UNIVERSITÀ DEGLI STUDI DI PADOVA

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

UNIVERSITY OF PADOVA

DEPARTMENT OF INFORMATION ENGINEERING

Ph.D. Course in Information Engineering

Intelligent Control Systems and Machine Learning Approaches for Particle Accelerators

Supervisor
Prof. Gian Antonio Susto

Ph.D. Student
Davide Marcato

Academic Year 2022/2023

September 2023

Abstract

Particle accelerators are used all around the world for fundamental physics research, medical diagnosis and industrial applications. These can be extremely complex machines, with control systems composed of thousands of sensors and actuators producing an enormous amount of data. By exploiting this data, it's possible to reach new levels of performance, improve the uptime of the accelerator and reduce the effort required to setup, control and maintain it. This thesis focuses on the application of Machine Learning and Deep Learning models to the field of particle accelerator control systems. Anomaly Detection is applied to the task of fault prediction, both using classical Machine Learning algorithms and Deep Learning time-series forecasting models. By discovering anomalies in the trends of the process variables we can predict the insurgence of fault conditions, or the breakage of a critical component, thus allowing to intervene in time to avoid it. Reinforcement Learning is applied to the task of beam emittance optimization, with the aim of training a model which is able to automatically tune the beam transport parameters online to reach the optimal beam dynamics. These methods are validated on the particle accelerators at the INFN National Laboratories of Legnaro, Italy, but can be extended to other facilities with similar challenges. Finally, we explore data-driven Dynamic Sampling strategies to optimize metrology plans for quality control in industrial manufacturing processes.

Contents

Scientific Publications

List of publications on international journals

- G. A. Susto, D. Marcato, M. Maggipinto, M. Donà, and S. McLoone. On temporally bounded spatial dynamic sampling for metrology optimization. *submitted to IEEE Transactions on Automation Science and Engineering*, 2023

List of publications on conference proceedings

- D. Marcato, D. Bortolato, V. Martinelli, G. Savarese, and G. A. Susto. Time-series deep learning anomaly detection for particle accelerators. In *Proceeding of the 22nd World Congress of the International Federation of Automatic Control (IFAC'23)*, 2023

- D. Marcato et al. Demonstration of beam emittance optimization using reinforcement learning. In *Proc. IPAC'23*, number 14 in IPAC'23 - 14th International Particle Accelerator Conference, pages 2838–2841. JACoW Publishing, Geneva, Switzerland, 05 2023

- D. Marcato, G. Arena, D. Bortolato, F. Gelain, V. Martinelli, E. Munaron, M. Roetta, G. Savarese, and G. A. Susto. Machine learning-based anomaly detection for particle accelerators. In *2021 IEEE Conference on Control Technology and Applications (CCTA)*, pages 240–246, 2021

Other publications (books, book chapters, patents)

- T. Barbariol, F. D. Chiara, D. Marcato, and G. A. Susto. A review of tree-based approaches for anomaly detection. In *Control Charts and Machine Learning for Anomaly Detection in Manufacturing*, pages 149–185. Springer, 2022

Other publications not included in the PhD thesis

These are publications developed during the PhD program that are not directly related to the main reaserch activity presented in this thesis or where my contribution was minor.

- D. Marcato et al. Upgrade of the alpi low and medium beta rf control system. In *Proc. IPAC'23*, number 14 in IPAC'23 - 14th International Particle Accelerator Conference, pages 4154–4157. JACoW Publishing, Geneva, Switzerland, 05 2023

- D. Marcato, G. Arena, M. Bellato, D. Bortolato, F. Gelain, G. Lilli, V. Martinelli, E. Munaron, M. Roetta, and G. Savarese. Pysmlib: A Python Finite State Machine Library for EPICS. *JACoW*, ICALEPCS2021:TUBL05, 2022

- G. Savarese et al. First installation of the upgraded vacuum control system for alpi accelerator. In *Proc. IPAC'23*, number 14 in IPAC'23 - 14th International Particle Accelerator Conference, pages 840–843. JACoW Publishing, Geneva, Switzerland, 05 2023

- L. Bellan et al. New techniques for the lnl superconductive linac alpi beam dynamics simulations and commissioning. In *Proc. IPAC'23*, number 14 in IPAC'23 - 14th International Particle Accelerator Conference, pages 1289–1292. JACoW Publishing, Geneva, Switzerland, 05 2023

- E. Fagotti et al. Upgrade of the heavy ion accelerator complex at infn-lnl. In *Proc. IPAC'23*, number 14 in IPAC'23 - 14th International Particle Accelerator Conference, pages 2169–2172. JACoW Publishing, Geneva, Switzerland, 05 2023

- L. de Ruvo, M. Allegrini, D. Benini, et al. Functional architecture of spes safety system. In *Proc. IPAC'23*, number 14 in IPAC'23 - 14th International Particle Accelerator Conference, pages 4682–4684. JACoW Publishing, Geneva, Switzerland, 05 2023

- V. Martinelli, L. Bellan, D. Bortolato, M. Comunian, E. Fagotti, P. Francescon, A. Galatà, D. Marcato, and G. Savarese. BOLINA, a Suite for High Level Beam Optimization: First Experimental Results on the Adige Injection Beamline of SPES. In *Proc. IPAC'22*, number 13 in International Particle Accelerator Conference, pages 933–936. JACoW Publishing, Geneva, Switzerland, 07 2022

- G. Savarese, L. Antoniazzi, D. Bortolato, A. Conte, F. Gelain, D. Marcato, and C. Roncolato. Vacuum Control System Upgrade for ALPI Accelerator. In *Proc. IPAC'22*, number 13 in International Particle Accelerator Conference, pages 744–746. JACoW Publishing, Geneva, Switzerland, 07 2022

- G. Savarese, G. Arena, D. Bortolato, F. Gelain, D. Marcato, V. Martinelli, E. Munaron, and M. Roetta. Design and Development of the New Diagnostics Control System for the SPES Project at INFN-LNL. In *Proc. ICALEPCS'21*, number 18 in

International Conference on Accelerator and Large Experimental Physics Control Systems, pages 428–432. JACoW Publishing, Geneva, Switzerland, 03 2022

- A. Galatà et al. First beams from the 1+ source of the adige injector for the spes project. *Journal of Physics: Conference Series*, 2244(1):012069, apr 2022

Chapter 1

Introduction

Particle accelerators host thousands of sensors and actuators with complex interactions between different subsystems; multiple control loops are required to maintain the correct operating parameters and to reach optimal ion beam properties.

A single malfunction of a component in the beam transport or in the beam acceleration subsystems is enough to disrupt the beam jeopardizing the experiment and blocking the whole accelerator. In fact, if a single transport or accelerating element is not working properly, the beam will deviate from its ideal trajectory and will be lost. This means that the control system must be able to detect and react to any malfunction in a timely manner, to avoid the loss of the beam and the consequent downtime of the accelerator. Given the cost of the beam time, this is a crucial requirement.

Beam time costs are hard to calculate but easily reach important amounts. In fact the machine is very expensive to build in the first place, being composed of high technology components and requiring a large number of highly specialized technicians and engineers. Furthermore, the facility requires constant maintenance and upgrades, in a context of experimental innovations that are often in the prototyping stage. Finally, the energy requirements associated with the operation of cryogenic lines or high current magnets can easily reach MW of power, even for a small scale, low energy accelerator as the ones in Legnaro. Thus the accelerator must be available as much as possible to maximize the scientific output.

Beam time cost is not the only reason to have a reliable control system. In fact, physics experiments often require a certain amount of beam energy, current, or low beam emittance to perform a certain measure. By increasing the performance of an accelerator it's possible to push its limits and to perform more complex experiments.

Another important factor to consider for the efficiency of a particle accelerator is the setup time. In fact, the accelerator is periodically powered off for maintenance and to save energy. When the accelerator is powered on again, it must be reconfigured to the desired operating parameters. This is a complex process which requires the intervention of many technicians and engineers, and can take days or weeks. The control system can help in this process by automating the configuration of the machine, thus reducing the setup time and increasing the efficiency of the accelerator. Ideally, a single routine should be able to configure the machine for any experiment, and the control system should be able to switch between different configurations in a fast and reliable way.

One of the most tedious tasks is the beam acceleration and transport phase. It requires to manually tune the beam transport elements one by one, to obtain the desired beam. For example, the quadrupoles must be tuned to focus the beam, while dipoles and steerers to center the beam on the optimal trajectory. Since the position, size and shape of the beam at a certain point depends on the setpoint of the transport elements preceding it on the beam line, this task is hard to parallelize and must be performed sequentially. Then again, during the acceleration phase all the cavities must be phased correctly, one by one.

Physics simulations are used extensively for this kind of tasks but they are not always accurate enough, especially in older accelerators where the real machine can differ from the simulation due to uncertainties on the beam line design and capabilities of the beam transport elements. Even a small error in the positioning of a triplet can have a large impact on the beam, so that the simulation results are not directly applicable to the machine. For these reasons, the simulation results are only used as a starting point, but they are fine tuned on the real machine. This is a very time consuming task, which could be automated by the control system. In fact, the control system could read the beam properties from the diagnostic elements and automatically tune the beam transport elements to optimize the beam. Similarly, the control system could read the beam properties and automatically tune the Radio Frequency (RF) cavities to accelerate the beam. For this reason, the research community is actively looking for new methods to automate these processes and reduce the time required to find the optimal configuration.

Unfortunately it's very hard to build an automatic system which is able to replace the intuition and knowledge of a trained technician. In fact, what may seem trivial to a human may become very hard to code. For example, the simple task of recognizing an increasing or decreasing variable may prove complex, as the signal may be noisy, the trend may be hidden by oscillations and the result depends on the time span observed as well as the resolution of the observation. Furthermore, the control system must be able to account for very different situations, such as different input beams, uncertainties on the positioning of the elements on the line and different operating conditions, such as external temperature or vacuum level, to the point that it becomes unfeasible to code all the possible scenarios.

Machine Learning (ML) and Deep Learning (DL) models have proven to be very effective in solving this kind of problems, displaying great results in many fields. They can learn high dimensional non-linear functions directly from the data, and can extrapolate to unseen situations. Reinforcement Learning (RL) instead, with its ability to learn optimal control strategies, has shown impressive results in decision-making processes and could prove very effective to tune the control system parameters based on online feedback. Furthermore, particle accelerators usually collect and record the trends of many Process Variables (PVs) from the thousands of sensors and actuators installed, meaning that a large dataset of raw signals is available or can be easily collected during a run of the accelerator. For this reason this thesis explores the possibility of using ML models to improve the performance of the control system of particle accelerators.

The thesis is organized as follows: chapter 2 introduces the reader to the world of particle accelerators, describing the accelerators at the Legnaro National Laboratories (LNL) in Legnaro, Italy and highlighting possible improvements to the control systems. Then an overview of the main concepts and algorithms of ML, DL and RL is presented in chapter 3, with a focus on the models used in this thesis. Chapter 4 offers an analysis of the literature in this field and tries to identify the main research areas and directions. Chapter 5 details two different works on Anomaly Detection (AD) for fault prediction, using both classical ML algorithms and DL time-series forecasting models. Instead, the work presented in chapter 6 focuses on the beam dynamics optimization problem and applies RL to the task of beam emittance minimization. Finally, chapter 7 address a slightly different topic, where data-driven strategies are employed to optimize the metrology plans for quality control in industrial manufacturing processes, following on the concept of augmenting the performance of a facility with ML tools. The thesis is concluded in chapter 8 with a summary of the results and a discussion on the future developments of this research.

Chapter 2

Particle Accelerators

2.1 Overview

Particle Accelerators are used all around the world to produce and accelerate ion beams for multiple applications. The main goal of a particle accelerator is to deliver a beam of particles to a target with a precise energy and with the required beam properties. They are extensively used in medicine both for diagnostics and therapy, in industry for material analysis and in physics research for nuclear physics experiments. Accelerators for physics research are usually very large, complex and expensive machines, composed of multiple subsystems, each one with its hundreds of high technology devices. The control system of these machines is responsible for the correct operation of all the subsystems, and thus it is crucial for the success of the experiments.

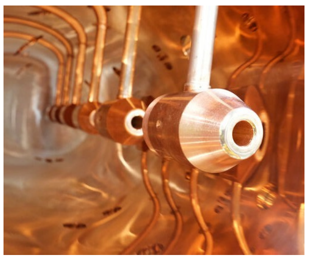
There are different types of accelerators, depending on the underlying technology used to accelerate the beam, as shown in figure 2.1. The most common ones are:

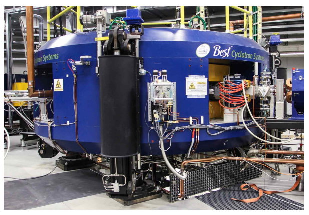
a) **Electrostatic accelerators**: they use static electric fields to accelerate the beam. They are the simplest and oldest kind of accelerators, but they are limited in the maximum energy they can deliver, as they require a high voltage terminal to accelerate the beam. This limits the maximum energy to a few MeV. Historically, Van de Graaff generators were largely used to produce such high voltage terminal. Now they are mostly used as injectors for other accelerators or for low energy experiments. Figure 2.1a shows the CN electrostatic accelerator at LNL [136].

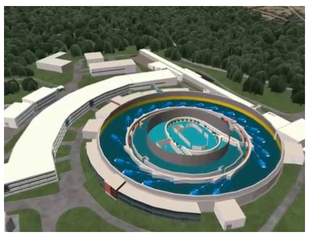
b) **Drift Tube**: they are composed of a series of cylindrical electrodes with variable length, with an alternating electric field between them. The beam is accelerated by the electric field which is synchronized so that a bunched beam always sees an accelerating field. They are used to reach medium energies, up to a few hundred MeV. Figure 2.1b shows the drift tubes of the European Spallation Source (ESS) accelerator [54].

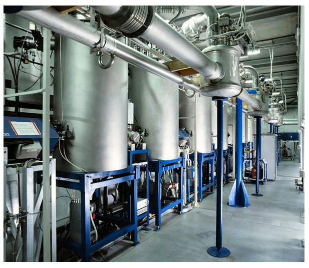
c) **Cyclotrons**: they are compact machines which use a static magnetic field to bend

(a) CN Electrostatic Accelerator.


(b) ESS Drift Tubes.


(c) SPES Cyclotron.


(d) Elettra Synchrotron.


(e) ALPI Linac.

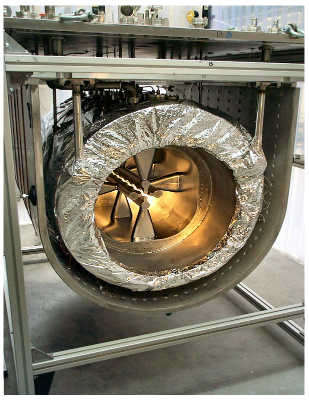

(f) PIAVE RFQ.

Figure 2.1: Examples of different types of particle accelerators.

the beam in a spiral path, and then use an RF field to accelerate the beam at each turn. They are common in the medical field, where they are used to produce radioactive isotopes for diagnostics and therapy. Figure 2.1c shows the SPES cyclotron [101].

d) **Synchrotrons**: they are composed by a ring of magnets which bend the beam in a circular path. The beam is accelerated by RF cavities located at one location along the circumference. They are used to reach high energies, and are common in high energy physics experiments where two beams are usually accelerated in opposite directions and then collided in a target. Figure 2.1d shows a rendering of the Elettra synchrotron in Trieste [46].

e) **Linear accelerators (Linac)**: they use RF cavities to accelerate the beam in a straight beam line. They can reach fairly high energies, depending on the number of available cavities. To reach high accelerating fields the cavities can be made of superconducting materials working at cryogenic temperatures. Figure 2.1e shows the *Acceleratore Lineare per Ioni* (ALPI) linac at LNL [27].

f) **Radio Frequency Quadrupoles (RFQs)**: in this case the beam is confined between four electrodes with a custom mechanical design, where an alternate quadrupole electric field is generated. This field is responsible both for the acceleration and the focusing of the beam. They are used as injectors for other accelerators, to increase the beam energy before the final acceleration. Figure 2.1f shows the RFQ of the *Positive Ion Accelerator for VEry low velocity ions* (PIAVE) injector at LNL [127].

The following sections will describe the particle accelerators available at Legnaro National Laboratories (LNL) and the control system used to operate them. They are used as case studies for the methods presented in this thesis, but many of the concepts presented are common to other accelerators.

2.2 Particle Accelerators at Legnaro National Laboratories

Legnaro National Laboratoriess (LNLs) are international physics laboratories of the Istituto Nazionale di Fisica Nucleare (INFN) located few kilometers away from the university of Padova. Their main focus is nuclear physics research, with multiple particle accelerators attracting researches from all around the world. A great effort is devoted to the development and maintenance of the accelerators, with innovative research projects on accelerator technology.

The main particle accelerators at the complex are:

- ***Acceleratore Lineare per Ioni* (ALPI)**: a linear accelerator based on superconductive RF cavities. This will be presented in detail in the following sections as the methods proposed in chapter 5 use this accelerator as a case study and rely on data from its control system for the experimental results.
- ***Positive Ion Accelerator for VEry low velocity ions* (PIAVE)**: an injector of ALPI, based on a superconductive RFQ. The complex includes a $250kV$ platform hosting an ion beam source based on Electron Cyclotron Resonance (ECR) technology, capable of producing positive ions with high charge.
- **TANDEM**: an electrostatic accelerator, which can deliver the beam directly to the target or as an alternative injector to ALPI. The name TANDEM is due to its peculiarity of accelerating twice the charges with one High Voltage (HV) potential: a negative ion source delivers a low-energy beam which is first accelerated towards the $+14.5MV$ terminal. Here the beam passes through a *stripper*, a thin carbon foil, which strips multiple electrons from the ions, which now become positive. Thus, they are now accelerated again by repulsion from the HV terminal. To achieve the maximum voltage, the whole accelerator is enclosed in a big tank, filled with Sulfur Hexafluoride (SF_6), which has a high dielectric strength.

- ***Selective Production of Exotic Species* (SPES)**: this is the latest accelerator installed at the laboratories and is still under development. It is composed of a Cyclotron capable of delivering $700\mu A$ of proton beam current and $70MeV$ particle energy. A primary target produces exotic species, which are then re-accelerated and brought to an experimental target to be analyzed. Other beam line are being developed to produce radionuclei for medical applications. While this accelerator is not yet ready to run and produce data, it is expected that its future control system will require advanced techniques like those presented in this thesis to achieve the design parameters.

- ***Acceleratore Di Ioni a Grande Carica Esotici* (ADIGE)** (ADIGE): this is a beam line designed to receive the 1+ radioactive ion beams, produced in the SPES Target Ion Source (TIS), in order to allow their post-acceleration with ALPI. ADIGE is also equipped with a 1+ ion source producing stable beams to characterize the beam line before the radioactive beams are available. This facility is currently under commissioning and will be described in more detail in the following sections, as it is the main target of the work presented in chapter 6.

2.3 ALPI

Here ALPI accelerator is described more in detail. This machine was built entirely at LNL during the first half of the 90's. Currently it is undergoing a renovation to be ready to accelerate the beams from the SPES primary target. Its main subsystems are:

- **Beam transport**: the beam is guided through the beam pipe with electromagnetic elements, like dipoles and quadrupoles. The dipoles are responsible for changing the direction of the beam, to follow the beam pipe layout. The quadrupoles are used to focus the beam, which is composed of bunches of particles with the same charge, and thus are subject to a repulsive force. It's important to keep the beam focused to minimize the percentage of beam current lost due to the collision with the beam pipe.

- **Beam Diagnostics**: a few instruments are dedicated to analysing the fundamental properties of the beam. There are multiple diagnostic boxes along the beam line. These are equipped with different instruments: the Faraday Cup (FC) is a metal target which can measure the charges of the particles colliding, thus giving a measurement of the beam current. A second instrument is the Beam Profiler which is composed of 40 small wires positioned horizontally and 40 vertically; when the beam passes through them, they can measure a small current, giving a measure of the beam shape, position and size, in both axis.

- **Acceleration**: the actual beam acceleration is provided by Radio Frequency (RF) cavities. Historically, the first accelerators used static HV potentials to accelerate charges. To increase the maximum energy, the RF cavities use an oscillating electric

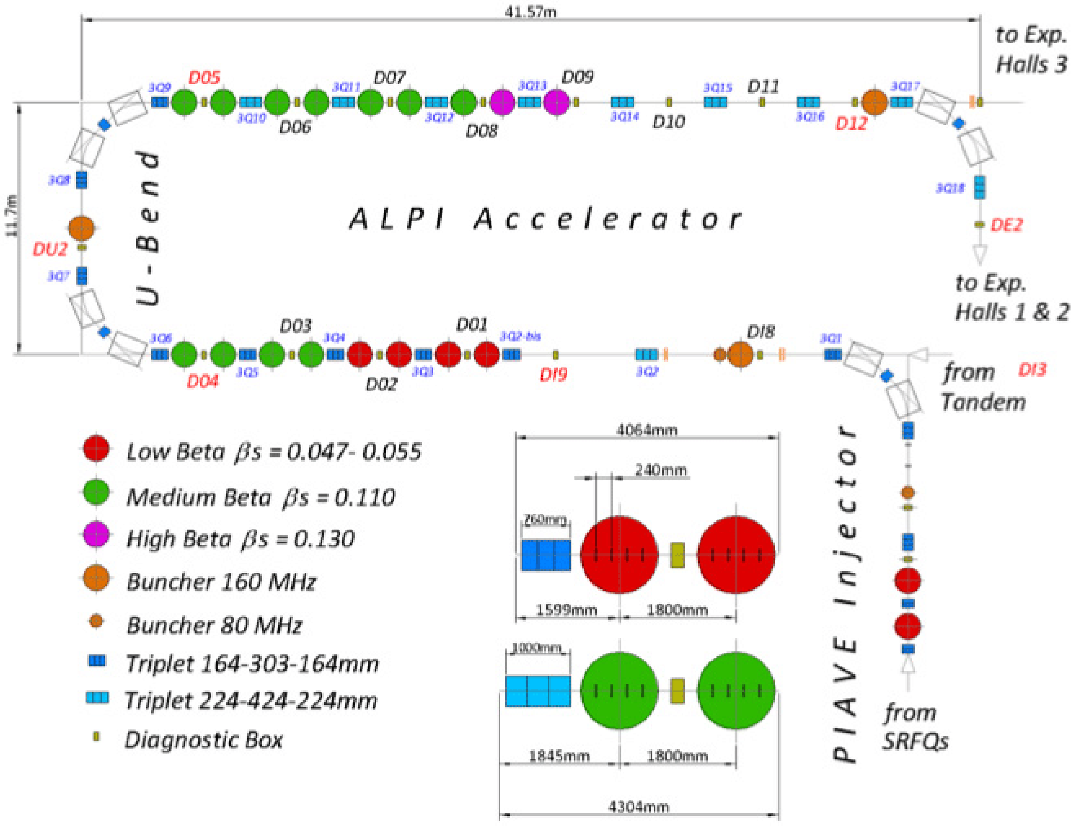


Figure 2.2: ALPI / PIAVE layout and main beam line elements. Source: [25]

field synchronized with the particle bunches. ALPI uses superconductive cavities, which can reach high gradients with low power consumption, and thus the cavities are grouped in about 20 cryostats of 4 cavities each. The work presented in this chapter 5 will focus on RF cavity operation, and thus more details are presented in the following section.

- **Vacuum**: the beam pipes are kept under high vacuum ($\sim 10^{-7} mBar$) to minimize the interactions of the particles with air and allow to operate RF cavities at high gradients avoiding discharges.

- **Cryogenics**: superconductive elements require cryogenics temperatures to operate, thus a cryogenic circuit is required, using liquid Helium (He) and Nitrogen (N). The cavities are operated at $4{-}6K$ using the He, while the N is used on the cryostats shields, to increase the temperature insulation.

2.3.1 RF cavities

RF cavities in the ALPI accelerator are able to transfer energy to the particles, and thus to accelerate it. They are made of Copper (Cu) with a superficial Niobium (Nb) coating. This material has been chosen since it has a critical temperature of $9.2K$ and thus becomes a superconductor at cryogenic temperatures. Since in a RF conductor the charges only flow on the surface of the material, a small layer of Nb is sufficient for superconductivity.

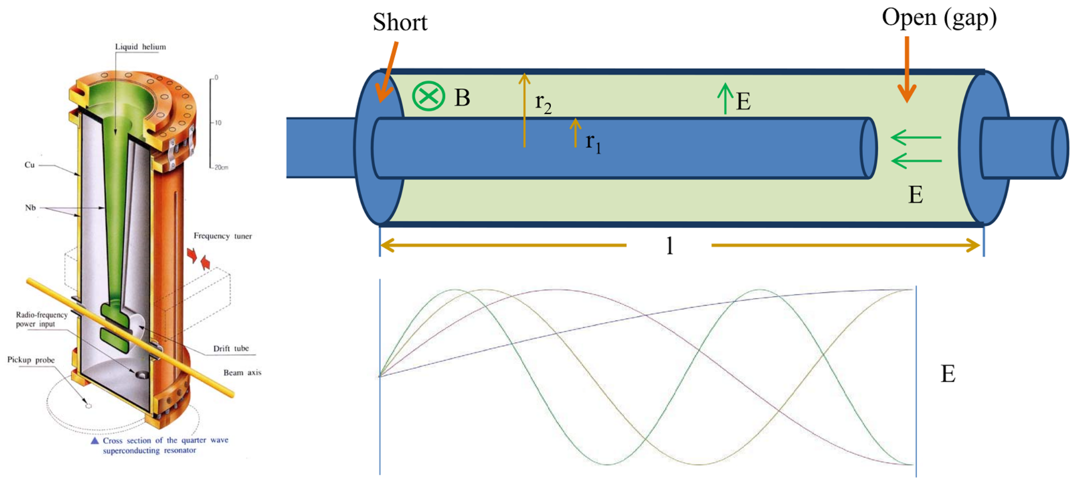


Figure 2.3: Quarter Wave Resonator. Source: [104]

ALPI uses a special kind of cavities, called Quarter Wave Resonators (QWRs). Their geometry is similar to a coaxial cable. As can be seen in figure 2.3, the RF power generates a alternate voltage between the cavity external walls and the inner conductor. This means that an alternating electric field is generated between the two poles, with its maximum intensity at the open extremity of the inner cylinder, where there is the beam line. These kind of cavities are able to accelerate only a bunched beam, meaning it is not a continuous flow of particles, but they are divided in packets, with a certain frequency of arrival. If the arrival of a packet of particles is synchronized with the electric field so that the electric field is directed toward the inner pole, the packet of (positive) charges is accelerated towards the center of the cavity. While the particles travel towards the center, the electric field is inverted so that when the particles reach the second half of the cavity they are subject to a second acceleration towards the exit of the cavity.

It is easy to understand that this implies a strong correlation between particles velocity, RF frequency and cavity geometry. Given all the constraint, the cavity can resonate only for the following family of frequencies:

$$f = \frac{c}{\lambda} = \frac{c(2n+1)}{4l}$$

where c is the speed of light, and l is the cavity length [104]. Given a certain frequency

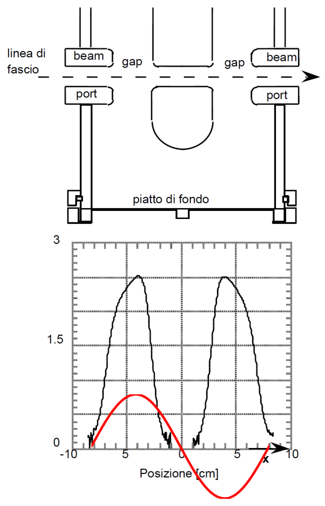


Figure 2.4: Electric Field along the beam axis. Source: [128]

of operation, the distance d between the centers of the two gaps where the particles are accelerated is fixed by:

$$d = \frac{\beta\lambda}{2} \qquad where \qquad \beta = \frac{v}{c}$$

where v è is the particles speed [104]. This means that the distance between the gap centers depends on the beam velocity (energy) and the cavity length depends on the bunches frequency.

For these reason in ALPI there are two different kind of cavities, first a group designed to work at $80MHz$ where the beam is less energetic, and then a second group at $160MHz$. The mechanical design and the geometry between the two groups is substantially different. This thesis focus on data from the first group, where the beam has a low β.

The second parameter, the cavity length (l) is not a fixed design parameter, but can be adjusted by the control system. By physically moving the metal extremity of the cavity with a motor, the control system can tweak the resonating frequency of the cavity and thus enable its operation at the desired setpoint. In the following paragraphs this mechanism is explained in detail.

2.3.2 Control System Architecture

Legnaro Control System is based on *Experimental Physics and Industrial Control System* (EPICS)[170], a free and open source framework. The architecture of the system is a distributed client server model, where the servers, also called Input Output Controllers (IOCs) are connected to the hardware and export all the available information and commands through a common network protocol, the Channel Access (CA). Each single piece of information exported by the IOC is called PV or *record* and all the PV on

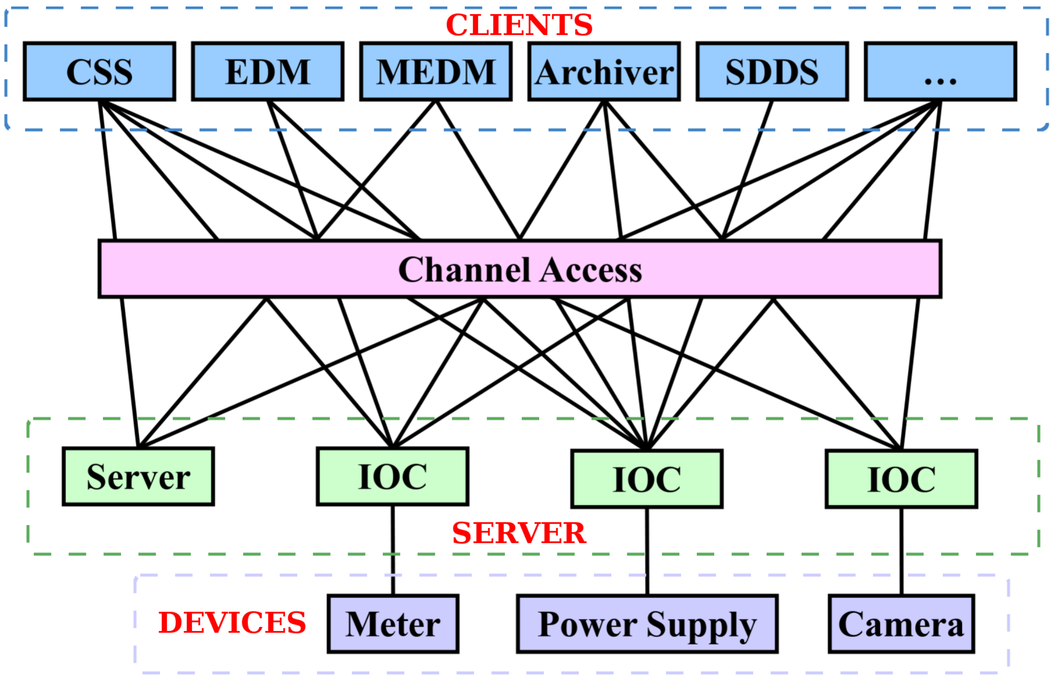


Figure 2.5: EPICS Architecture. Source: [75]

a network form what is called the *database*, even though it is not a traditional database software. Many clients then can access the database with standard commands to read and write the PVs, using the PV name as the only addressing mechanism. One example of client is the Graphical User Interface (GUI) in the control room, but there are libraries to access the CA for many programming languages.

2.3.3 The Archiver

This common layer, where all the process variables of all subsystems are available to all the clients, ease the exchange of information and a centralized logging process. In fact, one client is the *archiver*, a software which monitors the value of a set of PVs and save it to a database backend every time it changes. The archiver is composed of multiple *engines* which can be configured to monitor a different set of PVs, and each engine can be composed of multiple *groups* to organize the information.

The archiver database is PostgreSQL and figure 2.6 shows its internal schema. The most important table is the *sample* one, where all the historic values of all the PVs are stored, one per row, while other tables contain metadata and the groups and engines configuration. This service can be accessed through a basic web interface where the status of engines, groups and PVs is displayed, or directly through SQL queries.

This service has been crucial in the data-collection phase as it has allowed to store the

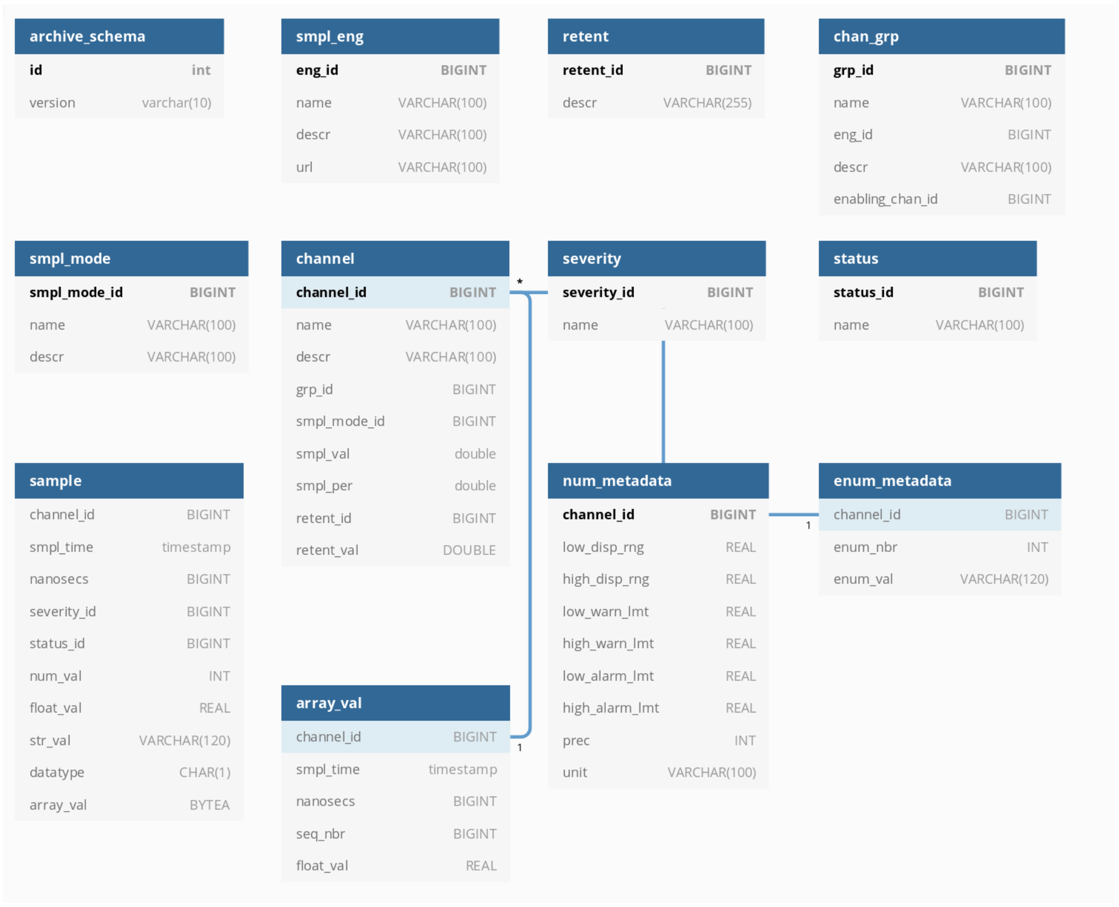


Figure 2.6: Archiver Database schema. Source: internal LNL control group wiki.

temporal evolution of hundreds of process variables in a single well-organized repository, with standard SQL commands available to retrieve the data.

2.3.4 RF control system

To fully understand the data used in chapter 5, the RF control system is here described more in detail. The purpose of this system is to achieve the greatest cavity gradient as possible to accelerate the beam and to do it in a stable way, otherwise beam transport would be impossible. This means that on each cavity the RF amplitude, frequency and phase must be controlled to be fixed to the desired setpoint, with minimal oscillations. This particular operating state is called the ***locked*** state of the cavity, while the lost of this condition is an *unlock* event. To lock the cavity and to keep it locked a hardware-based control loop is used, as highlighted by the orange loop in figure 2.7. A sample of the RF power is read from the cavity, its frequency, phase and amplitude are compared with the setpoint in a hardware controller and a signal is generated with the correction required to reach the desired setpoint. This signal is then amplified by the RF power amplifier and injected back into the cavity.

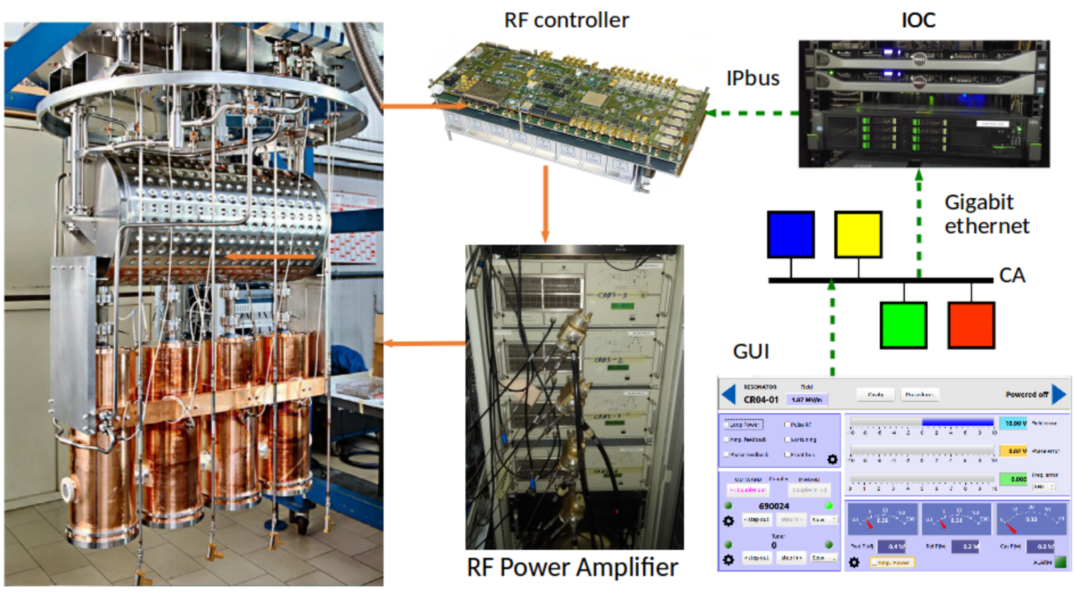


Figure 2.7: RF control system. Source: [104]

This system can apply corrections in a fast and accurate way, but has a limited operating range. The power required to apply a correction grows exponentially with the module of the error, and can easily reach the limit of the amplifiers and the cables. For this reason it's necessary to keep the natural resonating frequency of the cavity as close as possible to the setpoint. The cavities are manufactured very precisely to adhere to this requirement, but still there are some factors which can alter the cavity frequency:

- **geometry**: superficial deformations or vibrations
- **dimension**: with cryogenic temperatures the metal shrinks and alter the dimension of the cavity
- **energy**: electromagnetic fields inside the cavity create an electromagnetic pressure which compresses or expands the cavity
- **dielectric**: the vacuum level can impact the resonating frequency

For these reasons, a second control loop has been implemented in software, called *soft tuner*. While the cavity is locked the frequency and phase error should be zero, but in practice there is always a small residual error. The software can read those residual errors and decide to activate a stepper motor to move the metal extremity of the cavity, thus changing its length and its resonating frequency. This has the effect of reducing the correction required by the hardware controller, and thus reducing the RF power to be used. This is a much slower control loop, since it is done with software and involves the movement of a motor but the correction can be triggered only when the residual error is over a certain threshold. Since the hardware controller can work next to its ideal operating point, with low residual error, the result is that there are much fewer unlock events, where the feedback loop can't continue to operate and the field or phase setpoint are lost. Whenever this happens the beam is also lost because it will arrive out of time to the subsequent cavities, for this reason it is important to reduce these events because they represent an accelerator downtime.

2.4 ADIGE

This section describes in detail the ADIGE beam line, which is the main target of the work presented in chapter 6.

