces in the game of chess, it might do everything in its power to take high value pieces even though a certain move will result in the opponent checkmating. Thus, in chess it might be a good idea to model checkmate as $+1$, opponent checkmate as -1 and all other moves as 0 [22].

Returns

Informally the agent is maximizing the cumulative reward over time. To refer to this concept more formally, we begin by aggregating the rewards obtained by the agent over time in a sequence, $R_{t+1}, R_{t+2}, R_{t+3}, \ldots$. The return is simply the sum of the sequence up to time step T like follows:

$$G_t = R_{t+1} + R_{t+2} + R_{t+3} + \cdots + R_T \tag{3.3}$$

This approach works very well when the RL problem naturally breaks up into episodes, namely episodic tasks. Because in these cases there is a natural notion of what the final time step is, namely the time step occurring when the agent enters a terminal state. However, when the task is continuing infinitely as in continuing tasks, there is no final time step. In this case T would go to infinity, $T \to \infty$. The sequence of G_t would be an infinite sequence. Maximizing an infinite sequence is problematic since it could easily be infinite. To deal with this problem, the concept

of discounting is used. By utilizing this approach, rewards that are further in the future will be discounted by a larger factor. By expressing the return G_t as the sum of discounted rewards we obtain:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{3.4}$$

Here γ is called the *discount rate* and it is defined in the interval $\gamma \in [0, 1]$. By letting the discount rate $\gamma \to 1$, the agent will put more value on future rewards. When the discount rate is $\gamma = 0$, the agent only cares about immediate rewards. As long as the discount rate is $\gamma < 1$, the sum of the discounted rewards seen in Equation (3.4) will be finite as long as the rewards are bounded. To unify episodic tasks and continuous tasks, and hence, Equation (3.3) and Equation (3.4), we can rewrite Equation (3.4) slightly:

$$G_t = \sum_{k=0}^{T} \gamma^k R_{t+k+1} \tag{3.5}$$

Setting $\gamma = 1$ while the sum remains defined yields the equation for episodic tasks. Letting $T \to \infty$ while $\gamma < 1$ under the assumption that the sum remains defined, yields the equation for continuous tasks. When $T \to \infty$ and $\gamma = 1$, the sum is not defined, and is hence not a possibility [22].

.

Policies

When an agent makes actions over a long period of time in an environment, an observer may deduce a pattern in the agent's way of acting. A policy is simply how an agent should act. For example, an agent may follow a random policy, in which the agent's actions are based on a random distribution. A policy can be defined as a mapping from states to probabilities of selecting available actions in the given state. A policy is often denoted as π. An agent may be said to follow a certain policy π at a time t. As a policy is a mapping from state to action probabilities, a mapping function $\pi(a \mid s)$ is used to extract the behavior of an agent at time t [22].

Value Functions

To estimate how good a state is in terms of expected future rewards, *value functions* are used. These are functions of states or state-action pairs that are also estimated by the RL algorithm through experience. The value function is defined with respect to a certain policy, a certain way of acting, and is denoted as $v_\pi(s)$. Given that an agent starts in state s, the *state-value function* gives the expected return under the assumption that the agent follows policy π for subsequent states. The state-value function is formally defined as:

$$v_\pi(s) = \mathbb{E}_\pi[G_t \mid S_t = s], \quad \text{for all } s \in \mathcal{S} \tag{3.6}$$

Similarly, given that an agent starts in state s and takes action a, the *action-value function* gives the expected return under the assumption that the agent follows policy π in subsequent states:

$$q_\pi(s, a) = \mathbb{E}_\pi[G_t \mid S_t = s, A_t = a] \tag{3.7}$$

The difference between these two equations is that the action-value function hypothetically may choose an action that diverges from the policy π in the first time step. The goal of a RL algorithm is to find an optimal policy by training the learning agent. An optimal policy achieves the best possible return. For finite MDP, it is possible to define what optimal policy is. Informally, an optimal policy is always at least as good as all other policies. This implies that there may be more than one optimal policy.

An optimal policy is denoted as π_*. Value functions that operate with respect to an optimal policy are denoted as $v_*(s)$ and $q_*(s, a)$. Even though there may be multiple optimal policies they all share the same value functions. Once q_* is determined, the optimal policy is easy to find. An agent must choose the action that maximizes $q_*(s, a)$. Choosing actions based on this principle is the optimal policy. It is also possible to determine the optimal policy using v_*, but the agent has to do a one-step-ahead search through the possible actions [22].

3.3 Models

This section will describe different machine learning models used throughout the thesis.

3.3.1 Decision Tree

A decision tree is a hierarchical model with a tree-like structure. Each node represents a conditional statement. The branches of each node are the paths that the conditional statement may take. The leaf nodes represent the final "decisions", which are single output values. To determine how a decision tree reasons about a dataset it has been trained on, the tree takes the input vector and follows each conditional control statement from the root node to one of the leaf nodes depending on the features of the input vector and the features evaluated at each branch.

The nodes in a decision tree may have more than two children, but the rest of this section assumes at most two children for each node. Figure 3.3 shows an overview of how this might look like. In this figure, two nodes represent certain features x_1 and x_2. When a sample $x_1 = 3$ is passed to the decision tree, the decision tree will choose the left path and therefore its left child since this path corresponds to the conditional statement $x_1 \leq 5$ [23].

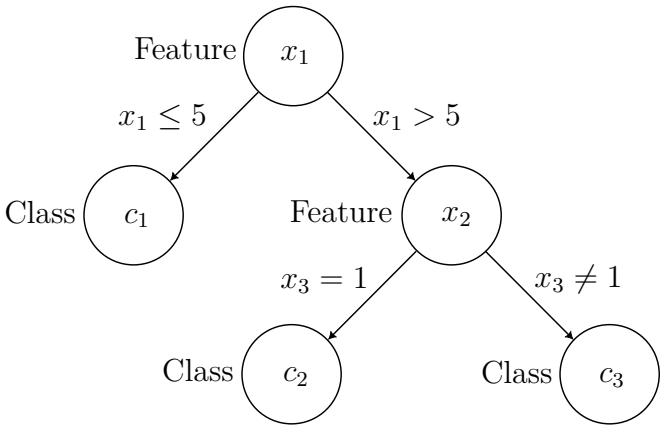


Figure 3.3: Decision tree example. The control flow begins in the node at the top and goes downwards to the left or right node depending on the conditional statements.

To train a decision tree, the decision tree learning algorithm adopts a greedy divide-and-conquer strategy. The decision tree will select the most important attribute first to evaluate if it makes the decision tree a good approximator. To determine which attribute is most important, the decision tree follows the intuition of finding the attribute that has the biggest impact to the outcome of the remaining training samples based on the reduction in entropy of choosing one attribute over another.

When the decision tree has determined the most impactful attribute and created the necessary branches to divide the training samples, the process starts over again with new root nodes, the leaves of the created branches, and a reduced training set, because the samples which have a clear answer can be ruled out. The goal of this approach, is to create an approximator with few paths so that the tree as a whole remains shallow [21].

3.3.2 Random Forest and AdaBoost

Random forest is an example of an ensemble learning algorithm. An ensemble learning algorithm uses multiple learning algorithms to obtain a better result by combining the output from each individual learning algorithm in a certain way. A random forest model consists of a finite set of decision trees and can be used in both classification and regression. For classification tasks, the combined output is the class that are selected by most decision trees. For regressions tasks, the combined output can be the average or mean of all the decision trees.

The random forest model is built by bootstrap aggregating the training data and subsequently training a tree on a subset of the features in that sample. More concretely, each tree is trained on a sample of the training data, and only some of the features are used. Each tree is therefore underfitted and biased, but the combined output of many such trees average out the errors. So instead of having a massive tree that is supposed to learn everything, easily leading to overfitting, each tree in the random forest can learn a small part of the data and provide emergent intelligence together [24].

AdaBoost is an ensemble technique similar to random forests. AdaBoost typically also consists of decision trees, but with shallow depth. The main difference is how the training data is sampled. In random forests it was randomly sampled, whereas in AdaBoost, each sample has a weight that increases if the sample was miss-classified by a previous tree. So the next tree trained is biased to learn that sample. Thus leading the ensemble to prioritize learning the difficult examples [24].

3.3.3 Support Vector Machine

SVMs are supervised machine learning models that create a mapping of the feature space by maximizing the distance between the categories in the output space. Different kernels, a set of mathematical functions, can be used for the SVM. For the linear SVM, the training phase consists of adjusting a hyperplane that separates the features with respect to the output space. Traditionally, only two classes are used in the output space, which makes it a binary classifier. As SVMs maximize the distance between the points of the different classes, no probabilities are involved, making SVMs a non-probabilistic binary classifier [25].

3.3.4 Artificial Neural Network

The artificial neural network has its roots in neuroscience. The network is modeled to mimic the behavior of the electrochemical activity found in the brain cells called neurons. Hence, artificial neural networks consist of nodes, the neurons, which are connected by directed links. A node has a set of inputs and outputs. Each input is aggregated with the summation function combined with a weight and bias. The input at node j is defined as $\mathcal{I}_j = \sum_{i=0}^{n} w_{i,j} a_i + b_j$. The weight $w_{i,j}$ is the link from node i to node j and determines the strength of the connection. The input of each node may be propagated through a function called an *activation function*. This function is denoted as g and serves to allow for nonlinear relationships of the data. It is used before passing the received value forward $a_j = g(\mathcal{I}_j)$ for a node j [21].

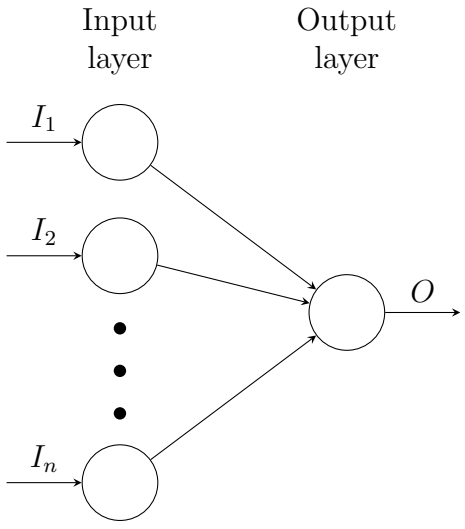


Figure 3.4: A single layer neural network known as a perceptron.

A single layer neural network consists of an input and output layer. This type of network is commonly known as a *perceptron*. Figure 3.4 shows a perceptron network with n inputs and one output. A perceptron network with more than one output, m is really m separate networks as the weights between the inputs to a certain output do not affect the weights between the inputs of another output. Thus, there are m separate training procedures for such a network, each which could be simplified to the structure shown in Figure 3.4. Multiple layers of neurons are used to construct the whole network. The way multiple layers of neurons are constructed in Figure 3.5 is called a *feed-forward network*. Feed-forward networks only have links in one directions, and hence, form an acylic directed graph. The model using the technique with multiple layers of perceptrons is conveniently called MLP [26].

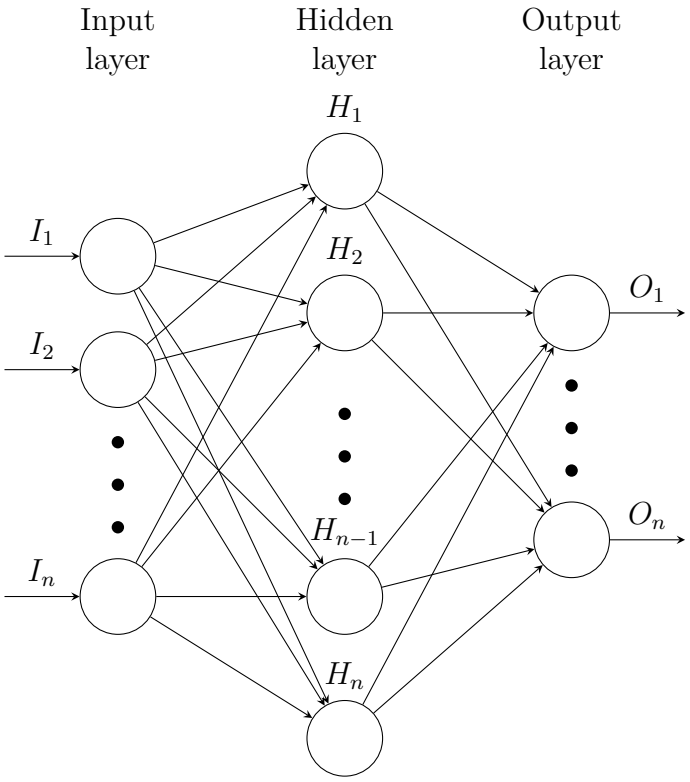


Figure 3.5: Neural network structure with an input layer, hidden layer and output layer. Each layer in the figure has n nodes.