Acceleratore Di Ioni a Grande Carica Esotici (ADIGE) is a beam line under commissioning at LNL, located in the middle between SPES and ALPI. It is designed to receive the 1+ radioactive ion beams, produced in the SPES TIS, with the goal of increasing their charge state thanks to a charge breeder, so that they can be accelerated more effectively in ALPI. The beam line is also equipped with a 1+ ion source producing stable beams to characterize the beam line before the radioactive beams are available. Currently, the beam line has been installed and commissioned up to the third diagnostic element, while the rest of line is being installed. The first part has been successfully tested for a prolonged period of time, characterizing the ion source in different operating conditions [48].

Figure 2.8 shows the synoptic of the beam line, as presented to the machine operators. The colors are used to highlight different type of devices, with the green ones being the electrostatic transport elements, the blue ones the magnets and, the orange ones the diagnostic elements, and the gray the RF elements. In detail the beam path the line is composed of:

1+ ion source This is the ion source of the beam line. It is composed of high current power supplies located on a high voltage platform. The power supplies are used to heat a oven, where the material to be ionized is placed. The oven is heated to a temperature where the material is evaporated and ionized. The ions are then extracted from the oven by the electric field of the platform and accelerated towards the beam line. A set of steerers is used to align the beam with the following elements. The source can also be used with a gas injection system to produce a plasma and extract the ions from it.

Electrostatic Triplets (AD.3EQ.xx) These are groups of three electrostatic quadrupoles, used to focus the beam both in the horizontal and vertical axis.

Diagnostics (AD.BI.xx) These are diagnostic boxes, where the beam can be characterized. They can include a Faraday Cup, a Beam Profiler and a Beam Emittance

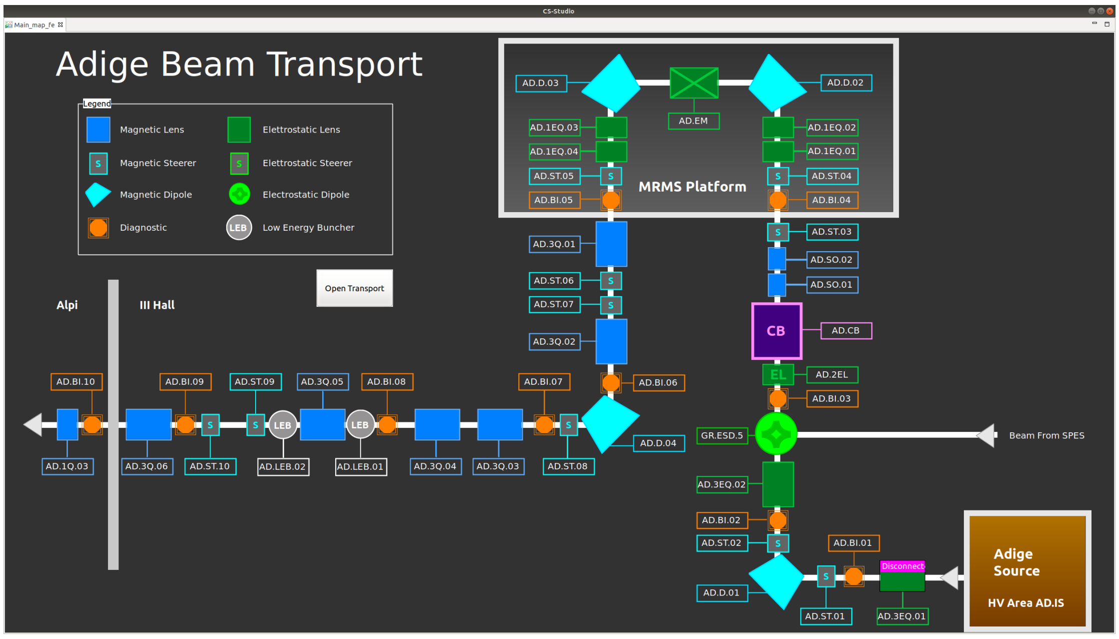


Figure 2.8: ADIGE beam line synoptic from the control room.

Meter. The Beam Emittance Meter can measure the beam emittance, which is a measure of the beam quality, which will be introduced in detail later. There exists different types of emittance meters, with the most basic being the Allison Scanner, composed of a set of slits which can be moved to select a portion of the beam, and a set of wires which can measure the beam current. By moving the slits and measuring the current at each position, the beam emittance can be calculated.

Magnetic Steerers (AD.ST.xx) These are couples of magnetic steerers, used to correct the beam trajectory.

Magnetic Dipoles (AD.D.xx) These are magnetic dipoles, used to change the beam direction. They are composed of a set of coils which generate a magnetic field, which is used to deflect the beam so that it follows the beam line curve. The coils are powered by a current power supply, which can be used to change the intensity of the magnetic field and thus the beam deflection.

Electrostatic Dipole (AD.ESD.5) This is an electrostatic dipole which can receive both the beam from the ADIGE source and from the SPES beam line and direct it towards the charge breeder. The operators can select the beam source by rotating the dipole.

Einzel Lens (AD.2EL) This is an electrostatic lens, used to focus the beam after the electrostatic dipole.

Charge Breeder (AD.CB) This is the charge breeder, used to increase the charge state of the ions. It is composed of a oven where the ions are injected, and a plasma chamber where the ions are ionized again. The plasma chamber is kept under vacuum and is heated by a radio frequency field, which ionizes the ions. The ions are then extracted from the plasma chamber and accelerated towards the beam line.

Solenoids (AD.SO.xx) These are magnetic solenoids, used to focus the beam after the charge breeder.

Multipole (AD.EM) This is an electrostatic multipole, used to correct the beam emittance between the two dipole of the Medium Resolution Mass Separator (MRMS).

Magnetic Triplets (AD.3Q.xx) These are groups of three magnetic quadrupoles, used to focus the beam both in the horizontal and vertical axis.

Bunchers (AD.LEB.xx) The Low Energy Bunchers (LEBs) are RF cavities used to bunch the beam before the injection in ALPI.

This thesis will focus on the MRMS platform, that is the part of the line with a U shape, and on its electrostatic multipole. The next section will present them and describe their purpose and behaviour.

2.4.1 Medium Resolution Mass Separator (MRMS)

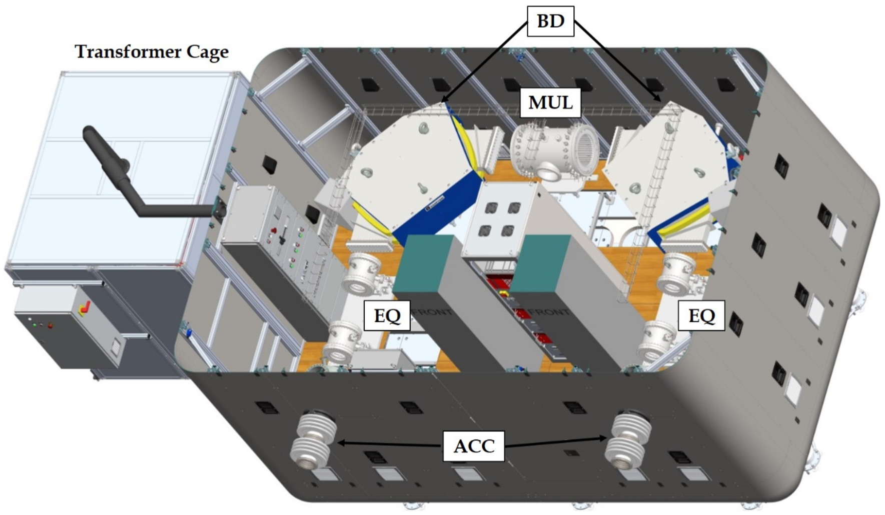


Figure 2.9: The MRMS platform [47]

The Medium Resolution Mass Separator (MRMS) [47] is installed on the ADIGE 1+ ion source beam line after a charge breeder, to separate the contaminants introduced by the breeding stage. It has a nominal resolving power of 1/1000 and is composed of a high voltage platform (see figure 2.9) operating at $-150kV$ with 4 electrostatic quadrupoles (up to 12kV), 2 bending dipoles and an electrostatic multipole between them. This is a cylinder composed of 48 high voltage terminals operating at $\pm 2.5kV$, so that it can be used as a high order multipole. Finally, on the beam line after the platform an Allison Scanner is available to measure the beam emittance.

Mass Separator

From the Lorentz force

$$\vec{F} = q\vec{v} \times \vec{B} \tag{2.1}$$

we know that a charged particle moving in a magnetic field is subject to a force which is perpendicular to both the velocity and the magnetic field. This means that the particle will follow a circular trajectory, with a radius proportional to the particle mass and inversely proportional to the charge of the particle $r = \frac{mv}{qB}$. This is the principle behind the mass separator, which is shown in figure 2.10. Thus, given a magnetic dipole built

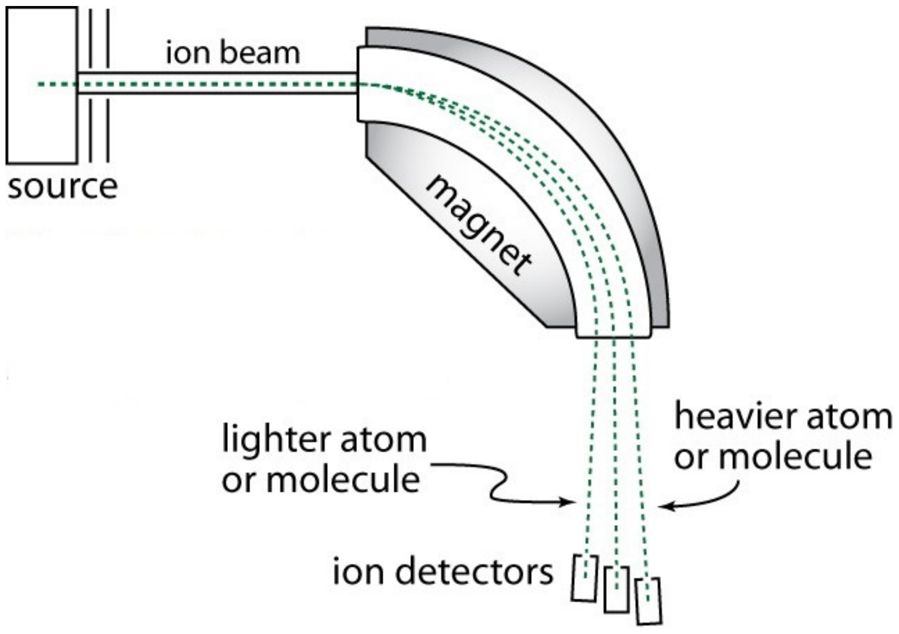


Figure 2.10: Mass Separator principle of operation.

with a certain radius and a beam with a specific energy and charge, by adjusting the magnetic field it is possible to select a specific mass of the ions. This principle is used twice in the MRMS, which is composed of two dipoles, to select the desired isotope.

Beam Emittance

The beam emittance is a measure of the beam transport quality, and is defined as the area occupied by the beam in a position and momentum phase space. It depends both on the variance of the particle positions and on the variance of the particle velocities.

To define the emittance first we should define the coordinate system, as shown in figure 2.11.

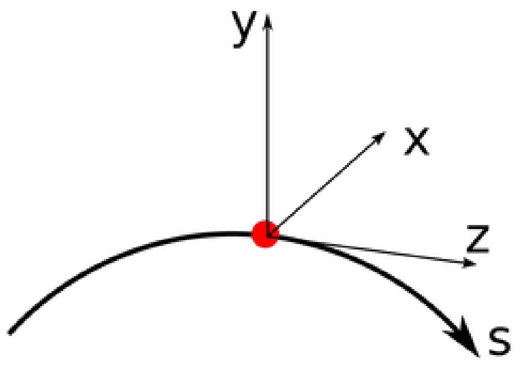


Figure 2.11: Coordinate system for the beam dynamics of a particle accelerator.

The coordinate system is centered with respect to the trajectory of a particle moving through the accelerator on the ideal trajectory s, with the nominal speed. This trajectory defines the *longitudinal axis* z, while the horizontal x and vertical y axis are called *transverse* axes.

Given this coordinate system, the beam emittance can be defined for each axis. Since in the MRMS the beam is spread along the x axis, we are interested in the x, x' beam emittance, which can be defined as:

$$\varepsilon_{RMS,x} = \sqrt{\left\langle x^2 \right\rangle \left\langle x'^2 \right\rangle - \left\langle xx' \right\rangle^2} \tag{2.2}$$

where $\left\langle x^2 \right\rangle$ is the variance of the particle positions, $\left\langle x'^2 \right\rangle$ is the variance of the particle velocities and $\left\langle xx' \right\rangle$ is the covariance between the two. $x' = \frac{dx}{dz}$ can also be interpreted as the angle that a particle makes with respect to the z axis. The beam emittance is usually expressed in *mm mrad*.

The beam is composed of particles with the same charge, and thus they repel each other. This means that the beam is not perfectly focused, the particles will drift apart and the beam will be lost. A low emittance value means that the beam is well focused and the particles are all travelling with similar velocities. This is important for the beam transport, since the closer the beam is to the ideal one, the more particles will reach the target and the more intense the beam will be.

In the x, x' plane the emittance is the area of the ellipses formed by the particle distribution, as shown in figure 2.12. Intuitively, the shape and orientation of the ellipses

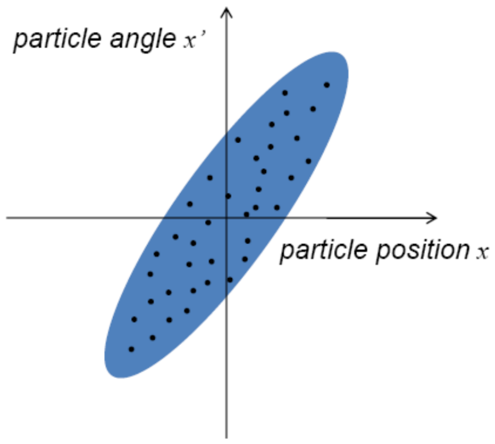


Figure 2.12: Beam emittance graph.

can be used to understand the beam quality. As shown in figure 2.13, when the beam is converging towards the focus point, the ellipse lies on the second and forth quadrant, because angle x' of the particles is negative. When the beam is diverging, the ellipse lies on the first and third quadrant, because the particles are moving away from the focus point, and thus the angle x' is positive. Finally, when the beam is not converging nor diverging, the ellipse is vertical, because the particles are moving parallel to the z axis, and thus the angle x' is around zero.

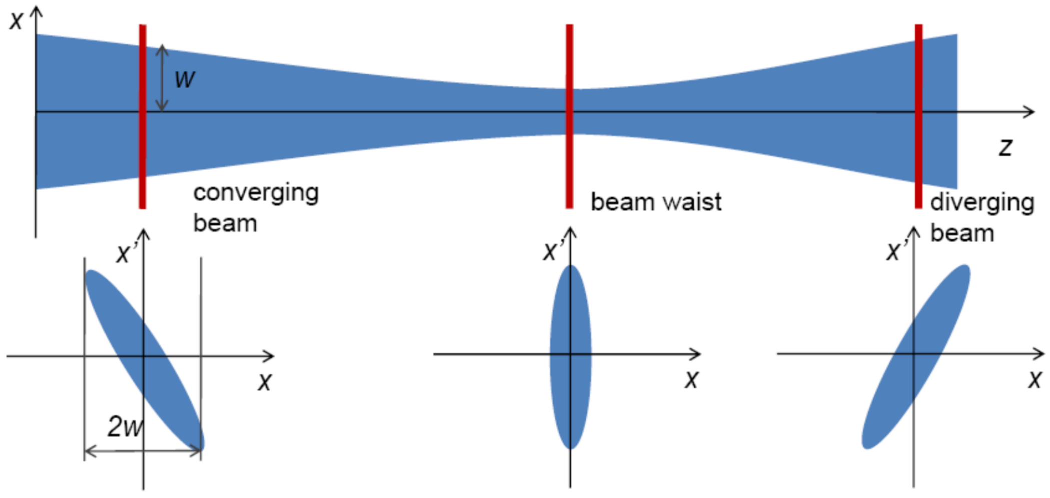


Figure 2.13: Beam emittance ellipses corresponding to the beam converging or not.

During the beam transport the beam emittance value changes whenever the beam is accelerated or decelerated. For this reason it is important to define a normalized emittance, which is invariant under acceleration. This is a constant value for a given beam, and thus it is a good measure to compare the beam quality at different energies.

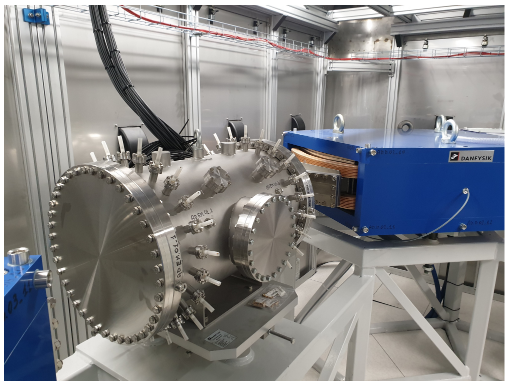
$$\varepsilon_n = \sqrt{\langle x^2 \rangle \langle (\gamma\beta_x)^2 \rangle - \langle x \cdot \gamma\beta_x \rangle^2} \tag{2.3}$$

Here the angle x' has been replaced by the normalized transverse momentum $\gamma\beta_x$ where γ is the Lorentz factor and $\beta_x = \frac{v_x}{c}$ is the normalized transverse velocity.

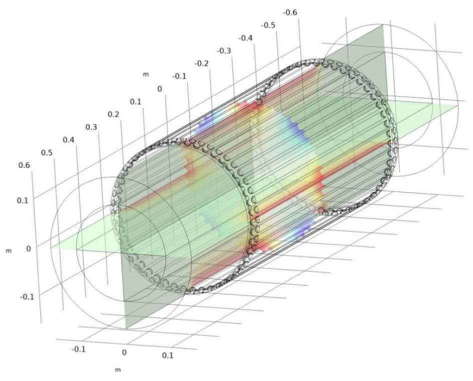
At the entrance of the MRMS platform the beam is defocused on the horizontal axis before the dipoles to maximize their resolving power. This means that the beam is spread on the x axis and when it enters the bending dipoles many particles follow a trajectory far from the ideal one. Here the magnetic field is not guaranteed to be uniform as in the center of the magnet. Furthermore, by passing far from the optimal ion path, the particles would require a different magnetic field to follow the ideal trajectory. Thus the magnetic field introduces non-linear effects to the beam, and the beam emittance increases.

Electrostatic Multipole

The electrostatic multipole of the ADIGE MRMS platform is placed between the two bending dipole. It is designed to compensate the non-linear effects on the beam introduced by the magnetic field of the dipole. Figure 2.14a shows a photo of the multipole installed in its final position, where it's also visible one of the bending dipoles in blu. The high voltage terminals are present on both sides of the multipole for a total of 48 terminals. Currently, the terminals are not yet connected to the power supply. Figure 2.14b instead shows the design schema of the multipole, where it's possible to see the metal bars connected to the terminals which bring the high voltage along the multipole length. The resulting electrical field is simulated to verify the design.



(a) Current status of the multipole.



(b) Design schema of the multipole.

Figure 2.14: ADIGE Electrostatic Multipole.

By having a large number of high voltage terminal the multipole can be easily reconfigured to act as a quadrupole, esapole, etc. or with a generic configuration up to 48 poles. Thus it is able to compensate higher order aberrations of the beam. In practice, the symmetry along the y axis is enforced by the planned cabling. In fact a 24 channel power supply will be used, and the terminals will be connected in pairs to the same channel, following the symmetry over y. This means that the electric field will be symmetric and will not have effects on the y axis. This is not a problem, since the multipole is only required to correct the aberrations on the x axis.

2.5 Control Systems Challenges

While many different application of ML to particle accelerators have been proposed in literature and can be further investigated, we decided to focus on two main problems which are of great importance for the accelerator complex at Legnaro: the detection of faults in the RF control system and the optimization of the beam emittance in the MRMS platform. The next two section will present these two problems in detail. These are not to be intended as final goals, but rather as a proof of concept of the effectiveness of ML in the accelerator control system. In fact, the same approach can be used to solve many other problems, and the results presented in this thesis can be used as a starting point for further development. The experience gained solving these tasks can help to highlight a path towards the development of intelligent control systems, able to perform many tasks automatically, and to optimize the accelerator performance. In particular the beam emittance optimization task can be seen as a first step towards an automatic beam transport system, capable of tuning all the beam transport elements on a beam line without human intervention. While the dimensionality of this problem may render it unfeasible for large beam lines today, it clearly shows the potential of ML in this field. With the advent of even more advanced techniques, such as new more efficient RL models, and improved computing power, this kind of systems may become a reality in the near future.

2.5.1 RF Runtime Faults

The RF control system presented in 2.3.4 is composed of two different control loops with the goal of keeping each cavity locked to the desired amplitude and phase setpoint. When all the cavities are locked, the beam is accelerated and transported to the target.

While the two feedback loops work most of the times, there are still situations where they can fail and the operation is interrupted. This can happen for different reasons like:

- Instabilities in the cryogenic lines, different pressure levels can alter the working conditions;
- Mechanical limitations of the motors and related gears can limit the effectiveness of the control loop;

- External temperature changes and slow drifts in the working conditions;
- HV discharges.

During one month of ALPI operation in 2020 we registered 169 lock failed events on 24 cavities in the first 6 cryostats working at $80MHz$. In all these cases the beam was lost and could not reach the target anymore, thus pausing the physics experiment. The number of faults is high enough to justify further development to improve the performance, but it's difficult to manually identify the problem. Hence the idea to analyze the historical data of the system and try to refine it further by using ML techniques.

This thesis will present a work in chapter 5 where the historical data of the RF control system is analyzed to detect faults in the system. The goal is to build a model which is able to predict if a fault is going to happen in the next few seconds. This can be used to improve the performance of the control system, by acting early to prevent the fault, with the final goal of improving the accelerator uptime.

2.5.2 ADIGE multipole configuration

The MRMS electrostatic multipole, presented in section 2.4.1, is used to correct the beam emittance between the two bending dipoles. It is composed of 48 high voltage terminals, which can be configured to act as a multipole of different order. In practice the multipole is cabled symmetrically over the y axis, so that only 24 independent voltages can be set.

Even so this is large number of parameters. If the operator had to tune them manually it would be lucky to find any good configuration, let alone the optimal one. In most of the cases the beam would degrade and it would be hard to find an error minimization direction to follow over 24 parameters. Furthermore, the only feedback available would be the beam emittance measured after the MRMS platform, which is slow to measure, requiring a scan of the Allison Scanner slits. This means that the operator would have to wait for the scan to finish before being able to evaluate the effect of the changes, thus slowing down the tuning process.

Another option would be to use a simulation to find the optimal configuration. In fact, the beam transport can be simulated with high precision, and the beam emittance can be calculated at the end of the line. This would allow to find the optimal configuration offline and apply it directly. Unfortunately, the simulation results are not always accurate enough to be used directly on the real machine. Moreover, the multipole is designed to correct high order aberrations introduced by the dipole, but those effects are hard to simulate. In fact, they are highly dependent on the actual position of the beam particles on the dipole and the non-uniformity of the magnetic field, which is hard to know with high precision. Thus, while simulation could be used as a starting point, the final configuration should be fine-tuned on the real machine. This could still require many trial and errors and may not result in the optimal configuration.

In chapter 6 of this thesis we propose a different approach, where the multipole configuration is optimized directly on the real machine with Reinforcement Learning. We develop a deep learning model which is able to iteratively update the multipole configuration converging towards the minimal beam emittance. The model exploits simulations for training and can be fine-tuned on the real machine, but should not learn a single optimal configuration. Instead, it should understand how to minimize the beam emittance even in new conditions, where the optimal configuration may change. In fact, such a model could be able to find the optimal solution with few iterations, thus reducing the time required to tune the multipole and improving the accelerator efficiency. As such this model can be seen as a first step towards an automatic beam transport system, which is able to find the optimal transport configuration of a complete beam line.

Chapter 3

Machine Learning Elements

3.1 AI and ML brief introduction

The emergence of intelligent systems capable of learning from data and making informed decisions has had a profound impact on many fields of our society [56, 161]. At the heart of this transformative process lies the domains of Machine Learning (ML) and Deep Learning (DL), part of the broader field of Artificial Intelligence (AI). These disciplines have gained unprecedented traction in the last decade, thanks to improvements both on the algorithmic and hardware fronts. The advent of Big Data has fostered the development of novel ML methodologies, while the availability of Graphical Processing Units (GPUs) has enabled the training of complex models on massive datasets. These advancements have brought impressive results on a large number of applications, ranging from image classification to natural language processing [81, 77], further fueling the research and applications interest.

The general goal of Machine Learning is to develop models with the ability to learn autonomously from data without explicit programming. These model are able to scrutinize large datasets to discern patterns, extract meaningful insights, and generalize knowledge for predictions and decisions. For this purpose ML builds upon the fields of probability theory, linear algebra, optimization, and algorithm design.

A first approach to this problem is the supervised learning scenario, where algorithms are trained on labeled datasets, learning mapping functions that correlate input and output variables. This was first applied to the problems of classification and regression tasks, with applications encompassing email filtering, medical diagnosis, and financial forecasting. Conversely, unsupervised learning unveils latent structures within unlabeled data. Examples are the problems of clustering and dimensionality reduction techniques, used for customer segmentation, anomaly detection, and exploratory data analysis.

While this field has been around for decades, the recent advancements in Machine Learning can be related to the introduction of Deep Learning. Deep Learning is a subfield of ML that leverages the power of artificial Neural Networks to learn complex representations from raw data. These models, which will be presented in detail in section 3.3, are characterized by complex architectures with many layers and a large number of parameters, thus being able to automatically learn salient features and abstractions. On the contrary classical ML models are typically shallow, with a limited number and parameters, and require hand-crafted features to be extracted from the data. This makes DL models more flexible and powerful, but also more computationally expensive and data-hungry. In real-world applications, however, the architecture of a deep learning model is typically tailored to the problem at hand, thus mitigating the computational and data requirements. For example, Convolutional Neural Networks (CNNs) manifest unparalleled efficacy in image analysis by using convolutional filter which reduce the number of training parameters and allow to hierarchically detect features such as edges, textures, and shapes. Recurrent Neural Networks (RNNs) instead focus on sequential data, thus making them potent tools in natural language processing, speech recognition, and time-series forecasting.

In the supervised setting such models are trained by optimizing the parameters (called weights) with the aim of minimizing a certain loss function. Different loss functions are used depending on the task at hand. For example, in the case of classification tasks, the cross-entropy loss is typically used, while in the case of regression tasks the mean squared error is often employed. The optimization of the loss function is performed using gradient-based methods, such as Stochastic Gradient Descent (SGD) and its variants. Each input is fed to the model, which produces an output which is in turn compared with the expected result to calculate the loss. The weights of the model are then updated using the gradient of the loss function with respect to the weights. In practice, however, data points are not fed to the model one at a time, but in batches. This means that the loss is calculated on the average of the losses of the points in the batch, and the weights are updated using the average of the gradients of the points in the batch. When all the points in the dataset have been used, an *epoch* is completed and the dataset is shuffled and used again. The whole process is repeated for a number of epochs, until the model converges to a stable solution. In the unsupervised setting, instead, the model is trained to learn the underlying structure of the data, without the use of labels. For example, in the case of clustering tasks, the model is trained to group similar data points together, while in the case of dimensionality reduction tasks, the model is trained to reduce the dimensionality of the data while preserving the most relevant information. In both cases, the model is trained to minimize a certain loss function, which is typically related to the reconstruction error of the data.

Once the model is trained, it can be used to make predictions on new data. In this phase, the model is typically evaluated on a test set, which is different from the training set and is used to assess the generalization capabilities of the model. This is a crucial step in the development of a ML model, since a model that is able to achieve high accuracy on the training set but performs poorly on the test set is likely to be overfitting the training data. This means that the model is learning the training data too well, thus failing to generalize to unseen data. Underfitting instead occurs when the model is not able to learn the training data, thus resulting in poor performance on both the training and test sets. Figure 3.1 shows an example of overfitting and underfitting on a classification and regression problem. While underfitting is simple to address by using a more complex model or increasing the training epochs, overfitting can be a tougher problem to solve. It can be tackled by using regularization techniques which aim to reduce the model complexity, which may reduce the overall performance, or by using more data, which may not be available. In the case of deep learning models, overfitting can also be mitigated by using early stopping, which consists in stopping the training when the generalization error starts to increase.

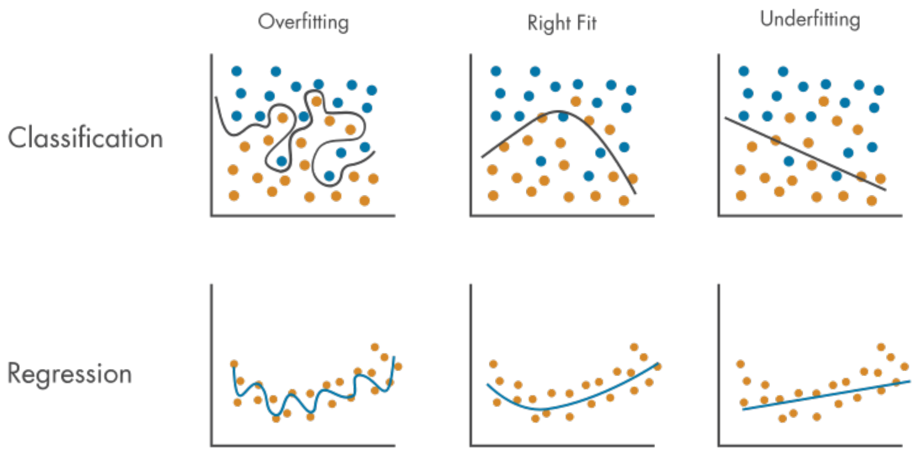


Figure 3.1: Examples of overfitting and underfitting in the Classification and Regression problems.

The ML developer thus has to choose the correct model and its best configuration, depending on the problem and the available dataset. These choices are called *hyper-parameters*, not to be confused with the weights of the model, as they are not learned during the training phase. This is typically done by using a validation set, which is different from both the training and test sets. The model is trained on the training set with different sets of hyperparameters and evaluated on the validation set. Finally, the most promising hyperparameters are used to train the model on the whole training set and evaluated on the test set. This process is called hyperparameter tuning and it is crucial to obtain a model that is able to generalize well to unseen data. One problem that may arise is that the results are dependent on the split between training, validation and test sets. To overcome this problem, the cross-validation technique can be used. As shown in figure 3.2 in this case, the dataset is divided into k folds, and the model is trained k times, each time using a different fold as validation set and the remaining folds as training set. The final results are then obtained by averaging the results of the k models. This technique is particularly useful when the dataset is small, as it allows exploiting better the available data.

After a model shows good results on the test set, it's important to evaluate it on

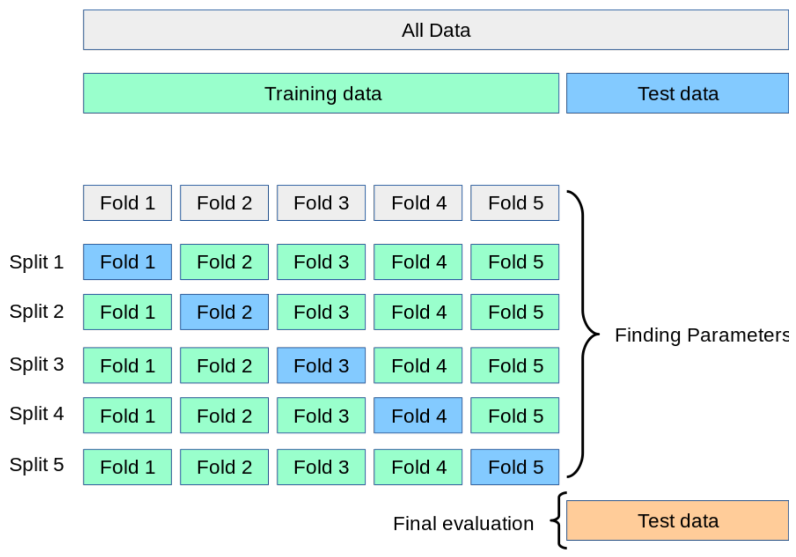


Figure 3.2: Example of 5-fold cross-validation.

external factors, which may not be completely captured by the loss function, such as explainability and fairness. In fact, while classical ML models are often based on simple mathematical functions, which are easy to interpret, deep learning models are based on complex architectures with millions of parameters, which are difficult to interpret. This is a problem, especially in critical applications, where it is important to understand why a certain decision has been taken. For this reason, in recent years, the field of explainable AI and the research on fairness of ML models has gained a lot of traction. This is particularly important in the medical field, where one wants to understand why a certain diagnosis has been given and avoid discriminations. In fact, it has been shown that ML models can be biased, for example, against people of color [121]. This is a problem that is often overlooked, but it is important to keep it in mind before using such models on real world applications.

Another important field of ML is Reinforcement Learning (RL). In this case, the model is not trained on a dataset, but it learns by interacting with the environment. The model is called agent and it is trained to maximize a reward function, which is a function of the state of the environment. The agent interacts with the environment by performing actions, which change the state of the environment, and receives a reward based on the new state. The goal of the agent is to learn a policy, which is a function that maps the state of the environment to an action, in order to maximize the reward. Thus, RL has proven very effective at solving complex games, such as Atari games [117] and Go [152]. RL has also been used in many other fields, such as robotics [87], finance [120], and healthcare [133].

While a complete dissemination on the vast fields of Machine Learning and Artificial Intelligence is out of scope for s, the rest of the chapter will focus on methods for Anomaly Detection and on the Reinforcement Learning framework. In fact, the main research activities carried out during the PhD are based on these methods. In particular, chapter 5 will present two different approaches for Anomaly Detection and Fault Prediction in the ALPI RF control system, while chapter 6 will show an application of Reinforcement Learning to optimize the beam transport in the ADIGE beam line.

3.2 Anomaly Detection

The problem of finding novel or anomalous behaviours, often referred as Anomaly or outlier Detection (AD), is common to many contexts: anomalies may be very critical in many circumstances that affect our everyday life, in contexts like cybersecurity, fraud detection and fake news [74, 96]. In science Anomaly Detection tasks can be found in many areas, from astronomy [103] to health care [115]; moreover, Anomaly Detection approaches have been even applied to knowledge discovery [29] and environmental sensor networks [60].

One of the areas that mostly benefits from the employment of AD modules is the industrial sector, where quality is a key driver of performance and success of productions and products. With the advent of the Industry 4.0 paradigm, factories and industrial equipment are generating more and more data that are hard to be fully monitored with traditional approaches; on the other hand, such availability of data can be exploited for enhanced quality assessment and monitoring [94, 155]. Moreover, products and devices, thanks to advancements in electronics and the advent of the Internet of Things, are increasingly equipped with sensors and systems that give them new capabilities, like for example the ability to check their health status thanks to embedded/cloud Anomaly Detection modules [4, 79, 168].

Despite the heterogeneous systems that may benefit from AD modules/capabilities, there are several desiderata that are typically requested for an AD module, since obviously looking for high detection accuracy is not the only important requisite. For example, in many contexts the delay between the occurred anomaly and its detection might be critical and the low latency of the model becomes a stringent requirement. One way to mitigate the detection delay problem is typically to embed the AD model in the equipment/device: this implementation scenario directly affects both the choice of the detection model and, in some cases, the hardware. As a consequence practitioners will typically have to find a compromise between detection accuracy and computational complexity of the model: while this is true for any Machine Learning module, it is typically more critical when dealing with AD tasks. Moreover, in presence of complex processes or products equipped with different sensors, data will exhibit high dimensionality and therefore practitioners will tend to prefer models that are able to efficiently handle multiple inputs. Summarizing, a good real-world AD solution: i) has to provide high detection performances; ii) has to guarantee low latency; iii) requires low computational resources; iv) should be able to efficiently handle high dimensional data.

The former list of desiderata for AD solutions is not exhaustive, and other characteristics can increase the model appeal in front of practitioners. In recent years, model interpretability is an increasingly appreciated property. Detecting an anomaly is becoming no longer enough and providing a reason why a point has been labelled as anomalous is getting more and more importance. This is particularly true in manufacturing processes where the capability of quickly finding the root cause of an anomalous behaviour can lead to important savings both in terms of time and costs. In addition, interpretability enhances the trust of users in the model, leading to widespread adoption of the AD solution.

Another attractive property is the ability of the model to handle data coming from non-stationary environments. Especially in early stages of AD adoption, available data are few and restricted to a small subset of possible system configurations; a model trained on such data risks to label as anomalous all the states not covered in the training domain, even if they are perfectly normal. In order to overcome this issue, the model should detect the changes in the underling distribution and continuously learn new *normal* data. In this process, however, the AD model should not lose its ability to detect anomalies.

Chapter 5 is dedicated to the application of AD techniques to the problem of fault prediction in particle accelerators. In this context, AD models are used to detect anomalies in the data stream produced by the sensors of the accelerator. For this reason, given the importance and diffusion of AD approaches, we deemed relevant to review the literature and to provide a comprehensive overview of the state of the art. After a formal definition of the AD problem, the following sections report the results of a systematic review of the literature on tree-based approaches to AD that we published in [10]. Furthermore, we include a general overview of ML and DL models for Anomaly Detection, with a specific focus on applications to time-series.

3.2.1 Outliers

The concept of *outlier*, or *anomaly*, can be easily perceived by intuition but it's difficult to give a formal definition. In fact, there is no single accepted definition, as it depends on the specific kind of data, field of application or interest in a certain feature. Consequently the methods and their results depend on the type of anomaly that is being searched and the ground truth is not always clear.

A first attempt to give a general definition comes from Grubbs [55]:

> *An outlying observation, or "outlier," is one that appears to deviate markedly from other members of the sample in which it occurs.*

Another widely accepted definition was given by Hawkins [58]:

> *An outlier is an observation which deviates so much from the other observations as to arouse suspicions that it was generated by a different mechanism.*

These definitions highlights two characteristics of anomalies: they are pattern in the data which are not common and that are probably generated by some unusual process. In the case of a particle accelerator, this unusual process can then lead to a fault condition, and thus to the stop of the machine. Identifying anomalies means identifying early the processes which can lead to a fault. For the purpose of the works presented in chapter 5, a more detailed definition of outlier can be given:

> *An outlier is an observation which does not appear during the normal operation of the accelerator and that is strongly correlated with a fault event few moments later.*

As we will see, this definition will be crucial to evaluate the performance of the various methods.

The problem of detecting outliers is a well-known problem, with vast fields of applications. For example, it can be useful to detect damages in industrial plants, network intrusion attacks or financial frauds. In all these cases the base idea is to detect data points that do not conform with the general distribution of the dataset. A certain measure of *non-conformity* is used to register the likelihood of a certain point of being an outlier. This measure is called *anomaly score* or *outlier score*. Thus, as detailed by [14], the anomaly detection procedure can be modeled as a function ϕ:

$$\phi : \mathbb{R}^n \to \mathbb{R}$$

$$\phi(x) \to \gamma$$

where γ is the outlier score, $x \in X \subseteq \mathbb{R}^n$, and X is the dataset. Given a continuous outlier score, to decide if a certain point is an outlier or not, a threshold δ is used:

$$\phi_{binary} : \mathbb{R}^n \to \{outlier, normal\}$$

$$\phi_{binary}(x) \hookrightarrow \begin{cases} outlier, & \text{if } \phi(x) > \delta. \\ normal, & \text{otherwise.} \end{cases}$$

Given this general approach, the whole research carried out in this work, and similar ones, focuses on the best method to calculate the outlier score and to decide the level of the threshold. This is a non-trivial task: for example if one tries to use a general binary classification method on a unbalanced dataset, where 99.9% of the point are not anomalous, the classifier could achieve 99.9% accuracy by just reporting every point as normal. This is obviously not wanted, as this classifier did not learn anything from the data and is completely useless.

Even when the model is actually learning from the data, it is not easy to decide where to set the threshold. For example, in medical applications, where not detecting an outlier could mean not diagnosing a serious disease, it may be better to have a low threshold to reduce the number of outliers not being detected. This will lead to a greater number of false anomalies, which can be investigated further, thus reducing the overall risk. In other cases, it may be better to reduce the number of normal points classified as outliers, while it may be acceptable not to detect some points. This means that there is no single best method or parameter value which works on all the cases, but the best solution depends on the application.

3.2.2 Taxonomy and classes of algorithms

Depending on the interpretation that each author gives to the definition in section 3.2.1, there exists different ways to measure anomaly. The only thing that brings together all the approaches is the use of an Anomaly Score (AS) assigned to each point. This score should serve as a proxy of the probability to be an outlier. Obviously, each method assigns a different anomaly score to the same point since it is based on different detection strategies.

A first classification of AD methods divides such methods into two categories:

- *Model-based* - It is the most traditional Anomaly Detection category and employs a predefined model that describes the normal or *all* the possible anomalous operating conditions. These approaches usually rely on physics or domain-knowledge heuristics; unfortunately they are often unfeasible and costly to be developed since they require extended knowledge of the system under exam.

- *Data-driven* - The approaches examined in this thesis rely instead on data availability. More precisely, such approaches make use of two ingredients: data sampled from the analyzed system and algorithms able to automatically learn the abnormality level of those data. Such category of approaches is often referred as data-driven anomaly detection and its advantages are the great flexibility and absence of strong assumptions that limit the model applicability.

Additionally, when looking at anomalous behaviours, two problem settings can be defined: the supervised and the unsupervised one. The former consists in classifying data, based on previously tagged anomalies. It is called supervised since training data are collected from sensor measurements and have an associated label that identifies them as anomalous or not; in industrial context, the supervised scenario is typically named Fault Detection. Unfortunately, supervised settings are seldom available in reality [18]: labelling procedures are very time consuming and typically require domain experts to be involved.

On the contrary, the unsupervised scenario is the most common in real world applications. In this case, data are not equipped with labels and therefore the learning algorithm lacks a ground truth of what is anomalous and what is not. Given that, the goal of the algorithm is to highlight the most abnormal data, assigning to each one an Anomaly Score (AS). In this chapter the focus will be on the unsupervised setting since it is the most applicable in real-world scenarios. To be more precise, the unsupervised setting can be further divided into two sub-categories, based on the nature of the available data. The fully unsupervised one relies on training set composed of both anomalies and normal data. However, in some applications obtaining training data with anomalies is quite complex, therefore in such cases data are composed only of normal instances: in this scenario semi-supervised approaches, sometimes named one-class setting, are the most natural ones to be adopted.

To achieve good results it's important to develop techniques which take into account the fact that labels are unbalanced and, in general, exploit all the known proprieties of the data. The anomaly detection problem is defined

- *static* if the analysis is performed on time-independent data (*static* datasets, where the order of the observations do not matter).
- *dynamic* if it is performed on time-dependent data (time-series data or dynamic datasets).

Working on time-series requires keeping into account the time correlation of the points. For example, an outlier on spacial data (such as an image) can be detected just by looking at low density regions or its distance to the other points, but this approach cannot be applied directly to the points in a time series because it does not keep into account the sequentiality of data. Having multiple streams of data recorded concurrently and derived from correlated signals in the real-world means that different PVs could be affected by the same underlying process. It's important thus to keep into account the correlation between different signals, in order to fully exploit their information. Furthermore, the sampling method, its frequency and precision, can influence the pre-processing phase of the analysis as well as the precision of the results. For example a noisy source could hide important information. Finally, in the context of anomaly detection it's usually difficult to obtain a labelled dataset due to the high amount of effort required to produce one, and so it's necessary to use unsupervised methods, able to learn the overall distribution of the normal points and to highlight the ones not conforming to it.

One of the most common approaches to deal with time correlation in time-series is to use sliding time windows. Instead of treating each point as independent, groups of subsequent points are grouped together and some properties, or *features*, are calculated on the points belonging to the group. These features now include the information about the time correlation, as they were calculated on subsequent point, and can be seen as a new dataset. The new points belonging to this dataset are independent between one another and it is possible to use generic outlier detection methods on them to find anomalous time windows. When this approach is used the width of the sliding windows, the number of point to include in each window, is an important design parameter which must be chosen carefully.

Another very basic discrimination is between univariate or multivariate anomaly detection, meaning that the analysis is performed on a single variable or on multiple variables at the same time. A more subtle distinction concerns the way in which the algorithm training is performed. The most traditional one is the batch training where the model is trained only once, using all the available training data. This approach might be unpractical when the dataset is too large to fit into the memory, or when sampled data do not cover sufficiently well the normal operating domain: in this case the model needs to be continuously updated as new data are collected; this is even more important in situations where the normal data distribution undergoes a drift, and samples that were used in the first part of the training now become anomalous. Therefore, in such case, the model has to adapt by learning the new configuration and by forgetting the outdated distribution.

Given these different formulations and requirements for the anomaly detection problem, many data-drive approaches have been proposed in literature over the years. These are usually divided into two categories: *classical* Machine Learning (ML) and Deep Learning (DL) models. The first ones are based on the use of classical machine learning algorithms, such as Support Vector Machine (SVM) or K-Nearest Neighbour (KNN), while the second ones are based on the use of Neural Networks (NNs). In fact, in the last decade DL methods and NNs have become popular in many fields and have spurred a variety of different models and architectures, which have been applied also to the AD problem.

Classical ML approaches can be further categorized into 5 classes. The distinction between classes is not strongly fixed, and some methods could be categorized at the same time in different classes. The most intuitive class of algorithms is the *distance-based* one. It is based on the assumption that outliers are spatially far from the rest of the points [3]. Also the *density-based* class is quite intuitive since assumes outliers living in rarefied areas [51]. The *statistics-based* ones are conceptually simple, but often make use of heavy assumptions on the distribution that generated training data [68]. *Clustering-based* employ clustering techniques in order to find clustered data, moreover, they are strongly susceptible to hyperparameters and often rely on density or distance measures [72]. Unfortunately, these approaches are expensive to compute or rely on too strong assumptions. Density and distance-based methods are hard to compute especially in high dimensional settings and when many data are available; such approaches are hardly applicable in scenarios where detection has to be performed online and on fast evolving data streams. Moreover, statistical methods are often restricted to ideal processes, rarely observed in practice.

Quite recently a new class of methods emerged: the *isolation-based.* This class is very different from the previous ones: it assumes that outliers are few, different and, above all, easier to be separated from the rest of data. This draws the attention from normal data to anomalies and allows to obtain much more efficient anomaly detectors. The primary goal of these methods is to quickly model the anomalies by isolating them, rather than spending resources on the modeling of the normal distribution. The seminal, and most popular, approach in this class is the Isolation Forest algorithm [91].

The following sections will present some of the most popular algorithms both from the machine learning and deep learning approaches. A complete section (section 3.2.5) is dedicated to the Isolation Forest algorithm and its extensions, as it is the focus of a review of the literature carried out during the PhD [10].

3.2.3 Performance Metrics

To evaluate and compare the performances of different design choices, it is necessary to establish common metrics. In the field of classification, the metrics are often based on 4 parameters:

1. True Positive (TP): number of points correctly classified as positives. In this case positive means outlier, negative means normal.
2. True Negative (TN): number of points correctly classified as negatives.
3. False Positive (FP): Number of negative points incorrectly classified as positives.
4. False Negative (FN): Number of positive points incorrectly classified as negatives.

In the case of unbalanced datasets, it would be useless to count the number of True Negatives as they would be some order or magnitude bigger than the other metrics, thus being incomparable. For this reason the precision and recall metrics are used:

$$precision = \frac{TP}{TP + FP}$$

$$recall = \frac{TP}{TP + FN}$$

Given a classification, the precision is a measure of how many points labeled as positives are well-classified. The recall instead indicates the proportion of real positives that have been detected. These are strictly correlated: when maximizing one of the two, the other is usually affected. Since it is advisable to maximize both of them, seeking the best trade-off for the application, a more compact metric can be used, the F-score:

$$F_\beta = (1 + \beta^2) \cdot \frac{precision \cdot recall}{\beta^2 \cdot precision + recall} = \frac{(1 + \beta^2) \cdot TP}{(1 + \beta^2) \cdot TP + \beta^2 \cdot FN + FP}$$

The β parameter is used to express the importance given to the recall respect to the precision, so it can be used to optimize the trade-off for the application. Usual values for β are 2, when recall is more important than the precision, and 0.5 when the precision is considered more important.

3.2.4 Machine Learning Methods

This section groups all those approaches based on classical machine learning methods which usually were originally developed for different purposes. With some adaptations they have proven successful for the task of anomaly detection.

K-Means Clustering

K-means [100] is a well know unsupervised clustering method, where data is grouped into k clusters, with the aim of minimizing the sum of the squared distances between each point and its closed center, also called centroid. Initially the centroids are chosen randomly, then each point is assigned to the cluster of the closest centroid and finally the centroids are computed again as the mean of the cluster. This repeated iteratively until convergence is reached.

This method has been adapted to work for anomaly detection in time-series by using a sliding windows approach. Given the time-series of N points $(x_1, x_2, ..., x_N)$, the windows length w and the sliding parameter γ, it is possible to extract the subsequences relative to each position of the sliding window:

$$S = (x_0, ..., x_w)^T, (x_{0+\gamma}, ..., x_{w+\gamma})^T, ..., (x_{N-w}, ..., x_N)^T$$

Given the design parameter k, the number of clusters, it is now possible to run the k-means algorithm on the set S. For each subsequence the final distance from the center of the cluster can be used as the outlier score. This method is also called *Subsequence time-series Clustering* (STSC) [67]. It requires to accurately set the k, w and γ parameters, as well as the distance function to use (usually the euclidean distance).

Local Outlier Factor (LOF)

Local Outlier Factor (LOF)[16] is an algorithm designed to detect anomalies on spatial data. The basic idea is to approximate the local density of each point based on the distance of its k nearest neighbours and to identify the outliers by finding the points whose local density is much lower than their neighbours. Figure 3.3 shows that anomalous (isolated) data points have higher LOF scores.

Formally, let's define the $k - distance(x)$ as the distance of the point x to its k-th nearest neighbour. Then the set:

$$N_k(x) = \{y | d(x, y) \leq k - distance(x)\}$$

contains all points in the k-distance neighbourhood. We can now define the reachability distance RD:

$$RD_k(x, y) = max\{k - distance(y), d(x, y)\}$$

which is the true distance of the two points, but at least k-distance of y. This means that all points belonging to the k-nearest neighbourhood of y $N_k(y)$ are considered to be equally distant. Then, it is possible to compute the *Local Reachability Density* (LRD) of x as:

$$LRD_k(x) = 1 / \left(\frac{\sum_{y \in N_k(x)} RD_k(x, y)}{|N_k(x)|} \right)$$

which is the inverse of the average reach ability distance RD from its neighbours. This is then compared with the LRD of the neighbours with the following LOF metric:

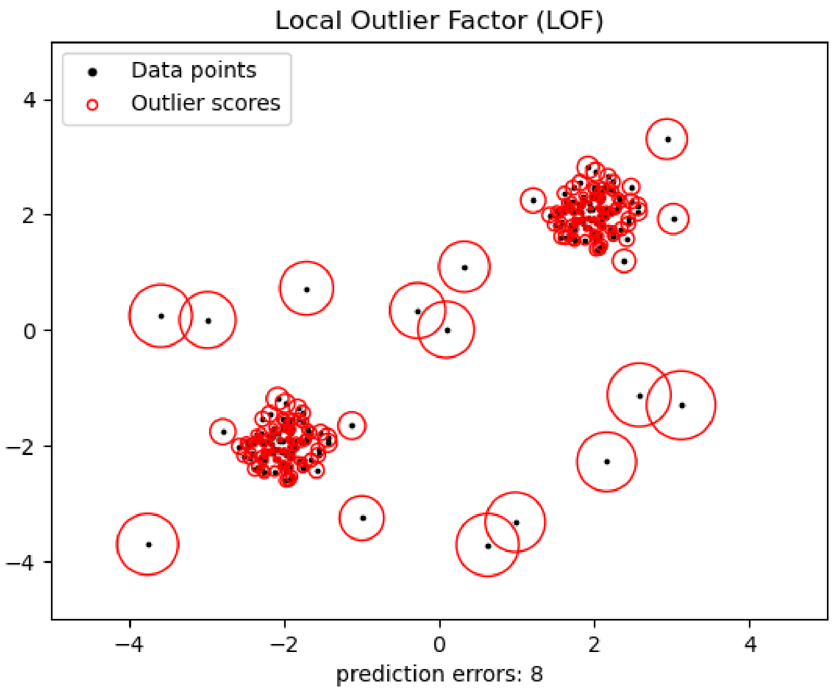


Figure 3.3: LOF score is higher on isolated data points. Source: [84]

$$LOF(x) = \frac{\sum_{y \in N_k(x)} \frac{LRD_k(y)}{LRD_k(x)}}{N_k(x)}$$

which is the average local reachability density of the neighbours over the point own local reachability density. Three cases are then possible:

- $LOF \sim 1$: similar density as neighbours
- $LOF << 1$: higher density
- $LOF >> 1$: lower density (outlier)

This methods has been adapted to work for time series by Oehmcke et al. [122]. They proposed using a sliding windows approach, with the original time-series X_T divided into a training set A and a test set B. The training set is split into a set of subsequences as follows:

$$A(w,t) \subseteq P(\{X_T\}) = \{(x_i, ..., x_{i+w}) | i \in \{1, ..., t-w\}, t \leq |\{X_T\}|\}$$

while $B(w,t)$ is the sequence to be analyzed:

$$B(w,t) \subseteq \{X_T\} = \{x_{t-\frac{w}{2}}, ..., x_{t+\frac{w}{2}}\}$$

It is not possible to compute the LOF value of $\{\{B(w,t)\} \cup A(w,t)\}$ which is a measure of how different $B(w,t)$ is from sets in $A(w,t)$. This can thus be used as an outlier score.

The main challenges with this method are related to the choiche of the different parameters, in particular the k parameter indicating the number of neighbours to consider and the distance function. Another important aspect is the execution time, which could become a stringent factor with large datasets. This algorithm has complexity $O(n^2)$, which is not always ideal.

Cluster Based Local Outlier Factor (CBLOF)

Cluster Based Local Outlier Factor (CBLOF) [59] model borrows ideas from both the clustering and LOF methods. It run a clustering algorithm and classifies the resulting clusters into small and large clusters. The outlier score is then calculated based on the size of the cluster the point belongs to as well as the distance to the nearest large cluster.

A formal definition of *large* and *small* clusters is required. Given a set of clusters $C = \{C_1, C_2, ..., C_k\}$ ordered by non-increasing element count so that $|C_i| \geq |C_{i+1}|, 1 \leq i < k$, and two parameters α and β the boundary b between large and small clusters must follow:

$$(|C_1| + |C_2| + ... + |C_b|) \geq N \cdot \alpha$$
$$\frac{|C_b|}{|C_{b+1}|} \geq \beta$$

where N is the total number of points. Then the large clusters are the one with index lower than b, while the others are considered to be small. The first formula ensures that the large clusters include together at least a fraction α of the total number of points, while the second requires that the small clusters are at least β times smaller than the large ones.

It is now possible to define the CBLOF metric for a certain point x:

$$CBLOF(x) = \begin{cases} |C_i| \cdot min(d(x, C_j)), & x \in C_i, C_i\ small, C_j\ large. \\ |C_i| \cdot min(d(x, C_i)), & x \in C_i, C_i\ large. \end{cases}$$

This value depends both on the dimension of the cluster of the point and on its distance from the nearest large cluster, or the center of the cluster it belongs to if it is large. This value can then distinguish between points inside large clusters and the one belonging to small remote clusters, which are to be considered outliers.

To adapt this method to work on time series it is possible to adopt similar techniques as the previous models, such as sliding windows.

3.2.5 Isolation tree based methods

Tree-based methods, as suggested by the class name, rely on tree structures where the domain of the available data is recursively split in a hierarchical way, in non-overlapping intervals named leaves. In the Anomaly Detection context, these models are seen as a tool rather than a separate detection approach. As a matter of fact, they are employed in both density-based and isolation-based approaches. The former approach is perhaps the most intuitive since at high densities one expects normal data clusters, vice-versa in low density regions. However this approach is in contrast with the simple principle of never solving a more difficult process than is needed [126]. Density estimation is a computationally expensive task since it focuses on normal data points, that are the majority. However, the ultimate goal of anomaly detection problem is to find anomalies, not to model normal data. Outliers are inferenced only at a second step. On the contrary, the isolation approach is less simple to formulate but also less computationally expensive. It directly addresses the detection of anomalies since points that are quickly isolated are more likely to be outliers.

The literature concerning anomaly detection using tree-based methods is quite vast, but we decided to focus on methods that naturally apply to the unsupervised setting, due to its relevance in industry. Not only we decided to exclude the supervised approaches, but also we excluded the ones that artificially create a second class of outliers like in [32]. These approaches often try to fit supervised models into unsupervised settings at cost of inefficient computations, especially at high dimensionality.

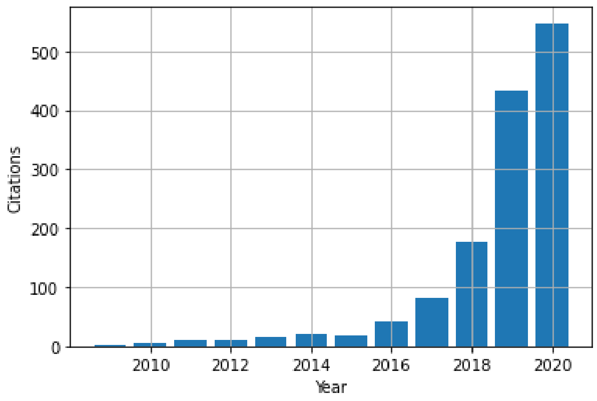


Figure 3.4: Combined citations of IF original paper [91] and its extended version [93] by the same authors. Source: Scopus. Retrieved on the 30th of March 2021.

Isolation forest (IF) is the seminal algorithm in the field of isolation tree-based approaches and it was firstly described in [91]: in recent years IF has received an increasing attention from researchers and practitioners as it can be noted in figure 3.4, where the evolution of citations of the algorithm in scientific papers has increased exponentially over the years.

Isolation Forest

As the name suggests, IF is an ensemble algorithm that resembles in some aspects the popular Random Forest algorithm revised in the unsupervised anomaly detection settings. Indeed, IF is a collection of binary trees: while in the popular work of Breiman [15] we are dealing with decision trees, here the ensemble model is composed by *isolation trees*, that aim at isolating a region of the space where only a data point lies. IF is based on the idea that, since anomalies are by definition few in numbers, an isolation procedure will be faster in separating an outlier from the rest of the data than when dealing with inliers.

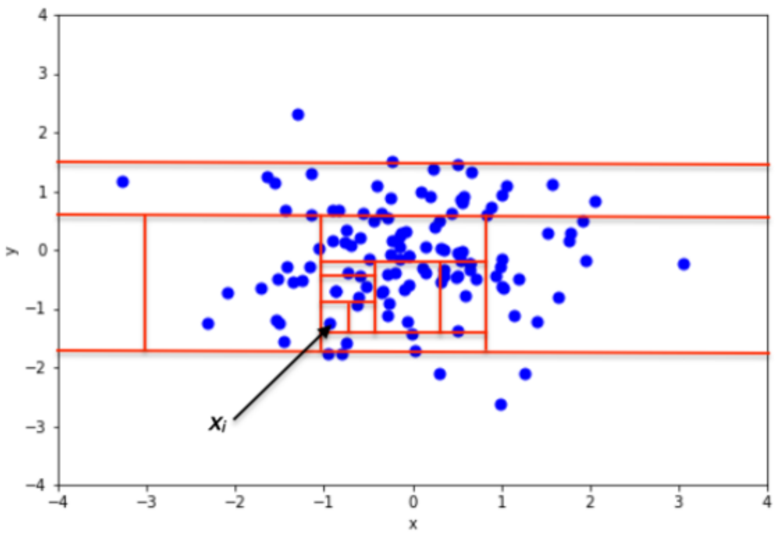


Figure 3.5: Isolation Forest recursive splitting. Source: [171]

More in details, the algorithm consists in two steps: training and testing. In the training phase, each isolation tree recursively splits data into random partitions of the domain. As said, the core idea is that anomalies on average require less partitions to be isolated, as can be seen in figure 3.5. Therefore inliers live in a leaf in the deepest part of the tree, while outliers in a leaf close to the root. More formally, the anomaly score is proportional to the average depth of the leaf where each datum lies. For the sake of clarity, we report the training and testing pseudo-codes (Algorithm 1, 2 and 3 - adapted from [91]).

Algorithm 1: IsolationForest(X, n, ψ)

```
Input: $X$ – data in $\mathbb{R}^d$, $n$ – number of trees, $\psi$ – sample size
Output: list of Isolation Trees
$forest \leftarrow$ empty list of size $n$;
$h_{max} \leftarrow \lceil \log_2 |X| \rceil$;
for $i = 1$ to $n$ do
    $\hat{X} \leftarrow sample(X, \psi)$;
    $forest[i] \leftarrow IsolationTree(\hat{X}, 0, h_{\max})$;
end
return forest
```

Algorithm 2: IsolationTree(X, h, h_{max})

```
Input: $X$ – data in $\mathbb{R}^d$, $h$ – current depth of the tree, $h_{\max}$ – depth limit
Output: Isolation Tree (root node)
if $h \geq h_{max}$ or $|X| \geq 1$ then
    return Leaf{
        $size \leftarrow |X|$
      };
else
    $q \leftarrow$ randomly select a dimension from $\{1, 2, \dots, d\}$;
    $p \leftarrow$ randomly select a threshold from $[\min X^{(q)}, \max X^{(q)}]$;
    $X_L \leftarrow filter(X, X^{(q)} \geq p)$;
    $X_R \leftarrow filter(X, X^{(q)} < p)$;
    return Node{
        $left \leftarrow IsolationTree(X_L, h+1, h_{max})$,
        $right \leftarrow IsolationTree(X_R, h+1, h_{max})$,
        $split\_dim \leftarrow q$,
        $split\_thresh \leftarrow p$
      };
end
```

The training phase starts subsampling the dataset composed of n data points, in t randomly drawn subsets of ψ samples. Then, for each subset a random tree is built. At each node of the random tree a feature is uniformly drawn. The split point is uniformly sampled between the minimum and maximum value of the data along the selected feature, while the split criterion is simply the inequality w.r.t the feature split point. The splitting procedure is recursively performed until a specific number of points are isolated or when a specific tree depth is reached. In principle, the full isolation tree should grow until all points are separated, unfortunately risking to grow trees with depth close to $n - 1$. However data that lie deep in the tree are the normal ones, not the target of the detector. For efficiency reasons, this is not practical and since anomalies are easy to be isolated, the tree only needs to reach its average depth $\lceil \log_2 n \rceil$.

Algorithm 3: PathLength(x, T, h)

```
Input: $x$ – instance in $\mathbb{R}^d$, $T$ – node of IsolationTree, $l$ – current length (to be
       initialized to 0 when first called on the root node)
Output: path length of x
if $T$ is a leaf node then
    return $h + c(T.size)$;
end
$q \leftarrow T.split\_dim$;
if $x^{(q)} < T.split\_tresh$ then
    return $PathLength(x, T.left, l+1)$;
else
    return $PathLength(x, T.right, l+1)$;
end
```

The testing phase is different and consists in checking the depth $h(\cdot)$ reached by the data point x in all the isolation trees, and taking the average.

The anomaly score $s(x, n)$ is defined as:

$$s(x, n) = 2^{-\frac{E(h(x))}{c(n)}}$$

where $c(n)$ is a normalizing factor and $E(h(x))$ is the average of the tree depths. Note that when x is an anomaly, $E(h(x)) \to 0$ and therefore $s(x) \to 1$, while when $E(h(x)) \to n-1$, $s(x) \to 0$. When $E(h(x)) \to c(n)$, $s(x) \to 0.5$.

The computational complexity of IF is $O(t\psi \log \psi)$ in training while $O(nt\psi \log \psi)$ in testing, where we recall that ψ is the subsampling size of the dataset. It is interesting to note that in order to have better detection results, ψ needs to be small and constant across different datasets.

Isolation forest has many advantages compared to the methods belonging to other classes. Firstly it is very intuitive and requires a small amount of computations. For this reason it is particularly suited for big datasets and for applications where low latency is a strict requirement. The use of random feature selection and bagging allows to efficiently handle high dimensional datasets. In addition, the use of tree collections makes the method highly parallelisable. Unfortunately the algorithm has some issues. The most severe is the *masking effect* created by the axis parallel partitions and anomalous clusters, that perturbs the anomaly score of some points. A closely related issue is the algorithm difficulty to detect the local anomalies.

Trying to make a summary: the standard isolation forest defines anomalies as few and different, and approaches their detection not modelling normal data but trying to separate them as fast as possible with the aid of bagged trees. The split criterion is based on randomly selected feature and split point, that create axis-parallel partitions. These characteristics will be challenged by the following methods but the structure of the algorithms will remain quite the same.

In the following subsections the focus will be on static approaches (Section 3.2.5), dynamic (Section 3.2.5), distributed (Section 3.2.5) and finally interpretable and feature selection methods (Section 3.2.5).

Static Learning

Static learning methods can be generally divided into two sub-categories, using two approaches. The first one groups i) methods that directly originate from the seminal work [91] slightly modifying the Isolation Forest, and ii) methods that start from a different but similar algorithms like the Half-Space (HS) trees or Random Forests (RF). The former group focuses on the importance of fast isolation, while the latter on the density approximation. The second grouping approach subdivides the static methods based on how is computed the anomaly score. The majority relies on the mean leaf depth but a growing number of algorithms employs some variation of the leaf mass.

SCiForest SCiForest [92] takes the assumptions of IF and tries to improve it, with special attention to clustered anomalies. Indeed Isolation Forest performs quite poorly on them. The two most important novelties that this method introduces are: i) the use of oblique hyper-planes, instead of axis-aligned, and ii) the use of a split criterion that replaces the random split. At each partition multiple hyper-planes are generated, but only the one that maximises a certain criterion is selected.

The intuition behind this criterion is that clustered anomalies have their own distribution and the optimal split separates normal and anomalous distributions minimizing the dispersion. Therefore the split criterion is formalized as:

$$\mathrm{Sd}_{\mathrm{gain}} = \frac{\mathrm{std}(X) - \mathrm{average}\big(\mathrm{std}(X^{\mathrm{left}}), \mathrm{std}(X^{\mathrm{right}})\big)}{\mathrm{std}(X)}$$

where $\mathrm{std}(\cdot)$ computes the standard deviation.

Due to the new computations, the complexity of the SCiForest increases reaching $O(t\tau\psi(q\psi + \log\psi + \psi))$ in the training stage, and $O(qnt\psi)$ in the evaluation stage, where t is the number of trees in the forest, τ the number of hyper-plane trials and q the number of features composing each hyper-plane dimensions.

ReMass Isolation Forest ReMass IF [8] starts from quite similar premises to the SCiForest's, but focuses on the poor performances of IF on local anomalies. Unlike SCiForest, it does not suggest to modify the training algorithms but the anomaly score: it proposes to substitute the tree depth with a new function, the *relative mass*.

The mass of a leaf $m(\cdot)$ is defined as the number of data points inside the leaf while the relative mass of the leaf is the ratio between the mass of the parent and the mass of the leaf. More precisely, the anomaly score for each tree is defined in this way:

$$s_i(x) = \frac{1}{\psi}\frac{m(X_{\text{parent}})}{m(X_{\text{leaf}})}$$

Note the authors suggest to modify only the anomaly score formula, keeping the rest of the algorithm untouched. This helps improving the anomaly score, while preserving the low computational complexity.

The time and space complexities are the same as IF.

Extended Isolation Forest (EIF) In the paper [57] the authors observe the masking effect created in the IF algorithm by the axis-aligned partitions. The intersection of multiple masks can even create some fake normal areas of the domain, leading to completely wrong anomaly detection. In order to overcome the described issue, the authors suggest a very simple but effective strategy already employed in SCiForest: the use of oblique partitions. However in this case the authors do not use a repeated split criterion, loosing its benefits but also the additional computational overload.

The time complexity is similar to the SCiForest, except for the saving of the τ repetitions.

LSHiForest The algorithm presented in [178] combines the isolation tree approach with the Locality Sensitive Hashing (LSH) forest, where given a certain distance function d, neighbours samples produce the same hash with high probability while samples far from each other produce the same hash with low probability. The probabilities can be tuned by concatenating different hash functions, so that an isolation tree can be constructed by concatenating a new function at each internal node. The path from the root to a leaf node is the combined key of the corresponding data instance. Since d is generic, this extension allows to incorporate any similarity measure in any data space. Moreover, the authors show that their framework easily accommodates IF and SCiForest when particular hash functions are selected. They adapted the method in this way: i) the sampling size is not fixed but variable, ii) the trees are built using the LSH functions, iii) the height limit and the normalisation factor are changed consequently and iv) the individual scores are combined after the exponential rescaling. The average-case time complexity in the training stage is $\Theta(\psi \log \psi)$, while in the evaluation phase it is $\Theta(\log \psi)$.

usfAD The work presented in [7] addresses the issue of different units/scales in data, starting by showing some examples where different non-linear scales lead to completely different anomalies. To solve this issue the authors propose *usfAD*, a method that combines Unsupervised Stochastic Forest (USF) with IF, and naturally born for the semi-supervised task. This hybrid model recursively splits the subsample until all the samples are isolated. However it is different from the IF since it grows balanced trees with leaf of the same depth. This is accomplished using a splitting rule that uses the median value as split point. The core idea is that the median, since relies on ordering, is more robust to changes in scale or units. After the tree growth, normal and anomalous regions are associated to each node: the former consists in the hyper-rectangle containing the training points, while the latter is the complementary region. All these modifications lead to a quite different testing phase. The anomaly score of a test point is the depth of the first node where it falls outside the normal region.

The time complexity is slightly higher than IF: the training is $O(nth + t2^h d)$ while the testing $O(t(h + d))$. Moreover, it needs $O(t2^h d)$ memory space.

PIDForest Classical IF relies on the concept of isolation susceptibility, which intuitively can be outlined as the average number of random slices that are needed to fully isolate the target data. This definition of anomaly has some great advantages, but also some pitfalls. In particular, in high dimensional data, many attributes are likely to be irrelevant and isolation may be sometimes very demanding.

PIDForest [52] is based on an alternative definition of anomaly. The authors assert that an anomalous instance requires less descriptive information to be uniquely determined from the other data. Then, they define their partial identification score (PIDScore) in a continuous setting as a function of the maximum sparsity over all the possible cubical subregions containing the evaluated data point x. Say X full data, and C a subcube of the product space and ρ a sparsity measure, PIDScore can be formalized as

$$\text{PIDScore} = \max_{C \ni x} \rho(X, C) = \max_{C \ni x} \frac{\text{vol}(C)}{|C \cap X|}$$

PIDForest builds a heuristic that approximates the PIDscore. The strategy is to recursively choose an attribute to be splitted in k intervals, similarly to k-ary variants of IF (authors suggest default hyperparameter $k < 5$). Intuitively, we would like to partition the space into some sparse and some dense regions. For this purpose, a possible objective is to maximize the variance over the partitions in terms of sparsity, that can be treated as a well-studied computational problem related to histograms and admits efficient algorithms for its solution. For each attribute the optimal splits are computed and the best attribute is chosen as coordinate for partitioning. Then, the iteration is repeated on each partition, until a data point is fully isolated or a maximum depth parameter is exceeded. Now the resulting leaf is labelled with the sparsity of the related subregion. In the testing phase, a data point can be evaluated on each tree of the PIDForest and the maximum score (or a robust analog, like 75% percentile) gives an estimate of the PIDScore.

In the words of its authors, the fundamental difference between IF and PIDForest is that the latter zooms in on coordinates with higher signal, being less susceptible to irrelevant attributes at the cost of more expensive computation time. Each

PIDTree takes $O(k^h d\psi \log \psi)$ as training time, while testing is pretty much equivalent to IF.

Dynamic Datasets

Dynamic datasets are made up of infinite data streams [162]. This poses new challenges that previously described methods cannot tackle. The most simple challenge is the continuous training: since the stream is infinite, the training data may be insufficient to fully describe it. In order to do that, the model should continuously learn from the incoming data stream, and at the same moment detect anomalous points. The most complex setting is represented by the distributional drift. In this case, the stream is not stationary and its distribution experiences time-dependent variations. Here, the model must adapt to the evolving data stream, but at the same time discern anomalies from new normal data points.

The weak point of these methods is the assumption about the rarity of anomalies. If they are too numerous, they can be confused with a change in the normal distribution of data points, leading to an erroneous adaptation of the model. Doing this the model will consider them as normal and it will not raise the necessary alarm.

iForestASD The method proposed in [36] is an adaptation of the original algorithm to the streaming settings. It is very simple: it splits the stream in windows, and checks each window to detect anomalies. If the ratio between normal data and anomalies is too high (exceeds the expected anomaly ratio), it assumes a concept drift is happening. In this case the model is re-trained on the new window. Obviously, the threshold on the anomaly ratio and the width of the window are delicate hyper-parameters that highly depend on the specific application.

Streaming HS In the paper [162] an adaptation of [163] to data streams is presented. Unlike other tree based approaches, it is not built starting from training data but its structure is induced only by the feature space dimension. It doesn't need split point evaluation, and therefore it is fast and able to continuously learn from new data. Contrary to the basic Isolation Forest, this method employs the concept of mass to determine the anomaly score. In practice, this method works segmenting the stream in windows, and working with two of them: *reference* and the *latest* windows (despite their name, they are immediately consecutive). The reference window is used to record the mass profile, while the latest one is used for testing. When this is done, the latest becomes the reference and a new mass profile is recorded. The authors show a time complexity of $O(t(h + \psi))$ in the worst case.

Streaming LSHiForest The model presented in [157] extends the LSHiForest algorithm for streaming data exploiting a dynamic isolation forest. The procedure can be split into three main phases: i) a dataset of historical data points is used to build a LSHiForest data structure, as presented in the original paper, then ii) the data points collected from multiple data streams are preprocessed to find "suspicious" samples, which are outlier candidates. Finally, iii) the suspicious data are updated into the LSHiForest structure and the anomaly scores of the updated data points are recalculated. To effectively extract suspicious points from the streaming data, Principal Component Analysis (PCA) and the weighted Page-Hinckley Test (PHT) are applied to a sliding window, to cope with the challenges of high dimensionality and concept drift. An update mechanism is proposed to iteratively update the LSHiForest by replacing the previous data points observed on a stream with suspicious ones.

Distribuited Approaches

Wireless Senor Networks (WSNs) pose new and more challenging constraints to Anomaly detection. Indeed sensor nodes are usually quite cheap but have multiple constrains on energy consumption, communication bandwidth, memory and computational resources. Moreover they are often deployed in harsh environments that can corrupt sensor measurements and communication [37]. Despite the distributed nature of the network, Anomaly Detection on such applications should minimize the communication burden as much as possible, since data transmission is the most energy intensive process.

Distributed Isolation for WSN The authors of [37] suggest the adaptation of IF to this distributed problem, considering the spatial correlation between neighbor sensor nodes in a local and global manner. They chose this base algorithm due to its already mentioned properties, that fits perfectly in this settings. However in the WSN context, data can be anomalous w.r.t. the single sensor node or w.r.t. the whole network. The local detector consists in a collection of isolation trees trained on a group of neighbouring nodes while the global one is made up of local detectors. When an anomaly is locally detected, it is marked as an error if is not detected by neighbor sensors, otherwise it is considered an event.

The space and time complexity is $O(km)$, where k is the number of trees on a local node, and m the number of leaves.

Optimal Weighted One-Class Random Forests (OW-OCRF) A similar approach has been exploited in [165], where a one-class random forest has been chosen as base detector. Here each sensor node builds his own model, but it is also augmented with the models belonging to neighbouring devices. In addition, a strategy to weight the most effective neighbor models has been implemented, based on the minimization of the model uncertainty. Uniform voting is reasonable in circumstances where all the learners arise from the same distribution, but when models come from heterogeneous data distributions this strategy shows its weaknesses. Larger weights are assigned to trees that are in accordance with the majority, while trees that increase the overall uncertainty are penalized. The optimization of these weights is performed in a fully unsupervised fashion. In presence of distributional drifts, the overall model can be easily adapted to the new conditions, optimising new weights or substituting the trees with lower weight importance. The communication between the node is employed just at early stages for the sharing of the detecting models, not for the sampled data sharing.

The time and space complexity of this approach are $O(th)$ and $O(t2^{h+1})$ respectively.

Interpretability and Feature Selection

The detection of anomalies is an important activity in manufacturing processes but it is useless if a corresponding action does not take place. That action is expected to be proportional to the gravity of the anomaly (encoded by the anomaly score), and to the *cause* that generated it. For doing that a tool to interpret that anomaly is needed. It is easy to understand that if unsupervised anomaly detection is challenging, interpretable models face even more complex issues. In real word scenarios anomalies are unlabelled and lack of proper interpretations.

Moreover [20] observes that interpretable models enhance the trust of the user in the anomaly detection algorithms, leading to a more systematic use of these tools.

Depth-based Feature Importance for the Isolation Forest (DIFFI) IF is a highly randomised algorithm and therefore the logic behind the model predictions is very hard to grasp. In the paper [20], a model specifically designed for the interpretability of IF outcomes is presented. In particular the authors developed two variants: i) a global interpretability method able to describe the general behaviour of the IF on the training set, and ii) a local version able to explain the individual IF predictions made on test points. The central idea of this method, named DIFFI, relies on the following two intuitions: the split of an important feature should a) induce faster isolation at low levels (close to the root) and b) produce higher unbalance w.r.t splits of less important features. This is encoded in a new index named *cumulative feature importance.* With this in mind, the authors formulate the global feature importance as the weighted ratio between the cumulative feature importance computed for outliers and inliers. The local interpretation of single detected anomalies is sightly different but relies on the same intuitions. DIFFI can also be exploited for unsupervised feature selection in the context of anomaly detection.

Anomaly explanation with random forests Authors in [80] developed an algorithm able to explain the outcome of a generic anomaly detector by using sets of human understandable rules. More specifically, the proposed model consists in a special random forest trained to separate the single anomaly from the rest of the dataset. This algorithm provides two kinds of explanations: the *minimal* and the *maximal*. The first is performed isolating the anomaly using the minimal number of necessary features. On the contrary the maximal explanation looks for all the features in which the anomaly is different, employing a recursive feature reduction. Once the forest are trained and the explanations are obtained, the decision rules are extracted in a human readable manner.

The time complexity of the algorithm is $O(n_t T_{\text{sel}} T_{\text{train}})$ where T_{sel} is linear with the number of normal samples in the data. For the minimal explanation n_t is the number of trees trained for each anomaly and $T_{\text{train}} = O(d|T|^2)$, where d is the number of features and T is the size of the training set. For the maximal explanation $n_t = O(d-1)$ while $T_{\text{train}} = O(d^2|T|^2)$.

3.3 Deep Learning Techniques

Deep learning techniques are the natural evolution of machine learning models. The distinct feature is the use of Neural Networks (NNs) with multiple hidden layers.

Neural Networks have long been proposed in literature, with seminal works dating back to 1943 [113]. In 1957 Frank Rosenblatt introduced the perceptron, a single layer neural network that was able to learn simple patterns [139]. However, the perceptron was not able to learn the XOR function, a simple non-linear function. This lead to a decline in the interest in neural networks, until 1980s when the backpropagation algorithm was introduced [169, 140]. This algorithm allowed to train multi-layer neural networks, and the interest in this field started to grow again. Between 1989 and 1998, Yann LeCun proposed the CNN architecture [85, 86], which was able to achieve state-of-the-art results in image classification tasks. This lead to a new wave of interest in neural networks, and in 2006 Geoffrey Hinton proposed the Deep Belief Nets (DBN) [61], a multi-layer neural network with a special architecture that allowed to train it layer by layer, making the training process much faster. In 2012, Alex Krizhevsky won the ImageNet competition [81] using a CNN architecture, and this was the starting point of the deep learning revolution. Since then, deep learning techniques have been applied to many different fields, achieving state-of-the-art results in many of them.