To train the network, some type of feedback is needed in the layers. The loss function defines how the network should change its parameter weights to minimize the error between the predicted values and the target values using a technique called backpropagation. Backpropagation computes the gradient of the loss function with respect to each parameter in the network one layer at a time using the chain rule, traversing the network backwards to avoid unnecessary computations [27].

A common loss function when dealing with multiple output classes is the categorical cross entropy loss function [28] which is used together with the softmax activation function [29]. The categorical cross entropy loss can be written as

$$\mathcal{L}(x, y) = -\log \frac{e^{x_y}}{\sum_{c=1}^{C} e^{x_c}} \tag{3.8}$$

where e^{x_y} is the score for the positive class. Another loss function is the smooth L1 loss function [30] which is defined as

$$\mathcal{L}(x, y) = \begin{cases} 0.5\,\frac{(x-y)^2}{\beta}, & \text{if} |x - y| < \beta \\ |x - y| - 0.5\beta, & \text{otherwise} \end{cases} \tag{3.9}$$

In smooth L1 loss, the loss is split up into two parts. The first part $0.5(x-y)^2/\beta$ is defined as the L2 loss in the interval $[0, \beta)$, whereas the second part $|x - y| - 0.5\beta$ is defined as the L1 loss in the interval $[\beta, \infty)$. The effect of using the L1 loss for samples above β reduces the over-penalization of outliers.

3.3.5 Q-learning

The Q-learning algorithm is centered around Equation (3.7). The algorithm directly approximates the optimal action-value function q_*, through the learned action-value function Q. The algorithm is defined by

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \max_a Q(S_{t+1,a} - Q(S_t, A_t) \right] \tag{3.10}$$

In the Q-learning algorithm, the Q-function is represented as a table of state and action values where the rows correspond to the states and the actions to the columns, see Figure 3.6 [22]. During training, a Q-agent adjusts its Q-values according to Equation (3.10) with a learning rate of α [31]. Q-learning is an off-policy learning algorithm because it updates its Q-values using a policy that it is not following the greedy policy.

During training, a Q-agent samples actions with exploration and exploitation in mind. With a certain probability a random action is selected for exploration. Otherwise, it exploits its current Q-value by greedily choosing the currently optimal action-value for the current state. The probability by which the agent select actions randomly can vary throughout the training phase by a factor of ϵ that decreases after each iteration. This is known as an ϵ-greedy policy, as the agent will choose actions more greedily towards the end of the training phase [22, 32].

$$\begin{bmatrix} Q(s_1, a_1) & Q(s_1, a_2) & \dots & Q(s_1, a_m) \\ Q(s_2, a_1) & Q(s_2, a_2) & \dots & Q(s_2, a_m) \\ \vdots & \vdots & \ddots & \vdots \\ Q(s_n, a_1) & Q(s_n, a_2) & \dots & Q(s_n, a_m) \end{bmatrix}$$

Figure 3.6: A Q-table containing n states and m actions.

3.3.6 DQN

As the number of states grow large, it will become infeasible to use tabular agents as in the Q-learning algorithm. One way to overcome this problem is to represent the Q-function using a function approximator such as a DNN. This approach is called DQN. The network is defined as $Q(s, \cdot; \theta)$, and outputs a vector of action values given a certain state s and the network parameters θ. To improve the performance of the algorithm, DQN make use of two techniques. The first technique is called experience replay [33]. With experience replay enabled, observations of the current and next iteration, actions and rewards are stored for a specified number of time steps T. The idea is to use uniform samples from the experience replay memory bank to update the network at every time step t. The second technique is to make use of a target network that is a replica of the layout of the original network, but with parameters θ^- instead of θ. The target network is updated at another frequency, every τ time steps, than the original network, which is updated every time step. At time step τ, the target network copies the parameters of the original network, $\theta_t^- = \theta_t$. The target network is used when defining the target Y_t^{DQN} as follows

$$Y_t^{\text{DQN}} \equiv R_{t+1} + \gamma \max_a Q(S_{t+1}, a; \theta_{\mathbf{t}}^-) \tag{3.11}$$

Just like in Equation (3.10), the update step can, with help of Equation (3.11), be defined as follows for the DQN algorithm

$$\theta_{t+1} = \theta_t + \alpha[Y_t^{\text{DQN}} - Q(S_t, A_t, \theta_t)]\nabla_{\theta_t} Q(S_t, A_t; \theta_t) \tag{3.12}$$

By defining the update step like this, the algorithm will utilize gradient descent to update the current value $Q(S_t, A_t; \theta_t)$ towards the target Y_t^{DQN} [34, 35].

3.3.7 DDQN

The problem with Q-learning and DQN is that both these algorithms tend to give overly optimistic estimates, because they use the same values to select and evaluate an action. To circumvent this issue, an additional set of parameters θ' can be used to select the greedy policy, while the original set of parameters θ determine the value of Q. This technique is referred to as Double Q-learning. Thus, the selected action a_{θ_t} is

$$a_{\theta_t} = \operatorname*{argmax}_a Q(S_{t+1}, a; \theta_t) \tag{3.13}$$

while the target Y_t^{DoubleQ} is

$$Y_t^{\text{DoubleQ}} \equiv R_{t+1} + \gamma Q(S_{t+1}, a_{\theta_t}, \theta_t') \tag{3.14}$$

The role for each of these parameters are switched continuously to keep both set of parameters updated. To combine the technique of Double Q-learning with DQN, we can let the additional parameter proposed in Double Q-learning θ' be the target

network with parameters θ^- in DQN. Hence, the target function $Y_t^{\text{DoubleDQN}}$ of DDQN is

$$Y_t^{\text{DoubleDQN}} \equiv R_{t+1} + \gamma Q(S_{t+1}, a_{\theta_t}, \theta_t^-) \tag{3.15}$$

The target is used instead of Y_t^{DQN} in Equation (3.12) for DDQN. When defining the target by setting $\theta' = \theta^-$, the process of selecting an action and value is not fully decoupled, but provide the benefits of DQN and and Double Q-learning without the large computation load of introducing an additional network [35].

4

Method

This chapter starts by summarizing previous beam management simulations in the scientific literature. Then, the parameters and method for our simulation are described, with some insight into the generated data. A brief explanation of the pipeline we used is given as well as how the RL environment was constructed.

4.1 Literature Review

The following section explores machine learning methods that different research papers highlight. Each method will be briefly summarized. Table 4.1 shows which methods each paper has investigated.

Model	Paper
Decision tree	[11]
Random forest	[11, 20]
AdaBoost	[11]
SVM	[11]
MLP	[11, 20]
Q-learning	[36]
DQN	[11]
DDQN	[37]

Table 4.1: A table showing the supported models in the pipeline and which paper each model is derived from.

In [11], a traffic simulator is combined with a ray-tracing simulator to model 5G channels in a realistic urban scenario. The data is generated by projecting rays with a predefined spacing in the three-dimensional angular space. The paths of each ray are ranked according to the received power at the UE. Moreover, the machine learning models they used (see Table 4.1) incorporate the positions and sizes of each vehicle as the only data required to perform beam selection. A receiver was defined as several coordinates that represented a vehicle. The resulting accuracy of the models is not the focus of the paper, but rather the simulation strategy. However, they do test their models on a small data set and conclude that the random forest and deep

neural network performed the best, with decision tree and AdaBoost slightly worse, and SVM about twice as poorly.

In [20], a beam training method assisted by machine learning is proposed where the method predicted the optimal Access Provider (AP) and the optimal beam based on the Global Positioning System (GPS) coordinates of the UE. The models have been trained with data generated from a RT simulation where the coordinates for each UE were transmitted through a lower-frequency Long Term Evolution (LTE) link. The selection models they used, see Table 4.1, were based on supervised classification. Their results showcase the accuracy of AP selection as well as beam selection, in both cases using perfect GPS data or with added noise. The random forest and MLP both give good results, but it is clear that the MLP does not overfit and is thus better in more noisy conditions. AdaBoost and SVM were also tested, but they chose to leave them out of the paper due to unsatisfactory performance.

In [36], a simulator from Ericsson was used to simulate moving UEs in a hexagonal cell with a radius of 100 m. The goal was to train a Q-learning algorithm in a scenario with moving UE to reduce the search space of the candidate beam set when selecting the best transmitter beam. Each UE has a predefined movement pattern of either one of two types. The first type of UEs always move straight and bounce in the opposite direction of arrivial when hitting a wall. The second type of UEs mimic the behaviour of how real world pedestrians or vehicles move along a road. The candidate beam set was calculated by taking the 5 closest beams among the closest 19 beams by either a baseline algorithm or the Q-learning agent. Additionally, the beam that was currently established between the BS and the UE was added to the candidate beam set for a total of 6 beams. Their results show that Q-learning provided slightly better throughput than a baseline algorithm.

In [37], a DDQN was used to dynamically adjust beam directions between the position of a high-speed train and a BS. To reduce overhead of the beam alignment procedure, multiple location bins were introduced that acted as trigger zones. The state space for the DDQN agent consisted of a three-dimensional tuple $(d, \theta, \phi) \in \mathcal{S}$ composed of the horizontal distance d, and the zenith and azimuth angles θ and ϕ between the transmitter and the receiver on the train. The action space consisted of a two-dimensional tuple $(\pm\sigma_b^\theta, \pm\sigma_b^\phi) \in \mathcal{A}$, where σ_b^θ and σ_b^θ are either fixed or zero, that denoted the total change of angle in the zenith and azimuth plane from the previous state. The reward was defined as the difference between received power of the current state and the next state. Their results showed that the DDQN beam tracking provided higher received power, and reduced the search time to 20 % of the original value.

4.2 Simulation Environment

Data has been generated using the QuaDRiGa channel model according to three different scenarios, UMi, UMa and RMa. These scenarios are described in specification 38.901 provided by the 3GPP [38]. QuaDRiGa eases the implementation of these scenarios as it comes with predefined configurations which are based on specification

38.901. General configuration settings can be seen in Table 4.1a. In each scenario, a BS, single transmitter antenna, has been placed at the origin of a site in a hexagonal grid. Moreover, the transmitter antennas have been configured to a single sector within the site, see Figure 4.2. Each scenario has different radius, height and carrier frequency, see Table 4.2. The radius is often referred to as Inter-Site Distance (ISD).

Environment	ISD (m)	Tx height (m)	Carrier frequency
UMi	200	10	30 GHz
UMa	500	25	30 GHz
RMa	1732	35	0.5 Ghz

Table 4.2: Simulation properties that differs between each scenario.

Receivers have been generated in different schemes depending on the scenario. However, each receiver had general constraints on where it was placed within the sector, see Table 4.1b. A receiver had to be placed at least 10 meters away from the transmitter. The maximum distance was also constrained to $0.93 \cdot \text{ISD}$ meters away from the transmitter. Every receiver has been placed at the same height of 1.5 m since beam sweeping was performed only in the azimuthal plane.

Property	Value
Layout type	Hexagonal
Number of sites	1
Number of sectors	1

(a) General layout properties.

Property	Value
Rx min distance	10 m
Rx max distance	$0.93 \cdot \text{ISD}$ m
Height	1.5 m

(b) Rx distance properties.

Figure 4.1: General properties for every scenario: UMi, UMa and RMa.