3.3.1 Vanilla Neural Networks

The building block of a NN is called neuron. A neuron is a mathematical function that takes as input a vector of values and produces a single value as output. The output is computed as the weighted sum of the inputs, plus a bias term. The exact formula is:

$$\hat{y} = \sigma(\sum_{i=1}^{n} w_i x_i + b) \tag{3.1}$$

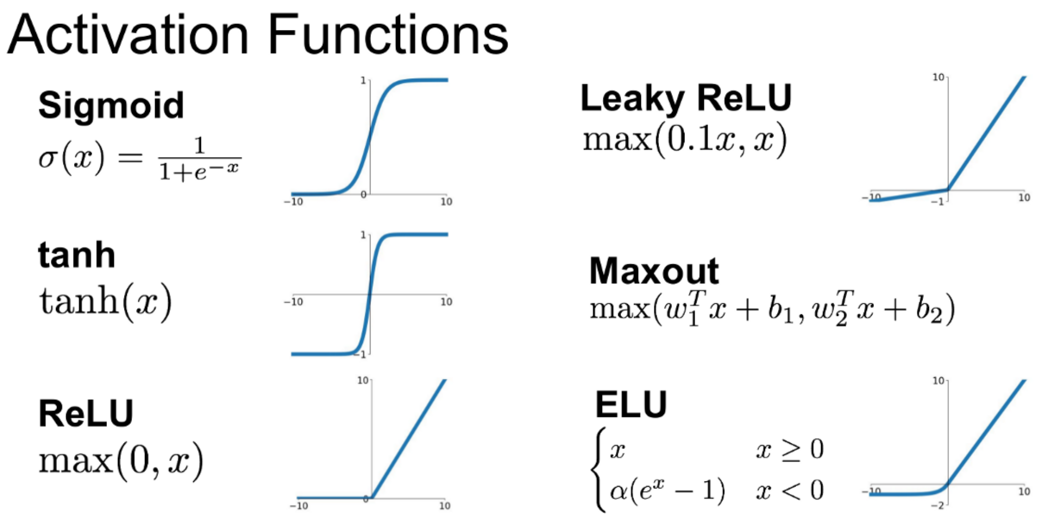


Figure 3.6: The most common activation functions.

where $\mathbf{x}$ is the vector of inputs and y is the output, $\mathbf{x} \in \mathbb{R}^n$, $\hat{y} \in \mathbb{R}$. The vector of weights $\mathbf{w} \in \mathbb{R}^n$ and the bias $b \in \mathbb{R}$ are the parameters of the neuron, and they are learnt during the training process. The output of the neuron is then passed through an activation function, which is a non-linear function that introduces non-linearity in the network. This is required to be able to learn non-linear functions. The most common activation function is the sigmoid function, but other functions like the hyperbolic tangent or the ReLU function are also used. Figure 3.6 shows the most common activation functions and their formulas.

A single neuron is not very powerful and can learn to approximate only trivial functions, but by combining multiple neurons together it is possible to create a more powerful model. The simplest architecture is the single layer feedforward NN, where multiple neurons are connected together. The output of each neuron is connected to the input of all the neurons in the next layer. The first layer is called input layer, and it receives the $\mathbf{x}$ vector. The last layer is called output layer, and it produces one or more $\hat{y}$ values. The layer in between is called hidden layer. Figure 3.7 shows an example of a single layer NN.

While such an architecture may seem arbitrary, its power comes from the universal approximation theorem [64]. This theorem states that a single layer NN with a finite number of neurons can approximate any continuous function on a compact subset of $\mathbb{R}^n$. This means that this model can learn to approximate any function and thus can be used to solve a large number of problems. However, the theorem does not say anything about the number of neurons required to approximate a function. In fact, the number of required neurons may grow exponentially, and this is one of the reasons why single layer NNs are not used in practice. For this reason, multi layer NNs have been introduced.

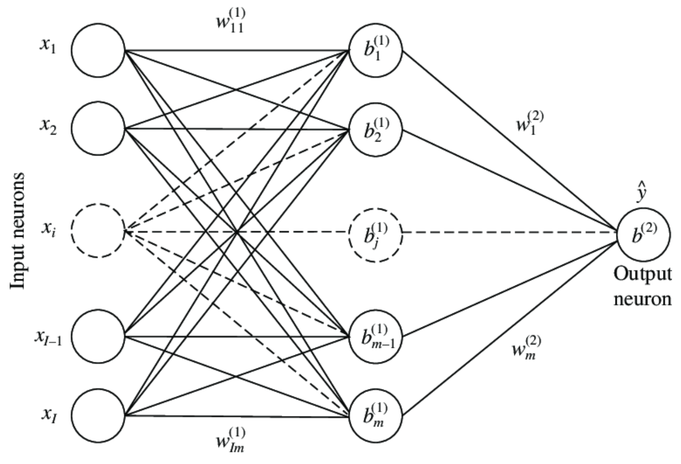


Figure 3.7: A single layer feedforward Neural Network.

A multi layer NN is a NN with multiple hidden layers, where the output of each layer is connected to the input of all the neurons in the next layer, as shown in figure 3.8. The number of hidden layers is called the depth of the network. While this architecture is more compact and is able to approximate complex functions, it's important to avoid very deep networks to avoid problems like the vanishing gradient problem [125].

To calculate the outputs of the last layer, the equation (3.1) is repeated for each neuron until the output is reached. Real world implementations rely on the matrix representation of such formula and are able to execute thousands of such operations in parallel exploiting the parallel architecture of GPUs. This process is called inference, or forward pass.

Usually, NN models belong to the supervised or semi-supervised learning categories, meaning that labeled data is expected to be available, in the case of supervised learning, or unlabeled data consisting only of *normal* points in the case of semi-supervised anomaly detection. The labels represent the ground truth that we want to learn ((y)), such as the real outlier score or the regression value to be predicted. The output of the last layer ($\hat{y}$) is then compared with the ground truth using a loss function. For example, in the case of a regression problem, the Mean Squared Error (MSE) is often used:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{3.2}$$

Observing this formula, one can notice that all elements are known, a part from the weights $\mathbf{w}$ and b parameters that are used to calculate the output $\hat{\mathbf{y}} = NN_{\mathbf{w},b}(\mathbf{x})$. The goal of the training process is to find the values of these parameters that minimize the loss function. This is done using the backpropagation algorithm, which is an algorithm that computes the gradient of the loss function with respect to the parameters of the network. The gradient is then used to update the parameters of the network using an

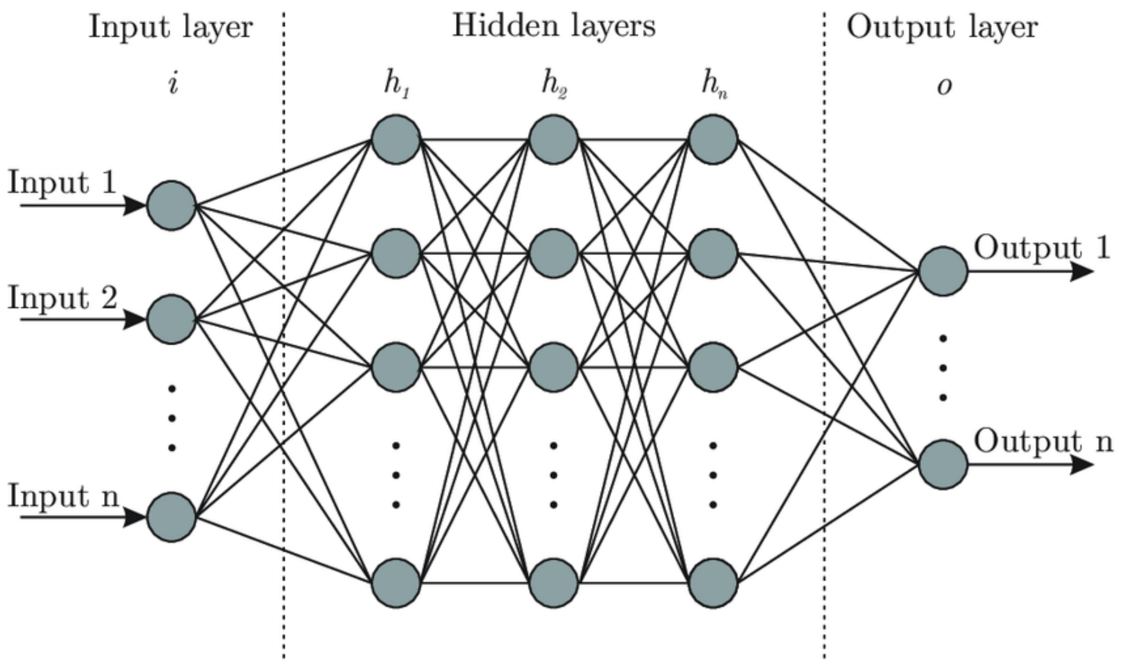


Figure 3.8: A fully connected Neural Network example. Source: [112]

optimization algorithm like Stochastic Gradient Descent (SGD) or its variations.

Probably the most important task in designing this kind of models is the choice of the neural network architecture. This means choosing all the parameters of the networks, like the number of hidden layers, the activation function and the type of layers. There is no single best choice but the results depend heavily on the problem, the data and computing resources available. In fact, developing a network too complex with a small training dataset can result in *overfitting*, where the networks is very good at predicting training data but fails to generalize for new data. The basic approach is to use dense layers of fully connected neurons as shown in figure 3.8, but in literature many designs have been proposed to improve the performances while reducing the computational cost. For example, CNNs, first used in image classification, include convolutional layers which take into account the spatial correlation between pixels in the images by grouping together close pixels. Other techniques have been proposed to work on data with time correlation like Recurrent Neural Networks (RNNs) and Long Short Term Memory (LSTM) networks. Another approach is the Autoencoder model, where the network architecture projects data into a lower dimensional space. The following sections will describe these architectures in more detail.

3.3.2 Convolutional Neural Networks (CNNs)

Convolutional Neural Networks (CNNs) have been proposed to solve computer vision tasks like object classification, object detection and image segmentation. The main idea is to exploit the spatial correlation between pixels in images, by grouping together close pixels. This is done using convolutional layers, which are layers that apply a convolution operation to the input. The convolution operation is a mathematical operation that can be seen as a sliding window that moves across the input and produces a new output. The size of the window is called kernel size, and it is a hyperparameter of the network. The output of the convolution operation is called feature map, and it is a matrix of values. The convolution operation is defined as:

$$S(i, j) = (I * K)(i, j) = \sum_m \sum_n I(i + m, j + n)K(m, n) \tag{3.3}$$

where I is the input matrix, K is the kernel matrix and S is the feature map. The convolution operation is repeated for each position of the kernel in the input matrix. The kernel matrix is a matrix of weights, and it is a parameter of the network that is learnt during the training process. The convolution operation is illustrated in figure 3.9.

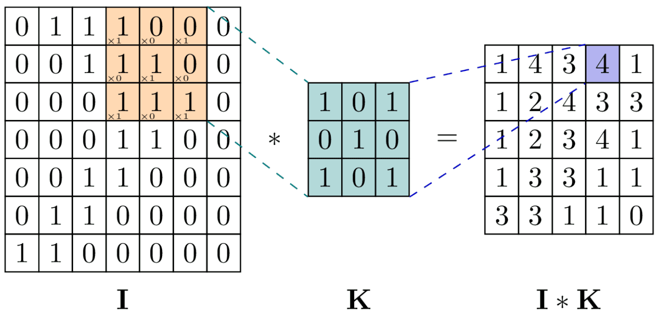


Figure 3.9: The convolution operation.

Since the kernel matrix does not change during the convolution operation, the weights are the same for each position of the kernel. This means that the number of parameters of the convolutional layer is much smaller than the number of parameters of a fully connected layer. This is one of the reasons why CNNs are so popular: they are able to achieve good results with a much smaller number of parameters, and thus they are faster to train and require less memory. Unfortunately this reduces the flexibility of the model, since a single matrix of weights may be insufficient to extract useful features from the input layer. To overcome this issue, multiple kernels can be used in the same convolutional layer, and the output of each kernel is a different feature map. By doing this the network moves from a layer with a single channel to a layer with multiple channels, each with reduced dimensions due to the convolution operation. The convolutional layer is then followed by a pooling layer, which is a layer that reduces the dimensionality of the input by applying a pooling operation. The pooling operation is similar to the convolution operation, but instead of a weighted sum it uses a function to reduce the values in the kernel to a single value. The most common pooling operation is the max pooling, which takes the maximum value in the kernel. By stacking convolutional and max pooling layers together, the input is progressively reduced in size and the number of channels is increased. The convolutional layers are used to extract features from the input, while the pooling layers are used to reduce the dimensionality of the features. The last layer of the network is a fully connected layer, which is used to produce the final output. Figure 3.10 shows an example of a complete CNN architecture.

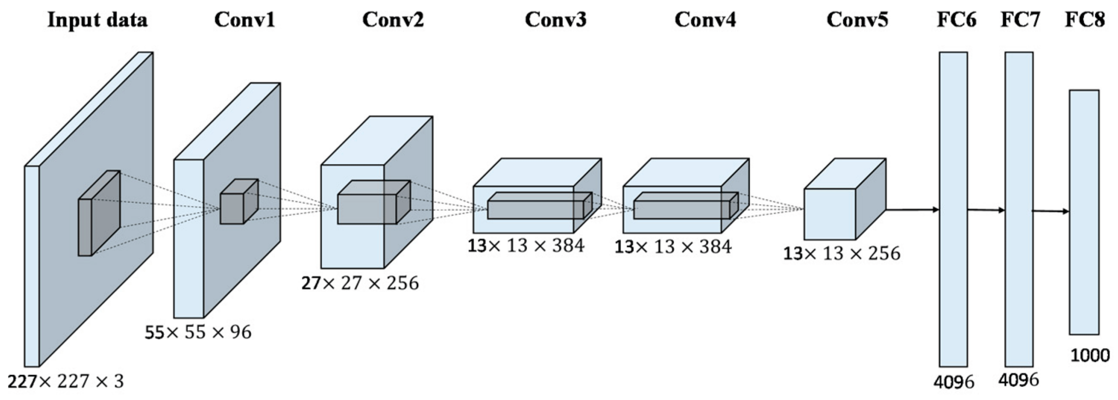


Figure 3.10: The AlexNet CNN architecture.

The final dense layers depend on the specific application: the number of outputs and the activation function will be designed according to the objective of the network. For example, for a classification task, the CNN will have a number of neurons on the output layer equal to the number of classes, and the activation function will be the softmax function. For a regression task, the output layer will have a single neuron and the activation function will be the identity function.

3.3.3 Autoencoders

In the context of unsupervised learning, one of the most interesting NN architectures is the Autoencoder. An Autoencoder is a NN that is trained to reproduce its input at the output layer. While this may seem a trivial task, the real objective of the network is to learn a compressed representation of the input, called *latent space* and re-generate the output from it. This is done by using a bottleneck layer, which is a layer with a smaller number of neurons than the input layer. The network is then trained to reconstruct the input from the bottleneck layer, and the reconstruction error is used as loss function.

Figure 3.11 highlights the main components of the Autoencoder architecture. The first part is called *encoder* and is composed of a few layers which reduce the dimensionality of the input. Depending on the input data type these can be dense of convolutional and max pooling layers, or even more advanced types. The second part is called *decoder* and is composed of a few layers which increase the dimensionality of the input. The last layer of the decoder is a dense layer with the same number of neurons as the input layer, and the activation function is the identity function. The bottleneck layer is the layer in between the encoder and the decoder, and it has a smaller number of neurons than the input layer. The number of neurons in the bottleneck layer is a hyperparameter of the network, and it determines the dimensionality of the latent space.

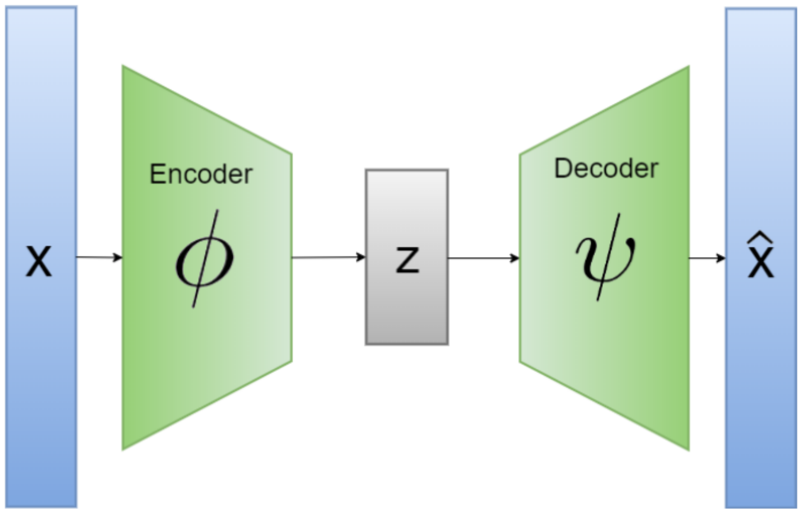


Figure 3.11: The Autoencoder NN architecture. Source: [14]

Once an Autoencoder is trained, the encoder and decoder can be used separately to encode and decode new data. This is useful to reduce the dimensionality of the input data, and it can be used as a preprocessing step for other machine learning models. However, the most interesting feature of Autoencoders is the ability to reconstruct the input from the latent space. Since the compressed representation will lose some information, only the main properties of the input will be preserved. Thus, an autoencoder will be able to reconstruct the input only if it does not deviate too much from training data, while it will fail to reconstruct anomalous data. For this reason Autoencoders are often used for Anomaly Detection, where the reconstruction error is treated as an anomaly score. The higher the reconstruction error, the more anomalous the input is.

Another approach to AD with deep learning models is the use of time series forecasting models. In this case, the NN is trained to forecast the next value in the time series, and the forecast error is used as anomaly score. This approach is similar to the one used with Autoencoders, but exploits the time correlation often found in time series data. The following sections will describe some of the most common forecasting models.

3.3.4 Deep Learning Forecasting

Given the time series $X = \{x_1, x_2, \ldots, x_n\}$, the goal of a basic forecasting model is to predict the next value x_{n+1}. Such problem has many applications in different fields, such as finance, weather forecasting, energy consumption and many more. In all these cases an accurate prediction of the future values can be very useful to make optimal decisions. Furthermore, a good forecasting model can be used to detect anomalies in the data, by comparing the predicted value with the real value. When dealing with future predictions it's important to keep in mind that a model can only be as good as the data it is trained on. Thus, if the data is not representative of the real world, the model will not be able to make accurate predictions. The model learns trends and historical behaviors from the available dataset and predicts the most probable outcome given the current state. In some cases, where the process is completely random, there may be no good model to predict the future. Also, the velocity of the underlying process and the volatility of the data must be taken into account. In fact, when predicting the future value of a slow-changing variable like the temperature, a model will be make accurate predictions for short time frames just by assuming that the value will not change at all. While the numerical results may look good, such a prediction is often useless.

While many statistical techniques have been developed during the years, deep learning methods have emerged as good candidates to address the time series forecasting problem given their ability to learn complex non-linear functions. In fact, even a vanilla NN can be used to forecast time series data just by expressing the problem as a regression task, where the label is the future value. While this may work for simple problems, it does not take into account the sequentiality of the input, which is treated as a simple vector, and thus the learning process is not optimal. More advanced DL architectures have been proposed tailored for time series data which were often able to achieve state-of-the-art results.

Another advantage of DL models is the ability to forecast multiple steps ahead. This is done by training the model to predict multiple values in the future, instead of just one. This is useful in many applications, for example in the stock market, where predicting the next value is not very useful, but predicting the next 10 values can be very useful to make investment decisions. The flexibility of DL models allows also to forecast multivariate time series, maximizing the information that can be extracted from the data. Furthermore, the software ecosystem around DL models is very mature and allows to efficiently work with large datasets, exploiting hardware accelerators like GPUs. Thus, it's easy to work even with very long or high frequency time series, which are often found in real world applications.

Recurrent Neural Network (RNN)

Recurrent Neural Networks (RNNs) are a class of NN architectures designed to work with sequential data. This is done by introducing a feedback loop in the network, which allows to store information about previous inputs. This is useful to learn time correlations in the data, and it's the main reason why RNNs are so popular in time series forecasting.

To compute $\mathbf{O}_t$ (the output at time t), RNNs take into account both the current input $\mathbf{X}_t$ and previous inputs $\mathbf{X}_{0:t-1}$. Following the notation in [147] and visualized in figure 3.12, we denote the inner layer state at time t as $\mathbf{H}_t \in \mathbb{R}^{n \times h}$, where n is the number of samples and h is the number of hidden units. Note that while a single hidden layer is shown in figure 3.12, the formulation can be easily extended to multiple hidden layers. Furthermore, we denote the input at time t as $\mathbf{X}_t \in \mathbb{R}^{n \times d}$, where d is the number of inputs of each sample. Two weight matrices are used to compute the inner layer state:

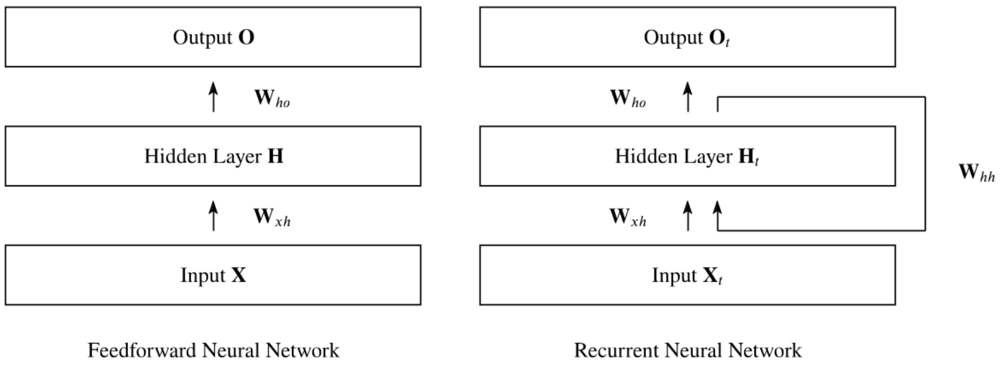


Figure 3.12: High level architecture of a feedforward NN compared with the RNN architecture. Source: [147]

the input matrix $\mathbf{W}_x \in \mathbb{R}^{d \times h}$ and the hidden matrix $\mathbf{W}_h \in \mathbb{R}^{h \times h}$. The inner layer state is then passed through an activation function σ to produce the output. Finally, the hidden state and the output are calculated as:

$$\mathbf{H}_t = \sigma(\mathbf{X}_t \mathbf{W}_x + \mathbf{H}_{t-1} \mathbf{W}_h + \mathbf{b}_h) \tag{3.4}$$

$$\mathbf{O}_t = \sigma(\mathbf{H}_t \mathbf{W}_o + \mathbf{b}_o) \tag{3.5}$$

As can be seen in equation (3.4), $\mathbf{H}_t$ depends recursively on $\mathbf{H}_{t-1}$, so that the output eventually depends on all the previous hidden states, and thus on all the previous inputs. This is the main difference between RNNs and feedforward NNs, where the output depends only on the current input. This allows RNNs to learn time correlations in the data, and thus they are very useful for time series forecasting.

However, RNNs have some limitations. In fact, to train the network the gradient of the loss function with respect to the output of the network must be computed. This is done using a slightly modified version of the backpropagation algorithm, called Backpropagation Through Time (BPTT), which unfolds the network and calculates the gradient of the loss function with respect to the output of each layer. This means that the gradient must follow a potentially long path through the network, due to the unfolding of the loop introduced by the RNN architecture. This can lead to the vanishing gradient problem [125], which occurs when the gradient of the loss function with respect to the parameters of the network becomes smaller and smaller with each layer. Thus the training process becomes very slow, the weights for past layers are barely updated and the network is not able to learn long term dependencies in the data. This problem is particularly evident when training RNNs on long sequences, and it can be mitigated using other architectures like LSTMs.

Long Short Term Memory (LSTM)

Long Short Term Memorys (LSTMs) were presented in 1997 [62] to overcome the vanishing gradient problem of RNNs. This is done by introducing a gated memory cell in the network, which is a special unit that is able to store information for long periods of time. The memory cell is then used to compute the hidden state, instead of using the previous hidden state as in RNNs. This allows the network to learn long term dependencies in the data, and thus they are very useful for time series forecasting.

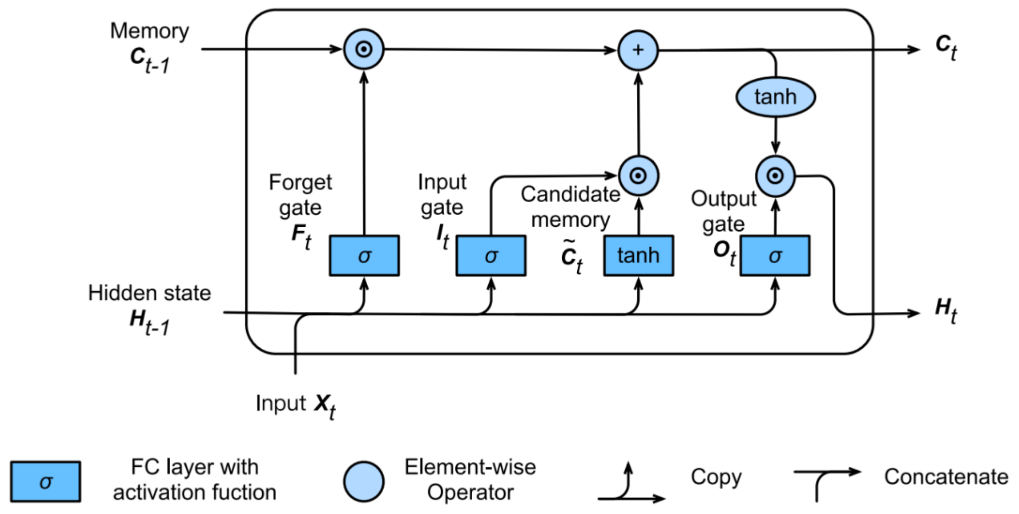


Figure 3.13: The LSTM cell. Source: [147]

First we define the output gate $\mathbf{O}_t$, the input gate $\mathbf{I}_t$ and the forget gate $\mathbf{F}_t$ as:

$$\mathbf{O}_t = \sigma(\mathbf{X}_t\mathbf{W}_{xo} + \mathbf{H}_{t-1}\mathbf{W}_{ho} + \mathbf{b}_o) \tag{3.6}$$

$$\mathbf{I}_t = \sigma(\mathbf{X}_t\mathbf{W}_{xi} + \mathbf{H}_{t-1}\mathbf{W}_{hi} + \mathbf{b}_i) \tag{3.7}$$

$$\mathbf{F}_t = \sigma(\mathbf{X}_t\mathbf{W}_{xf} + \mathbf{H}_{t-1}\mathbf{W}_{hf} + \mathbf{b}_f) \tag{3.8}$$

with $\mathbf{W}_{xo}$, $\mathbf{W}_{xi}$, $\mathbf{W}_{xf} \in \mathbb{R}^{d\times h}$ and $\mathbf{W}_{ho}$, $\mathbf{W}_{hi}$, $\mathbf{W}_{hf} \in \mathbb{R}^{h\times h}$ being the weight matrices, and $\mathbf{b}_o$, $\mathbf{b}_i$, $\mathbf{b}_f \in \mathbb{R}^h$ being the bias vectors. In this case the activation function σ is the sigmoid function. These gates are used to control the flow of information in the memory cell. The candidate cell state $\tilde{\mathbf{C}}_t$ is then computed as:

$$\tilde{\mathbf{C}}_t = \tanh(\mathbf{X}_t\mathbf{W}_{xc} + \mathbf{H}_{t-1}\mathbf{W}_{hc} + \mathbf{b}_c) \tag{3.9}$$

As show in figure 3.13, the cell receives as inputs the previous cell state $\mathbf{C}_{t-1}$, the previous hidden state $\mathbf{H}_{t-1}$ and the current input $\mathbf{X}_t$. In output it produces the current cell state $\mathbf{C}_t \in \mathbb{R}n \times h$ and hidden state $\mathbf{H}_t$. The cell state is computed as:

$$\mathbf{C}_t = \mathbf{F}_t \odot \mathbf{C}_{t-1} + \mathbf{I}_t \odot \tilde{\mathbf{C}}_t \tag{3.10}$$

where $\odot$ is the element-wise multiplication. Finally, the hidden state is computed as:

$$\mathbf{H}_t = \mathbf{O}_t \odot \tanh(\mathbf{C}_t) \tag{3.11}$$

The tanh activation function is used to force the output between (-1, 1). Figure 3.13 summarizes the LSTM cell architecture described by equations (3.6) to (3.11).

The cell state is then passed to the next cell, and the process is repeated for each time step. The hidden state is then used to compute the output of the network as in equation (3.5). The main difference between LSTMs and RNNs is that the hidden state is computed using the cell state instead of the previous hidden state. This allows the network to learn long term dependencies in the data, and thus they usually achieve better performance.

Temporal Convolutional Network (TCN)

Temporal Convolutional Networks (TCNs) have been introduced more recently [83] for action segmentation in videos. They can be used for time series forecasting thanks to dilated 1-D convolutions, which are a special type of convolutional layers that allow to increase the receptive field of the network without increasing the number of parameters.

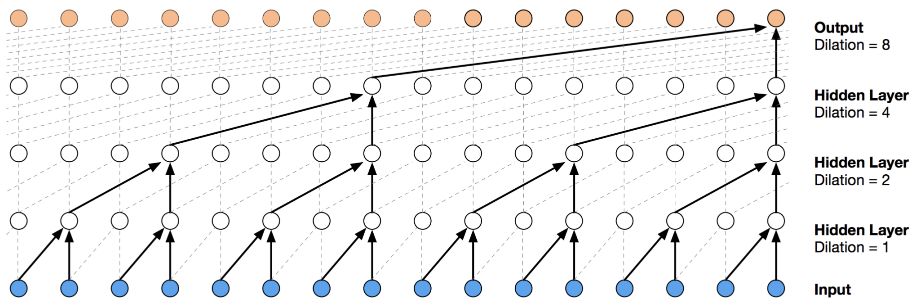


Figure 3.14: The TCN layer architecture.

While RNNs introduce a loop into the network, TCNs architecture is similar to feedforward NN. As stated earlier, a simple feedforward NN can be used to time series forecasting by designing the problem as a regression task on the future value. This has the disadvantage of not keeping into account the sequentiality of the data and, if the input is not chosen correctly, risks using data in the future make the prediction, which is not reasonable.

TCN solves these problems by using dilated causal convolution layers. The term causal means that the output at time t depends only on the inputs at time t and before, thus guaranteeing that the network does not use future data to make predictions. Diluted convolutions instead are convolutions where the kernel is applied to the input with a certain step, called dilation rate. Such kernel is calculated on input data which may not be adjacent, thus linking a node on a layer with nodes in the previous one far in the past without increasing the number of parameters. The final result is an increase of the receptive field of the network, that is the number of inputs that influence the output. Figure 3.14 shows the architecture of a TCN layer when applying a dilated 1-D convolution with filter size 2 and dilation factors 1, 2, 4 and 8. As can be observed, the receptive field of the network increases at each layer thanks to the dilation factors, so that the final output depends both on the last few inputs and also on the ones long in the past.

A d-dilated layer with a kernel size k has a receptive field of $1+d\cdot(k-1)$. If d is fixed, the number of layers required to cover a certain input length increases linearly with the input length. For long input vectors, this results in a very deep NN, which suffers from the vanishing gradient problem. Thus TCNs are often used with exponentially increasing d values. If we choose a *dilation base* b, d can be expressed as a function of the hidden layer index i: $d = b^i$. In this case the receptive field can be expressed as:

$$r = 1 + \sum_{i=0}^{n-1}(k-1)\cdot b^i = 1 + (k-1)\cdot \frac{b^n - 1}{b-1} \tag{3.12}$$

The example in figure 3.14 follows this convention. Now the number of layers required to cover a certain input length increases logarithmically with the input length, thus avoiding very deep networks and reducing the vanishing gradient problem. TCNs further mitigate this problem by using residual connections, which are connections that skip one or more layers. This allows to train very deep networks, and it's one of the reasons why TCNs are so popular in time series forecasting.

Many more models and architectures have been proposed in literature for time series forecasting, but they are out of the scope of this thesis. The interested reader can find more information in [164].

3.4 Reinforcement Learning

Reinforcement Learning (RL) is a machine learning approach where the model does not learn from a dataset, as in supervised learning, but by interacting with an environment. A complete mathematical framework has been developed to enable such experience-driven autonomous learning. Then, with the advent of deep learning and thus the ability to approximate complex functions, RL has been able to scale to decision-making problems that were previously intractable, with large state and action spaces, often beating competing strategies or human experts. For example, in [118] the authors presented an algorithm that was able to learn to play a range of Atari 2600 games by just observing the screen pixels, achieving super-human performance. This was a major breakthrough in the field, and it opened the door to many new applications. After that, another deep reinforcement learning model, AlphaGo, was able to defeat the world champion in the game of Go [153]. This was a notable achievement, since Go is a very complex game with a huge state space, and it was thought that a computer would never be able to beat a human player. Thus RL has emerged as a promising technique to solve control problems in many different fields like robotics, autonomous driving, finance and many more.

The main idea behind RL is to make an *agent* learn through interaction with an external environment. By observing the consequences of its actions, the agent learns to choose the best action in each situation. This is done by adapting its behaviour based on the received reward. This is similar to how humans learn, and it's one of the reasons why RL is so powerful. Figure 3.15 shows the RL loop.

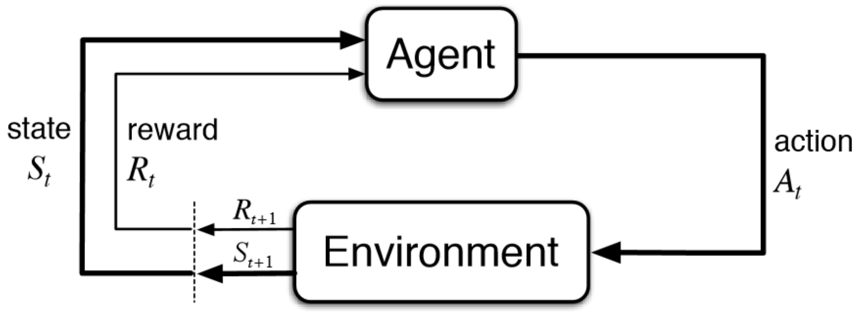


Figure 3.15: The RL loop.

In the RL terminology an autonomous agent, which can be implemented with a NN model, observes a state s_t from the environment and chooses an action a_t to perform based on such observation. At this point the environment and the agent transition to the state s_{t+1}, which depends on the chosen action. The state contains all the information about the environment at a given time step, and it can be as simple as a single number or more complex, like as a set of images. Upon the state transition, the agent receives also a reward r_{t+1} from the environment, which represents the quality of the action. The goal of the agent is to maximize the cumulative reward over time, and this is done by learning a *policy* π, which is a function that maps the current state to an action. The policy is learnt by trial and error, since the state transition dynamics model is not available. The agent observes the current state of the environment, chooses an action according to the policy, and receives a reward from the environment. The reward is then used to update the policy, so that the agent will choose better actions in the future. This process is repeated until the agent learns the optimal policy, maximizing the cumulative reward over time.

The RL framework can be described as a Markov Decision Process (MDP), which is a mathematical framework that describes the interaction between the agent and the environment. A MDP is defined as a tuple $\langle \mathcal{S}, \mathcal{A}, P, \mathcal{R}, \gamma \rangle$, where:

$\mathcal{S}$ is the set of states and $p(s_0)$ is the initial state distribution.

$\mathcal{A}$ is the set of possible actions.

P is the state transition probability function, which is defined as $P(s_{t+1}|s_t, a_t)$ and represents the probability of transitioning to state s_{t+1} given the current state-action s_t, a_t.

$\mathcal{R}$ is the reward function, which is defined as $\mathcal{R}(s_t, a_t, s_{t+1})$ and represents the reward received by the agent when transitioning from state s_t to state s_{t+1} after performing action a_t.

γ is the discount factor, which is a value between 0 and 1 that represents the importance of future rewards. A discount factor of 0 means that only immediate rewards are considered, while a discount factor of 1 means that future rewards are as important as immediate rewards.

The policy π is a function that maps states to a probability distribution over actions. $\pi : \mathcal{S} \to p(\mathcal{A} = a|\mathcal{S})$. In episodic MDPs the agent interacts with the environment for a finite number of steps, and the episode ends when a terminal state is reached, or when the maximum number of iterations T is reached. In this case the sequence of states, actions and rewards is called a *trajectory* or a rollout of the policy. During the trajectory the agent accumulates the rewards from the environment, and at the end calculates the *return*:

$$R = \sum_{i=0}^{T} \gamma^t r_{t+1} \tag{3.13}$$

The goal of the agent is learn the optimal policy π^*, which is the policy that maximizes the expected return:

$$\pi^* = \arg\max_{\pi} \mathbb{E}[R|\pi] \tag{3.14}$$

If we consider instead nonepisodic MDPs, the discount factor γ can be used to avoid an infinite sum of rewards by setting $\gamma < 1$. In this case methods that rely on complete trajectories are not applicable, since the agent will never reach a terminal state, whereas those that rely on a finite set of transitions must be used.

The MDP framework assumes that the environment satisfies the Markov property, which assumes that the current state contains all the information about the past. It means that only the current state affects the next state, and the future depends only on the current state. This is a reasonable assumption in many real world applications, and it allows to simplify the problem by not considering the history of the states. Any decision can be taken considering only the current state, and the history of the states can be discarded. This is very useful to reduce the computational complexity of the problem, since the state space can be very large. For example, in the case of a chess game, the state space is the set of all possible configurations of the board. If the Markov property is not satisfied, the state space would be the set of all possible configurations of the board and the history of the moves. This would make the problem much more complex and computationally expensive. Still, this assumption requires the current state to be fully observable, which may not be the case in many real world applications. In these cases a generalization of the MDP framework called Partially Observable Markov Decision Process (POMDP) is used, where the agent receives an observation $o_t \in \Omega$ which is a partial representation of the state s_t. The probability distribution of such observation $p(o_{t+1}|s_{t+1}, a_t)$ depends on the current state and the previous action.

While RL models have achieved great results they come with some challenges. First, the agent must explore the environment to learn the optimal policy. This is done by taking random actions, which is called *exploration*. However, if the agent takes random actions it will receive low rewards, and thus it will not learn anything. If instead it focuses too much on actions which produce good rewards, may end up never trying actions which may result in even better rewards.This is called the *exploration-exploitation dilemma*, and it's one of the main challenges in RL. The agent must find a balance between exploration and exploitation, so that it can learn the optimal policy without wasting too much time exploring the environment. Another challenge is the sample efficiency: the agent must learn the optimal policy with as few samples as possible. This is particularly important in real world applications, where the agent must learn from experience, and thus it must interact with the an environment where each interaction may be expensive or dangerous. For example, in the case of a robot learning to walk, each interaction may result in a fall, which may damage the robot. Thus, the agent must learn the optimal policy with as few falls as possible. Finally, it's not trivial to correlate a positive reward with the action which caused it, since the agent may have taken many actions before receiving the reward. This is called the *credit assignment problem*, and it's one of the main challenges in RL.

RL methods can be classified into two main categories: approaches based on *value functions* and approaches based on *policy search*. The following sections will describe the two categories and present selected algorithms from each one.

3.4.1 Value Functions Methods

Value functions method are based on the idea of estimating the expected return of being in a certain state. We can define the state-value function $V^{\pi}(s)$ as the expected return when starting in state s and following the policy π:

$$V^{\pi}(s) = \mathbb{E}_{\pi}[R|s, \pi] \tag{3.15}$$

The optimal policy then corresponds to an optimal state-value function $V^*(s)$, which is defined as:

$$V^*(s) = \max_{\pi} V^{\pi}(s), \forall s \in \mathcal{S} \tag{3.16}$$

This means that if one knows the optimal state-value function, the optimal policy can be computed simply choosing the action a (among the available ones) that maximizes $\mathbb{E}_{s_{t+1}\sim\mathcal{P}(s_{t+1}|s_t,a)}[V^*(s_{t+1})]$. Since the transition probability function is not known, the state-action value or quality function $Q^{\pi}(s,a)$ is used instead. This function is defined as the expected return when starting in state s, performing action a and following the policy π. It differs from $V^{\pi}(s)$ in that the action a is known, and the policy π taken in state s.

$$Q^{\pi}(s,a) = \mathbb{E}_{\pi}[R|s,a,\pi] \tag{3.17}$$

Once $Q^{\pi}(s,a)$ is known, the optimal policy can be choosen with a greedy approach by selecting the action a that maximizes $Q^{\pi}(s,a)$ at each state, while the corresponding value function is $V^*(s) = \max_a Q^{\pi}(s,a)$.