The UE have been sampled according to three different distributions, one for each scenario. In the RMa scenario, the uniform distribution was used while the UMi and UMa scenarios used custom city distributions. Thus, each scenario generated a separate dataset. In the RMa and hence the uniform dataset, the UE are sampled uniformly across a circle sector, see the greyed area in Figure 4.2. The samples are created using polar coordinates. The random variables are defined by constructing $u \sim 2U[0, 1]$ and defining the angle and radius as follow

$$\theta \sim U[0, 1] \cdot 120^\circ, \qquad r \sim \begin{cases} u \cdot (r_{\max} - r_{\min}) + r_{\min}, & u \leq 1 \\ (2 - u) \cdot (r_{\max} - r_{\min}) + r_{\min}, & u > 1 \end{cases}$$

where $r_{\min}$ is the minimum allowed radius for a scenario and $r_{\max}$ is the maximum allowed radius for a scenario. Figure 4.2 shows how the BS was placed at the center of a site and pointed in a certain sector. The grayed area is the region where the UE

resided. The colored beams, illustrate how the transmitter sweeps the area. The radius of the circle sector is dependent on the ISD defined for each scenario, which can be found in Table 4.2.

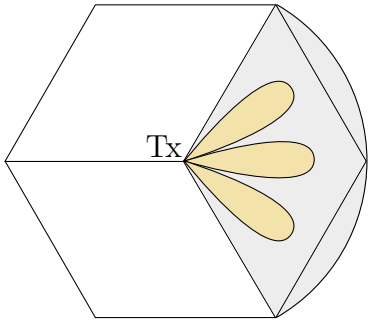
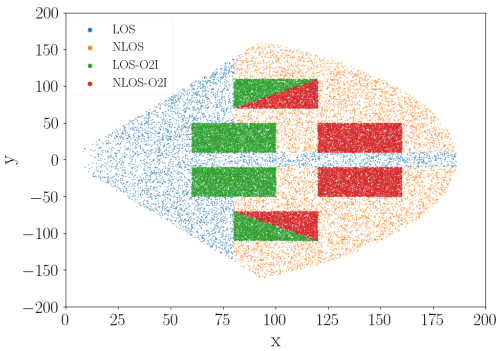
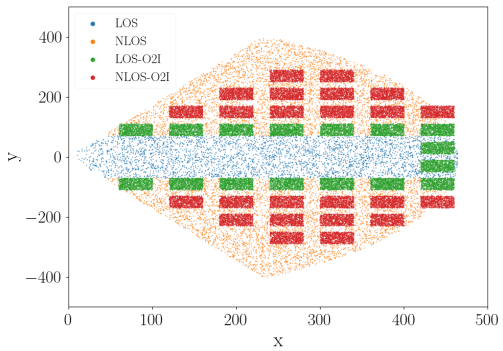


Figure 4.2: A single sector of a site is covered by the transmitter. The sector is 120 degrees wide.

Custom cities have been constructed for the UMi and UMa scenarios. Positions for indoor and outdoor conditions have been manually set in order to achieve more realistic and consistent data. Although, there is no specific layout for the RMa scenario, much of the blocking is due to the terrain. The distribution of the UE for the UMi and UMa scenario can be seen in Figure 4.3a and 4.3b respectively. The UE have been color-coded depending on the type of the transmission. The green and red colors means that they were placed within a building while blue and orange colors means that they were placed outside a building. There are a total of four colors for the combinations of LoS or NLoS and Outdoor to Indoor (O2I) or not O2I. As stated by the 3GPP, approximately 80% of the UE should be placed indoors, while 20% of the UE should be placed outdoors. This is the explanation behind the higher concentration of green and red UE in Figure 4.3 for both scenarios.



(a) The distribution of receivers for the UMi city layout.



(b) The distribution of receivers for the UMa city layout.

Figure 4.3: Custom receiver distributions according to two city layouts. The blue dots are receivers in LoS, the orange are in NLoS, the green are in O2I LoS and the red are in O2I NLoS. The scenario distribution is not exactly geometrically accurate but it serves as a decent approximation.

The antenna of the BS and UE are defined according to Table 4.3. The table shows what differs between the antenna of the BS and the antenna of the UE. The only difference between the antennas is that the antenna of the BS had a certain level of downtilt while the antenna of the UE had none. In QuaDRiGa, the antenna type specified as `3gpp-mmw` has the argument shown in Table 4.3. M refers to the number of vertical elements and N refers to the number of horizontal elements. The polarization refers to the type of polarization used, defined between 1-6. By setting the polarization to 6, the elements will be 45 degrees cross polarized. The spacing refers to the element spacing defined with regards to the wavelength. For example, setting the spacing to 0.5 results in a spacing of 0.5 λ. Mg refers to the number of nested panels in a column, whereas, Ng refers to the number of nested panels in a row. The V panel spacing determines the spacing in the vertical direction, and the H panel spacing determines the spacing in the horizontal direction.

Property	BS	UE
M	4	4
N	4	4
Polarization	6	6
Downtilt	**12**	**0**
Spacing	0.5	0.5
Mg	1	1
Ng	2	2
V panel spacing	2.5	2.5
H panel spacing	2.5	2.5

Table 4.3: QuaDRiGa's `3gpp-mmw` antenna configuration for the BS and UE. The only difference between the BS and UE is the downtilt.

A brief explanation of the time invariant QuaDRiGa simulation follows. The UE positions were generated and placed on the layout according to the scenario's scheme. A burst of beam sweeps were performed by rotating all the antenna for all UE at once for each defined UE azimuth angle, after which the antenna of the BS rotated to its next azimuth angle. Then, the process started all over again until all angles had been cycled through. Each beam pair was measured at different times, so there is some variation in the parameters from the random distributions in QuaDRiGa. However, the received power during a single burst was measured at the same time which is necessary for consistency in the data.

The time variant dataset is similar to the time invariant dataset. It was an extension of the RMa scenario with moving UE along linear tracks of 60 meters. A track is a segment of paths that the receiver followed during the simulation. At a fixed time step a snapshot was recorded, containing the simulated paths between the transmitter and the receiver. A linear track was applied to every receiver with a random angle. The velocity of each receiver was 3 m/s and 20 time steps were recorded on each receiver. The snapshot frequency was set to 1/3 samples per meter giving a total of 60 m in path length. Figure 4.4 shows two UE with different angles

along a linear track.

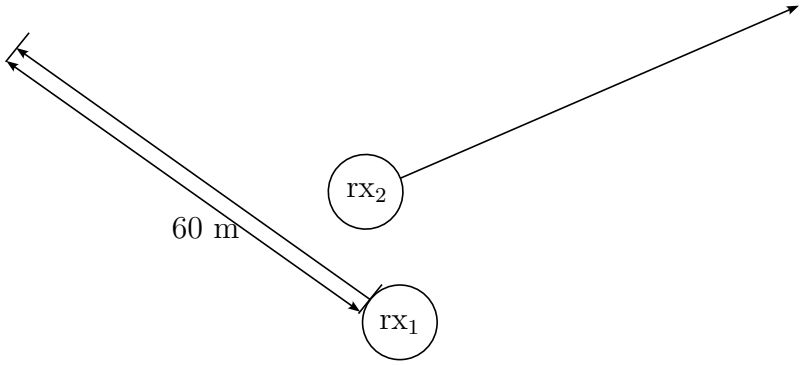


Figure 4.4: Two UE along their linear track marked as rx_1 and rx_2.

The type of data that the QuaDRiGa simulation outputs is presented in Table 4.5. The receiver positions were floating point values placed in the sector according to the layout. There are 5 valid angles for the BS and 3 valid angles for the UE, resulting in a total of 15 beam pairs. The number of paths that a beam was scattered through differed between the scenarios. In the UMi scenario a maximium of 58 paths were allowed, in the UMa scenario a maximum of 61 paths were allowed while in the RMa scenario a maximum of 11 paths were allowed. Table 4.4 maps each beam pair with its corresponding azimuth angles. The UE in the time invariant datasets were set to an initial direction pointing west, whereas, the UE in the time variant dataset had an initial direction straight forward along the track.

Beam pair	Tx-azimuth	Rx-azimuth
1	-60	-60
2	-60	0
3	-60	60
4	-30	-60
5	-30	0
6	-30	60
7	0	-60
8	0	0
9	0	60
10	30	-60
11	30	0
12	30	60
13	60	-60
14	60	0
15	60	60

Table 4.4: This table summarizes the beam pair indices with their corresponding angle for the BS and the UE. The angles are given in degrees. Tx-azimuth represents the angle for the BS while the Rx-azimuth represents the angle for the UE.

Rx-x	Rx-y	Tx-azimuth	Rx-azimuth	Path power
$x \in \mathbb{R}$	$y \in \mathbb{R}$	$\theta_{tx} \in \{-60, -30, 0, 30, 60\}$	$\theta_{rx} \in \{-60, 0, 60\}$	$p \in \mathbb{R}$

Table 4.5: Output data from QuaDRiGa simulation. Rx-x and Rx-y stands for the x and y coordinates of the UE. Tx-azimuth and Rx-azimuth stands for the azimuth angles that each respective antenna sweeps. Path power is the received power from the different paths that each beam travels due to scattering.

4.3 Preprocessing

Preprocessing of the time invariant dataset was minor. The only steps were to compile labels, aggregate the path dimension and split the dataset into train, validation and test sets. Applying an optimal and realistic method of aggregating the paths is a complex problem. Averaging was performed because it provides consistent results. Another simple approach is to take the maximum path but it has more power variance. That is due to QuaDRiGa intentionally generating power samples around an average [13, p. 82].

The sliding window dataset was created based on the time variant RMa dataset. The difference between the datasets is in how they were preprocessed in the pipeline. The sliding window dataset broke up the sequences of time correlated data into time independent data by sliding a window of k size over the sequence [39]. That is, consider a certain time sequence p of positions and beam pair labels (x_p^t, y_p^t) and l_p^t respectively for $1 \leq t \leq 20$. By sliding a window with $k = 2$ the sequence is reordered to $(x_p^t, y_p^t, x_p^{t+1}, y_p^{t+1}, l_p^t)$ with the corresponding label l_p^{t+1}. The steps are similar with $k \geq 2$. By transforming the dataset in this way, models like random forest and decision tree are able to learn the time correlation indirectly.

4.4 Reinforcement Learning Environment

The RMa time variant dataset was also used in the construction of the RL environment. As the environment used offline data, it was split up into a training set and test set. Intuitively, the training set and test set were split up to two different phases, the training phase and the evaluation phase. During the training phase, the agent tried to learn the optimal policy using both exploration and exploitation. During the evaluation phase, the agent was set to achieve the best possible reward by exploiting its learned optimal policy.

The environment was modelled as a "hybrid" episodic task where one episode consisted of the whole time sequence. Each time sequence was 20 time steps. The task acts like an episodic task when the agent interacts with it because of the preprocessing. However, when UE is moving in a real life sector, it is a continuous task. This will become apparent when describing the discretization of the state space.

A MDP was created based on the time sequences of the data, see Figure 4.5. The state space consists of a 5-tuple $(x^k, y^k, x^{k+1}, y^{k+1}, a^k) \in \mathcal{S}$ where $(x^{k+i}, y^{k+i})_{i \in \{0,1\}} \in$

$\mathbb{R}$ are the previous position and current position respectively in the time sequence, while $a^k \in \mathcal{A}$ represents the chosen serving beam for position (x^k, y^k) at time $t = k$. The action space consisted of all beam pairs as in the SL problem formulation. Hence, the size of the action space was $|\mathcal{A}| = 15$. As the environment was based on offline data, all beam pairs have already been established between each time step in the sequence. The agent was set to predict the best serving beam when moving through the positions in the time sequence. A sequence began with an initial state $(x_s, y_s, x^k, y^k, a_{\max})$ where (x_s, y_s) was the starting position and $a_{\max}$ was the best beam pair for that position. In this state, the agent had to predict a new chosen beam pair given the additional information of the movement in the time sequence (x^k, y^k).