To estimate the quality function $Q^{\pi}(s,a)$, we exploit the Markov property and the Bellman equation. In fact, we can define the function as a Bellman equation, that is a recursive equation that expresses the value of a state in terms of the value of its successor states.

$$Q^{\pi}(s,a) = \mathbb{E}_{s_{t+1}}[r_{t+1} + \gamma Q^{\pi}(s_{t+1}, \pi(s_{t+1}))] \tag{3.18}$$

This formulation shows that we can use the current estimation of Q^{π} to update the estimation of Q^{π} at the next state. This is the main idea behind the *Q-learning* algorithm, which is one of the most popular RL approaches.

The policy π used to estimate the quality function $Q^{\pi}(s,a)$ can be the same policy used to choose the actions, or it can be a different policy. In the first case we talk about *on-policy* methods, while in the second case we talk about *off-policy* methods. In the case of on-policy methods, the policy used to estimate the quality function $Q^{\pi}(s,a)$ is the same policy used to choose the actions, and thus the agent learns the optimal policy π^* directly. In the case of off-policy methods, the policy is learned from actions collected following a different policy, or even with random actions. For example, q-learning learns the optimal policy with the help of a greedy policy, and thus it's an off-policy method.

While the quality function can be used directly, it's often more convenient to use the *advantage function* $A^{\pi}(s,a)$, which is defined as the difference between the quality function and the value function:

$$A^{\pi}(s,a) = Q^{\pi}(s,a) - V^{\pi}(s) \tag{3.19}$$

It represents the advantage of taking action a in state s over the average action. Intuitively, the absolute value of the reward of an action is less interesting than how well it compares with other actions to choose the best one. Furthermore, it's easier to learn that an action will have better consequences than another, than learning the exact return of each action. From the point of view of the NN model, this operation is equivalent to removing a baseline from the input data, which is always a good practice on NN models. Thus, this idea has been implemented in many RL algorithms.

DQN

DQN [117] is an off-policy RL algorithm based on the idea of approximating the quality function $Q^{\pi}(s, a)$ with a NN model. Unfortunately, in contrast with standard supervised learning, in RL the distribution of the input data constantly changes during the exploration phase, as well as the target value of Q. This introduces a lot of instability in the learning process, and thus it's hard to successfully train the network. DQN introduces two main ideas to solve this problem: *experience replay* and the *target network*.

Experience replay is a technique that allows to store the transitions $(s_t, a_t, r_{t+1}, s_{t+1})$ in a *replay buffer*, and then sample a batch of transitions from the buffer to train the network. This means that the training process can be done offline, reducing the number of interactions with the environment and thus increasing the sample efficiency. Using batches of transitions also reduces the variance of the gradient and breaks the correlation between consecutive samples, and thus it makes the learning process more stable. Training is also easily parallelizable, since the samples are independent from each other, increasing the computational efficiency. Furthermore, the replay buffer can be used to store the best network parameters found so far, and thus it can be used to restore the network in case of catastrophic forgetting.

The second idea is the target network, which is a copy of the main network which is not updated for a certain number of iterations. Instead of calculating the loss function with respect to a higly unstable estimate of Q as produced by the main network, the target network is used to compute the target value of Q. This is done to stabilize the learning process, since it's important that the target value does not change too much during the training process.

The final loss function can be expressed as:

$$L(\theta) = \mathbb{E}_{s,a,r,s' \sim D}[(r + \gamma \max_{a'} Q(s', a'; \theta^-) - Q(s, a; \theta))^2] \tag{3.20}$$

where D is the replay buffer, θ are the weights of the main network, θ^- are the weights of the target network, and $Q(s, a; \theta)$ is the quality function approximated by the main network.

3.4.2 Policy Gradient Methods

Policy gradient methods are based on the idea of directly searching for the optimal policy π^* instead of learning the value function $V^{\pi}(s)$ or the quality function $Q^{\pi}(s, a)$. This is done by parameterizing the policy $\pi_\theta(a|s)$ with a set of parameters θ, and then learning the optimal parameters θ^* that maximize the expected return $\mathbb{E}[R|\theta]$. These can be obtained by computing the gradient of the expected return with respect to the parameters θ, and then updating the parameters in the direction of the gradient.

Usually, the policy is parameterized as a NN model, and the parameters θ are the weights of the network, which is trained with SGD. The network takes as input the state s and outputs a probability distribution over the actions. For example, for an action which is represented by a single value, this could be the mean and standard deviation of a Gaussian distribution from which the action can be sampled. Instead, for a discrete action space, this could be a softmax layer with a number of neurons equal to the number of actions, where each value represents the probability of the corresponding action. The advantage of a stochastic policy is that is allows more exploration, since the agent can take different actions in the same state. Furthermore, we need to average over the trajectories induced by a certain policy parameterization, since the return is a random variable. This is done by sampling a batch of trajectories, and then computing the gradient of the expected return with respect to the parameters θ.

REINFORCE

We already introduced the concept of trajectory τ as:

$$\tau = (s_0, a_0, r_1, s_1, a_1, r_2, \dots, s_{T-1}, a_{T-1}, r_T, s_T) \tag{3.21}$$

and we want to define a function which measures how good a certain policy π_θ is. This can be expressed as the expected return of a trajectory τ induced by a policy π_θ parameterized by θ:

$$J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta}[R(\tau)] = \sum_{\tau} P(\tau; \theta) R(\tau) \tag{3.22}$$

where $P(\tau; \theta)$ is the probability of the trajectory τ induced by the policy π_θ. This can be expressed as the product of the probability of each state-action pair in the trajectory:

$$P(\tau; \theta) = P(s_0) \prod_{t=0}^{T-1} \pi_\theta(a_t|s_t) P(s_{t+1}|s_t, a_t) \tag{3.23}$$

Then we can compute the optimal parameters θ^* as:

$$\theta^* = \arg\max_{\theta} J(\theta) \tag{3.24}$$

To find the maximum of the objective function, we can calculate its gradient with respect to the parameters θ:

$$\nabla_\theta J(\theta) = \sum_{\tau} \nabla_\theta P(\tau; \theta) R(\tau) \tag{3.25}$$

and then iteratively update the parameters using gradient ascent:

$$\theta_{t+1} = \theta_t + \alpha \nabla_\theta J(\theta) \tag{3.26}$$

where α is a tunable learning rate.

Unfortunately the gradient of the objective function is intractable, since it depends on the probability of all possible trajectories, which is hard to compute. Furthermore, the transition probabilities $P(s_{t+1}|s_t, a_t)$ are not known, and thus we cannot compute the gradient. To solve this problem, we can use the *policy gradient theorem*, which states that:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta}[R(\tau)\nabla_\theta \log \pi_\theta(a_t|s_t)] \tag{3.27}$$

This new formulation is differentiable and does not require the differentiation of the state distribution.

REINFORCE [173], also known as Monte Carlo Policy Gradient, estimates the gradient of the objective function using Monte Carlo sampling, which is a technique that allows to approximate the expected value of a random variable by taking the average of a finite number of samples. In this case, we can approximate the gradient using an estimated return from an entire episode:

$$\nabla_\theta J(\theta) \approx \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) R(\tau) \tag{3.28}$$

A widely used variant of REINFORCE improves the algorithm by subtracting a baseline from $R(\tau)$. This is done to reduce the variance of the gradient, and thus to make the learning process more stable. For example, the baseline can be the average return of the trajectories in the batch, or more complex advantage functions, as will the described in the next method.

Advantage Actor-Critic (A2C)

The previous algorithm uses the return $R(\tau)$ to increase or decrease the probability of a certain action. If the return is high, the probability of the action in the given state is increased, and vice versa. Since the environment and the policy are stochastic, trajectories can lead to highly different returns even when starting from the same state. This implies a high variance on the gradient, which makes the learning process unstable. To mitigate this problem one can calculate the average return over a number of trajectories, but this reduces the sample efficiency of the algorithm. As stated before, another solution is to use a baseline. The actor-critic methods combine the policy gradient approach with the estimation of a value function, which is used as a baseline.

The idea is to learn to function approximations:

- The actor, which is the policy $\pi_\theta(a|s)$, parameterized by θ.
- The critic, which can a value function $V_w(s)$ or and action-value function $Q_w(a|s)$, parameterized by w.

The two can optionally share some parameters. The actor learns the policy, while the critic learns to evaluate the actions taken by the actor. This can then be used to update the policy, so that the actor will take better actions in the future. The training process then consists of the following steps:

1. The actor takes an action a_t in state s_t according to the policy $\pi_\theta(a_t|s_t)$.

2. The environment transitions to state s_{t+1} and returns a reward r_{t+1}.

3. The critic evaluates the action a_t taken by the actor in state s_t and returns the estimated $Q_w(s_t, a_t)$ value.

4. The actor updates the policy $\pi_\theta(a|s)$ by updating its parameters θ in the direction of the gradient: $\theta_{t+1} = \theta_t + \alpha_\theta \nabla_\theta \log \pi_\theta(a_t|s_t) Q_w(s_t, a_t)$.

5. The Temporal Difference error (TD error) is calculated as $\delta_t = r_{t+1} + \gamma Q_w(s_{t+1}, a_{t+1}) - Q_w(s_t, a_t)$. This is then used to update the parameters w of the action value function $w_{t+1} = w_t + \alpha_w \delta_t \nabla_w Q_w(s_t, a_t)$.

The TD error is an alternative of the monte carlo approach to update the estimate of the value function or the policy. Monte carlo waits until the end of the episode, calculates the return and uses it as a target of the update. Instead, TD error estimates the return by summing the reward of an action to the discounted value function of the next state, and thus it can be used to update the value function at each time step.

To further stabilize the learning process, the advantage function $A(s, a)$ can be used as the critic model. We recall that an advantage function measures how much better an action is compared to other choices, as opposed to measuring the absolute value of the action. The problem is that the advantage function depends on both the value function and the quality function:

$$A(s, a) = Q(s, a) - V(s) \tag{3.29}$$

Thus, it's not trivial to estimate it. However, we can use the TD error to estimate the advantage function as:

$$A(s, a) = r_{t+1} + \gamma V(s_{t+1}) - V(s_t) \tag{3.30}$$

This approach is called Advantage Actor-Critic (A2C) [116], and it's one of the most popular actor-critic methods.

PPO

As we have seen in the previous methods, controlling the training stability is crucial to obtain a good RL model. In fact, when updating the policy one must choose carefully the learning rate:

- If the learning rate is too high, the policy will change too much at each iteration, and thus the training will be unstable and it becomes hard to learn the optimal policy.

- If the learning rate is too low, the policy will change too little at each iteration, and thus the training will be very slow.

Proximal Policy Optimization (PPO) [148] constrains the policy update to avoid large changes at each iteration, thus making the training more stable. This is done by clipping the probability ratio between the new and the old policy, so that the policy can change only by a certain amount at each iteration. The final loss function can be expressed as:

$$J^{CLIP}(\theta) = \mathbb{E}_t[\min(r_t(\theta)\hat{A}_t, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)] \tag{3.31}$$

where the probability ratio $r_t(\theta)$ is defined as:

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} \tag{3.32}$$

This is the ratio between the probability of taking action a_t in state s_t according to the new policy π_θ and the probability of taking the same action according to the old policy $\pi_{\theta_{old}}$, that is the policy before the update. If the ratio is greater than 1, the new policy is more likely to take action a_t in state s_t than the old policy, while if the ratio is less than 1, the action is less likely in the new policy. It basically represents how much the two policies differ between them.

Equation (3.31) chooses the minimum between two terms:

- The first term is the probability ratio multiplied by the advantage function $\hat{A}_t$.
- The second term is the clipped probability ratio multiplied by the advantage function $\hat{A}_t$.

In the second term the clipped ratio is forced to be between $1-\epsilon$ and $1+\epsilon$, thus limiting the change of the policy. ϵ is a hyperparameter, usually set to 0.2. Finally, the minimum between the two terms is chosen, so that we choose either one based on the clipping and relative advantage, but we guarantee that the policy will not change too much at each iteration.

PPO has gained a lot of popularity since it's extremely versatile, and it has been used in many applications with great results. It's also very easy to apply to different problems, since it doesn't require to tune many hyperparameters, like the learning rate, which is automatically tuned by the clipping. Even so, it shows poor performance on discrete action spaces with sparse high rewards, where it often gets stuck on local optima.

Chapter 4

Machine Learning for Particle Accelerators

4.1 ML in physics laboratories

The advent of innovative data-driven approaches based on Machine Learning (ML) and Deep Learning (DL) has opened new possibilities in many fields. In the case of physics laboratories around the world, the main applications of these techniques can be grouped under the following macro categories:

- **Data analysis and physics simulations**: the large amount of data generated by the physics experiments are traditionally stored and analyzed offline by physicists. The application of ML and DL techniques to this field has led to the development of new methods for data analysis and pattern recognition, which are able to extract useful information from the data and to perform complex tasks such as particle identification and event classification [9]. Furthermore, the simulation of the physics processes involved in the experiments is a very computationally intensive task, which can be sped up by the usage of ML and DL models. In particular, the simulation of particle collisions and their interaction with the detector are essential to compare the experimental results with the theoretical predictions. This is a very complex task, which is usually performed by Monte Carlo (MC) methods. The usage of ML and DL models for this task has been extensively studied in the literature, for example in [123], where a generative adversarial network (GAN) is used to generate simulated particle showers in a calorimeter, and in [31], where generative neural networks are used to simulate clusters produced by particles in the ALICE experiment at CERN. Finally, the speedup provided by these new technologies opened the possibility to run complex analysis online in real time, during the data acquisition phase, which enhances the quality of the data acquisition system [53].
- **Computing and data storage**: Physics experiments generate a huge amount of data, which requires extensive computing facilities to store and analyze, along with

fast network connectivity. Consequently, interconnected computing facilities have been developed all around the world, such as the Worldwide LHC Computing Grid (WLCG) [12], which is used by the experiments at CERN. ML models have been used to optimize the usage of these computing facilities, for example in [24] the authors use ML models to analyze file transfer errors in the WLCG. [34] instead presents a more general overview of smart automation systems for the computing infrastructure.

- **Control systems**: The control system of a particle accelerator can record the trend of all its variables. ML models can thus be trained on such data to perform high-level control system tasks like advanced automations, smart alarm systems or virtual sensors. This is the field of research which is most relevant to this thesis and it will be discussed in more detail in the next section.

The first category represents a vast field of research, which builds directly upon traditional data analysis and simulation techniques and concerns directly the physicists working on the experiments. The ML models here are used directly for physics research, with the goal of improving the quality of the data analysis and the accuracy of the simulations, or to speed up the computation. The ground truth is usually known and the models can be compared with the traditional methods to ensure their correctness. Moreover, their application is usually specific to the particle physics community or even to a certain experiment.

The second category is related to the computing infrastructure of the experiments and it is usually studied by computer scientists and engineers. This is narrower field of research, with fewer applications and a more limited impact. Nevertheless, it is relevant not only to the particle physics community, but to all the scientific community and, in general, to all the fields which require large computing facilities.

Finally, the third category deals with ML for the control system of particle accelerators. This is a nascent field of research, which aims to improve the reliability, efficiency and performance of the accelerators to enhance their scientific output. From an engineering perspective, particle accelerators are similar to industrial plants, since they are composed of complex equipment which needs to work together to produce the final product. The main difference is that particle accelerators are usually less standardized, with many experimental or prototype components, and they undergo continuous research and development. For these reasons the control systems of particle accelerators must usually be more flexible and must be able to adapt to compensate the low reliability of the underlying equipment. Still, the research on ML models in this field can easily benefit the industrial sector, since many of the problems faced by the control systems of particle accelerators are similar to the ones faced by industrial plants. At the same time the industrial sector can provide useful insights and solutions to the problems faced by the control systems of particle accelerators.

The main goal of ML models in the control systems of particle accelerators is to improve the performance, reliability and efficiency of the accelerators. One way to achieve these general goals is to have *smart* automation systems, which are able to perform complex tasks autonomously. For example, the best way to recover from a fault in the accelerator depends on the type of fault and on the current state of the accelerator. A smart automation system would be able to predict or detect the fault, to classify it and to perform the best recovery action autonomously, imitating the behaviour of an expert operator. This chain of actions involves common ML tasks, such as anomaly detection, fault prediction and classification, root cause analysis and optimal decision-making. Another example is the optimization of the beam transport, which is a complex task that requires the knowledge of the accelerator state and of the beam dynamics physics. A DL model can be trained to evaluate the state of the accelerator without coding all the implicit and explicit relationships between the different variables and the diagnostic elements, and may be able to have an insight on the actual state even better than a human looking at a single beam profiler. Then another model may be able to use such information to optimize the setpoint of the different components to maximize the performance of the accelerator.

Another approach is the introduction of models with the aim of helping the human operators in their tasks. For example, a model could infer the value of a variable which is not measured directly, but which is related to other measured variables. For example, the number of beam diagnostics on a beam line is limited, but if it was possible to know the beam current or the beam profile on any given position along the beam line, that would greatly benefit the experts to optimize the beam transport. Virtual sensors could also be used to summarize the state of a complex machine, highlighting the information that matter the most for the operators. For example, often it's not really interesting to know the exact value of a variable, but it's more important to know if it's increasing or decreasing, or just to be alerted when it's not behaving as expected. By reducing the number of variables to monitor, the operators can focus on the most important ones and can react faster to unexpected events.

As we can see, the application of ML models to the control systems of particle accelerators is a very promising field of research, with many exciting paths to explore and large potential benefits for the scientific community. Already in 2018 [40] highlighted the potential of ML models in the field of particle accelerators. In the next section we will present a brief overview of the literature in this field.

4.2 Literature Review

In this thesis and during the Ph.D. we focused on the application of ML models to the control systems of particle accelerators. Here we will present a brief overview of the literature in this field. We will distinguish between four main categories of applications: anomaly detection and fault prediction, virtual diagnostics, beam dynamics optimization and industrial applications.

4.2.1 Anomaly detection and Fault Prediction

We already introduced the importance of anomaly detection and fault prediction in the context of particle accelerators. In the literature we can find many examples of ML models applied to this field.

In [33] the authors propose a complete pipeline for anomaly detection and evaluation of the fault causes in beam transfer equipment at CERN. They deal with real data from different asynchronous source, and they use Gaussian Mixture Models and Isolation Forests as unsupervised detectors. The results are evaluated by comparing with the logbook entries. [154] instead focuses on fault classification on the RF cavities at CEBAF (Jefferson Lab). Several classical ML models as well as a LSTM network are compared on the task of identifying the cause of the fault as well as the initial fault in a cascade of faults. Random Forest and Decision Tree were the best performing models. [38] instead presents an AD approach based on single time-series. In this work, which is validated on data from the CERN LINAC4 RF plasma generators, the authors use an statistical and machine learning methodologies to detect and analyze jitter and long time drifts in the data which leads to reduced beam quality, as opposed to fault detection.

One of the most straightforward approaches to anomaly detection is the usage of the residual prediction error of a forecasting or regression model. With the advent of DL and especially recurrent neural networks, time series forecasting models have become very powerful and could be applied directly to particle accelerators data. Different approaches have been proposed in this field: [90] presented a classification model to predict safety interlocks based on Recurrence Plots and Convolutional Neural Networks (CNN). An anomaly detection method based on Recurrent Neural Networks is used by [156] to predict faults on the RF system. In this case the facility accelerates a photon beam and the faults are due to quenches, when a cavity loses its superconductive status. [89] offer a review of time series forecasting methods in the field of particle accelerator, highlighting both linear and non-linear models and their importance in different applications in particle accelerators. An application on RF cavity faults based on a CNN classifier is presented, while methods based on recurrent neural networks are only theorized and left for future works.

More recently, [98] presented a method based on a LSTM network to predict faults on the power supplies of a storage ring. The model predicts the temperature of various components of the power supply based on its power output and room and cooling water temperatures, and when the observed temperature deviates from the predicted one, a fault is detected. [50] instead shows that anomaly detection techniques can have an impact on accelerator performance even when applied to ancillary systems such as the air cooling system. In this case an autoencoder architecture is used to detect precursors of trips in the cooling system of the Collider Accelerator Complex at Brookhaven National Lab. An autoencoder is implemented also in [65] to detect malfunctionings on the CERN SPS Beam Dump System. In this case the model works on real time images and is trained both on real data and on simulations.

Another way to prevent faults is predictive maintenance. Whenever a fault is due to the degradation or aging of a component, it's often much cheaper to replace the component before it fails, rather than waiting for the fault to happen and then repairing it. In [21] the authors propose a predictive maintenance model with the aim of reducing the downtime of a proton therapy accelerator at PSI. They use a regression model to predict the remaining useful lifetime of a component. Obviously this approach is only possible if the assumption that the fault depends on aging of a component holds true, and it's possible to measure the degradation level. Faults due to other causes, such as misconfigurations or unexpected events, cannot be predicted with this approach.

When dealing with anomaly detection, it's often interesting to know the underlying cause that generated a fault, or at least the variable that is most correlated with it. In fact, this can be used by experts and engineers to understand the system behavior and design a fix. For example, in [45] a supervised approach based on explainable machine learning fault classification has been presented; in such work, several convolutional neural networks are used and compared for binary classification of faults and then the Layer-wise Relevance Propagation (LRP) method is used to identify the causes of the faults. The work exploits multiple time-series coming from the Proton Synchrotron Booster (PSB) at CERN.

4.2.2 Virtual Sensors

Virtual sensors are a very interesting application of ML models to particle accelerators. The idea is to use a model to predict in real time the value of process variables which are not measured. These variables are usually related to the measured ones, and the model can be trained on historical data or on simulations to learn the relationship between the variables. This approach can be used to augment the information available to the operators, to replace faulty sensors or to display the information in a more intuitive way.

An interesting application of this approach is presented in [176]. In ultrafast electron diffraction and microscopy a bunched electron beam is used, and the exact properties of each bunch are required, such as the energy and energy spread. While it's unfeasible to measure such properties for each bunch in real time, ML models are used to infer the properties of the unmeasured bunches based on the measured ones. A similar approach is used in [43] to provide an estimate of photon pulse characteristics related to beam positioning in a free electron laser. This can be used when real diagnostics are unavailable or destructive. In this context ML models can provide the same results of a physics simulator in a fraction of the time. [99] shows how a ML model can replace a destructive diagnostic with a virtual one to measure the longitudinal phase space of a beam.

In [1] instead the authors use ML to model the rise in temperature on a target at J-PARC when irradiated by a high power proton beam. The model is trained on historical data and then can be used to know in advance the temperature of the target and detect anomalous values.

4.2.3 Beam Dynamics Optimization and Optimal Control

Controlling the beam dynamics is a challenging task in many accelerators. Given the large number of variables involved, it's often difficult to find the optimal setpoint of each beam transport element. Physics simulations are used extensively but they are very computationally intensive and they are not always accurate. The field of research to tackle these challenges with ML models is very active and many different applications can be found in literature. Depending on the type of accelerator the optimization task is defined on different metrics, but the general goal is to achieve a beam dynamics as close as possible to the ideal one, which is usually defined by the accelerator design.

One noteworthy researcher in this field is A. L. Edelen, currently head of the Machine Learning Department for the Accelerator Research Division at SLAC National Accelerator Laboratory in California. In her Ph.D. thesis [41] she explored the use of ML and especially neural networks for several applications in particle accelerators. For example, she applied NNs to speed up physics simulations so that they could be used in real time for optimal control and she studied transfer learning techniques to train a model with simulations and then use it on real machines. She worked also on control for temperature and resonant frequency of RF cavities. In [40] she presented an overview of the opportunities for ML in particle accelerators, highlighting the potential of ML models in this field. Later works from her and her research group focus on automatic evaluation of image-based diagnostics with CNNs to be incorporated in feedback controls [39] or Bayesian optimization aided by neural networks for optimal tuning of injectors [175]. In fact, when dealing with direct optimization of control system parameters they focus on methods based on Bayesian optimization, but use NNs surrogate models as a prior mean for Gaussian processes.

A broad investigation on the use of ML models for optimal control of particles accelerators was presented also in [145]. The authors present the case study of an accelerator which is designed to operate over a wide range of complex beam phase space distributions, where it's impossible to switch between configurations just with a lookup table of all the parameter settings. In fact, limited diagnostics, time-varying performance and complex interactions between the different components make it difficult to guess the optimal setpoint in all the possible scenarios. The authors propose a hybrid approach mixing ML models and feedback-based model-independent methods. The former are used to get between a neighborhood of the required machine settings for a given set of desired beam properties and the latter are used to get to the exact setpoint and to track it continuously.

More recently, [149] shows the use of deep neural networks as predictive components in a control loop, with the aim of controlling the linac output energy and its phase oscillation. [42] instead used NNs to build a surrogate model of a synchrotron for orbit correction, which maps orbit distortions into correction settings. In fact, in a synchrotron the beam travels in a closed circular loop and even a small deviation from the ideal trajectory can lead to a large error after many turns, and thus it's important to correct the orbit continuously. Similarly, [141] uses a neural network to calculate the orbit correction in real time. [174] shows the usage of Reinforcement Learning to control the beam trajectory, with a focus on how to transfer a model trained on a specific line to similar beam lines. At CERN machine learning has been considered for a variety of beam dynamics applications for the Large Hadron Collider (LHC) [6]. This includes an autoencoder-based model to improve the quality of betatron-phase measurements, which is fundamental to compute the optics correction, and a model to predict the beam lifetime based on the control system configuration. This creates a data-driven surrogate model of the accelerator, which can be used to find the optimal control system parameters by using a standard optimization algorithm. A similar approach is presented also in [70], where a beam dynamics model of a storage ring is built with a physics informed neural network, so that the training becomes easier and more accurate. Reinforcement Learning has been used to optimize trajectory steering at the AWAKE and LINAC4 beam lines at CERN [76], and free energy-based reinforcement learning (FERL) model with clamped quantum Boltzmann machines (QBM) and multidimensional continuous state-action space environments has been proposed as an upgrade [146].

4.2.4 Industrial applications

Many of the challenges in control systems of particle accelerators are similar to the ones faced by industrial plants. For this reason, the research on ML models for particle accelerators can benefit from the research on industrial applications and vice versa. For example, Anomaly Detection is a very active field of research in the industrial sector, where it's used to improve the reliability of a plant or for quality assessment on the final product. While a comprehensive overview of the literature on industrial applications of ML is out of the scope of this thesis, we will cite here just a few examples of relevant applications as an example of potential cross benefits between the two fields.

In this respect AD is surely one of the tasks where the industry has built a lot of expertise, which could be brought to the particle accelerator community. For example [102] presents a anomaly detection pipeline on industrial data from the semiconductor manufacturing. An autoencoder model is deployed to detect defects on the final product, in order to minimize the number of broken chips delivered to customers. The concept of residual prediction error on time series forecasting in exploited in [95], where a Generative Adversarial Network (GAN) is used to reconstruct the original data. A similar approach is presented by [180], using a Variational LSTM model to learn a latent representation of the data and then reconstruct it. A number of surveys of this field is available, like [26] which focuses on unsupervised AD for industrial images and [134] which deals with AD in wireless sensor networks.

Another area where the industry research is especially active is predictive maintenance. In fact this is a crucial task to achieve resilient productive processes whenever there are tool subject to wear, which is very common in industrial settings. For example, [160] presents a multiple classifier ML method for predictive maintenance and demonstrates its effectiveness on a semiconductor manufacturing maintenance problem. A literature review on this topic is published in [182].

Other areas of interest for industrial applications which can have an impact on particle accelerators are root cause analysis, explainability and interpretability of ML models, which can contribute to understand the origin of faults in the machine. [19] offers an example of research in this field in the context of industrial plants. Furthermore, methods for design of experiments [5], transfer learning [111], and continual learning [28] can have several applications in particle accelerators.

4.3 Summary and future directions

Throughout this chapter we have conducted a non-exhaustive review of the literature on ML and DL research in the field of particle accelerators. We have seen how ML models are being to improve the performance, reliability and efficiency of particle accelerators, and how they can be applied to different aspects of the control system. We also highlighted the close relationship between the industrial and the particle accelerator fields, and how the research in one field can benefit the other.

In general, a growing number of researchers is exploring ML techniques for an increasing number of applications, with different levels of success. This demonstrates the interest and potential of this field of research. Nevertheless, many works appear to focus on very narrow problems, with a limited impact on the overall performance of the accelerator. Applications to more challenging tasks, such as the automatic optimization of the beam dynamics on a complete beam line are still sparse and hardly generalizable. Reinforcement learning techniques are still rarely used, despite their potential to solve complex online optimization tasks. Furthermore, many researchers are working on similar problems from different point of views, but we lack a clear comparison between the different results. The community is still fragmented, and often each research group starts from scratch, without building upon previous works.

In this context, two initiatives have the potential to focus the efforts of the community: RL4AA is a new collaboration which aims to bring together researchers from different laboratories to share their knowledge and to work together on Reinforcement Learning for autonomous control of accelerators [138]. The ICFA Beam Dynamics Mini-Workshop on Machine Learning Applications for Particle Accelerators [17] instead is an event which focuses on applications of ML to beam dynamics. Initiatives like these can help the community to grow and to focus on the most promising research directions, while also providing a common ground for comparison between different approaches. For example, it would be interesting to develop a common benchmark for Reinforcement Learning applications in particle accelerators, or common datasets which could be used to compare different approaches and to evaluate the performance of new models.

Chapter 5

Anomaly Detection and Fault Prediction

This chapter will describe in details the work performed to predict fault conditions on the ALPI RF control system, introduced in section 2.3.4. Two approaches will be presented, one based on classical ML section 5.2 and one based on DL in section 5.3. The ML theoretical foundations presented in chapter 3 are here used in practice to achieve the best results.

Accelerator fault conditions can be seen as anomalies in the stream of data produced by the sensors. For this reason, the data analysis here presented focuses on Anomaly Detection, with the purpose of identifying as early as possible trends or points which significantly differ from the normal conditions. These can be early indicators of possible fault conditions, with the ability both to predict a fault and potentially to indicate its root cause. We choose a semi-supervised approach, where artificial labels are automatically generated from the data and are used only to optimize the final threshold on the model output. While the resulting models have been tailored to a specific use case, the methods can be generalized to work on different subsystems of a particle accelerator and in general can be useful where there are complex control systems with multiple interconnected systems, even in industries.

Before explaining the two approaches, we will first describe the dataset and the computing setup and present the data acquisition and preprocessing phase.

5.1 Prerequisites

In this work, since the data is generated continuously by multiple sensors during the operation of the machine, each data point has a signal name, a value and a timestamp. Thus, the objects of analysis are time-series, with the following properties:

dynamic : in time-series, each point is not an independent choice from a random distribution, but its values is influenced by the previous values, or the overall trend.

discrete : the measurement is not continuous but, as all digital systems, is sampled at specific timestamps.

multivariate : this term indicates the dimension of the series, or the number of different sensors concurrently acquiring different signals. This means that the data does not represent a single stream of a certain PV but multiple PVs during the same period of time.

non-aligned : all the PV values are stored in the database by the archiver when they change value, to reduce the space requirements. This means that the data is not aligned, with all the values of all the PVs sampled periodically, but each point can occur at any given time.

unsupervised : a dataset with ground-truth labels is not available.

Each of these point influences the choices on the methods to utilize and the preprocessing phase, as will be discussed in the following sections.

5.1.1 Signals description

Here the main Process Variables involving the RF control system of ALPI will be presented, as they will be used as they represent the dataset of our ML analysis.

First, the controlled variables:

- **Frequency Error**: It's the difference between the cavity frequency setpoint ($80MHz$) and the actual frequency. The hardware controller is only able to measure it absolute value, and thus cannot be used by the soft tuner procedure to decide the direction of the motor movement for the frequency correction.
- **Phase Error**: It's the difference between the cavity phase setpoint, as set by the operator, and the actual phase. Since the phase is the derivative of the frequency, it is used as the controlled variable in the soft tuner PID. The control loop is active only when this goes over a certain threshold, to avoid moving continuously the motors.

The cavity is a non-perfectly coupled RF load, and thus the power injected into the cavity is partially reflected back to the amplifier. So the RF power is measured as:

- **Forward Power**: It's the power injected into the cavity by the amplifier. Usually in the range of $7 - 30W$ during normal operation.
- **Reflected Power**: It's the power reflected back from the cavity to the amplifier. This may vary from a small percentage of the forward power to 100% depending on the operating conditions.

- **Net Power** or **Cavity Power**: It's the difference between the forward and reflected power, and measure the power stored in the cavity.

The power is interesting to measure since it gives an indication of how well the control loop inside the RF controller is working: if there is a surge in power this usually means that the controller sees a big discrepancy with the setpoint.

Other useful process variables to monitor are the one referring to the cryogenic and vacuum subsystems. While previous variables where relative to a single cavity, these last variables are relative to a whole cryostat, so they affect 4 cavities.

- **He tank level**: It's the filling percentage of the liquid Helium tank.
- **He tank pressure**: It's the pressure of the liquid Helium. Fast surges or decrease of this variable are often associate with lock failed events.
- **Cryostat vacuum level**: It's the vacuum level inside the cryostat, usually around $10^{-8} mBar$.

Other less related PVs include the gradient setpoint, the field error, and the whole setpoint and status of the cavity.

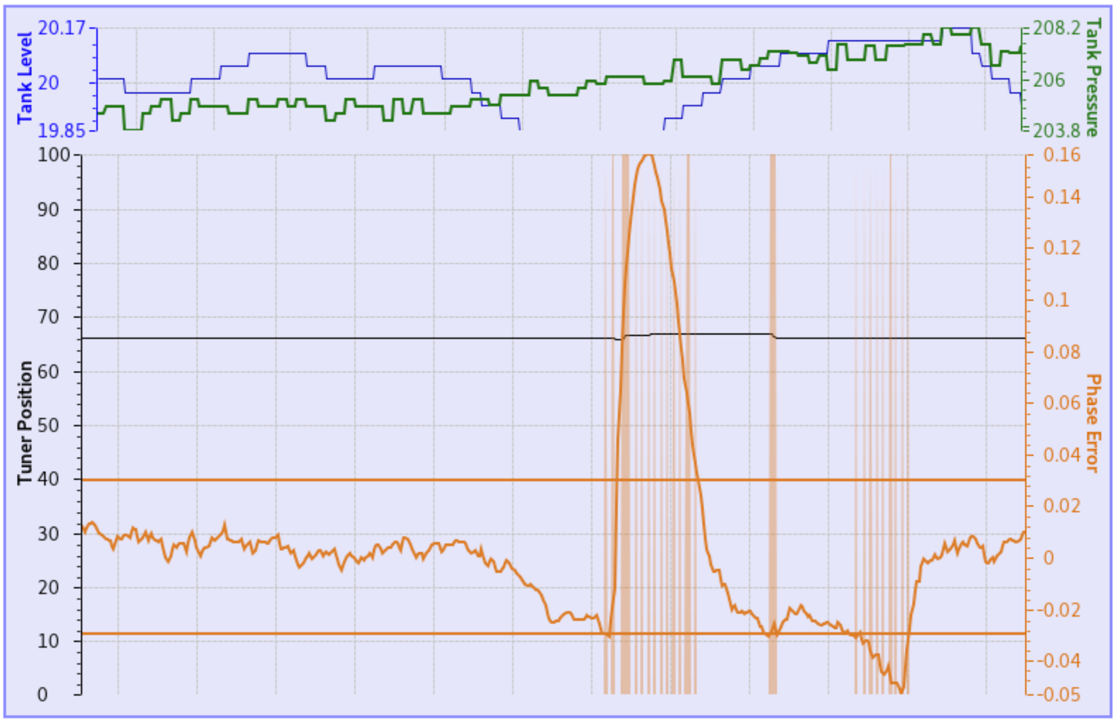


Figure 5.1: Soft Tuner GUI with one minute trend of the phase error and related PVs. Source: Control Room GUI.

An example of these variables can be seen in Figure 5.1. It shows two graphs from the GUI as shown to the machine operator, displaying one minute trend of the Process Variables involving the soft tuner. The upper trend presents the cryogenic variables of He level and pressure, while the lower one shows the phase error variation over time, and the PID thresholds. The black line shows the tuner motor position in percentage and the vertical lines indicates the motor moving to correct the phase error. It's easy to see that, thanks to the soft tuner control loop, when the phase error exits its deadband the motors are activated to bring it back between the thresholds. Even so, some unexpected events can bring the phase error quite distant from its setpoint, as can be seen in the middle of the graph. This is how the lock failed events introduced in section 2.5.1 can originate.

5.1.2 Computing setup

The code developed for this methods has been written in python, as it is the *de-facto* standard programming language for ML and it provides multiple useful libraries. The main ones used throughout this work are:

- `Pandas` [135], a data analysis and manipulation library which handles natively time-series data. Its main data structure is called `DataFrame` which represent a table with multiple columns and an index. Many methods are available to load data into dataframes, and then to filter, merge, resample, slice them, among many others. As a backend the popular scientific library `Numpy` is used, since it offers many performant primitives to work on arrays. The index can store the timestamps or time deltas associated with the values on the same row, which is very convenient.

- `TensorFlow with Keras` [22] Keras is a popular library to implement DL algorithms. It provides a high-level interface to define neural networks and train them on a variety of hardware, including GPUs. It is built on top of `TensorFlow`, a low-level library which provides primitives to define and train neural networks. The `Keras` library is used to implement the DL approach presented in section 5.3.

- `Pyod` [179] is a Python toolkit for detecting outlying objects in multivariate data. It implements a wide range of algorithms, both belonging to the classical machine learning and deep learning categories. The algorithms offer a standard interface to the user and work on data with multiple features, even though they are not specific for time-series, and thus cannot be applied directly to this use case. The `Pyod` library is used to implement the classical ML approach presented in section 5.2.

- `Matplotlib` [66], used to create graphs and visualizations.

- `Psycopg2` [130], a popular library to access a PostgreSQL database.

- `Joblib` [73], used to run python code in parallel on multi-core CPUs.

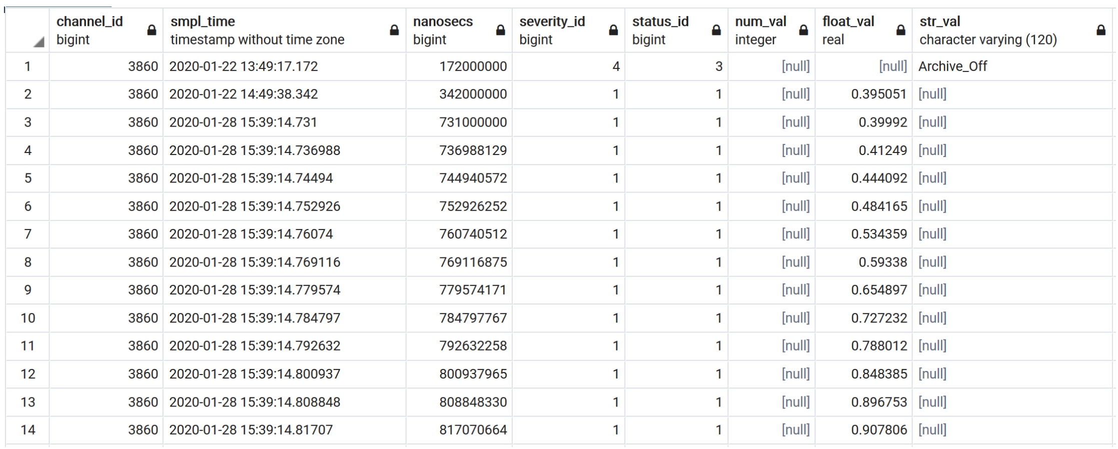
The development and prototyping of the code was carried out using *JupyterLab*, a web-based environment to write and run pieces of code. This is a widely used tool in the Machine Learning community as it allows to display together the code, its output including graphs, and rich text to explain it. Initially the web server was run locally on a desktop computer, but then a *JupyterHub* server has been installed on the Cloud-Veneto [2], a cloud-based computing infrastructure hosted between Padova and Legnaro supporting the research activities of the University of Padova and INFN. This allowed to reach the development environment from any client, and to access much more powerful computing capabilities. A virtual machine with 8 high performance CPU cores and $90GB$ of main memory was allocated. This allowed to run some algorithms in parallel, fully exploiting the CPU power without worrying about memory limitations. An NVIDIA Tesla T4 GPU instead was used to run the DL training and inference. This GPU has $16GB$ of memory, which means that very large dataset could not fit in memory. Even on this cloud infrastructure, often times computing time has been a limiting factor. Here the data covered only one month of accelerator runtime, but when dealing with bigger time slots a more powerful computing facility is suggested.

5.1.3 Data acquisition and preprocessing

The first step of this work consists of the data acquisition phase. Following the schema of the EPICS architecture presented in chapter 2.3.2, the Process Variables (PVs) read by the field sensors are exported to a common interface, the Channel Access (CA). The Archiver is then setup to listen to this interface and store the values of the PVs to the database. The Archiver keeps its configuration into an XML file which specifies a list of PVs and their group. For each PV it's possible to choose the sampling mechanism between polling, where the value is stored periodically, and *monitoring*, where the value is written to the database only upon changes. This second option has been chosen to reduce the wasted storage space.

Figure 5.2 shows how the values of each PV is stored in the `sample` table of the database. Each row of the table contains one point of the time-series of this particular PV. The columns instead contain the `channel_id` which is an internal ID for the database, the `smpl_time` and `nanosecs` which represent the timestamp with nanosecond precision, the `severity_id` and `status_id` which is the status and alarm level of the PV, and three columns for the actual PV value. In fact, the value can be both an integer, a floating point or a string value. The three columns `num_val`, `float_val` and `str_val` contain the value depending on its type, or null. Since python is able to handle different data type dynamically, we want to import them as a single column.

For this purpose we can use the `COALESCE` SQL function, which chooses the non-null value between `float_val` and `num_val` and returns it as a new column called `val`. Moreover, we added a check that at least one of those is not null, to remove anomalous data from the dataset. To run the query from python, the `psycopg2` library is used. First a connection to the database is created, and then multiple *cursors* can be opened on this connection to execute queries and retrieve results. We decided to use a special copy



	channel_id bigint	smpl_time timestamp without time zone	nanosecs bigint	severity_id bigint	status_id bigint	num_val integer	float_val real	str_val character varying (120)
1	3860	2020-01-22 13:49:17.172	172000000	4	3	[null]	[null]	Archive_Off
2	3860	2020-01-22 14:49:38.342	342000000	1	1	[null]	0.395051	[null]
3	3860	2020-01-28 15:39:14.731	731000000	1	1	[null]	0.39992	[null]
4	3860	2020-01-28 15:39:14.736988	736988129	1	1	[null]	0.41249	[null]
5	3860	2020-01-28 15:39:14.74494	744940572	1	1	[null]	0.444092	[null]
6	3860	2020-01-28 15:39:14.752926	752926252	1	1	[null]	0.484165	[null]
7	3860	2020-01-28 15:39:14.76074	760740512	1	1	[null]	0.534359	[null]
8	3860	2020-01-28 15:39:14.769116	769116875	1	1	[null]	0.59338	[null]
9	3860	2020-01-28 15:39:14.779574	779574171	1	1	[null]	0.654897	[null]
10	3860	2020-01-28 15:39:14.784797	784797767	1	1	[null]	0.727232	[null]
11	3860	2020-01-28 15:39:14.792632	792632258	1	1	[null]	0.788012	[null]
12	3860	2020-01-28 15:39:14.800937	800937965	1	1	[null]	0.848385	[null]
13	3860	2020-01-28 15:39:14.808848	808848330	1	1	[null]	0.896753	[null]
14	3860	2020-01-28 15:39:14.81707	817070664	1	1	[null]	0.907806	[null]

Figure 5.2: Example result of a basic query to the Archiver database.

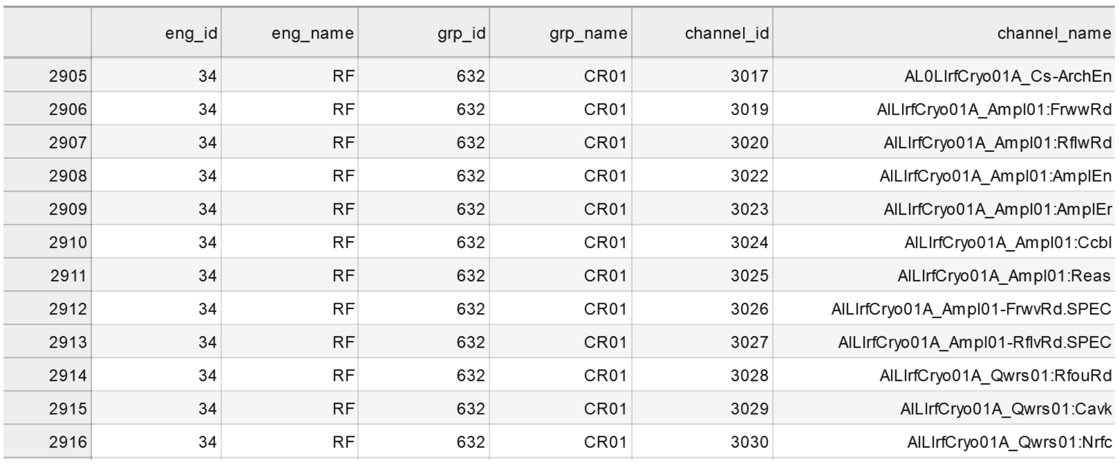
query of PostrgreSQL, where the output is returned in the form of Comma Separated Values (CSV). The `copy_expert` method of the cursor allows to copy the result directly to a file handler, which is then compressed with `gzip` and written to disk. This allows to execute a query which returns a lot of rows without loading them into memory all together, which in this case required multiple gigabytes per cavity. Having the data locally also has the advantage of not requiring to connect to the database each time and to speed up execution. Finally, we saved a different file for each cavity and cryostat, reducing the load times and memory requirements compared to a single large file. CSV writes data in plain text, so it is not optimized for disk usage, but has the advantage of being easily readable by a human.

Executing all the queries took about 8 hours, probably due to some missing indexes on the database. The data is composed of about 670 million points totaling about $20GB$ of CSV files, reduced to $7GB$ when compressed.

Once all the data is available in local, it is possible to load and start inspecting it. One of the problems is the missing PV name column. In fact, the database does not store the full name in the `sample` table, but just a reference ID, and then stores all the names just once in the `channel` table. Since it is easier to reference each PV by its name, the two tables must be merged. One option would be to join the tables during the SQL query, but this would have slowed down the query even further. For this reason, an external CSV file called `db_ids.csv` has been produced with `pgAdmin` where for each PV there is information about its name, IDs, group and engine. In figure 5.3 some example values from this file are shown.

Now this file can be used to add the PV names to the dataset. To do so, both the file and the data are first loaded into Pandas dataframes, which can easily be joined together. While loading the `db_ids.csv` file, a filter is applied on the PV names, to load only the PVs relative to the RF control system which are needed for the analysis.

The merge operation to create the final dataframe requires the following steps:



	eng_id	eng_name	grp_id	grp_name	channel_id	channel_name
2905	34	RF	632	CR01	3017	AL0LIrfCryo01A_Cs-ArchEn
2906	34	RF	632	CR01	3019	AILIrfCryo01A_Ampl01:FrwwRd
2907	34	RF	632	CR01	3020	AILIrfCryo01A_Ampl01:RflwRd
2908	34	RF	632	CR01	3022	AILIrfCryo01A_Ampl01:AmplEn
2909	34	RF	632	CR01	3023	AILIrfCryo01A_Ampl01:AmplEr
2910	34	RF	632	CR01	3024	AILIrfCryo01A_Ampl01:Ccbl
2911	34	RF	632	CR01	3025	AILIrfCryo01A_Ampl01:Reas
2912	34	RF	632	CR01	3026	AILIrfCryo01A_Ampl01-FrwvRd.SPEC
2913	34	RF	632	CR01	3027	AILIrfCryo01A_Ampl01-RflvRd.SPEC
2914	34	RF	632	CR01	3028	AILIrfCryo01A_Qwrs01:RfouRd
2915	34	RF	632	CR01	3029	AILIrfCryo01A_Qwrs01:Cavk
2916	34	RF	632	CR01	3030	AILIrfCryo01A_Qwrs01:Nrfc

Figure 5.3: Some values of the database internal references between PV names and IDs.

1. First the `data` object, which is the dataframe with the PV values, is merged to `event_pvs`, which contains the PV names. This means that a column is added to `data` with the PV name corresponding to the database ID.

2. The rows are sorted in chronological order.

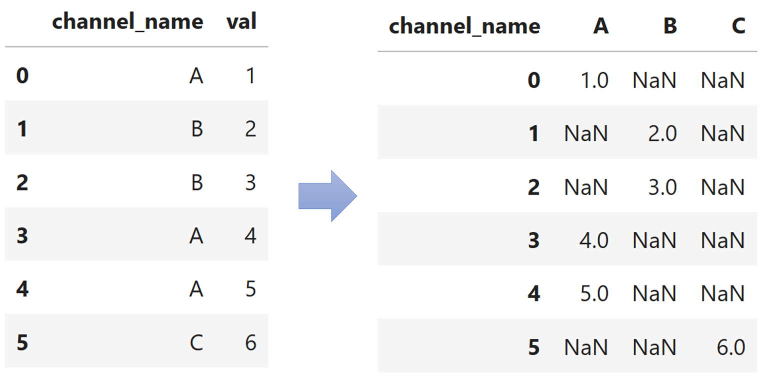
3. A pivot operation is applied. This means that a new table is created, with one column for each PV name from the `channel_name` column, the timestamps as row indexes and the PV values from the original dataset in the intersection of the corresponding PV name and timestamp. All the other values are set to `NaN`. Figure 5.4 shows an example of a pivot operation.

4. The column names are changed to keep only the suffix of the name. Since these operations are applied for each cavity distinctly, there are no overlapping suffixes.

5. Finally, a new column with the net cavity power is added by subtracting the values in the column of the reflected power from the ones of the forward power. This is an interesting value often calculated by domain experts that will be part of the following analysis.

6. With the `ffill()` method all the `NaN` values are replaced with the preceding value in their column. This way, for each timestamp, the table contains the full set of PV values at that moment in time.

We also compute an external list `fail_ts` containing all the timestamps where the column *Lkdf* has becomes 1. This is the case when an "unlock" event is happening at the corresponding timestamp, so we want to save this list to be used afterwards. A second list is saved with the timestamps where the "lock" events are happening. The cavity is in locked state, its normal operating mode, between a "lock" event and an "unlock" event.

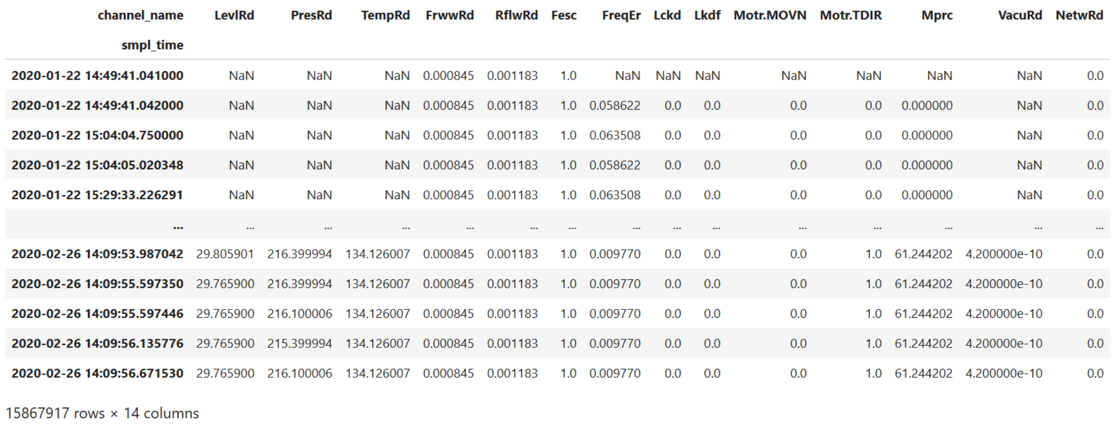


	channel_name	val
0	A	1
1	B	2
2	B	3
3	A	4
4	A	5
5	C	6

channel_name	A	B	C
0	1.0	NaN	NaN
1	NaN	2.0	NaN
2	NaN	3.0	NaN
3	4.0	NaN	NaN
4	5.0	NaN	NaN
5	NaN	NaN	6.0

Figure 5.4: An example of a pivot operation using Pandas.

The result of all the previous operations is a single dataframe for each cavity, containing all the data about it, with timestamps as index, column names which indicate the signal name, and the rows containing the most recent value of each signal. This is a compact form, which is convenient for the analysis and easy to understand, as it mimics multiple time-series flowing in parallel. Figure 5.5 shows some rows from the data of the first cavity of the third cryostat.



channel_name	LevlRd	PresRd	TempRd	FrwwRd	RflwRd	Fesc	FreqEr	Lckd	Lkdf	Motr.MOVN	Motr.TDIR	Mprc	VacuRd	NetwRd
smpl_time														
2020-01-22 14:49:41.041000	NaN	NaN	NaN	0.000845	0.001183	1.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	0.0
2020-01-22 14:49:41.042000	NaN	NaN	NaN	0.000845	0.001183	1.0	0.058622	0.0	0.0	0.0	0.0	0.000000	NaN	0.0
2020-01-22 15:04:04.750000	NaN	NaN	NaN	0.000845	0.001183	1.0	0.063508	0.0	0.0	0.0	0.0	0.000000	NaN	0.0
2020-01-22 15:04:05.020348	NaN	NaN	NaN	0.000845	0.001183	1.0	0.058622	0.0	0.0	0.0	0.0	0.000000	NaN	0.0
2020-01-22 15:29:33.226291	NaN	NaN	NaN	0.000845	0.001183	1.0	0.063508	0.0	0.0	0.0	0.0	0.000000	NaN	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
2020-02-26 14:09:53.987042	29.805901	216.399994	134.126007	0.000845	0.001183	1.0	0.009770	0.0	0.0	0.0	1.0	61.244202	4.200000e-10	0.0
2020-02-26 14:09:55.597350	29.765900	216.399994	134.126007	0.000845	0.001183	1.0	0.009770	0.0	0.0	0.0	1.0	61.244202	4.200000e-10	0.0
2020-02-26 14:09:55.597446	29.765900	216.100006	134.126007	0.000845	0.001183	1.0	0.009770	0.0	0.0	0.0	1.0	61.244202	4.200000e-10	0.0
2020-02-26 14:09:56.135776	29.765900	215.399994	134.126007	0.000845	0.001183	1.0	0.009770	0.0	0.0	0.0	1.0	61.244202	4.200000e-10	0.0
2020-02-26 14:09:56.671530	29.765900	216.100006	134.126007	0.000845	0.001183	1.0	0.009770	0.0	0.0	0.0	1.0	61.244202	4.200000e-10	0.0

15867917 rows × 14 columns

Figure 5.5: The data from CR03-01 after the merge operations.

5.1.4 Event visualization

To better understand the data available, a useful first step of the analysis is data visualization. In fact, just by plotting the time-series it is possible to visually inspect them and start to evaluate possible ideas on how to proceed. For example, one may notice that a certain variable is fixed at a certain value for the whole time, and thus it is probably not very informative in regard to the advent of a fault.

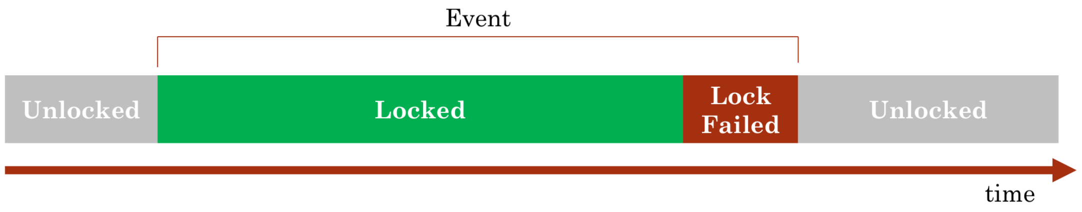


Figure 5.6: The timestamps of the lock and unlock events are used to split the data into Events.

The most interesting parts of the data are the last few minutes before a fault event happens, where it is probably possible to notice unusual patterns. Conversely, all the data collected while the cavity was not in the "locked" state, its normal operating mode, must be discarded as it represent a completely different setup. For this reason the dataset is now split into subsequences, called *Events*, beginning at the moment of a lock event and ending with an unlock event, as shown in figure 5.6. This way, each section represents a single unlock event and the temporal evolution of all the PVs before it. Obviously, since each cavity is mostly independent from the others, only the PVs of the same cavity and its cryostat are included.

To split the data into subsequences, or "events" as each subsequence represents the data leading to an event, the two lists containing the timestamps of the lock and unlock events are used. For each timestamps of an unlock events, the preceding timestamp of a lock event is searched. We also decided to filter out events where the cavity remained in the locked state for less than 2 minutes. In fact, in those cases the cavity was probably not properly setup and thus it was never able to work in *normal* conditions. We are interested in predicting patterns which bring a cavity from normal operating conditions to an unlock event, so these events are not interesting. After this filter, the data for the one-month accelerator run contains 169 lock failed events. With these events, an `Event` object is instantiated and saved to a list. The `Event` class is a custom class which keeps track of all the info about a single event, such as its preceding data and timestamps and can be used to implement algorithms on this data as methods of the class.

The first method of the class which is here presented is a method to plot a synthetic representation of the event. This is implemented using the `subplots` method of matplotlib, which enables to create multiple subplots on the same graph, sharing the x axis, which represents the time. The result for one specific event on CR03-01 is shown in figure 5.7.

This plot covers the last 5 minutes before the unlock event. The moment of the event can be noted thanks to the purple line on the second subplot, representing the locked status, going to zero. Furthermore, the yellow area indicates the lock failed status. On the first graph the cryogenics Process Variables are shown, that is the He tank level and pressure. In the second graph, the red line represent the forward power level, the orange one is the reflected power and the pink line is the net power, that is the difference between forward and reflected. The area highlighted in green show the time when the tuner motor

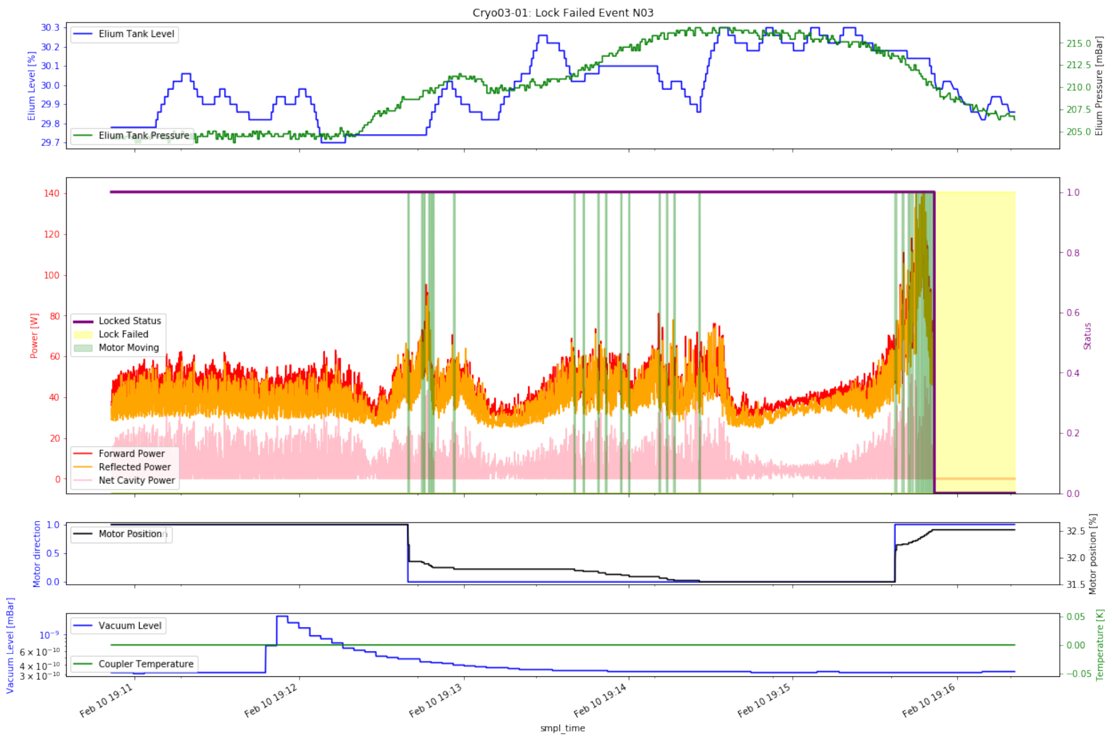


Figure 5.7: A compact graph representing an unlock event, with the trend of the main PVs on the last 5 minutes before the event.

was moving, to correct the phase error. On the third graph the tuner motor direction a position are shown. It is important to show the direction as a lot of instabilities happen when the motor inverts direction, due to its mechanical backlash. Finally, the last graph shows the cryostat vacuum level and the coupler temperature.

As can be seen, the coupler temperature is fixed at $0K$ which indicates a broken sensor, so this variable can't be used. The vacuum level shows some oscillations, but does not appear to be correlated with the cavity instabilities, or the probe is not accurate enough to show them. The instabilities can easily be seen on the RF power. At the beginning of the time period the forward and reflected power are almost constant, then there is a period with some turbulence where the power is unstable and the motor has to move to correct the error and at last there is a pick of power linked to the fault event. Looking at the cryogenic values, the Helium level seems pretty random, while the pressure shows a certain grade of correlation with the instabilities. In fact, when its derivative increases, so does the RF power.

5.2 Classical ML approach

This chapter will describe an Anomaly Detection method for fault prediction based on classical ML algorithms. Furthermore, we present an approach that exploits feature importance to guide the definition of an optimal sliding window and feature extraction on time-series.

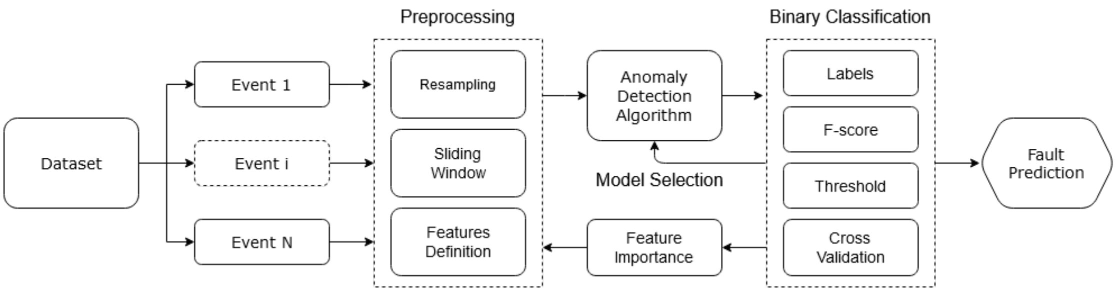


Figure 5.8: The main steps of the proposed approach presented as a flowchart.

Figure 5.8 shows a flowchart which summarizes the main steps of the proposed fault prediction pipeline. We start from the historical data divided into Events, as described in the previous section. Then, a preprocessing phase is used resample the time series to a constant sample frequency, to apply a sliding window and to calculate the feature which will form the final training dataset. These features are then passed to an unsupervised AD algorithm which will learn the normal behaviour of the system and will be able to output and anomaly score for each sample. A high outlier score is used as an indicator of an upcoming fault, since we expect to observe anomalous behaviours in the process variables before a fault. So, by applying a threshold on the prediction loss it's possible to obtain a binary classification which distinguish normal trends from upcoming faults. Finally, feature Importance is used to select the most important features and the best sliding window length. All these steps will be described in detail in the following sections.

5.2.1 Feature Extraction

The basic idea is to use `Pyod` models, which are designed to work with multivariate data, adapted to accept time series. To do so, the sliding window approach will be used, as it usually the solution used in literature. Given a certain window, data is not fed directly to the model but a number of features is calculated on the data, such as the average value of a certain PV to reduce the dimensionality to a single array, without losing important information.

The functions to apply to the window to extract the features can be of any kind, and thus it is possible to come up with different ideas. Following the considerations about the graph expressed above, the following features are calculated:

- **Average value** of the forward, reflected and net power, as well as the helium pressure.

- **Standard deviation** of the same variables.
- **Delta**, the Difference between the last value and the first for each of the four variables above. This is to capture the increasing, decreasing or flat trend of the PVs.
- **Motor moving time**: the amount of time with the motor moving out of the whole time window.
- **Motor direction inversions**: the number of times the motor has inverted direction during the time window.

To calculate these features a dedicated method can be implemented on the `Event` class. Initially a window length of 15 seconds and a step increase of 1 seconds are chosen. This means that the ending result has a point each second, calculated using the data from the 15 seconds earlier. It is important to use only data from earlier timestamps as in a real world scenario using this software in real time during the accelerator runtime, the future data is not available, obviously.

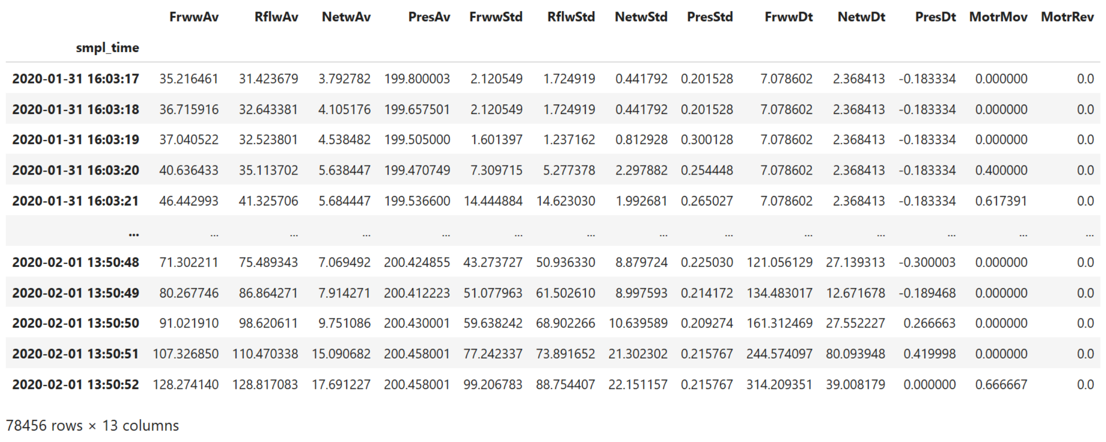
Usually in these cases the data is aligned to a certain time period, for example one point each second, but in this case the events are recorded as they happen, so there could be seconds with multiple points recorded and other where no event has happened. This makes the calculations of the features not so trivial. For example, to calculate the exact average value of a time series in a certain time window, one can use the following formula:

$$avg = \frac{(x_0 \cdot (t_{x_1} - t_{init})) + (x_1 \cdot (t_{x_2} - t_{x_1})) + ... + (x_n \cdot (t_{end} - t_n))}{T}$$

where T is the window time period, 15 seconds, x_0 is the last value in the time-series before the begin of the window, t_{init} and t_{end} are the timestamps corresponding to the beginning and ending of the window. This is basically a weighted average, the weight is the amount of time a value remains the same. If we avoid to adjust for time, fast spikes with values much greater than the average could completely distort the measurement. This formula is not easy to calculate since it requires one value outside of the time window, x_0, to calculate the time deltas, and to perform an inner product between two non-aligned arrays.

Considering that this is an operation to be performed quite often during the analysis, for example to test different parameters, it is important to implement the operation to be as fast as possible. Source code A.1 shows one possible implementation. First we define a `feature` dictionary, which is a data structure to represent a feature and all the required information to calculate it. The dictionary keys represent the function to apply to the columns in the second value of the tuple to obtain new columns with names in the first part of the tuple. The feature names uses the same prefixes as the data. A suffix is added according to their type, such as `Av` for average, `Std` for standard deviation or `Dt` for delta. Then the `resample` method of `Pandas` is used to substitute all the rows belonging to the same second with a single value which is the mean of the original values. After this step the data has a fixed period and is aligned, with one observation each second. This introduces a certain approximation with respect to the previous approach since the values are not time weighted, but since the time deltas in the data are small and have small variance, this is an acceptable approximation.

After that, a column is added to the output dataframe for each feature. The values are calculated using only pandas native methods, like `sum()`, `diff()`, `bfill()`, which are really fast. One noteworthy method is `rolling()` which provides an easy way to implement rolling window calculations. For example to calculate an array where each value is the sum of the values inside the sliding window for one of its positions, it is sufficient to write `df.rolling(window).sum()`.



	FrwwAv	RflwAv	NetwAv	PresAv	FrwwStd	RflwStd	NetwStd	PresStd	FrwwDt	NetwDt	PresDt	MotrMov	MotrRev
smpl_time													
2020-01-31 16:03:17	35.216461	31.423679	3.792782	199.800003	2.120549	1.724919	0.441792	0.201528	7.078602	2.368413	-0.183334	0.000000	0.0
2020-01-31 16:03:18	36.715916	32.643381	4.105176	199.657501	2.120549	1.724919	0.441792	0.201528	7.078602	2.368413	-0.183334	0.000000	0.0
2020-01-31 16:03:19	37.040522	32.523801	4.538482	199.505000	1.601397	1.237162	0.812928	0.300128	7.078602	2.368413	-0.183334	0.000000	0.0
2020-01-31 16:03:20	40.636433	35.113702	5.638447	199.470749	7.309715	5.277378	2.297882	0.254448	7.078602	2.368413	-0.183334	0.400000	0.0
2020-01-31 16:03:21	46.442993	41.325706	5.684447	199.536600	14.444884	14.623030	1.992681	0.265027	7.078602	2.368413	-0.183334	0.617391	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...
2020-02-01 13:50:48	71.302211	75.489343	7.069492	200.424855	43.273727	50.936330	8.879724	0.225030	121.056129	27.139313	-0.300003	0.000000	0.0
2020-02-01 13:50:49	80.267746	86.864271	7.914271	200.412223	51.077963	61.502610	8.997593	0.214172	134.483017	12.671678	-0.189468	0.000000	0.0
2020-02-01 13:50:50	91.021910	98.620611	9.751086	200.430001	59.638242	68.902266	10.639589	0.209274	161.312469	27.552227	0.266663	0.000000	0.0
2020-02-01 13:50:51	107.326850	110.470338	15.090682	200.458001	77.242337	73.891652	21.302302	0.215767	244.574097	80.093948	0.419998	0.000000	0.0
2020-02-01 13:50:52	128.274140	128.817083	17.691227	200.458001	99.206783	88.754407	22.151157	0.215767	314.209351	39.008179	0.000000	0.666667	0.0

78456 rows × 13 columns

Figure 5.9: The dataframe with the feature values. Each column has the name of the feature, as presented in the code snippet A.1.

The execution time of the feature calculation with this function is just 3 seconds, as compared to about 3 minutes when using the trivial python loops, which means that it is 60× faster. The function is fast enough that it is possible to run it multiple times with minimal slowdowns. The resulting dataframe, with all the values of the features is shown in figure 5.9.

5.2.2 Model selection

Once all the features have been computed, `Pyod` models can be used to calculate the outlier score. The models have a `fit()` method which uses the training data to set the internal parameters. It is then possible to use this fitted model to get the outlier score of new data. Finally, the output is re-scaled linearly to be in the $[0-1]$ range.

Initially three models have been selected among the available ones: Isolation Forest, LOF and KNN. The first two have already been presented in chapter 3, while KNN is a model which uses the distance from the Kth nearest neighbor as an anomaly score, because it can be viewed as a density measure. In figure 5.10 the outlier score of the event in event of figure 5.7 is shown.

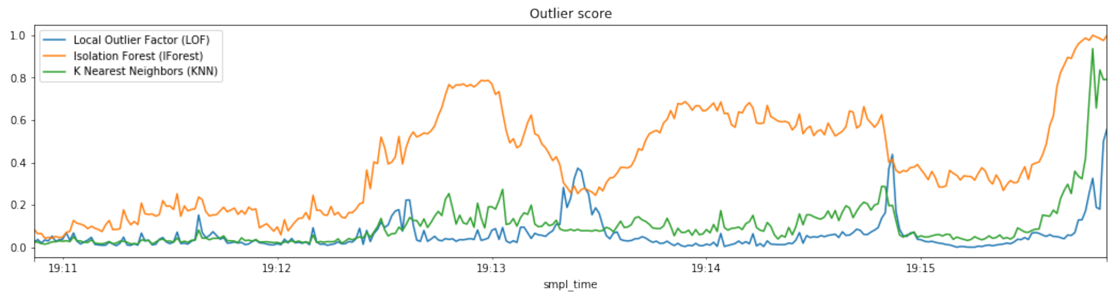


Figure 5.10: The outlier score of the event CR03-01 N03 with three different outlier detection models.

In this graph, the fault event happens on the far right of the plot, and the outlier score of the last 5 minutes is shown. The outlier score is clearly increasing towards the fault event, but the results vary a lot depending on the model and the particular event.

To check if there is a clear trend, the outlier score of all 169 events from all the cavities has been calculated and the following representation (figure 5.11) has been produced. In these graphs, the x axis represent the relative time remaining before each event. This means that the fault event is always aligned to the far left of the chart, and the time flows from the right to the left. On the right the time is 5 minutes before the event, while going towards the left the time goes toward zero seconds before the event. The black lines show the mean outlier score for each model, while the red one is the median. On the background each individual score is plotted.

This graph is interesting as it shows that actually, there is a correlation between the time remaining to the next event and the mean outlier score value. In fact, the mean and median values are much higher next to the zero, than in the rest of the graph. This gives hope to find a model able to calculate a score good enough to predict most faults, while minimizing the false positives.

To complete the classification process, the following step is to use the outlier score to classify each point either as an outlier or not. To do this, the standard technique based on a threshold can been used, as presented in chapter 3.2.1. When the outlier score is above a certain threshold the point will be classified as an anomaly. The value of the threshold must be chosen to maximize the number of correctly detected anomalies while minimizing the false positives. The F-score defined in section 3.2.3 represents both these goals and can be used as a single benchmark to maximize.

This procedure requires to know the real correct label of each point, to be compared to the one predicted by the model, to calculate the F-score. In this case the real label is not available, but a good way to estimate the true label can be deducted from the

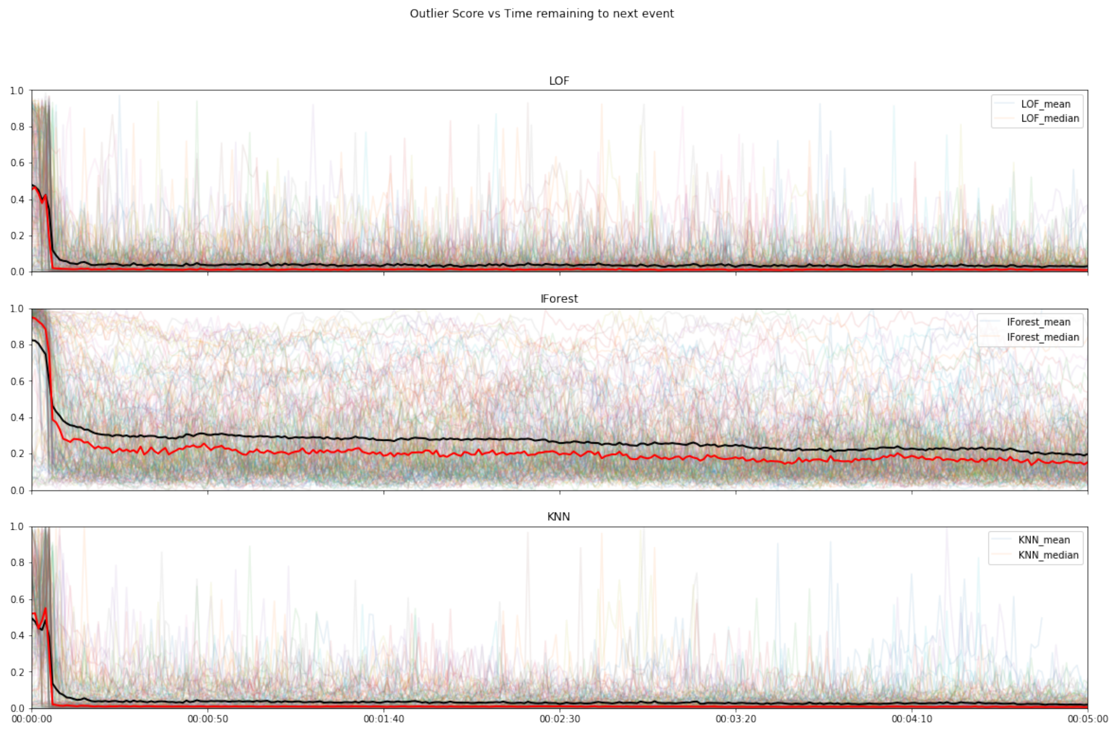


Figure 5.11: The mean (black) and median (red) outlier score for all the events, plotted against the time remaining to the next event.

definition of outlier that we introduced in chapter 3.2.1. In fact, since we are interested in anomalous points that precede a fault event, we can classify the points in the last few seconds before an event as outliers. All the other points, occurring during the normal operation of the machine will be classified as non-anomalous. The number of seconds where the points are considered outliers can be chosen with some experiments, and in this case it is set to 8.

Differently from other classification problems, here the data has time correlation. The final goal is to detect a fault event, and not to correctly classify each point. This means that it is sufficient to detect one point preceding an event as an outlier to reach the optimal result. For this reason the calculation of the F-score must treat each event as a single point. To check if any of the preceding points has been detected, the labels are not binary values but are constructed to carry the number of the event, while the normal points are set to -1.

$$Y_i = \begin{cases} -1 & t_i < t_{event} - 8s \\ event.ID & t_{event} - 8s \leq t_i \leq t_{event} \end{cases} \tag{5.1}$$

The F-score can then be calculated with the source code A.2. In this piece of code, the `available_events` are all the events that are present in the labels, meaning all the labels with a non-negative value without repetitions. The TPs are calculated as the intersection of the available events and the points predicted as outliers. The FPs are the points predicted as ouliers whose real label has a negative value, since it is a normal point. In this case the consecutive duplicates are removed as they refer to the same underlying anomaly. FNs is the number of events present in the labels minus the number of detected events. With these component it is possible to calculate the precision and recall, and finally the F-score. Here the beta value is set to 0.75 as it produces a good trade-off between FPs and FNs.

The F-score is now used to set the threshold to its optimal value. Since it is advisable to run this procedure each time the model is fit, a new class called `DeepicsModel` (see source code A.3) has been developed which wraps a `pyod` model and extends the `fit()` method to search for the best value of the threshold. While `pyod` models are unsupervised models, this new class represent a supervised machine learning model, since the labels are used to set one of its internal parameters. As can be seen in the `fit()` method, the value of the threshold is decided simply trying all possible values in a certain range and choosing the one which gives the best F-score. The range depends on the average value of the outlier score on the last few seconds before the fault, to accommodate for different models which gives vastly different outlier scores.

With these methods it is now possible to compare the performance of different models. The code above is run with different models from `pyod` and the training scores at different threshold levels are recorded. The training data consists of all the events of a single cavity, $CR06-01$, as this single cavity accounts for 70 events out of 169. The graph in figure 5.12 shows the resulting trend for the precision and recall scores.

In this graph it is easy to notice that when the precision is maximized, the recall is reduced, and vice versa. The goal is to obtain a value as close as possible to the upper right corner. The CBLOF model stands out as a clear winner. The `pyod` implementation of this algorithm follows its theoretical design, as presented in chapter 3.2.4, but the score only depends on the closest large cluster center. Looking at the numbers in table 5.1, the best F-score on the training data is 0.95 corresponding to a threshold of $1.66 \times 0.477 = 0.796$. 66 out of 70 events are correctly detected and just 3 false events are found.

Table 5.1: Results of the model selection.