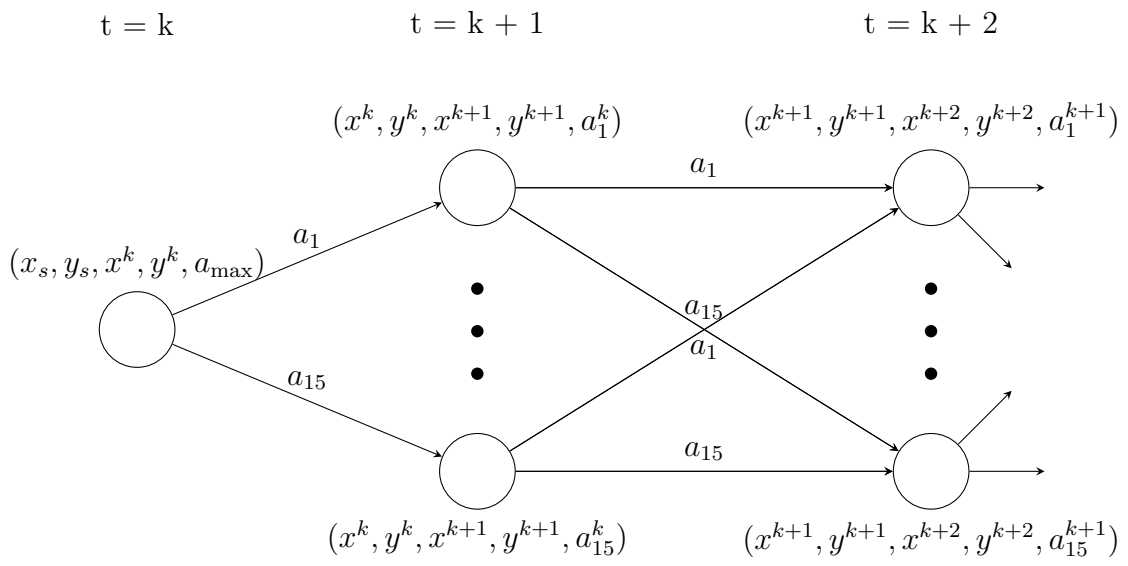


Figure 4.5: This is the MDP used as the environment. Each node represent the state while the arrows between the nodes represent the actions.

The agent was rewarded with higher rewards the closer its predicted beam was to the optimal beam according to

$$r = \frac{1}{2^{|i-j|}} \tag{4.1}$$

where i and j are the indices for the actions a_i and a_j respectively given that j represents the optimal index. This implies that an agent was rewarded with a score of 1 if it predicted the optimal beam, and that the reward was halved for each index from the optimal, converging to 0. Furthermore, a penalty was incorporated as well. If the difference was higher than that for the current serving beam, a penalty of 0.25 was subtracted from the reward. Similarly, if the current serving beam and the newly chosen beam was greater than 2, implying that the newly chosen beam did not account for spatial locality, a penalty of 0.25 was subtracted from the reward. To enable spatial locality for the beam indices, the beam indices had to be reordered in the preprocessing step for the environment.

As the positions in the state tuple are defined by real numbers $(x^{k+i}, y^{k+i})_{i\in\{0,1\}} \in \mathbb{R}$, the state must undergo discretization to work with the tabular representation used in Q-learning. The continuous space for each axis was split up into n different bins, resulting in a state space of the size $|\mathcal{S}| = n^4 \cdot 15$. The size of each bin was calculated by taking the maximum and minimum value along each axis. For example, the x-axis size was $(x_{\max} - x_{\min})/n$. Due to the memory size scaling with the number of states, n was limited to 10. Using 10 bins along each axis resulted in a state space of $10^4 \cdot 15 = 150000$ states, and a bin size of $190 \cdot 320 = 60800$ m^2, as the RMa dataset was used. Discretizing the state space by dividing the plane into a grid according to these bins was naive, as it had the effect of many empty states. The positions of UE are contained within a circle sector and the grid represents the rectangle that encloses the circle sector.

To reconnect to the definition of the "hybrid" episodic task in the case of discretization, one can observe that a state most likely is used by multiple sequences, but not necessarily with regard to the same time step in each sequence. The last time step in each sequence should be defined as absorbing since no further state can be reached from this time step. By definition, the Q-value for an absorbing state is equal to zero which means that the agent stops there. However, setting the value to zero for an absorbing state was problematic since the state may be used in another sequence in a non-absorbing way resulting in the wrong Q-value for this particular sequence. Thus, the value for absorbing states was not set to zero.

4.5 Pipeline

The pipeline was implemented with Python. To support all models found in Table 4.1, two common Python libraries were used: Scikit-learn [40] and PyTorch [41]. The decision tree, random forest, AdaBoost and SVM were implemented using the Scikit-learn library [40]. These models are supported out of the box and have been encapsulated with wrapper classes to fit along the other models in the pipeline. The MLP, DQN and DDQN agents were implemented using PyTorch. Q-learning did not use any third-party library. The pipeline has been implemented as a command-line utility to ease the training and evaluation phases. Listing 4.1 shows how the utility was invoked. In the case multiple models were specified, the pipeline trained and evaluated each model on the specified dataset. If multiple datasets were specified, the pipeline ran the models sequentially on each dataset.

```
python -m <program> -d [datasets] -m [models] --OPTIONS
```

Listing 4.1: An example showing how to invoke the pipeline from the command-line.

The pipeline consisted of a hierarchical structure of loaders and models, see Figure 4.6. A loader is a wrapper class around a dataset. As the thesis involves two paradigm, SL and RL, wherein SL models are implemented using two different libraries, a loader must keep track of its supported models. That is, a loader may

support a Scikit-learn model, PyTorch model or reinforcement agent, or any combination of such. This is indicated by how the data of the loader was preprocessed. Similarly, a model is a wrapper around a specific learner. A model is required to have the `fit()`, `evaluate()` and `run()` methods.

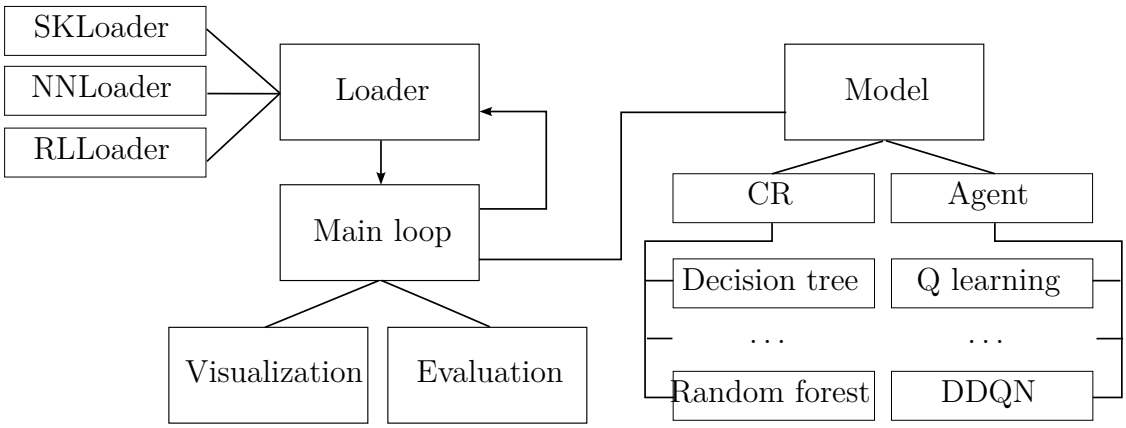


Figure 4.6: The structure of the pipeline. It consists of two main objects, the loader and the model. The "CR" means classifier and regressor.

4.6 Models

This section will describe model specific characteristics for each model as well as their implementation details. The implemented models were adapted to either the time invariant or time variant datasets. Table 4.6 shows the relation between each model and the datasets.

Model	Dataset
Decision tree	I, S
Random forest	I, S
AdaBoost	I, S
SVM	I, S
MLP	I, S
Q-learning	V
DQN	V
DDQN	V

Table 4.6: The table shows the supported datasets for each model. The "I" represents the time invariant dataset, the "V" represents the time variant dataset while the "S" represents the sliding window preprocessing step of the time variant dataset.

After the data was generated and a pipeline was setup to run the models, the models were tuned. For the Scikit-learn models, grid search was used to find good parameters. Table 4.7 shows the different parameters that were searched for each

model. The grid search performed a 5-fold cross validation on the parameters and the results are shown in the results chapter. After tuning these models, the MLP was built and tuned in PyTorch using a validation set.

Parameter	Model
Max depth	DT, RF, AB
#estimators	RF, AB
C	SVM
#iterations	SVM

Table 4.7: The parameters that were grid searched. DT, RF and AB stands for decision tree, random forest and AdaBoost. #estimators is the number of decision trees that were used in the random forest and AdaBoost. C is a regularization parameter for the SVM model.

In the implementation, the Q-learning, DQN and DDQN algorithms are derived from an abstract class called Q-agent. The Q-agent implements all base features that are inherent in these algorithms. This includes all general base parameters of the models and can be found in Table 4.8. Parameter ϵ_{decay} needs further explanation. By setting $\epsilon_{\text{decay}} = 0.8$, the agent reaches the minimum epsilon value ϵ_{min} by the time $0.8n_{\text{iterations}}$ had passed. Moreover, the Q-agent abstraction also includes an evaluator used to calculate the average reward and average episodic reward for the test set.

Parameter	Value
$n_{\text{iterations}}$	1 000 000
γ	0.99
ϵ	0.95
ϵ_{min}	0.05
ϵ_{decay}	0.80
α	0.01

Table 4.8: The set of base parameter values for the Q-agent. In order, the parameters refer to the number of training iterations, discount factor, exploration probability, minimal exploration probability, exploration probability decay and learning rate.

The implementation details regarding DQN and DDQN build upon the Q-agent abstraction. They use some further parameters for the target and policy networks. For both algorithms, instead of updating the target network at fixed time steps τ, the target network was updated gradually with a τ-factor according to

$$\theta^{-}_{t+1} = \tau \cdot \theta_t + (1 - \tau) \cdot \theta^{-}_t \tag{4.2}$$

By applying the gradual soft update rule, the stability of the learning algorithm is improved as the target network's parameters are changing more slowly [42].

Both the policy network and target network use a network layout according to Table 5.4 presented in the result chapter. The input shape to the network structure shown in Table 5.4 represent the continuous 5-tuple state introduced in Section 4.4. The network layouts used the smooth L_1 loss along with the Adam [43] optimizer.

5

Results

The first section will showcase the datasets that resulted from the QuaDRiGa simulations. Then, the second section will showcase the results of hyperparameter tuning for the models. Finally, the third section will showcase the results of evaluating the models on the datasets.

5.1 Simulation Results

A visualization of the target distribution in the time invariant datasets is illustrated in Figure 5.1. The goal of the figure is to visualize bias in the samples. In all three subfigures it is noticeable that the outer three beam pairs have fewer examples. The three beam pairs on either side, namely 1, 2, 3, 13, 14 and 15 correspond to the outermost transmit azimuth angles. Therefore, the reduction in sample frequency of those beam pairs can be explained by the fact that those beam pairs are directed along the edges of the sector, which leads to less coverage. For the UMi scenario, increased targets are visible in the 8th, 9th, 10th and 12th beam pair. Considering that the layout of the city in the UMi scenario is symmetrical, the reason for a right leaning bias can only be explained by the randomness in UE position generation and the QuaDRiGa channel coefficient samples. It is reasonable that the high frequency samples are close to the middle because of the four large houses in the layout.

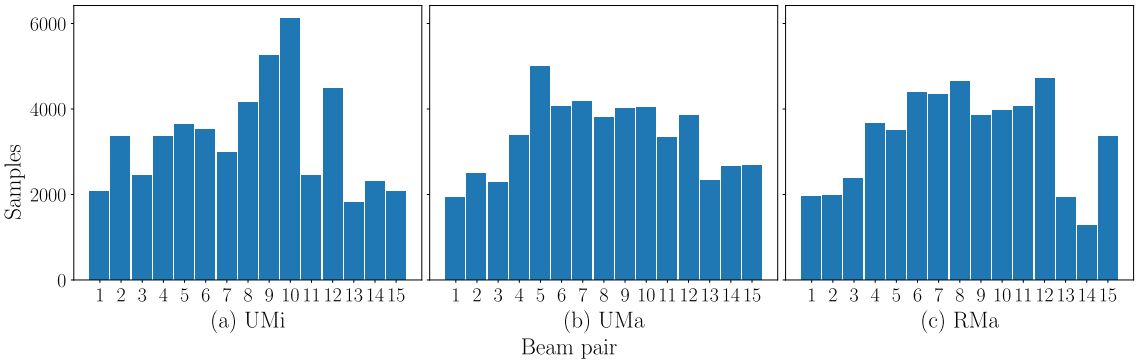


Figure 5.1: Distributions of target labels in UMi, UMa and RMa datasets. The x-axis shows the 15 beam pairs and the y-axis shows the number of samples.