Model	TP	FN	FP	Precision	Recall	F-score	Time [s]
CBLOF	**66**	**4**	**3**	**0.95**	**0.94**	**0.95**	**77.2**
KNN	64	6	7	0.90	0.91	0.90	2.0
FeatBag	57	13	7	0.89	0.81	0.86	29.8
LOF	55	15	7	0.88	0.78	0.84	2.7
LODA	61	9	23	0.72	0.87	0.77	0.6
iForest	46	24	25	0.64	0.65	0.65	5.9

Another important parameter is the fit time, that is the number of seconds required to fit the model. Usually more complex models perform better but require a lot of

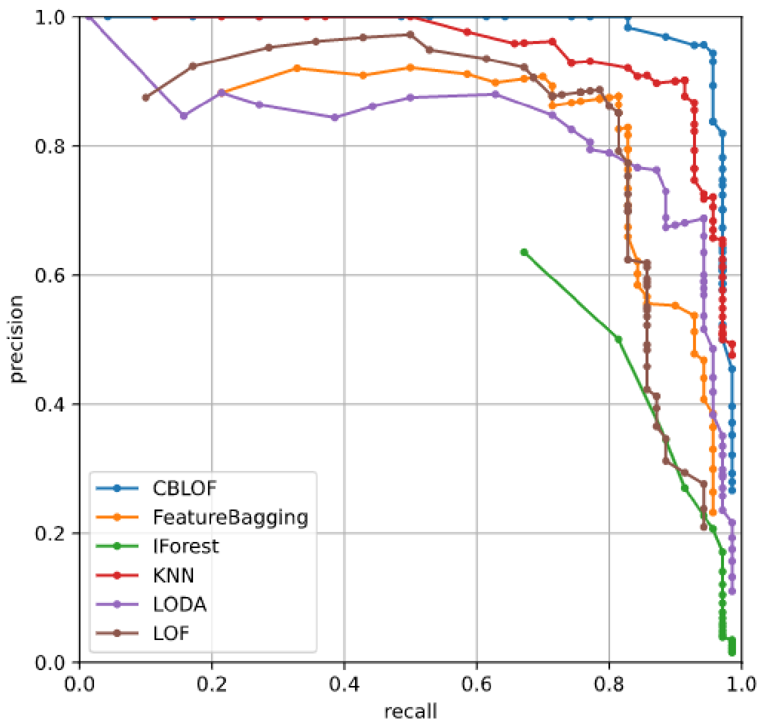


Figure 5.12: Precision Recall plot comparing different models on the training score.

computing power. In this case, the CBLOF model is slower than the other models in the list, but still requires a small amount of time. Other models are not included in the comparison since they are far too slow to be used in a reasonable amount of time on the given hardware. When the dataset grows, it may be reasonable to choose the KNN model as its performance are still good, but the fit time is much faster. For the following steps of this analysis, the CBLOF model will be used.

5.2.3 Experimental Results

In the previous section the models are compared based on the training score. This represent the ability to predict data which is already known to the model. To assess the real performances of the model it is necessary to test it with new data, and calculate F-score of predicted values.

The first step is to split the data in two parts: the training and test dataset. The first one is used to fit the model, while the second one to calculate its F-score. Since the data is already split into events, these are further split into the two datasets randomly. The 70 event from the cavity CR06-01 are here used, with 49 events in the train dataset and 21 in the test one. The resulting training F-score is 0.84, lower than the previous one since less data is now available. When predicting the test set, the resulting F-score is 0.89, as shown in table 5.2.

Table 5.2: Results when predicting the test dataset.

F-score	0.899
Precision	1.000
Recall	0.762
TP	16
FP	0
FN	5

In the previous case the score is quite good, as it is not lower than the training score, but it depends heavily on the random split of the dataset. There are some events that are easier to model and other which are more complex, thus they can make a big difference when they are assigned to the test or training set.

To avoid this kind of problems, cross validation can be used. This is a procedure where the data is split into k parts: $k - 1$ parts are used as training set while the remaining as test set. This is repeated k times, using each time a different part as test set. The average F-score is then returned as a good approximation of the performance of the model on new data. The `joblib` library is here used to execute the run function on all the k split configurations in parallel. With this procedure the average F-score obtained with 5-fold cross validation is 0.83 ± 0.24.

5.2.4 Permutation importance

To try and improve the previous result, one idea is to select the best set of features to use. In fact, the features to calculate were chosen based on the knowledge of the underlying physical processes, but they may not be optimal. If a feature does not bring any useful information about the upcoming faults, it is better to remove it to reduce the dimensionality of the input and the complexity of the model.

For this reason, the permutation feature importance procedure is used. In this technique the values of one single feature are randomly shuffled and the decrease of the F-score is calculated. By executing it for each feature, it is possible to detect the features that have most impact on the final F-score. By running this procedure, the results in figure 5.13 are obtained.

These results show that only one single feature is significative, the forward power increase during the time window, with the other mostly useless. This is not a good result, as it means that the model is not good enough to extract information from the other features available. It is thus necessary to understand why this is happening and how to fix this situation.

The first idea is to test a reduced set of features. In fact, many of the features are highly correlated, and this could mean that shuffling one single feature is not enough to affect the F-score. Unfortunately, this is not the case, as can be seen in figure 5.14

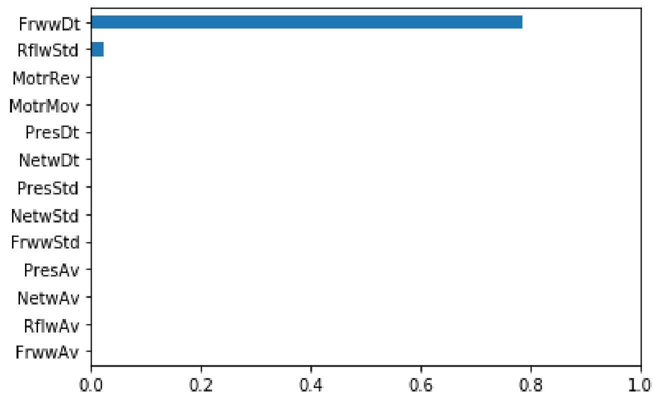


Figure 5.13: Feature importance.

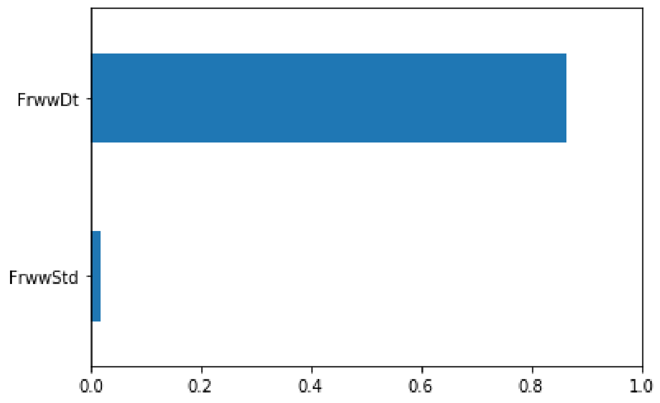


Figure 5.14: Feature importance on a reduced set of features.

The F-score is still affected only by one feature, by a big margin. The overall F-score calculated with cross validation is basically unchanged, with a value of 0.81 ± 0.30.

5.2.5 Window length optimization

The window length was set to 15 seconds, as it was deemed a reasonable value by looking at the graphs, but there is no evidence that this is the optimal value. In fact, different features may be more meaningful on shorter windows. To test this hypothesis, the permutation importance procedure is run multiple times, with different window length ranging from 2 seconds to 15. The resulting graphs is shown in figure 5.15.

This graph is quite interesting as it shows how each feature importance varies at different window lengths. It is easy to notice that at 15 seconds only one feature is important, coherently with previous observations, but with shorter windows other features gain some relevance. The forward power delta feature gains importance when using longer windows, probably because the delta value becomes bigger. Many other features

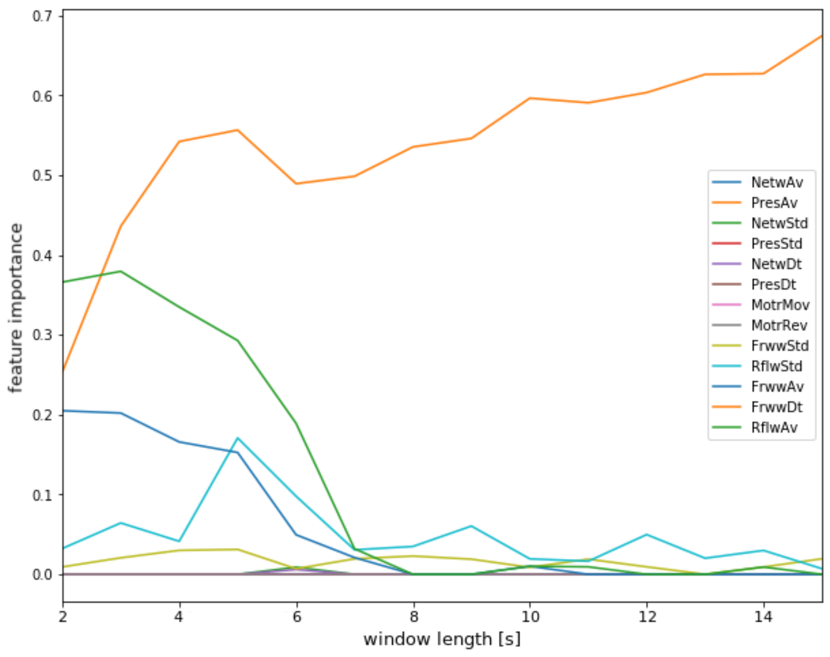


Figure 5.15: Feature importance with different window lengths. On the x axis there are the different window lengths, while on the y the corresponding feature importance.

instead seem to be most significant below the 6 seconds length, probably because the anomalous data is not appreciable much before the event, as can be seen in the graphs of the outlier score (figure 5.11).

From this graph it is then possible to choose the best window length to use. In this case 5 seconds is chosen because at this length at least 4 features have an importance above 0.1. In figure 5.16 it is possible to see how the window length has changed the bar plot of the feature importance compared to figure 5.13. By using this new window length and the reduced set of features with an importance of 0.1 or more, the cross validation F-score increases to 0.87 ± 0.19. This is small but clear improvement.

5.3 Deep Learning approach

This section presents a work on the same anomaly detection and fault prediction task addressed in the previous section, but using a different approach. The goal is to detect anomalies in the RF control system data and use them to predict unlock events. Instead of using classical ML models which require manually extracted features, a deep learning approach is used. This approach is more flexible since it can be more easily applied to different systems with different process variables. Furthermore, deep learning models have shown the ability to learn complex patterns in the data, which may result in more accurate predictions. The flowchart in figure 5.17 summarizes the main steps of the

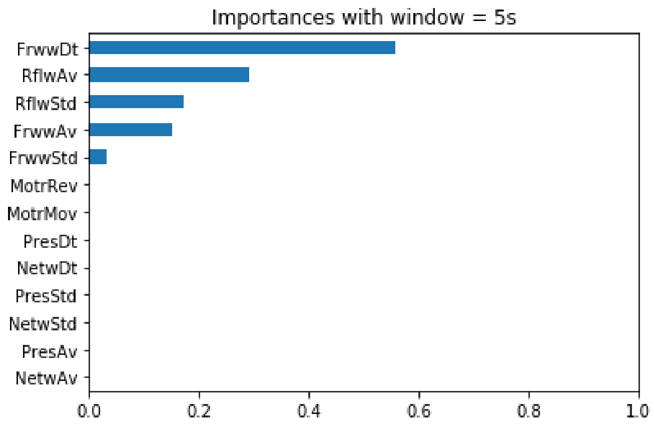


Figure 5.16: Feature importance with the optimized window length of $5s$.

proposed approach. This pipeline builds upon the work presented in section 5.2; with

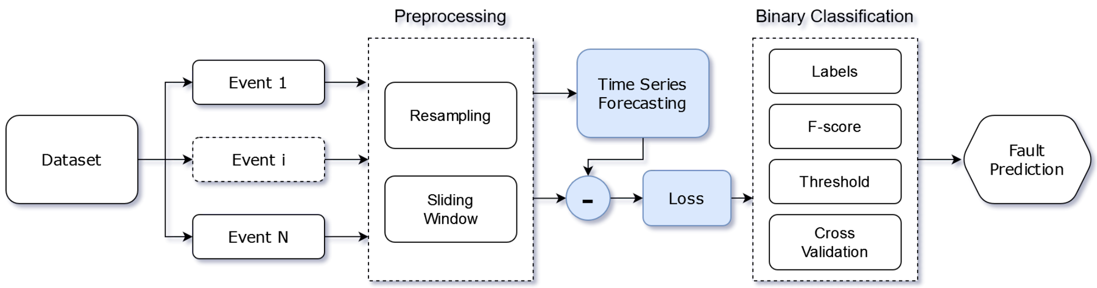


Figure 5.17: The anomaly detection pipeline with time-series forecasting.

respect to the original work, the main differences are highlighted in blue: previously an anomaly detection model was run on a set of manually extracted features to obtain the outlier score, now the score is calculated from the prediction loss of a time series forecasting model fit on the original data. The main idea is that a good prediction model should be able to forecast the future value with a small error during normal operation, when all the variable behave as expected, while the prediction error could be much bigger for anomalous values. Thus, by comparing the prediction with the next value, as soon as it is available, it's possible to compute the prediction loss and use it as an outlier score. The following paragraphs explain the details of this approach.

5.3.1 Dataset preparation

The dataset used in this work is the same as the one used in the previous section. It is composed of the evolution over time of multiple process variables recorded by the RF control system, and thus it is structured as multiple concurrent time series. Each data point includes the PV name, the acquisition time and the value. In this case we decided to work with the following PVs:

- *Lock* and *Lock Failed* statuses;
- Forward RF power;
- Tuning motor direction and moving flag;
- Cryostat liquid helium tank pressure.

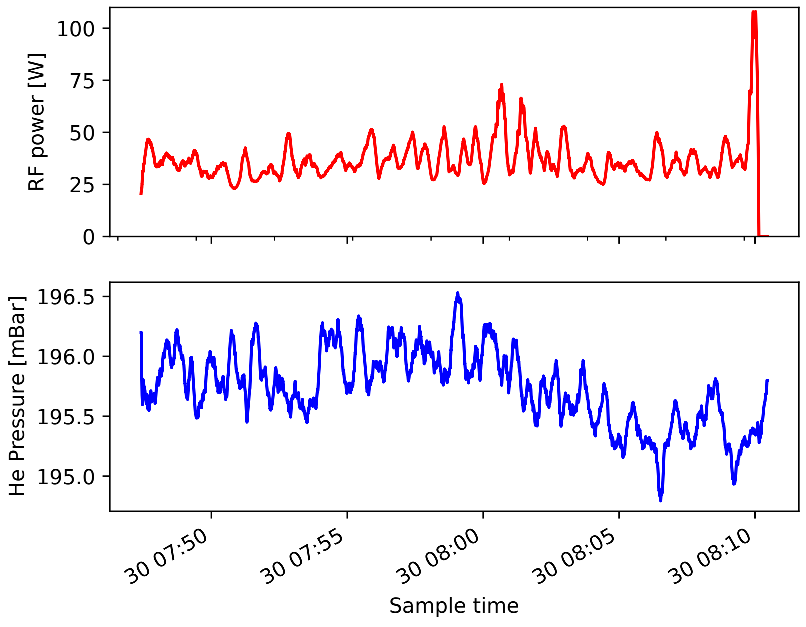


Figure 5.18: The trend of the RF power and He Pressure before an unlock event.

After splitting the data into events, the classical ML methods required to calculate the custom features on a sliding window. In this case, we have to prepare the data to be fed into a neural network, and thus three transformations are required.

As can be seen in Figure 5.18, different PVs have different engineering units and different scales, which is not ideal to be used as input to a neural network because it may result in a slow or unstable learning process. For this reason, the data is first normalized using the Standard Scaler, which imposed the data to be centered in zero, with unit variance. As shown in equation (5.2), this is done by subtracting the mean value and dividing by the standard deviation. The standard scaler is used in this case as the data do not have an obvious maximum value and thus it's not trivial to normalize it based with respect to the maximum value.

$$\mathbf{z} = \frac{\mathbf{x} - \mu}{\sigma} \tag{5.2}$$

The second transformation is the resampling operation. In fact, time series models expect to have an input with a constant and aligned sampling rate. Instead, the data was recorded during the operation in an event driven way, meaning that each value is written to the database as soon as it changes. This is useful not to lose any value and to save storage space for slow-changing PVs, but is not very convenient during the analysis of the data. Thus, all the data was resampled to the same sampling rate, simply using the mean value in the sampling period as the new value. When no data was recorded in the sampling period, the last valid value was propagated. The resampling operation is performed with `Pandas` for performance reasons as explained in section 5.2.1. The resampling period is an important parameter to choose: choosing a very fast sampling rate the dataset size and the execution time increases exponentially, without adding any meaning information, but a very long sampling period means losing a lot of information due to the averaging.

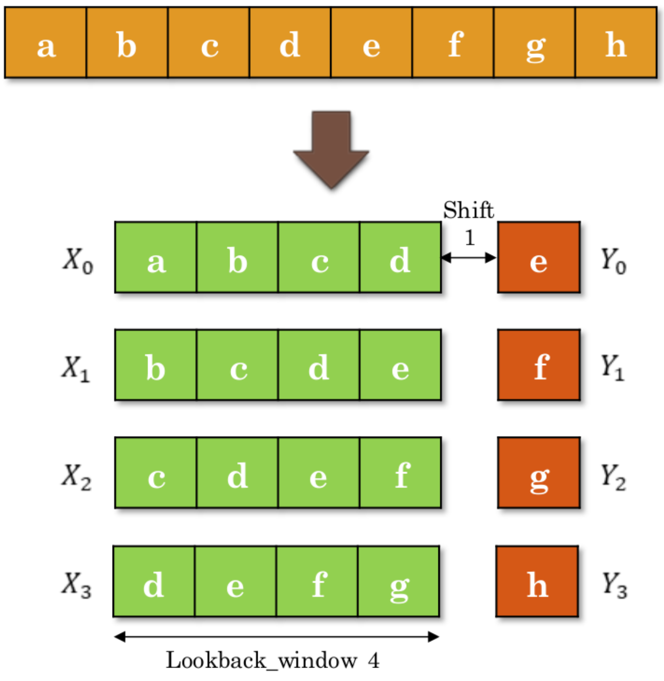


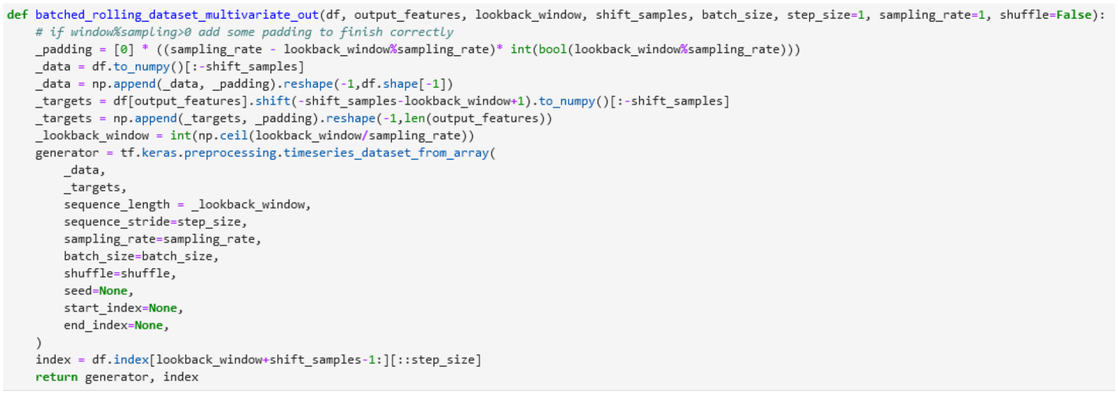
Figure 5.19: A sliding window on time series example.

The final step is the creation of the actual dataset for forecasting using a sliding window approach. This is a crucial part of the process to build the model, as it can have a direct impact on the training of the neural network and the final result. An example of sliding window application with a single feature is shown in Figure 5.19. From the initial sequence of values of the time-series we want to create a set X where each sample is a subsequent portion of the original data (with the correct time order), and a set Y with the future value of each window. Given an element from the X set, our model will be trained to predict the corresponding value in Y. There are many parameters that can be adjusted:

- **input PVs**: this indicates which PV to include in the X samples;
- **output PVs**: which PV to include in the Y samples;
- **lookback window**: the length of the window;
- **shift samples**: how *far* in the future to predict in number of samples;
- **batch size**: the number of samples in a training batch;
- **step size**: how many samples to slide between one window and the next;
- **window sampling rate**: optionally subsample the values in the window to reduce its size while keeping old values;

These parameters determine the forecasting horizon, which is selected aiming both at proving accurate predictions in useful time and at considering typical dynamics of a failure (ie. the amount of time on which anomalous behaviour start to verify before the unlock event). Furthermore, the length of the window is adjusted to be able to capture trends far in the past, while subsampling is then applied to limit the number of points in the window. This enables the use of smaller models, which have a limited number of nodes in the input layer. By applying the subsampling after the windowing, the number of samples in (X, Y) is not artificially reduced. Finally, the obtained dataset correlates one observation window with a future value, and thus can be shuffled without any problem in order to obtain a more robust result.

A side effect of the sliding window operation is the explosion of the size of the data, since each original point is repeated many times in the observation windows. For real world use case, like the one presented in this paper, this can become a problem as it's easy to incur into memory limitation issues, even when using modern hardware. In fact, with the naive approach the dataset is fully loaded on the device memory and can exceed its size, in this case the 16GB of memory of the available GPU. For this reason the implementation of the dataset should use streaming data type which loads the data in memory in chunks to avoid hitting the memory limit. In this case a custom rolling window function with all the aforementioned parameters was developed, exploiting the TensorFlow *Dataset* data type. This guarantees that data is streamed to the GPU in chunks, instead of being loaded all at once. Furthermore, having a single function with all the parameters allows to easily change the dataset parameters and test different configurations. The figure 5.20 shows the implementation of the function, which shows the complexity of dealing with the flexibility introduced by all the rolling window parameters.



```python
def batched_rolling_dataset_multivariate_out(df, output_features, lookback_window, shift_samples, batch_size, step_size=1, sampling_rate=1, shuffle=False):
    # if window%sampling>0 add some padding to finish correctly
    _padding = [0] * ((sampling_rate - lookback_window%sampling_rate)* int(bool(lookback_window%sampling_rate)))
    _data = df.to_numpy()[:-shift_samples]
    _data = np.append(_data, _padding).reshape(-1,df.shape[-1])
    _targets = df[output_features].shift(-shift_samples-lookback_window+1).to_numpy()[:-shift_samples]
    _targets = np.append(_targets, _padding).reshape(-1,len(output_features))
    _lookback_window = int(np.ceil(lookback_window/sampling_rate))
    generator = tf.keras.preprocessing.timeseries_dataset_from_array(
        _data,
        _targets,
        sequence_length = _lookback_window,
        sequence_stride=step_size,
        sampling_rate=sampling_rate,
        batch_size=batch_size,
        shuffle=shuffle,
        seed=None,
        start_index=None,
        end_index=None,
    )
    index = df.index[lookback_window+shift_samples-1:][::step_size]
    return generator, index
```

Figure 5.20: The custom rolling window function.

5.3.2 Forecasting Models

Given the dataset described in the previous paragraph, three Neural Networks were tested as forecasting models. The first one is based on Long Short Term Memory (LSTM) layers ([63]). This architecture, introduced in section 3.3.4, can propagate a "state" between inputs so that the output depends both on the input and on the state, effectively exploiting the time dependency between the data points. In particular, LSTM layers are designed so that the output does not depend only on the state from the last few nodes, but can have long term dependencies.

As shown in Fig. 5.21, we chose to use a Neural Network with 2 stacked LSTM layers with 64 units each, both returning the whole sequence of outputs of each unit. Thus, the second layer receives in input these output and reprocesses them sequentially to obtain new outputs. These are finally fed into a fully connected layer with a single output to perform regression on the prediction value. A linear activation function is used on the output neuron.

The second model was built using a Temporal Convolutional Network (TCN) from [82]. We already introduce this type of NN in section 3.3.4, but we recall that this model is based on 1D causal convolution layers arranged so that the output depends only on the inputs from earlier in the sequence. Furthermore, by stacking the layers it's possible to obtain a large receptive field, meaning that the input sequence can be long and the data dimensionality is reduced by the convolution. Skip connections can be added to propagate the gradient between deep stacks of layers. In this case the complete network is composed by a stack of 3 layers with dilations of 1, 4 and 16, 64 filters and skip connections. After that a fully connected layer with 16 nodes precedes the single output node with linear activation.

The aforementioned models both use the trend of multiple input time-series to forecast a single value: the RF power. The choice of the single variable to forecast is arbitrary and can have a big impact on the final results. An alternative approach is to forecast multiple values at once and then combine the losses of each prediction to obtain a more robust

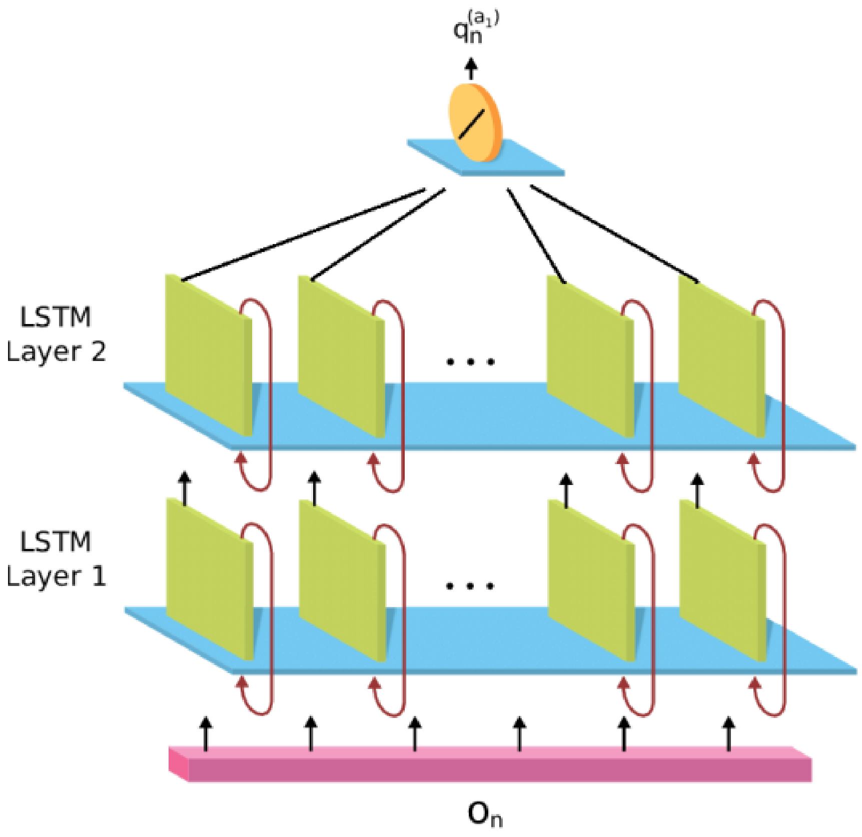


Figure 5.21: A simplified schematic of the LSTM based neural network architecture.

outlier score. Thus, we developed a third model, based on a TCN network, to predict the forward and reverse RF power and the helium pressure. The dataset is changed to include the three variables in Y and the output layer of the network is increased to 3 nodes. Finally, the average of the prediction loss in the three variables is used as the final score.

The hyperparameters of the three model were chosen empirically balancing accuracy and fit times, since the final goal is not to obtain a perfect forecasting model but to produce a good anomaly score. These models are trained with the Adam optimizer for 10 epochs using the Mean Squared Error (MSE) loss function. The parameters of the dataset presented in the previous section are chosen as shown in Table 5.3. This means that the LSTM model predicts 30s in the future by looking at 1 value every 30 seconds in the last 256 minutes, while the TCN models look at one value every 10s in the last 30 minutes and predict 10s in the future.

The three models achieve similar performance, with the LSTM network obtaining a MSE 0.723 and a Mean Absolute Error (MAE) of 0.5222, while the TCN achieved a MSE of 0.702 and MAE of 0.5229. The TCN predicting multiple variable reaches a MSE of

Table 5.3: Dataset preprocessing parameters

	LSTM	TCN	TCN Multi
Data sampling period	0.5s	1s	1s
Input PVs	All	All	All
Output PVs	RF power	RF power	RF power, He pressure
Lookback window	30720	1800	1800
Shift samples	60	10	10
Batch size	128	128	128
Step size	3	1	1
Window sampling rate	1/60	1/10	1/10

0.503 and a MAE of 0.4044. By computing the absolute value of the difference between the predicted value and the actual value, it's possible to obtain an outlier score for each point. In fact, we expect the forecasting model to be less accurate on anomalous data point. This can be observed in Figure 5.22, where the time series of all the prediction errors in the last 300 samples before each event are shown. The prediction errors explode in the last few samples before an event, indicating the presence of anomalous data and thus anticipating the fault event.

5.3.3 Experimental Results

The outlier score obtained from the prediction error, as presented in the previous paragraph can thus be used as an outlier score to predict faults, following the same steps of the classical ML approach. In fact, given the outlier score the following steps are the same.

A simple threshold function is used to classify each point between anomalous or not. The value of the threshold must be chosen to maximize the number of true positive while avoiding false positives and false negatives. For this purpose artificial labels were computed from the Y dataset using equation (5.1): all the datapoints which are far from the next events are assigned a label of -1, meaning that they are considered non-anomalous. The samples in the last 5 seconds are assigned a non-negative value, identifying the corresponding event, to indicate that they are considered outliers. After that a threshold is applied on the prediction loss and all the points are classified as either 0 or 1, where 1 is considered an anomalous point. Now, a fault is considered correctly classified if at least one point in the last 5 seconds before its occurrence is classified as anomalous. Vice versa, a false positive is counted for each sequence of 1s in the predicted labels which are not in the last 5 seconds before the event. Finally, the correct threshold must be chosen. This can be done by testing the F-score (defined in section 3.2.3) at different threshold levels, and choosing the one that minimized it.

The results of this procedure for the three models are shown in Table 5.4. The metrics

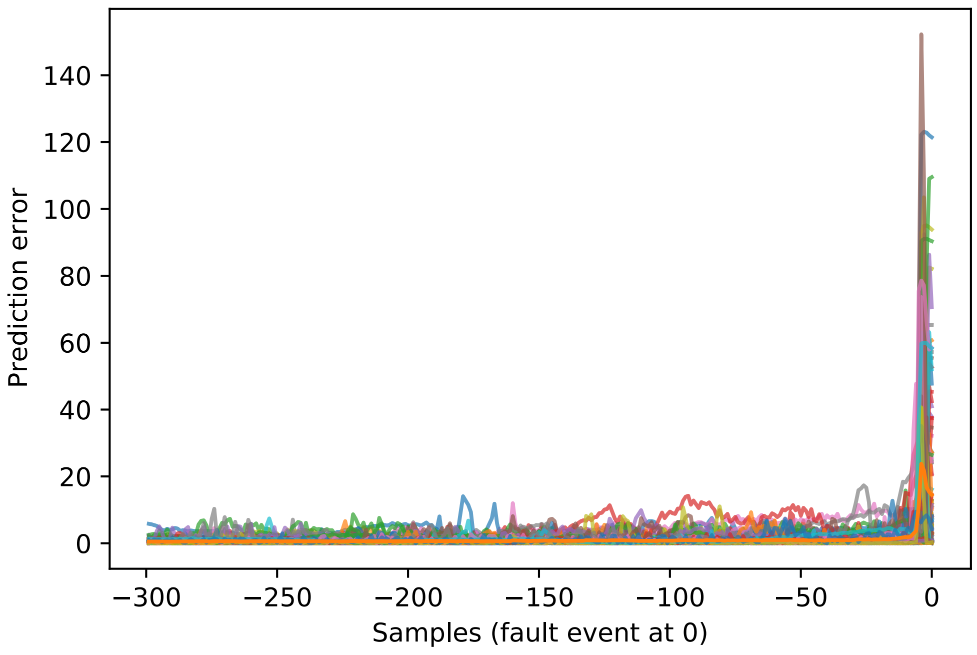


Figure 5.22: The prediction errors on the last 300 samples of all the events, aligned with fault at sample 0.

are calculated using a 5-fold cross validation procedure, where the dataset is split into 5 folds and the training is repeated 5 times, each time using a different fold as test set.

Table 5.4: Fault prediction results

	LSTM	TCN	TCN Multi
Fscore	0.578 ± 0.004	0.642 ± 0.006	0.691 ± 0.006
Precision	0.713 ± 0.025	0.743 ± 0.043	0.831 ± 0.023
Recall	0.419 ± 0.019	0.507 ± 0.036	0.516 ± 0.019
TP	25.6 ± 1.20	66.4 ± 4.92	67.6 ± 2.57
FP	10.4 ± 1.85	23.6 ± 6.65	13.8 ± 2.78
FN	35.4 ± 1.20	64.4 ± 4.71	63.4 ± 2.57

The results show that these models are able to correctly predict many faults with few false positives. When compared, TCN models are more robust and obtain a higher F-score than the LSTM one, and a further advantage is gained by predicting multiple variables. The Precison-Recall plot in Fig. 5.23 shows the behaviour of the models when changing the threshold level, and the TCN Multi outperforms the other on the whole range of threshold. Moreover, the graph highlights that by accepting a lower recall value it's possible to reach high levels of precision, which may be useful depending on the use case.

These results are comparable with the ones of a few classical ML models presented in section 5.2, trained on the same dataset, even though they do not reach the F-score of 0.87 of the best one. Even so, DL models have the unique advantage of being usable without any manually crafted feature, which means that they are directly portable to different environments and different kinds of faults. Furthermore, while the dataset includes millions of data points, not all of them are usable to build the training models, and they pertain only to a short time span of 1 month of operation, meaning that they did not include much variability. Even if the DL training did not take into account the faults, the data included less than 200 events, a very small number to build a confident model. This is probably an advantage for simpler models, but by collecting more data and recording more events DL models are expected to reach better performance.

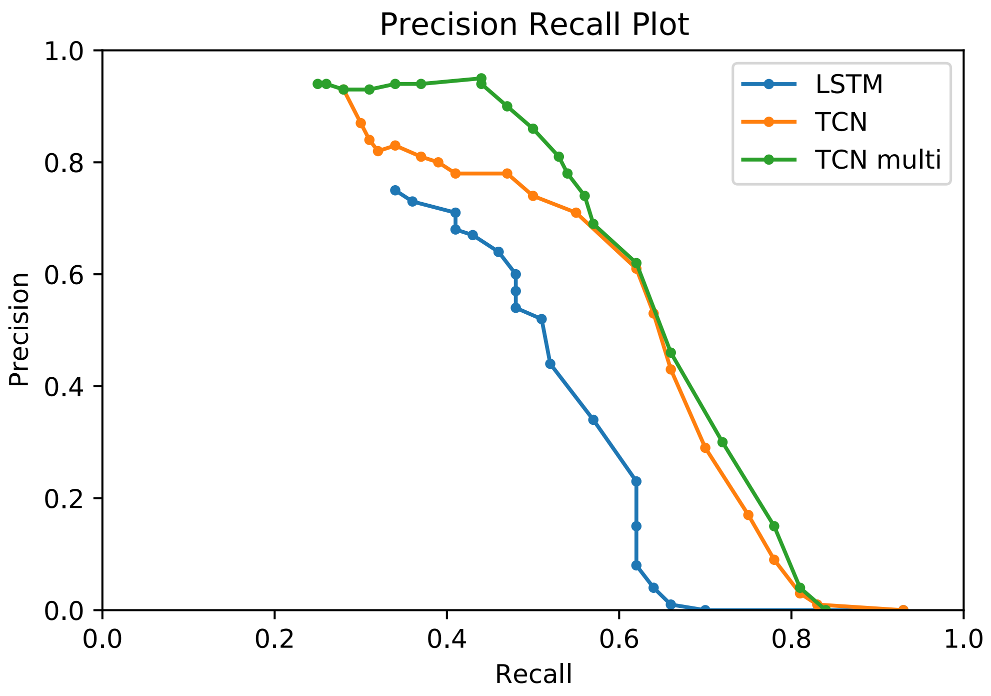


Figure 5.23: The Precision-Recall plot of the three models.

5.4 Conclusions

In this chapter we presented two separate approaches to the problem of fault prediction in the RF control system of the ALPI particle accelerator. The first one is based on the CBLOF method, while the second one uses LSTM and TCN networks. The original data is composed of multiple time series of the process variables from a one month-long accelerator run. After a preprocessing phase, where the data is split into events and resampled to a constant frame rate, the actual training dataset is computed. In the first approach, the dataset is composed of manually extracted features, while in the second one the dataset is composed of sliding windows of the original data. Classical ML models are trained to output an outlier score for each point, while DL models predict the future value of the process variables and the prediction loss is used as anomaly score. In both cases, the F-score is used to select the best threshold on the outlier score to distinguish between normal and anomalous points.

The first method has led to good results, especially after the permutation importance procedure which helped to identify the most relevant features and the optimal length of the sliding window. The final F-score is 0.87 ± 0.19, which is a good result considering the small number of events available. Even so, there are some aspects that require some more investigation. For example, when dealing with events from different cavities the model is less accurate. One possible improvement is to use a different threshold level for each cavity or for each cryostat. More data is also required to gain enough confidence in cavities with fewer events.

The second method, based on DL models, has shown promising results, with an F-score of 0.691 ± 0.006 for the best model. This is comparable with the results of the classical ML models, even though it does not reach the best result. The main advantage of this approach is that it does not require manually crafted features, and thus it can be easily applied to different systems and different faults. Furthermore, it is expected to scale better with more data and more faults, which are not available at the moment.

At the same time, it would be extremely interesting to integrate these models in the control system of the accelerator and to test them during the next run of the accelerator. Since the control system is easily accessible and scriptable with python, the integration is straightforward. By analyzing the behaviour of the models on real-world scenarios it is possible to better understand their strengths and limitations to plan future upgrades. Even if the models are not perfect, they could already bring some benefits to the accelerator operation. For example, the models could be used to show a live graph of the anomaly score to the operators, to help them understand how well the accelerator is working. In fact, even if everything is running correctly, there is a big difference between a system working at the limit of its capabilities and a system working with a lot of margin. An operator could use this information to decide to adjust the operating parameters of the RF control system to be more conservative to avoid possible faults or instead to push the performance to the maximum. In this case the anomaly score could be considered as a sort of *health indicator* of the accelerator and could fall under the category of virtual sensors introduced in section 4.2.2.

In general, this experience highlighted the opportunity to introduce anomaly detection in the control system of ALPI and similar accelerators to improve the operation and reduce the number of faults. This work could be extended to different subsystems of the accelerator, with the aim of building a complete fault prediction and alarm system. Furthermore, by analyzing the prediction models it could be possible to understand the root cause of the faults and to improve the accelerator design to avoid them in the future.

Chapter 6

Reinforcement Learning for Beam Emittance Optimization

6.1 Introduction

This chapter proposes a novel approach based on a RL model to tune online the control system parameters of a particle accelerator and obtain the minimal beam emittance. The approach is tested on the ADIGE MRMS beam line at INFN Legnaro National Laboratories to demonstrate its feasibility. As presented in chapter 2, this beam line includes an electrostatic multipole with 48 independent voltage terminals, and thus represents the perfect example of a beam transport element which is hard to tune manually.

As reported in section 2.5 and section 4.2.3, beam dynamics is one of the areas where ML models can have a big impact on the performance of a particle accelerator. In fact, it requires to compute the optimal setpoint of each beam transport element to optimize a number of beam properties. In particular, the beam emittance is a fundamental parameter which determines the quality of the beam and thus the performance of the accelerator. We recall that the emittance is a measure of the spread of the beam in the phase space, and thus a low emittance corresponds to a narrow beam with a small spread of the particles velocities and positions. This is fundamental for the performance of the accelerator, since a narrow beam can be focused more easily and thus can be accelerated to higher energies.

Finding the configuration of all the voltage values of the multipole that minimize the beam emittance is not trivial, given its high parameter space and its strong dependency on the details of the real machine. As we highlighted in section 2.5.2, while it's possible to simulate the non-linearities introduced by the MRMS dipoles, they are highly dependent on the actual position of the beam compared with the dipole, its focusing, energy spread and the actual distribution of the magnetic field inside the dipole magnet. In practice, the optimal solution can be found only by iteratively adjusting the configuration on the real machine. For this reason, this problem is used as a first demonstration of the usage of RL models for beam commissioning.