Figures 5.2-5.5 show illustrations of both the UE positions and their corresponding

target labels. They are showcasing a subset of samples from the test set to show what the models are going to be tested on. The left figures show many receivers to give an understanding of their layout distribution and the transmit beam pattern. The right figures also show the optimal receiving beam for each UE to give an understanding of that pattern.

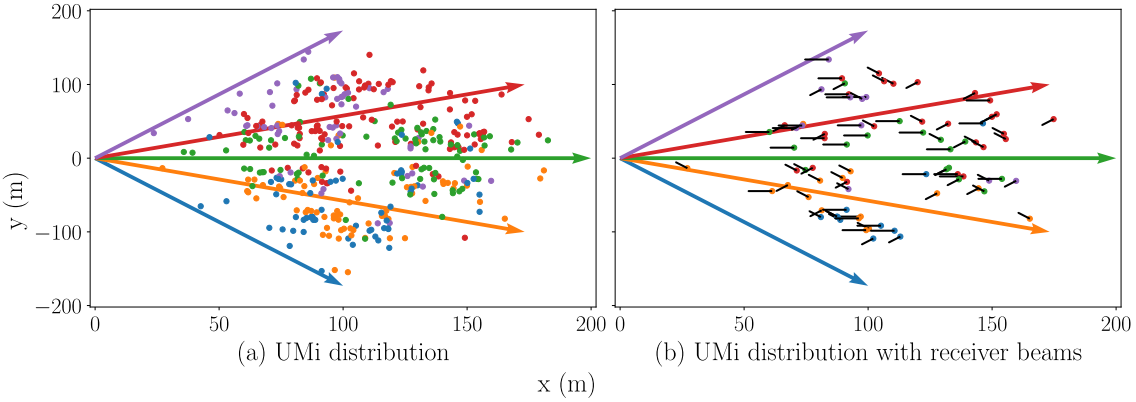


Figure 5.2: UMi dataset target labels visualized on the layout. The five big arrows are the transmit beam directions from the BS at the origin. Each dot represents a UE, which is colored by its optimal transmit beam connection. Figure (b) also shows the target receiver beam but fewer samples are shown to avoid overlap. On the x-axis are the x-coordinates of the layout and on the y-axis are the y-coordinates.

A pattern shown by the figures is that UE is biased to connect with the closest transmit beam, but with significant deviation in the buildings. The optimal receiver beam direction appears to be rather random. In the simulator, 80% of UE is placed indoors which is noticeable since the data points are more dense there. It is also clear that conditions change in just tiny distances, so that nearby UEs have different optimal beam pairs. That will affect the tuning of the model as it needs to be able to pick up on such detailed information, and still avoid overfitting.

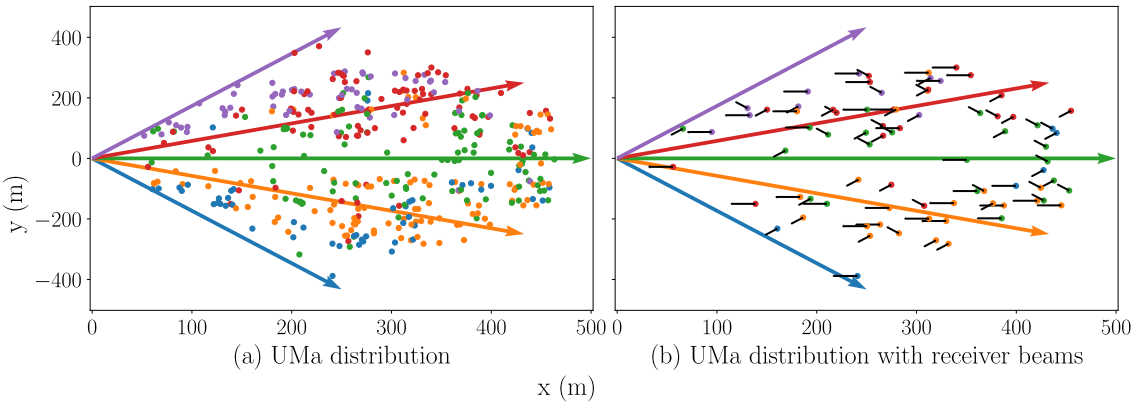


Figure 5.3: UMa dataset target labels visualized on the layout.

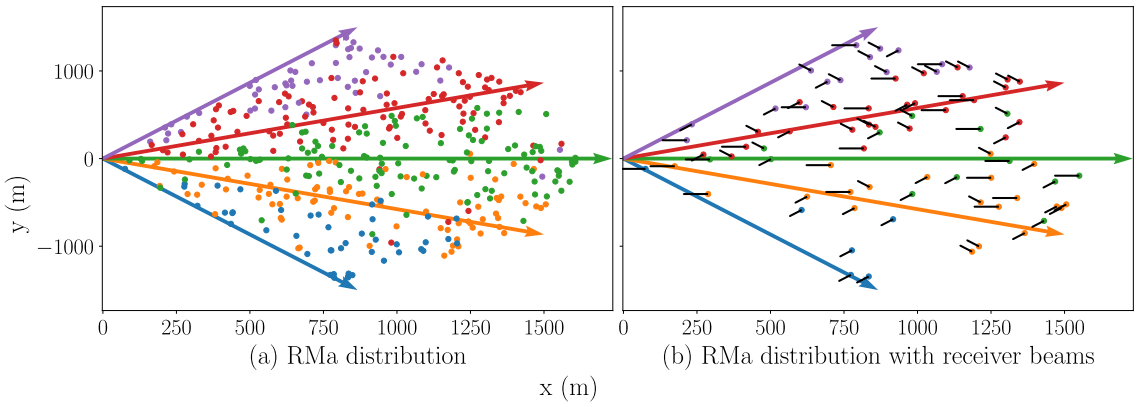


Figure 5.4: RMa dataset target labels visualized on the layout.

For the time variant RMa dataset, see Figure 5.5, the number of unique beam pairs was on average 2.6 for the whole sequence over all UE tracks. This implies that the optimal beam pair does not change much as UE moves along the track, but rather stays the same for about one third of the track, before switching to another optimal beam pair or toggles between beam pairs. By investigating how the sequences look like, it is clear that many UE tracks only use one or two optimal beam pairs, while few tracks have more rapid fluctuations.

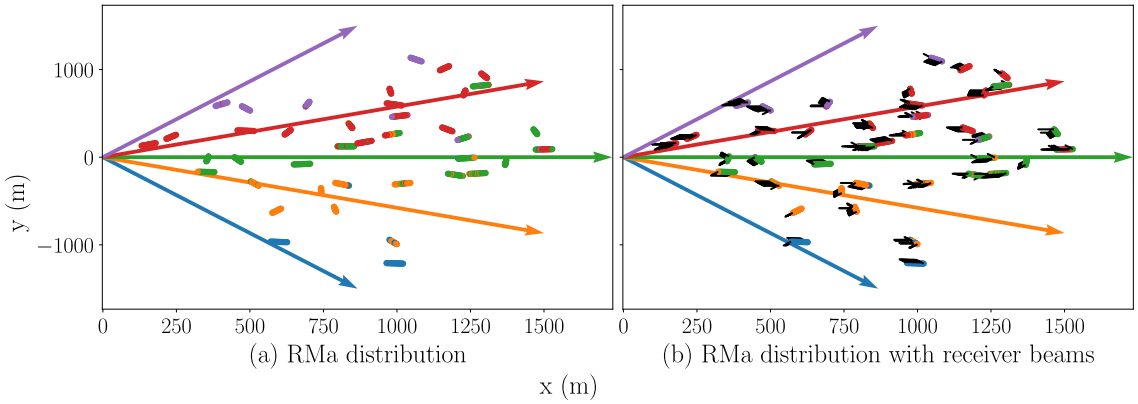


Figure 5.5: Target labels for the time variant dataset in the RMa scenario.

5.2 Hyperparameter Tuning

All time invariant models except the MLP were tuned by grid search to find the best parameters. The grid search performs 5-fold fitting on the training set for a set of parameters. The parameter of interest for the decision tree is the max depth. For the random forest and AdaBoost models the number of estimators were considered as well. The output of the grid search can be seen in Table 5.1 and Table 5.2. They contain the average accuracy during the 5 folds for all three time invariant scenarios. The grid search for SVM did not have an impact at all for the accuracy, meaning that all parameters gave the same results.

Max depth	UMi	UMa	RMa
1	0.24	0.18	0.17
5	0.43	0.30	0.37
10	0.75	0.59	0.55
20	0.90	0.83	0.68
50	0.90	0.83	0.68

Table 5.1: Grid search for the max depth parameter of the decision tree on all different scenarios.

In Table 5.1, the first column corresponds to the max depth parameter while the other columns correspond to the accuracy for each respective scenario. By looking at the accuracy, the overall trend is the same for all scenarios. The accuracy stagnates at a max depth of 20. The UMi scenario tend to have slightly better accuracy than the UMa scenario which in turn has slightly better accuracy than the RMa scenario. In Table 5.2 the overall pattern is approximately the same. The accuracy is only shown for the UMi scenario, as the UMa and RMa scenarios showed about the same overall trend as the UMi scenario except with a slightly lower accuracy as in the grid search for the decision tree. To make the table more readable, the number of estimators is not shown for Table 5.2 as varying the parameter did not have nearly as big of an impact on the accuracy.

Max depth	Random forest	AdaBoost
5	0.46	0.36
10	0.80	0.86
20	0.89	0.90
50	0.90	0.90

Table 5.2: Grid search for the max depth parameter of the random forest and AdaBoost models on the UMi dataset. The accuracy for UMa and RMa showed the same overall pattern, but with slightly lower accuracy.

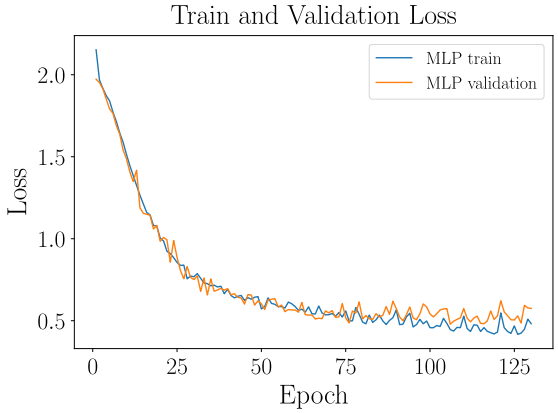


Figure 5.6: The MLP's train and validation loss curves when training for 130 epochs with the layout presented in Table 5.3 and using cross entropy loss.

For the MLP, hyperparameter tuning used a validation set and its loss graph is illustrated in Figure 5.6. The loss is reducing to a much lower value and then stabilizes which means that the model is learning something from the data, thus it is not underfitting. The model is not overfitting since the training loss is not much lower than the validation loss on the last epoch.

Experiments with the MLP led to the layout presented in Table 5.3. Additional configurations such as dropout, batch normalization, fewer, smaller, more and larger layers were tested but all such changes reduced the validation accuracy. Most of such measures are intended to reduce the chances of overfitting, but the time invariant datasets are not prone to the problem of overfitting. This will be discussed more in the discussion. The layout is presented in Table 5.3.

Layer	Structure	Activation function	Size
1	FC	ReLU	2
2	FC	ReLU	64
3	FC	ReLU	256
4	FC	ReLU	512
5	FC	ReLU	512
5	FC	ReLU	1024
6	FC	ReLU	2048
7	FC	ReLU	4096
8	FC	ReLU	2048
9	FC	ReLU	1024
10	FC	ReLU	512
11	FC	ReLU	256
12	FC	None	15

Table 5.3: The neural network has a fully connected layout with ReLU activation functions on each layer except the last one. Its optimization algorithm is Adam and the loss function is cross entropy loss, as they are popular for these kinds of problems and gave good results.

As for the experiments with the best DQN and DDQN network layout for the target and policy network, the network layout is highlighted in Table 5.4. The network is smaller than the optimal network for the MLP due to differences in training times. Along with different network layouts, three different sizes for the memory replay buffer were tested: 10000, 20000 and 30000. The best size was 20000.

Layer	Structure	Activation function	Size
Layer 1	FC	ReLU	5
Layer 2	FC	ReLU	128
Layer 3	FC	ReLU	128
Layer 4	FC	None	15

Table 5.4: The network layout for the policy network and the target network in the DQN algorithm and DDQN algorithm.

5.3 Models

This section shows the results of testing the models on the datasets. The metric used for accuracy in the majority of the graphs are the top-k accuracy metric. Illustrations of the predicted beam pairs by the best performing model, and the subset of wrong predictions is shown for the UMi dataset. As baseline, a random beam pair selection method was used which acts like a dummy model without intelligent selection.

5.3.1 Time Invariant

The random forest was selected to produce Figure 5.7 illustrating the outcome of predicting beam pairs with the model compared to the target outcome. The k value is set to 1 so only one candidate beam is allowed. Since it only made a few errors, they are visualized separately in Figure 5.8 for clarity. Observing the figure shows that there is no clear pattern on the incorrect predictions. These beam plots are omitted for the other scenarios since they show no additional patterns. By looking at the target beams in Figures 5.2-5.4 and their corresponding top-1 accuracy, the general idea of the plots is the same.

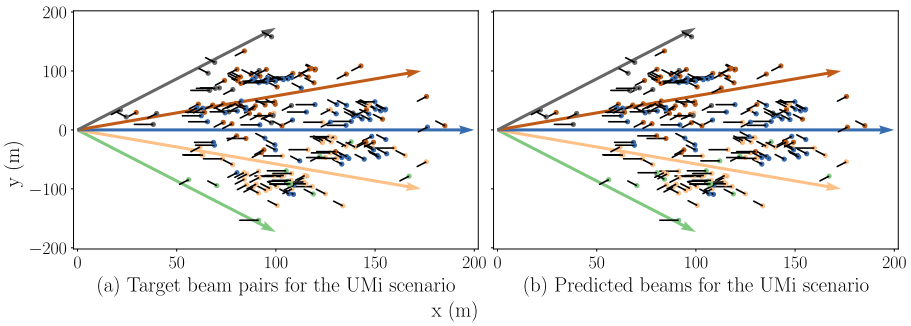


Figure 5.7: Target and predicted beam pairs for the UMi scenario. (a) shows the beam pairs that are correct and (b) shows the predictions of the random forest. A subset of 200 samples from the test set are visualized.

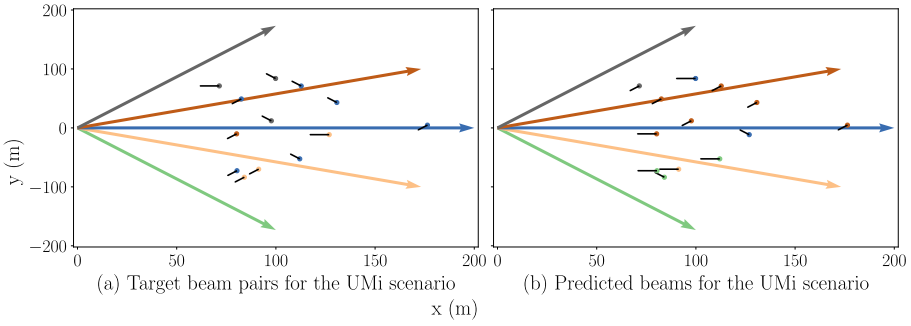


Figure 5.8: (a) shows the target labels for the incorrectly predicted beams and (b) shows the incorrect predictions by the random forest. The samples are the incorrect predictions from the subset of 200 samples above.

In Figure 5.9, the top-k accuracy for the UMi dataset is shown for all time invariant models. All models except SVM performed very well with a top-k accuracy of over 80%. The SVM was lagging behind, and only managed to achieve a top-k accuracy above 80% at $k = 6$. Also, the MLP is noticeably worse than the decision tree based models. The top-k accuracies for the other scenarios are illustrated in Figures 5.10 and 5.11. They showcase the same patterns except with lower accuracies.

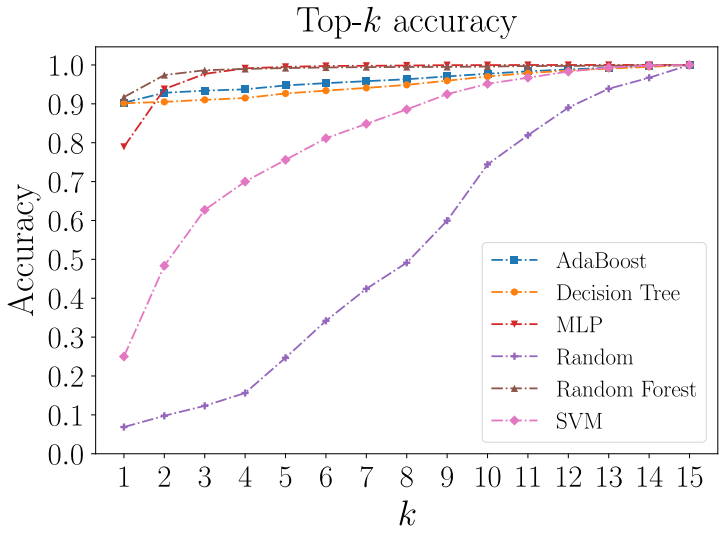


Figure 5.9: Top-k accuracy for the UMi scenario with an explicit city layout. On the x-axis is the number of candidate beams selected by the model. On the y-axis is the probability that the optimal beam is among the k selected candidate beams. Random is used as baseline.

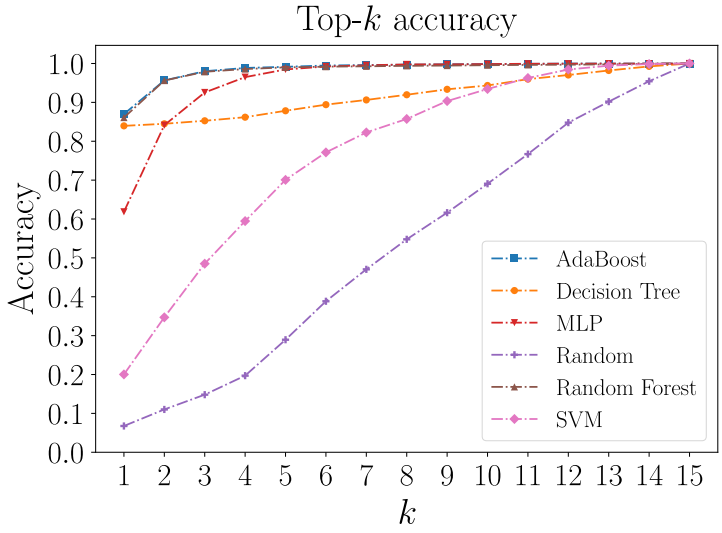


Figure 5.10: Top-k accuracy for the UMa scenario with an explicit city layout. On the x-axis is the number of candidate beams selected by the model. On the y-axis is the probability that the optimal beam is among the k selected candidate beams. Random is used as baseline.

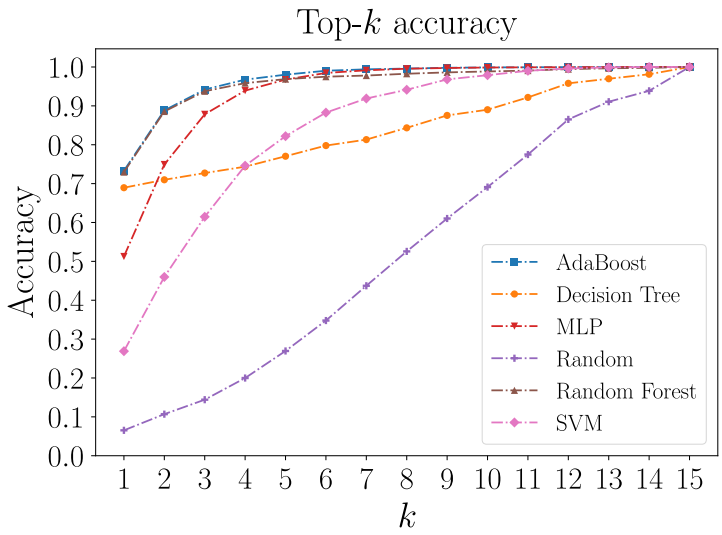


Figure 5.11: Top k-accuracy for the time variant RMa scenario. On the x-axis is the number of candidate beams selected by the model. On the y-axis is the probability that the optimal beam is among the k selected candidate beams. Random is used as baseline.

5.3.2 Time Variant

Figure 5.12 shows the moving average result of the best runs with a window size of 1000 for the RL algorithms Q-learning, DQN and DDQN. Clearly, the Q-learning algorithm outperforms the DQN and DDQN algorithms for this environment with around 0.2 more in average reward during the final iterations of the training phase. Table 5.5 also confirms that the Q-learning algorithm in fact has around 0.2 more in average reward than the DQN and DDQN algorithms during the test phase, that is, when the environment used the test dataset. The Q-learning algorithm has a stable increase throughout the training process that plateaus just below 0.6 in average reward. The stability of the DQN and DDQN algorithms is a bit more chaotic, but seems to converge to around 0.3. Interestingly, both algorithms increase in reward until 200 000 iterations, to drop almost back to the initial reward performance, to rapidly gain a lot of performance at around 400 000 iterations.

Table 5.5 also shows the average episodic reward. The difference being that the average reward is the mean reward over all iterations in the test phase while the average episodic reward is the mean reward for all finished episodes during the test phase. Obviously, these are directly correlated, but it is interesting to highlight that all of the algorithms get above 6.98 in average episodic reward. The implication is that the algorithms on average manage to correctly utilize the correct beam in 7 of out 20 time steps. The reward starts below zero at around -0.25 because of the penalty described in the method chapter. That is, if the difference between the current beam and each newly chosen beam is greater than 2, the penalty is induced.

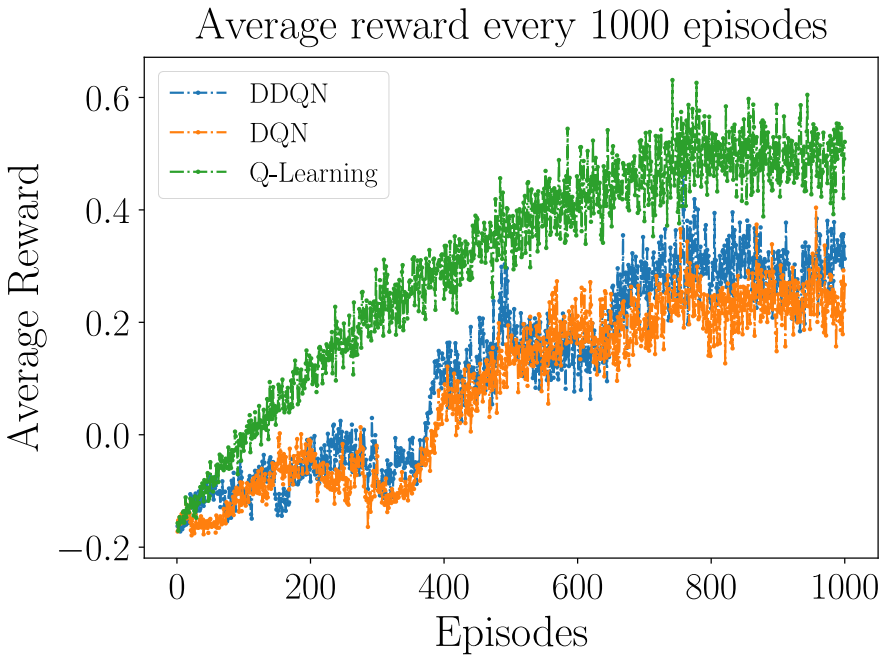


Figure 5.12: The moving average reward for one million iterations with an average sliding window of 1000 episodes.

Model	Avg reward	Avg episodic reward
Q-learning	0.61	10.91
DQN	0.41	7.32
DDQN	0.39	6.98

Table 5.5: The average reward and average episodic reward using the test dataset.

To compare the models against the Scikit-learn models and PyTorch models, the reward can roughly be converted to an accuracy estimate since the environment is kind of based upon the classification scenario. By viewing the reward from this perspective, it functions as an upper bound for the accuracy. Hence, the accuracy of these algorithms lay around but not above 0.4 for DQN and DDQN while 0.6 for Q-learning. The maximum reward and highest possible accuracy would be achieved when the correct beam is chosen for every time step, which would result in an average reward of 1.0 and an upper accuracy of 1.0.