We want thus to develop a model which, given the real machine, is able to iteratively converge to the multipole configuration which minimizes the beam emittance after the MRMS. This model can be trained on a simulation but over time it can be fine-tuned on the data from the real machine. The model should be generic enough to be able to reach the correct solution even when the optimal value changes: our goal is for the model to not learn a single optimal configuration but to be able to iteratively move towards the optimal solution. In fact, the optimal solution is not static and depends on the beam parameters, which can change over time. Learning a single solution with RL would be equivalent to use any other optimization algorithm, such as the simplex method, while learning a policy to move towards the optimal solution is a more complex task which requires the ability to generalize to different situations. Furthermore, a good model could be able to converge to the solution in far fewer step than any optimization algorithm, as it incorporates the knowledge of the physics of the problem, and thus could be used online much more efficiently. Given the time required to measure the beam emittance, which is in the order of the minutes, moving from a method that requires hundreds of iterations to one that requires just a few would mean moving from a timescale of hours to minutes. This would be a huge improvement for the beam commissioning process.

6.2 Physics simulation

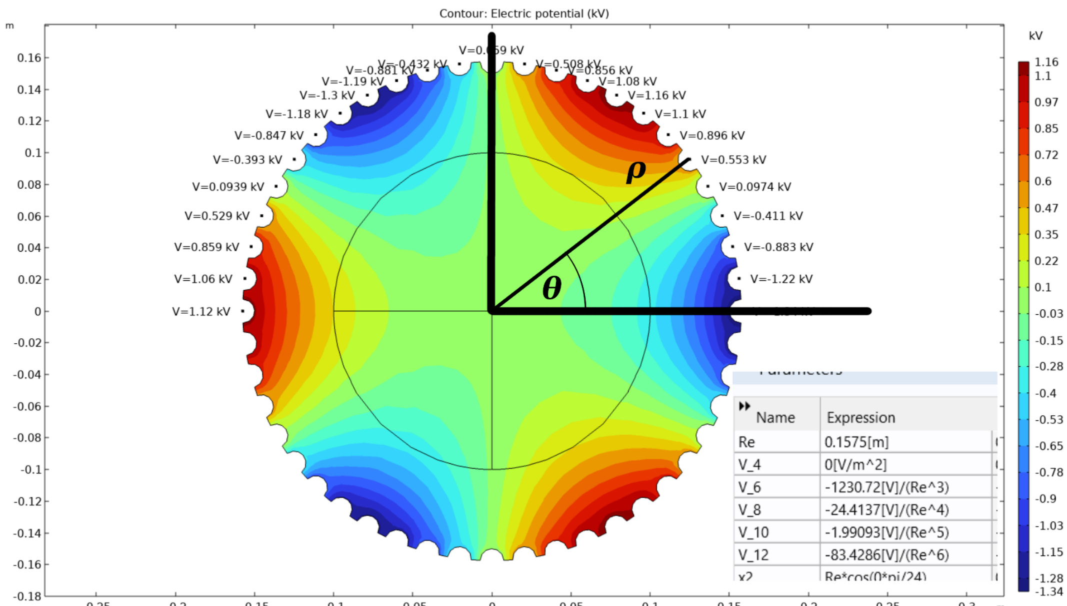
The standard approach to solve this kind of problems is to use an optimization algorithm on a physics simulator. In particular, the TraceWin [166] simulator is used, a common tool in the community. This simulator is able to compute the beam dynamics of a particle beam along a beam line, with a configurable number of particles. Given a high enough number of particles, the simulator outputs a beam distribution which can represent the real beam.

First we need to define the beam line shown in figure 2.8 using the syntax of the simulator. This is usually provided by the physicists who designed the line and is reported in source code A.4. While a complete description of the syntax is out of scope for this thesis, it can be retrieved on the TraceWin documentation [167]. The beam line is simulated from the output of the charge breeder, which acts as an ion source for the beam dynamics, up to the beam emittance meter at the exit of the MRMS. In particular, we can highlight:

- lines 4, 7: the *AD.SO.01* and *AD.SO.02* solenoids
- line 12: the electrostatic field of the MRMS platform
- line 17, 55-59: the *AD.BI.04* and *AD.BI.05* beam diagnostics
- lines 20, 50: the *AD.ST.04* and *AD.ST.05* steerers
- lines 22, 24, 44, 46: the *AD.1EQ.01*, *AD.1EQ.02*, *AD.1EQ.03* and *AD.1EQ.04* quadrupoles

- lines 27, 41: the *AD.D.02* and *AD.D.03* dipoles
- lines 29 to 38: the *AD.EM* multipole

As can be seen the multipole is obtained as the superimposition of 4 different field maps. The multipole has 48 independent terminals, and thus in theory our configuration should contain 48 parameters. In practice, we are not interested in generating a skewed field, and thus the terminals are connected to 24 power supplies symmetrically over the y axis. Furthermore, since the non-linear effects introduced by the dipoles follow the order of x^2, x^3, x^4, etc. where x is the size of the beam envelope on the x axis, we chose to correct them by using a linear combination of the fields of sextupole, octapole, decapole and dodecapole. This effectively reduces the number of parameters to 4 and forces the solution to follow the physics of the problem.



Name	Expression
Re	0.1575[m]
V_4	0[V/m^2]
V_6	-1230.72[V]/(Re^3)
V_8	-24.4137[V]/(Re^4)
V_10	-1.99093[V]/(Re^5)
V_12	-83.4286[V]/(Re^6)
x2	Re*cos(0*pi/24)

Figure 6.1: Example configuration of the multipole voltages.

Then, with the following formula we can derive the voltage value on all the terminals (see Figure 6.1):

$$\phi(\rho, \theta) = \sum_{n \in [3,4,5,6]} A_n \rho^n \cos(n\theta) \tag{6.1}$$

where ρ is the multipole cilinder radius, θ is the angle of each high voltage terminal, and A_n is the parameter to learn. This acts as a weight for each of the basic configurations (sextupole, octapole, decapole and dodecapole), so that the final solution is a linear combination of those.

6.2.1 Traditional optimization methods

Given the setup presented in the previous section, we can now use the physics simulator to compute the beam emittance after the MRMS for a given set of A_n values. Then, we can use an optimization algorithm to find the optimal configuration. In particular, we can use the simplex method, which is a well known algorithm for the minimization of a function and is integrated into TraceWin. The optimization is shown in figure 6.2. As

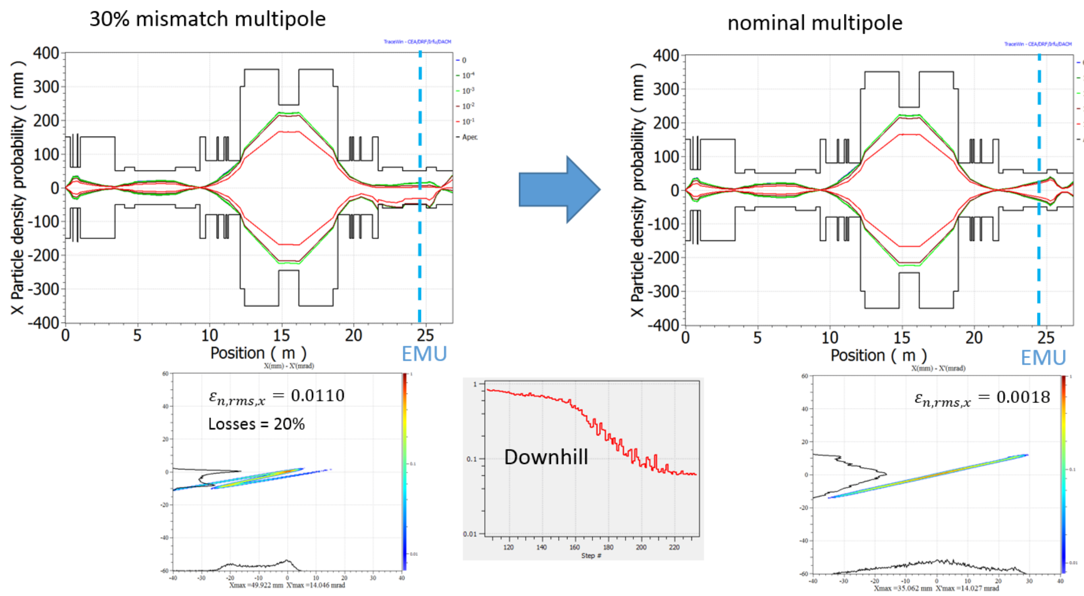


Figure 6.2: simplex method for the minimization of the beam emittance.

can be seen, the simplex method starts from a random configuration of the A_n values and then iteratively moves towards the minimum. Initially the multipole does not correct the aberrations of the beam and thus the beam emittance is high and its graph is not a simple ellipse shape but appears distorted. Furthermore, the beam envelope after the dipoles is not centered on the x axis, and thus the beam is partially lost. As the simplex method iterates, the beam emittance is reduced and the beam envelope is centered on the x axis. On the graphs on the right of the figure, we can see how the final configuration is able to correct the aberrations of the beam and thus the beam emittance is reduced to a minimum.

The algorithm is able to find the minimum in about 220 iterations, which can handily be performed on simulation, but would be unfeasible on the real machine. Furthermore, some experiments showed that the algorithm is not always able to converge to the minimum when the number of parameters is increased, or the computation time grows exponentially. For these reasons, this kind of methods can only be applied to small portions of the beam line, with a limited number of parameters, and are not directly usable online. Since no *learning* happens in the optimization, the algorithm is not able to transfer the knowledge to a different beam line or to a different beam. Thus, the algorithm should be re-run from scratch every time the beam line or the beam changes, which is computationally expensive and time-consuming.

6.3 Reinforcement Learning approach

In this Section we present our approach for the development of a model able to find the optimal configuration of the multipole high voltage values. The general idea is illustrated in figure 6.3. Given a configuration of the multipole values we can simulate the beam

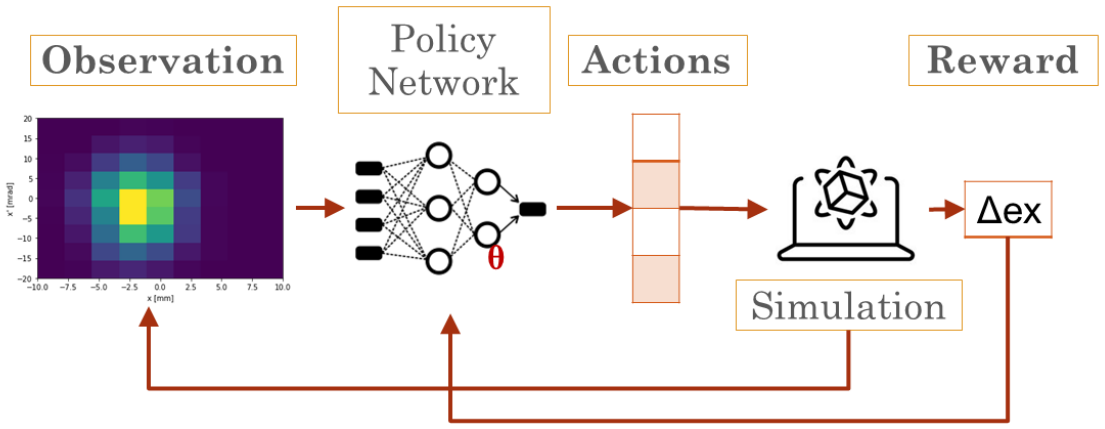


Figure 6.3: Reinforcement learning training and evaluation.

with the physics simulator and compute the beam emittance. Then, we can train a model to perform actions, that is to change the multipole values, in order to reduce the beam emittance. If the resulting emittance is lower than the previous one, then the model is rewarded, otherwise it is penalized. The model is trained to maximize the reward, thus it should learn to converge to the minimum beam emittance configuration. Since we don't want to learn a single solution, we build a RL model that learns to *move* the parameters towards the optimal solution. Finally, once the model is trained, it can be used online to change the multipole values on the real machine, hopefully converging to the optimal configuration in few steps.

6.3.1 Python wrapper

To be able to execute the RL loop illustrated in figure 6.3, it's necessary to integrate the physics simulator into the RL algorithms, which are run from python. To do so, a python wrapper was developed to run the simulator from python code and retrieve the results.

The wrapper is implemented as a standalone python module, called `pyTraceWin`. It provides a class called `TraceWin` with a few useful methods to interact with the simulator.

The main method of this class is `run()` which launches the simulator with a specific set of parameters and waits for the simulation to complete. It exploits the possibility to run TraceWin from the command line, without the graphical user interface, and uses the `subprocess` python module to run the simulator as a subprocess. When run from the command line, TraceWin accepts a few arguments to specify the project file to load, the output folder and the parameters to change. The project file includes the description of the beam line presented in source code A.4, while the output folder is used to store the results of the simulation. The remaining arguments can be used to overwrite the values in the project file, like the configuration of the multipole. The wrapper then builds the command line arguments and runs the simulator. Unfortunately, even though the GUI is not executed, the simulator still requires a display to run, which is often not available when running on a remote server. To overcome this limitation, the `pyvirtualdisplay` python module is used to create a virtual display that the simulator can use. This allows to run the simulator anywhere, but introduces some computing overhead due to the creation of a virtual display every time a simulation is run.

Finally, other methods allow retrieving and parsing the results from the output files. A python module from ESS (`ess—python—tools`) is used to parse the particle distribution files. To minimize the I/O operations on the disk, which is often slow, the output path is set to a ramdisk, which is a portion of the memory used as a file system. This allows to speed up the simulation and the parsing of the results.

6.3.2 REINFORCE

The first RL algorithm tested was REINFORCE, a policy gradient algorithm introduced in section 3.4.2. It was chosen for its simplicity and because it is a good starting point to understand the basic concepts of RL. Furthermore, it can be easily implemented in python from scratch.

To verify the feasibility of the approach and validate the code, we initially consider a reduced version of the problem, where we make the following assumptions:

- The total number of parameters is reduced from 48 to 4, corresponding to the 4 basic configurations of the multipole as expressed by equation (6.1).
- We further reduce the parameters to just 1 by setting the value of the other 3 to their optimal value. This is equivalent to consider only the sextupole component of the multipole.
- As observation we use the x, x' beam emittance graph image, which is a 2D histogram of the particle distribution in phase space on the diagnostic box at the end of the MRMS. To further reduce the complexity of the problem, we reduce the resolution of the image to 9x9 pixels.
- Our agent can only select two actions, that is to increase or decrease the value of the parameter. The amount of the increment is fixed to 50, which is a reasonable value for the operating range of the sextupole.

- We limit the number of steps in an episode to 10, which is enough to reach the minimum beam emittance. Furthermore, we terminate an episode successfully if the beam emittance is below a threshold of 0.15 mm mrad, or we abort the episode if the emittance is above 0.7 mm mrad, as it is considered a minimization failure. These little tricks are used to speed up the training process.

Now the optimization problem becomes a 1D problem, where we want to find the optimal value of A_3 to minimize the beam emittance. This is a much simpler problem, which can be solved with a simple minimization algorithm, but it allows us to test the RL framework and verify that it works as expected.

Another important design choice is the reward function. As we highlighted in section 3.4, the reward function is a fundamental part of the RL algorithm, since it determines the behavior of the agent. In our case, we want to minimize the beam emittance, thus we can use the following formula as reward:

$$r^n = \varepsilon_x^{n-1} - \varepsilon_x^n \tag{6.2}$$

The reward is positive when the beam emittance decreases, negative otherwise, thus by maximizing the reward the agent should converge towards the minimum beam emittance.

The next step requires to build the environment, that is the interface between the simulator and the RL algorithm.

Environment

We define a class which implements the standard RL methods by using the TraceWin simulator to perform an observation of the environment, perform an action and receive a reward. This class is called `TraceWinEnv` and its main methods are shown in source code A.5.

This class implements the `reset()` method, which is called at the beginning of each episode to reset the environment to its initial state. In our case, this means to reset the multipole parameters to a random value and run the simulator to obtain the initial observation. The `step()` method is called at each iteration of the episode and performs the following operations:

- It calls the `_perform_actions()` method to compute the new multipole parameters given the actions of the agent. This method simply increments or decrements the parameter values by 50, depending on the action.

- It runs the simulator with the new parameters and retrieves the results.

- It calls the `_observation()` method to compute the new observation, reward and done flag following the logic presented in the previous paragraph.

Training

Given the environment defined in the previous paragraph, we need to define a policy. This is a ML model which will receive the observation from the environment and will output the action to perform.

In our case, we implement the policy in pytorch as a simple CNN with a conv2d layer with 3 3x3 kernels and a single hidden fully connected layer of 10 neurons. The input of the network is the observation, that is the 9x9 image of the beam emittance, while the output layer is a vector of 2 values with the softmax activation function, corresponding to the probability of performing the two actions. Figure 6.4 shows a schema of the network.

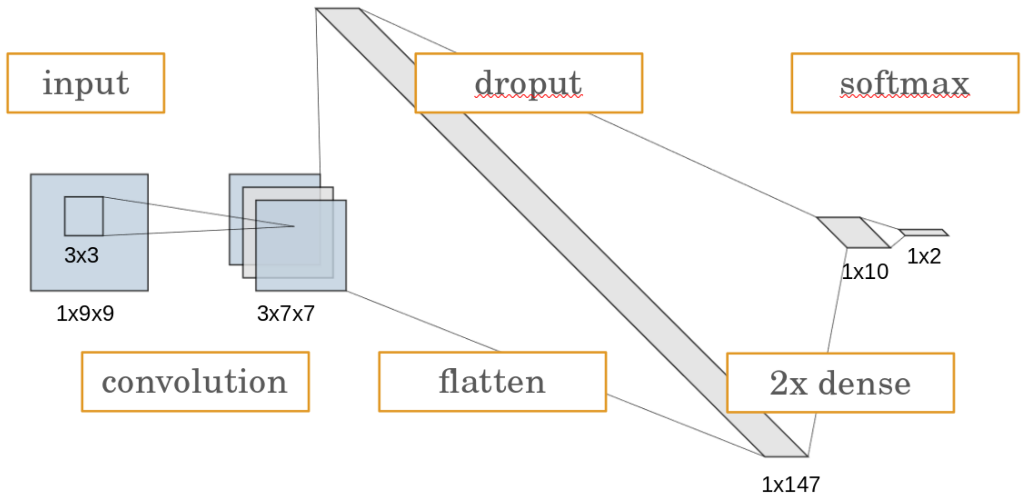
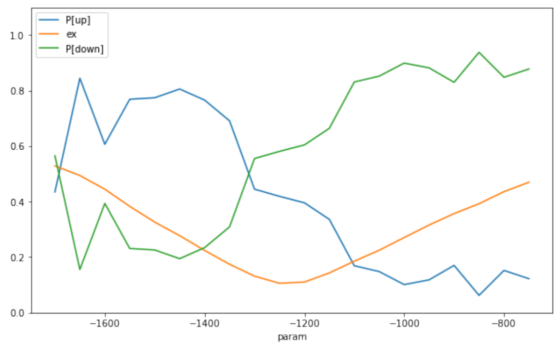


Figure 6.4: Policy network for the REINFORCE algorithm.

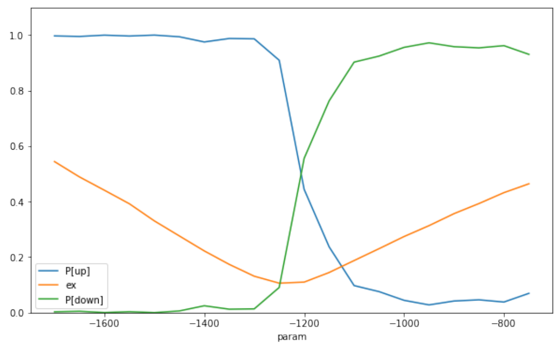
Then, the main training loop is implemented as shown in source code A.6. Initially the environment is reset and the first observation is retrieved. Then, the policy network is used to retrieve the probability of performing each action. The `Categorical` class from pytorch is used to sample an action from the probability distribution. The action is then performed on the environment and the new observation, reward and done flag are retrieved. This process is repeated until the episode is concluded. Then, the the policy network is updated by following the gradient descent formula 3.28.

Evaluation

The minimization problem as formulated in the previous paragraphs is quite simple to solve, as the only free parameter has a clear minimum around -1200. While using RL for such a problem is overkill, it allows us to test the code and train the model in a reasonable amount of time. In fact, even thought the NN training is offloaded to the GPU, the TraceWin simulator can only run on the CPU and, having to simulate a beam composed of 10000 particles, it takes about 15 seconds to complete a single simulation, even on modern hardware. This means that, even with a simple problem like this, the training can take many hours. Unfortunately this is a big limiting factor and must be carefully considered when designing the RL model.



(a) After 3 hours of training, $\sim$ 300 episodes and $\sim$ 1700 steps.



(b) After 20 hours of training, $\sim$ 3100 episodes and $\sim$ 11000 steps.

Figure 6.5: Policy learned by the REINFORCE algorithm.

An advantage of the simplified setup is that the policy network outputs two simple probabilities, which can easily be visualized. In figure 6.5a we can see the learned policy after 3 hours of training, while in figure 6.5a we can see the policy after 3 hours of training. The orange line which shows the beam emittance when varying the multipole parameter, with a clear minimum around -1200. As can be seen, the policy learns to perform the correct action, that is to decrease the parameter value when it is above -1200, and to increase it when below -1200. Even so, the probabilities of performing the correct actions are not close to 1 and are quite noisy, even when the beam emittance curve is very smooth. In fact, the policy does not know the current beam emittance value and has to rely only on the observation.

In figure 6.5b instead we can see the results after 20 hours of training, corresponding to about 3100 episodes and 11000 simulations. The policy is now much more stable and the probabilities are closer to 1. This is due to the fact that the policy network has learned to recognize better the beam emittance graph and thus it can perform the correct action with higher probability.

Overall these results show that the RL approach is applicable to the beam emittance minimization task and that it's possible to learn a policy to perform the correct actions. Nevertheless, the model has to scale to more parameters, to a higher resolution of the observation and to perform steps of variable length. When these factors are taken into account, the training becomes much more difficult and the REINFORCE algorithm is not able to converge to a solution. One solution to improve performance is to replace the return in the policy gradient formula 3.28 with and advantage function. This is a common technique in RL and the simplest way to implement it is to subtract the average return. This means that each action is rewarded positively only if it performs better than the average. With this addition the learning process becomes more powerful and can be extended to more parameters. Nevertheless, the training is still very slow and more advanced techniques can be used to improve the performance. For this reason, we decided to use a more efficient algorithm to scale to multiple parameters.

6.3.3 Proximal Policy Optimization

The Proximal Policy Optimization (PPO) algorithm was chosen for its flexibility and popularity in the RL community. In fact, it requires very little tuning and it is able to scale to complex problems. Furthermore, it is implemented in the Stable-Baseline3 [132] library, which implements many RL algorithms in PyTorch with a unified interface.

Gymnasium Environment

The algorithms from Stable-Baseline3 require an environment which adheres to the `gymnasium` interface, a standard API for creating RL environments. This is similar to the environment in source code A.5, having the following methods:

- `reset()`: reset the environment to its initial state and return the initial observation.
- `step(action)`: perform the given action on the environment and return the new observation, the reward, the done flag and some additional information.
- `render()`: render the environment, usually by plotting the observation.
- `close()`: close the environment and free the resources.

When implementing the environment, we decided to restrict the problem the 4 parameter A_3, A_4, A_5, A_6 following equation (6.1) as this simplification injects some physics knowledge in the model and guarantees that the solution is valid. Whenever it's possible to constrain an optimization problem with knowledge of the system it makes sense to do so, as it reduces the search space and thus the complexity of the problem. Then the policy network has to decide how to update each parameter. In the previous approach the policy network had to decide whether to increase or decrease the parameter, but this is not ideal when each parameter is close to its optimal value, as it may overshoot. We decided to use a continuous action space, where the network can decide both the direction of parameter update (increase or decrease) and also its amplitude. For this reason, the action space is set to a `gym.spaces.Box` space, that is a vector consisting of 4 real values between -1 and 1. This value is then multiplied by a vector of coefficients ($[200, 100, 50, 50]$) to obtain the values to adjust the multipole setpoints. These coefficients were chosen by dividing the expected operating range of each parameter by the maximum number of iterations per episode.

Furthermore, we decided to use a higher resolution for the observation, that is a 36x36 image of the beam emittance graph. This is the same observation used in the previous section, but at a higher resolution so that the agent can better recognize small variations in the graph. In fact, when correcting the high order parameters (A_5 and A_6) the effect on the beam emittance graph is a distortion on the tails of the ellipse, which are not visible at low resolution. On the other hand increasing the resolution means that the policy network will have many more parameters and thus it will be more complex and slower to train. Since the observation is a 2D image, the policy network was configured to use a CNN with an architecture which satisfies the constraints given by the input and output specifications. For simplicity the default CNN policy from the library was used. This is composed of 3 convolutional layers, followed by a fully connected layer with 512 neurons and a final linear layer with 4 outputs.

The reward is calculated with the following formula:

$$r^n = \left(\varepsilon_x^{n-1} - \varepsilon_x^n + \frac{I_l^{n-1} - I_l^n}{a} \right) * b \tag{6.3}$$

where ε_x^n is the emittance value at iteration n, I_l is the number of lost particles on the beam transport through the MRMS, a is a calibration factor which was set to $a = 10000$ (number of simulated particles) and $b = 10$ is a gain used to amplify the reward value. Compared to equation (6.2), this last term was introduces to avoid a solution where the beam emittance reaches its minimum due to heavy transmission losses. As before, an episode is concluded when the emittance reaches a low threshold or when the number of iterations in the episode reach a limit of 10.

Now, with this setup it's possible to train the model using the PPO algorithm and evaluate its ability to converge to the minimum beam emittance.

Results

The physics simulator is configured with a Sn19+ beam at 0.76MeV. To correctly simulate the beam distribution the simulator calculates the complete dynamics of 10000 particles along the ADIGE beam line. As we noted before, even on a high performance server, a single simulation takes about 15 seconds to complete. GPU acceleration is used for the neural network training, but the simulator does not support it and the simulation step is responsible for most of the computing time. This is obviously a big limiting factor on the amount of simulations that can be performed and thus on the complexity of our model.

Given this setup, the agent is trained for a few days and about 25k episodes. As can be seen in Figure 6.6 the mean reward reaches a fairly constant positive value in relatively few episodes. Instead, the average episode length requires more episode to stabilize to a small value. This means that the training is actually successful and the model is able to reach the minimum beam emittance in few steps. Around the $9000th$ episode the reward decreases drastically and the mean episode length increases correspondingly. While this behaviour was not expected, it's probability due to the training escaping from a local minima: the model explored different areas of the state action space, which resulted in a lower reward for some episodes. Nevertheless, the training was able to recover by learning the optimal behavior in all the situations, resulting in a more robust policy overall.

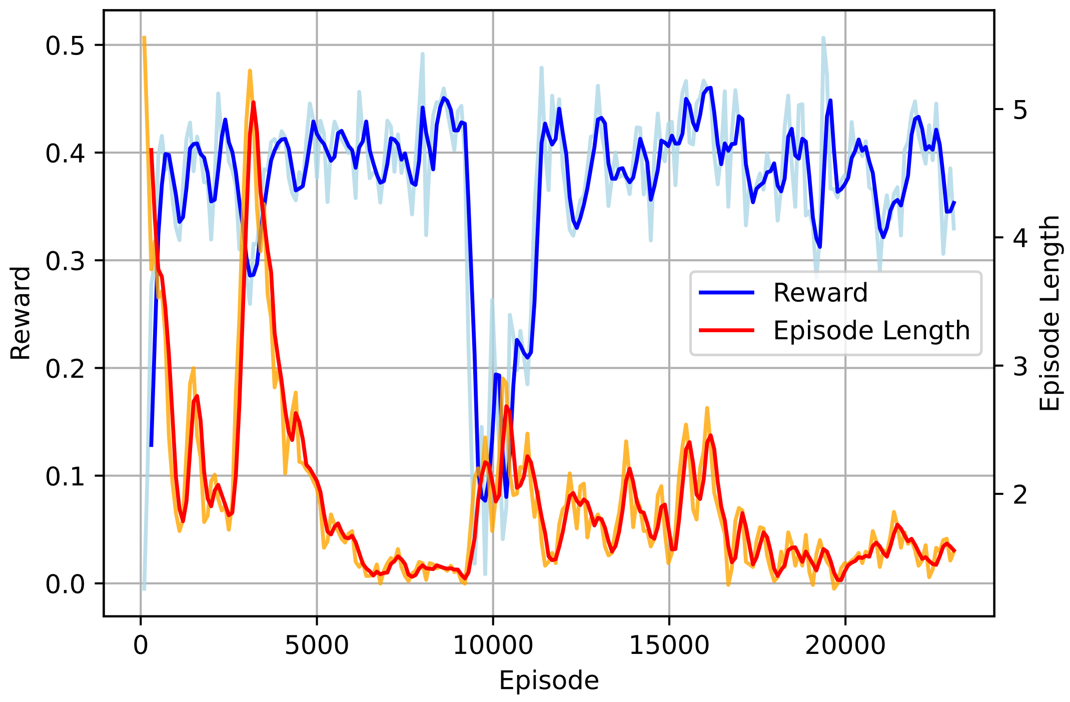


Figure 6.6: Mean reward and episode lenghts during training.

Finally, we evaluated the model by running the trained agent on new simulations, each starting from a random configuration of the multipole. We observed that the model is able to converge to a minimum 97% of the times with an average of just 2.2 steps. This is a great result because, if it translates to the real machine, it means that the model is able to converge to the optimal configuration in just a few minutes. This is a huge improvement over the traditional optimization methods, which require hundreds of iterations to converge to the minimum.

6.4 Conclusions

In this chapter, we presented a method based on reinforcement learning to train an agent which is able to iteratively optimize the parameters of an electrostatic multipole on the ADIGE beam line to minimize the beam emittance. This demonstrates the feasibility of using reinforcement learning models to automatically explore in a smart way the parameter space of a complex beam line and converge towards the optimal beam dynamics solution.

A first approach based on the REINFORCE algorithm was presented, but it was not able to scale to more parameters and to a higher resolution of the observation. Then, a more advanced algorithm based on Proximal Policy Optimization was used to train a model which is able to converge to the minimum beam emittance in just a few steps. This is a great result, as it means that the model can be used online to optimize the beam line in a few minutes, instead of the hours required by traditional optimization methods.

Nevertheless, the proposed method requires further research to scale it to more parameters and evaluate its performance with a greater environment variability. In particular, it would be interesting to test how much a trained model is able to adapt to a different input beam or slightly different beam line, thus generalizing to different use cases. The current results are obtained in simulation, but the actual performance on the real machine have yet to be tested. Finally, data from the real machine could be used to further fine tune its behaviour and actually learn the peculiarities of a specific beam line. The policy network architecture could be fine-tuned to the specific problem, for example by decreasing its complexity to ease the training process. Furthermore, other algorithms could be tested to improve the performance and the training time, especially Q-learning based algorithms which are known to be more sample efficient.

From the experience gained on this work we can conclude that reinforcement learning is a promising approach to the optimization of beam lines, but currently we are limited by the computational complexity of the physics simulator. In fact, the simulator is the bottleneck of the whole process, as it requires a lot of computing time to simulate even a small beam line. Exploring the use of different physics simulators, especially GPU accelerated ones, may be required to scale further the approach.

Chapter 7

Dynamic Sampling

We report here a work carried out during the last period of the Ph.D., which concerns the application of data-driven techniques to the optimization of metrology plans. While this project is not directly related to the main topic of the thesis, it is still relevant to the research activity carried out during the Ph.D. as it is a good example of the application of data-driven and ML algorithms to optimize industrial processes. Such techniques are studied primarily for quality assessment in the manufacturing industry, but they can be applied to a wide range of different fields. In the field of particle accelerators they can be used to optimize the measurement plan when verifying the alignment of the beam line, which means reducing the number of measures and speeding up the process. More broadly, these techniques can be applied for the *design of experiment* task, where the goal is to minimize the number of experiments or measures required to obtain a certain level of accuracy. This is a very important task in the field of particle accelerators, where certain measures are very expensive and time-consuming.

7.1 Introduction

Metrology operations are fundamental for quality assessment of production processes [35]. However, these operations are often expensive both in terms of time and money hence, it is pivotal to optimize metrology plans in order to guarantee a good trade-off between production costs and final product quality. In many cases a set of measures is required to assess a certain property of the measured object over its physical extension, or between different objects produced by the same process. For example, if one wants to verify that the thickness of the sheets of metal produced by a plant is within the accepted range, measurements of the thickness need to be taken at many points across each sheet [13]. When a dataset of past measurements is available, it is possible to exploit data-driven techniques to build predictive models that exploit the spacial correlations between measurements, allowing a subset of the available locations to be measured with the values at the remaining sites estimated using the prediction models [124].

This optimization can have a great impact on a wide range of different industries.

As in the previous example, in many mechanical manufacturing applications different quantities and properties need to be measured over a spatial domain to guarantee the quality of the final product [119]. When working with particle accelerators a measurement plan is used to verify the alignment of a beam line over hundreds or thousands of meters with a precision of $\sim 100\mu m$, which is critical for the correct transport of a beam to the physics experiment [88]. In this case a temporal correlation is expected between different alignment campaigns.

Another paradigmatic example of the need for good metrology plans is in the semiconductor manufacturing industry. Here electronic circuits are built up on wafers using a complex sequence of chemical and photo-lithographic processes - including Chemical Vapor Deposition (CVD) [150], plasma etching [137], ion implantation [172], and planarization [151]. For each of these processes, key parameters such as the height of a deposited layer, depth of an etched trench, etc., need to be measured at multiple locations (called sites) distributed over the wafer surface to ensure they are within specified tolerances [69], otherwise the wafer is considered defective. In all these cases optimizing the measurement plan is of critical importance, as the measurement equipment is typically expensive and can only measure sites sequentially resulting in a very time-consuming operation.

For these reasons we design a measurement plan that allows the number of measured sites on each *unit* to be reduced. We use here the term *unit* to denote the item being measured, e.g. a silicon wafer or a metal sheet in our examples. We reduce the number of measured sites from v (the total number of available sites) to k (with $k < v$). The value of the measure on the remaining $v - k$ unmeasured sites will be predicted using a regression model exploiting the expected spatial and temporal correlations with the measured sites. Thus the k parameter allows the measurement process to be adjusted to be more precise (with higher k values) or faster and cheaper (using smaller values of k). Given a certain k, set by the constraints of the manufacturing process, we can compare the measurement plans in terms of prediction error.

In [129] the authors presented a method for optimal site selection in wafer metrology based on an algorithm called Forward Selection Component Analysis (FSCA). Given a unit, it iteratively selects the site which, taken with the previously selected sites, minimizes the reconstruction error. FSCA is a "static sampling" algorithm, which means that it only identifies a single set of sites to be used for measurement of all future units. This approach reduces the number of selected sites, but there is the risk of missing anomalies or important process information by having sites that are never measured on any unit.

Dynamic strategies are thus preferable, where the set of measured sites changes at every unit so that all the sites are eventually measured at least once, increasing the unit coverage temporally, and thus creating a more robust and reliable measurement plan. In [114, 158] the authors proposed spatial dynamic sampling techniques that are able to guarantee that all candidate measurement sites are visited periodically, however, these methods do not allow the user to specify a priori the allowable temporal sampling interval (i.e the maximum number of units that can be processed before all sites are visited at least once), or to directly trade-off spatial visibility for temporal visibility.

In this chapter we consider the temporally constrained dynamic sampling problem. This formulation is especially useful when a production process runs with a certain periodicity and we want to guarantee that each site is visited at least once in that period or to satisfy regulatory constraints. To guarantee that the temporal constraint is satisfied it is necessary to force the measurement of a number of previously unmeasured sites for each unit over the measurement plan. To do this efficiently, while retaining the best possible reconstruction accuracy, it is necessary to determine how best to distribute the forced sites over the measurement plan and how to select the forced sites for each unit. In the following section we explore a number of different approaches to addressing these problems and, with the aid of extensive experimental evaluation, establish that an even distribution of forced sites over the temporal window is optimal, and that a generalized FSCA algorithm is the best methodology for selecting the sites on each unit. Thus, the main contributions of this work are:

- A new formulation of the spatial dynamic sampling problem that allows the a priori specification of the maximum number of iterations required to visit all the available sites in the measurement plan.

- A novel dynamic sampling algorithm that optimizes spatial visibility (i.e. minimizes the measurement reconstruction error per unit) while satisfying a temporal visibility constraint.

- A generalized version of the FSCA algorithm that extends the algorithm presented in the literature [131] by adding the capacity to preselect sites, and to specify the sets of candidate sites from which sites can be selected.

7.2 Methodologies

In this section we provide an overview of existing methodologies used for spatial metrology optimization. Typical optimization pipelines are composed of three main phases: the first (optional) is to use Principal Component Analysis (PCA) to determine how much redundancy there is among the candidate measurement sites - we can obtain a lower bound on the number of sites that must be measured to achieve reliable results. If this redundancy is high enough, we can proceed to the second phase, that is, the selection of a subset of sites with one of the algorithms described in the following sections. The third phase is the reconstruction of the remaining (redundant) sites using virtual metrology models. Here, linear regression models are adopted, but more complex models such as those found in [177, 71, 23, 78], can also be employed for this task.

7.2.1 Preliminaries

Let $\mathcal{P} = \{\mathcal{P}_1, \mathcal{P}_2, ..., \mathcal{P}_M\}$ be an M-unit measurement plan with $\mathcal{P}_i$ denoting the set of k measured sites from the v available sites (with index set $\mathcal{I}_V$) for the i-th unit. In general, the t-th unit in a production sequence will be measured using the sites specified by $\mathcal{P}_{(1+t \bmod M)}$. Static sampling corresponds to the case where the same set of sites is measured on each wafer, i.e, $M = 1$.

For static sampling the optimum site selection problem can be formulated as

$$\mathcal{I}_S^* = \underset{\mathcal{I}_S \subset \mathcal{I}_V}{\arg\min} ||\mathbf{X} - \hat{\mathbf{X}}(\mathcal{I}_S)||_F^2, \quad \text{s.t.} \ \ |\mathcal{I}_S| = k, \tag{7.1}$$

where $\mathbf{X}$ is an $m \times v$ matrix of historical measurements and $\hat{\mathbf{X}}(\mathcal{I}_S)$ is the reconstructed version of $\mathbf{X}$ using only measurements taken at the locations specified in $\mathcal{I}_S$. Here we employ linear regression based reconstruction, that is, $\hat{\mathbf{X}}(\mathcal{I}_S) = \mathbf{X}_{\mathcal{I}_S}\mathbf{\Theta}$, where

$$\mathbf{\Theta} = (\mathbf{X}_{\mathcal{I}_S}^\top \mathbf{X}_{\mathcal{I}_S})^{-1}\mathbf{X}_{\mathcal{I}_S}^\top \mathbf{X} \tag{7.2}$$

and $\mathbf{X}_{\mathcal{I}_S}$ is the subset of columns of $\mathbf{X}$ indexed by $\mathcal{I}_S$. FSCA provides a greedy search approximate solution to this NP hard subset selection problem. We hereafter consider this to be the optimal solution.

The resulting reconstruction accuracy (process visibility) may be quantified in terms of the normalized mean squared reconstruction error, E_R. This is defined over the training set as

$$E_R(\mathbf{X}, \mathcal{I}_S) = \frac{1}{\sigma_{\mathbf{X}}^2}\frac{1}{mv}||\mathbf{X} - \hat{\mathbf{X}}(\mathcal{I}_S)||_F^2 \times 100\% \tag{7.3}$$

and over the m' unit test set $\mathbf{Z}$ as

$$E_R(\mathbf{Z}, \mathcal{I}_S) = \frac{1}{\sigma_{\mathbf{X}}^2}\frac{1}{m'v}||\mathbf{Z} - \hat{\mathbf{Z}}(\mathcal{I}_S)||_F^2 \times 100\%, \tag{7.4}$$

where, $\hat{\mathbf{Z}}(\mathcal{I}_S) = \mathbf{Z}_{\mathcal{I}_S}\mathbf{\Theta}$, with $\mathbf{\Theta}$ as defined in (7.2). The normalization factor $\sigma_{