5.3.3 Sliding Window

The sliding window preprocessing of the RMa dataset gave overall better performance than using the RMa and the time variant version of RMa alone as seen in Figure 5.13. The sliding window preprocessing step presented in Figure 5.13 used a window size of 2, meaning that two positions and one beam pair were used to predict a beam pair.

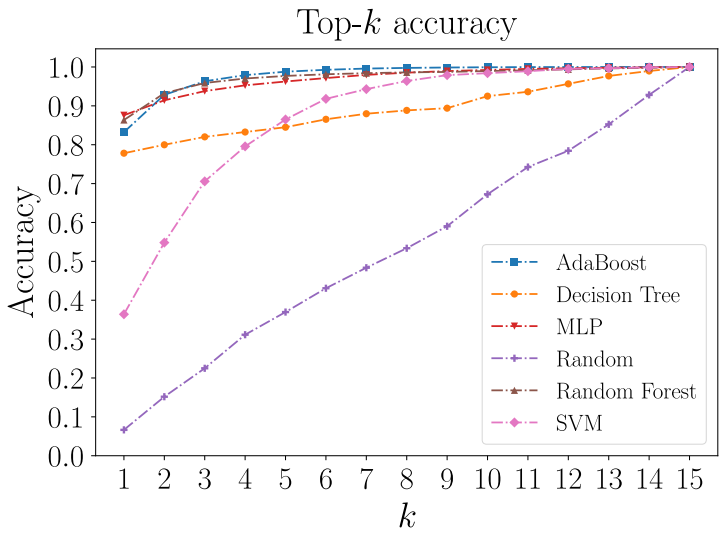


Figure 5.13: Top-k accuracy for the time variant RMa scenario using a sliding window of size 2.

By looking at Figure 5.14 and comparing it against Figure 5.11, the performance increased by around 15 percentage points for the best models random forest and AdaBoost. However, the comparison is not completely fair as k in Figure 5.13 represents two positions and one label while k in Figure 5.11 represents one position because of the preprocessing steps of the datasets.

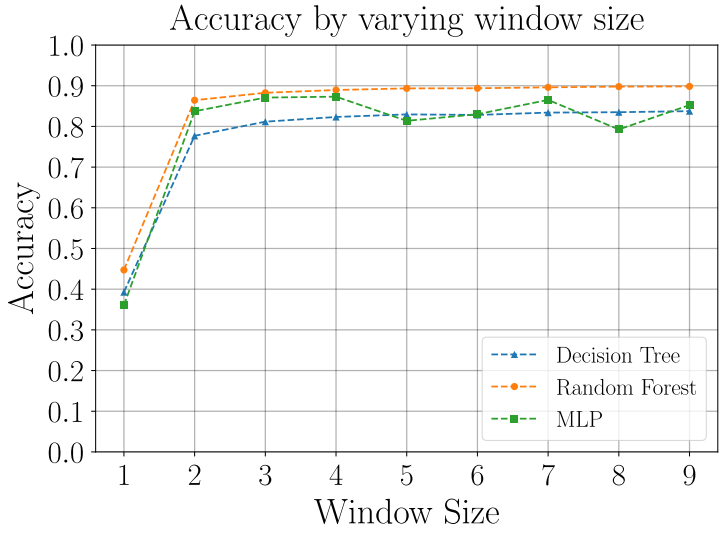


Figure 5.14: The performance of random forest, decision tree and MLP for the RMa dataset with different window sizes.

In Figure 5.14, the decision tree, random forest and MLP were trained on the sliding window pre-processed dataset using different sizes for the window to get a grasp of the improvements by adding more context. The random forest model performed best

overall and slowly converged to 90% accuracy for a total of nine different window sizes. The decision tree and MLP settled for a slightly lower accuracy, after achieving almost the same performance as the random forest. Thus, the performance degraded by adding a bigger window size, probably due to the model having the same layout structure for all window sizes.

6

Discussion

The following section discusses the result. It starts with the time invariant datasets, followed by the sliding window dataset, and finishes with the time variant dataset.

6.1 Time Invariant

In the time invariant dataset, there are city layouts for UMi and UMa, and a uniform layout for RMa. Even though there are preset scenarios in each building of the cities, many different beam pairs are optimal in a given building. That causes the data to have great detail, and subsequently the models need to be able to capture more detail. The reason for this is partly that the frequency is very high so the signal is greatly affected by objects in the environment. There is a risk that much overhead is needed to manage the beams if the allocation scheme is highly complex. Using this unprocessed data without transformation would perhaps be less efficient than smoothing it out a bit.

Another factor that increases the detail is that the simulation includes UE beam sweep which many other papers do not. UE beam sweeps lead to more complexity in the data as UEs that are close together can yield further different optimal beam pairs. Thus, a larger model is needed to represent this increased complexity. For example, there are data points in the buildings in Figure 5.2 that have different optimal beams even though they are mostly overlapping, and some that are surrounded.

Also in the time invariant datasets, the possible positions in the sector are associated with their own unique target label. That is because the random number generator seed is constant between the measurements, so conditions never change on the exact same position. Thus it will always have the same optimal beam pair. In more realistic conditions, the optimal beam pair may change over time due to changes in the environment. So the time invariant dataset works for an initial understanding of the situation, not as a long term full solution. One way to mitigate the constant conditions is to introduce noise in the data. For example, the UE coordinates could be noisy and result in some mistakes by the models as done in [20].

The reason why the simpler models perform better than the MLP is reasonable. The decision tree, random forest and AdaBoost are built of decision trees which split the features, namely the x and y positions into buckets. The more depth and

more estimators used, the more precise those buckets will be, until it is as precise as the QuaDRiGa simulated data itself. Seemingly, the decision tree structure is slightly better than the MLP at representing that detailed distribution of data, and much better than the SVM. The SVM performs poorly, probably because it is implemented as a linear SVM whereas the data is very nonlinear. Since the data is very interspersed, any kind of struggle with complex kernels is likely worse than using random forest or MLP.

Similar patterns in performance of the supervised models are seen in papers [11, 20] from the literature review. The random forest and MLP are best in both papers and SVM is significantly worse. In paper [20], AdaBoost and SVM were tested but their results were not included because they were unsatisfactory. Most sources say that AdaBoost is meant to use very shallow trees, perhaps even tree stumps meaning 1 depth, but around 1000 of them. That configuration was tested as well but it gave very poor results and was not included. Instead, AdaBoost was presented with the same parameter tuning as the random forest, but then AdaBoost and random forest are quite similar. This is why they gave similar results.

A difference from the referenced papers is that this simulation has receiver side beam sweep. As explained above it creates more detail in the data and thus more complex models are needed. Hence, this simulation used random forest with 20 estimators instead of 10 as in [20], and the MLP is significantly larger. The MLP in [20] had 5 hidden layers whereas this simulation required 10 hidden layers with high neuron counts to maximize accuracy.

6.2 Sliding Window

The sliding window approach gives better performance compared to the corresponding time invariant dataset for $K > 1$. As discussed in the result section, the comparison does not seem fair as Figure 5.13 contains more information about the dataset in the x-axis. However, since the models trained on the sliding window dataset are expected to keep the history of recently tracked beam pairs, the comparison is not unreasonable.

The sliding window models are assumed to have moved to the next time step, and thus, have access to two position data points as well as one target beam pair data point to make an informative judgement. Hence, the comparison is valid and reflects that the models using the sliding window dataset, under the assumption that history has been stored, are able to predict the optimal beam pair with better performance than using the time invariant dataset by almost 15 percentage points. By utilizing both models, there is a potential to increase the overall performance of the system by switching over to a model trained on the sliding window dataset after establishing a link using a model trained on the time invariant dataset.

6.3 Time Variant

There are some differences in the data when using moving UEs, visible by comparing Figures 5.4 and 5.5. Specifically, optimal beam pairs do not change much along tracks. These tracks use a technique called drifting from the QuaDRiGa implementation. Drifting alters the path-delays, powers and angles when the UEs move, but they originate from their initial values at the starting position. This results in some inconsistencies when tracks overlap, because at the point of overlap they will not necessarily have the same received power. This problem is somewhat mitigated by the usage of two consecutive positions in the RL environment states, thus allowing a separation between tracks of different paths.

Another point about the rural dataset is that the antenna array on the receiving side is unrealistically large to move. Calculating the size of the antenna array using the frequency of 0.5 GHz, the antenna spacing of half the wavelength and the four antenna elements along each axis, it turns out to be about two square meters. This issue could be mitigated by not using beamforming in the UE of the rural scenario. Omnidirectional UE would cause the data to be less complex and perhaps easier for the RL to learn. The expected impact on the resulting comparison of ML models is low because the patterns that the ML models learn are the same because the general structure of the data is the same, just less complex. There are also no references to absolute real world values in the results. Everything is about reducing the search space by a proportion.

The time variant models, Q-learning, DQN and DDQN have proven to perform poorly in comparison with the other models. Ideally, the RL algorithms should perform similar to the sliding window models. While the causes remain somewhat unclear, it is clear that Q-learning achieves the best performance among the models. As the optimal beam pairs along each track on average are 2.6, it is not surprising that Q-learning with its discretization steps performs quite well. The number of bins after the discretization process is 100 where each bin occupy an area of $190 \times 320 = 60800\,\mathrm{m}^2$. It is safe to say that almost all tracks fit within a certain bin and as the average optimal beam pairs along each track are quite low, the state representing each bin can quite accurately describe the optimal beam pairs.

Increasing the number of bins would increase the precision of the grid, allowing the Q-learning algorithm to be more precise along each track by dividing the track into different bins more frequently. However, as stated shortly in the description of the environment, see Section 4.4, it is not feasible to increase the number of bins much further due to memory constraints. This shortcoming of the Q-learning algorithm is mitigated by the DQN and DDQN algorithms. In the description of the environment, it is also mentioned that the binning process was a bit naive. That it gave rise to a lot of unnecessary states as some bins were placed outside the circle sector where no receivers could have moved. These states are not a problem for the Q-learning algorithm because the algorithm only updates the states it can reach. However, the states occupy unnecessary memory.

The DQN and DDQN algorithms have very similar performance, and it is hard to tell them apart by just looking at the cumulative average reward shown in Figure 5.12. The problem with these approaches are most likely that they have not been exposed to enough data samples, and would benefit with many more training iterations. However, these models were also the slowest to train, and hence, iterate upon. For reference, training the models on 1.5 million iterations took about 16 hours which does not seem much with a working pipeline, but builds up because of data generation, training of other models and overall technical debt of maintaining the pipeline. This is the reason to why a bigger network layout was not considered for the DQN and DDQN even though the MLP empirically showed that a bigger network layout increased the performance of the model.

The models are also very dependent on the initial conditions of the environment and random states, and while the best performance is visualized in Figure 5.12, the average performance when tuning the models were a bit lower than that even including very similar tuning parameter configurations. In theory, the DQN and DDQN should perform better than Q-learning because they are not limited by the discretization process of the continuous state. Again, with more training the models presumably would have scored better.

Comparing the results of our RL approaches, with the methods described in the literature review, is not trivial as the environments and underlying data differ a lot even though some ideas have been integrated into our environment. In [37], more of a regression perspective was incorporated into the environment with the goal of estimating the optimal angle difference needed to align the beams. This presumably makes the environment better defined as it constrains the beam selection phase in the action space instead of inducing a penalty for rapidly fluctuating beam pair selections.

In [36], position data was not used at all to predict the next serving beam. Instead, the Q-table is defined as the currently serving beam and a switch to another beam as the state and action respectively. However, one potential problem with this approach is that a serving beam always leads to the same beam switch regardless of the position of the UE. UE located near the BS would have the same beam switch as UE located far away from the BS given that they have the same serving beam. This might lead to errors due to blocking along the way for the UE farther away, which would result in another beam switch. To mitigate this problem, the position data was also incorporated into the state.

Another aspect of the definition of the environment is concept of the “hybrid” task. In a real world setting it does not make much sense for the problem to be formulated as a episodic task, as serving the correct beam is an ongoing process. However, it is easier to generate many receivers moving short distances than one receiver moving and bouncing along the circle sector bounds in order to cover as much of the area as possible in different directions. It may be more appropriate to view the shorter distances as episodic.

6.4 Ethics

While the specific problem studied in this paper, the problem of beam management, may not have much ethical implication on its own, it is a key factor in the development of 5G in general. 5G is a key enabler for IoT devices. By incorporating cloud technologies 5G aims to make everything smart and connected [44]. However, that gives rise to privacy concerns. With technological advancements in 5G, systems are capable of capturing, processing and analyzing huge amounts of data. Being deployed in critical settings such as healthcare makes it important to protect personal data from an adversary. Breaches of privacy might easily arise due to lack of security in especially IoT devices as they often have outdated firmware susceptible for attacks [45, 46]. To protect future users from misuses of personal data, rigid frameworks must be the center of focus when implementing the technology [47].

A big part of the world population has still no internet as of today [48]. These people are cut off from a myriad of opportunities. The acceleration of even faster communication technologies like 5G and 5G being rather expensive to deploy increases the gap between between users and non-users of the technology often referred to as the digital divide [48, 49]. United Nations (UN), a division for sustainable developments, has made one of its 17 goals to ensure a resilient infrastructure that foster innovation and sustainable innovation. In their report from 2022 [50], they acknowledge that most of the world's population are covered by mobile-broadband signals, but universal and affordable access have still not been met in the least developed countries. While 5G has the potential to reduce the digital divide and provide social and environmental benefits, efforts need to be made to include everyone; not just the rich countries [49, 51].

7

Conclusion

In conclusion, a number of different machine learning approaches are presented to the problem of beam management. Each method has been applied to a number of different scenarios provided in the specification by 3GPP including UMi, UMa and RMa. The scenarios have been generated using a stochastic channel model called QuaDRiGa. The RMa scenario has further been composed into two datasets, the time invariant dataset and the time variant dataset.

Two machine learning paradigms are studied: SL and RL. Of the studied models, decision tree, random forest, AdaBoost, SVM and MLP belong to the SL paradigm while Q-learning, DQN and DDQN belong to the RL paradigm. All models successfully reduce the search space for the beam management problem. However, the models in the SL paradigm perform much better than the models in the RL paradigm, perhaps due to a poorly defined environment.

The best models are random forest and AdaBoost which achieves an accuracy of 90% for the UMi scenario. The worst performing supervised model was the SVM which is reasonable when inspecting its function. The performance of the other scenarios are similar, but with slightly lower accuracy. The implication of having 90% accuracy means that on average a model would select the optimal beam pair 9 out of 10 times without searching at all.

When making the models of the SL paradigm compatible with the time varying RMa scenario as in the sliding window preprocessing step, the accuracy further increases with a window size greater than or equal to two. This suggests that models trained on the static and dynamic datasets can be combined to utilize an overall better performance gain.

7.1 Future work

A future comparison of ML solutions can be carried out in a more holistic 5G simulator to provide more applicable results. In the QuaDRiGa simulation that was carried out in this thesis, only the received power was used. However, in reality there are reference signals that are used to more accurately measure the quality of a connection. Higher layer logic such as load balancing and reduced beam switching that are necessary for higher quality of service is also useful to incorporate.

This thesis focused on the beam selection part of beam management, so it does not provide a full solution. Some combination of techniques that provide full beam management function is also future work. During IA, one kind of method can be used that performs optimally when there is no historical data. Then, after a connection is established it can possibly be maintained by another method that takes into account the UE's live movements. If the connection breaks, recovery can be accomplished using yet another method, which incorporates the historical data during the previous connection with the particular UE.

8

References

[1] E. Dahlman, S. Parkvall, and J. Skold, *5G NR: The Next Generation Wireless Access Technology*. USA: Academic Press, Inc., 2st ed., 2018.

[2] F. W. Vook, A. Ghosh, E. Diarte, and M. Murphy, "5G New Radio: Overview and Performance," in *2018 52nd Asilomar Conference on Signals, Systems, and Computers*, pp. 1247–1251, Oct. 2018.

[3] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. van den Driessche, T. Graepel, and D. Hassabis, "Mastering the game of Go without human knowledge," *Nature*, vol. 550, pp. 354–359, Oct. 2017.

[4] J. L. Gustafson, "Moore's Law," in *Encyclopedia of Parallel Computing* (D. Padua, ed.), pp. 1177–1184, Boston, MA: Springer US, 2011.

[5] S. Zhu, T. Voigt, J. Ko, and F. Rahimian, "On-device Training: A First Overview on Existing Systems," May 2023.

[6] S. Sagiroglu and D. Sinanc, "Big data: A review," in *2013 International Conference on Collaboration Technologies and Systems (CTS)*, pp. 42–47, IEEE, May 2013.

[7] L. Zhou, S. Pan, J. Wang, and A. V. Vasilakos, "Machine learning on big data: Opportunities and challenges," *Neurocomputing*, vol. 237, pp. 350–361, May 2017.

[8] V. N. Tsakalidou, P. Mitsou, and G. A. Papakostas, "Machine learning for cloud resources management – An overview," Jan. 2021.

[9] C. Cox, *Introduction to 5G - The New Radio, 5G Network and Beyond*. John Wiley & Sons.

[10] M. R. Minar and J. Naher, "Recent Advances in Deep Learning: An Overview," 2018.

[11] A. Klautau, P. Batista, N. Gonzalez-Prelcic, Y. Wang, and R. W. Heath, "5G MIMO Data for Machine Learning: Application to Beam-Selection Using Deep Learning," in *2018 Information Theory and Applications Workshop (ITA)*, pp. 1–9, IEEE, Feb. 2018.

[12] J. Turkka, P. Kela, and M. Costa, "On the spatial consistency of stochastic and map-based 5G channel models," in *2016 IEEE Conference on Standards for Communications and Networking (CSCN)*, pp. 1–7, Oct. 2016.

[13] S. Jaeckel, L. Raschkowski, L. Thiele, E. Eberlein, T. Heyn, and F. Burkhardt, "QuaDRiGa - Quasi Deterministic Radio Channel Generator, User Manual and Documentation,"

[14] S. Ju, O. Kanhere, Y. Xing, and T. Rappaport, "A millimeter-wave channel simulator nyusim with spatial consistency and human blockage," pp. 1–6, 12 2019.

[15] V. Desai, L. Krzymien, P. Sartori, W. Xiao, A. Soong, and A. Alkhateeb, "Initial beamforming for mmWave communications," in *2014 48th Asilomar Conference on Signals, Systems and Computers*, pp. 1926–1930, Nov. 2014.

[16] A. Capone, I. Filippini, and V. Sciancalepore, "Context Information for Fast Cell Discovery in mm-wave 5G Networks," in *Proceedings of European Wireless 2015; 21th European Wireless Conference*, pp. 1–6, May 2015.

[17] 3GPP, "5G; NR; Physical channels and modulation," Technical Specification (TS) 38.211, 3rd Generation Partnership Project (3GPP), January 2023. Version 17.4.0.

[18] M. Martínez-Ramón, A. Gupta, J. L. Rojo-Álvarez, C. Christodoulou, and K. Cools, *Machine Learning Applications in Electromagnetics and Antenna Array Processing*. Artech House, 2021.

[19] Abbasi Qammer H., Jilani Syeda F., Alomainy Akram, and Imran Muhammed A., "10.2.1.3 Hybrid or Analog/Digital Beamformers," in *Antennas and Propagation for 5G and Beyond*, Institution of Engineering and Technology.

[20] Y. Heng and J. G. Andrews, "Machine Learning-Assisted Beam Alignment for mmWave Systems," p. 6.

[21] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*. Upper Saddle River: Pearson, 3rd edition ed., Dec. 2009.

[22] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, Mass: A Bradford Book, second edition ed., Mar. 1998.

[23] J. R. Quinlan, "Induction of decision trees," *Mach Learn*, vol. 1, pp. 81–106, Mar. 1986.

[24] L. Breiman, "Random Forests," *Machine Learning*, vol. 45, pp. 5–32, Oct. 2001.

[25] C. Cortes and V. Vapnik, "Support-vector networks," *Mach Learn*, vol. 20, pp. 273–297, Sept. 1995.

[26] F. Rosenblatt, "The perceptron: A probabilistic model for information storage and organization in the brain.," *Psychological Review*, vol. 65, no. 6, pp. 386–408, 1958.

[27] Hecht-Nielsen, "Theory of the backpropagation neural network," in *International 1989 Joint Conference on Neural Networks*, pp. 593–605 vol.1, 1989.

[28] Z. Zhang and M. R. Sabuncu, "Generalized Cross Entropy Loss for Training Deep Neural Networks with Noisy Labels," Nov. 2018.

[29] J. Bridle, "Training stochastic model recognition algorithms as networks can lead to maximum mutual information estimation of parameters," in *Advances in Neural Information Processing Systems* (D. Touretzky, ed.), vol. 2, Morgan-Kaufmann, 1989.

[30] C.-Y. Fu, M. Shvets, and A. C. Berg, "RetinaMask: Learning to predict masks improves state-of-the-art single-shot detection for free," Jan. 2019.

[31] C. J. C. H. Watkins and P. Dayan, "Q-learning," *Mach Learn*, vol. 8, pp. 279–292, May 1992.

[32] M. Rawson and R. Balan, "Convergence Guarantees for Deep Epsilon Greedy Policy Learning," Jan. 2022.

[33] L.-J. Lin, "Self-improving reactive agents based on reinforcement learning, planning and teaching," *Mach Learn*, vol. 8, pp. 293–321, May 1992.

[34] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, pp. 529–533, Feb. 2015.

[35] H. van Hasselt, A. Guez, and D. Silver, "Deep Reinforcement Learning with Double Q-learning," Dec. 2015.

[36] J. Bill and G. Fahlén, "Machine Learning Technique for Beam Management in 5G NR RAN at mmWave Frequencies," 2020.

[37] L. Wang, B. Ai, Y. Niu, M. Gao, and Z. Zhong, "Adaptive Beam Alignment Based on Deep Reinforcement Learning for High Speed Railways," in *2022 IEEE 95th Vehicular Technology Conference: (VTC2022-Spring)*, pp. 1–6, IEEE, June 2022.

[38] 3GPP, "5G; Study on channel model for frequencies from 0.5 to 100 GHz," Technical Specification (TS) 38.901, 3rd Generation Partnership Project (3GPP), April 2022. Version 17.0.0.

[39] H. S. Hota, R. Handa, and A. K. Shrivas, "Time Series Data Prediction Using Sliding Window Based RBF Neural Network,"

[40] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[41] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*, pp. 8024–8035, Curran Associates, Inc., 2019.

[42] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," July 2019.

[43] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," Jan. 2017.

[44] T. Muluk, "Importance of 5G for Developing Countries,"

[45] E. Bertino, "Privacy in the Era of 5G, IoT, Big Data and Machine Learning," in *2020 Second IEEE International Conference on Trust, Privacy and Security in Intelligent Systems and Applications (TPS-ISA)*, pp. 134–137, Oct. 2020.

[46] J. Cook, S. U. Rehman, and M. A. Khan, "Security and Privacy for Low Power IoT Devices on 5G and Beyond Networks: Challenges and Future Directions," *IEEE Access*, vol. 11, pp. 39295–39317, 2023.

[47] V. B. Viswanathan and K. A. Nagarajan, "Building Privacy First 5G Networks," in *2022 IEEE International Conference on Electronics, Computing and Communication Technologies (CONECCT)*, pp. 1–5, July 2022.

[48] Qualcomm, "5G and the digital divide," April 2021.
[49] M. G. Macra, "Will the 5G aggravate the digital hole or separation?." https://www.sofrecom.com/en/news-insights/will-5g-exacerbate-the-digital-gapdivide.html. (accessed May 30, 2023).
[50] United Nations, "The Sustainable Development Goals Report," 2022.
[51] M. M. Mim, M. Karmokar, K. M. Tarikul Imam, M.-U.-S. Chowdhury, and D. Kabir, "5G Network Implementation in Least Developing Countries: Possibility, Barriers and Future Opportunities," in *2022 IEEE 3rd Global Conference for Advancement in Technology (GCAT)*, pp. 1–9, Oct. 2022.

DEPARTMENT OF ELECTRICAL ENGINEERING
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY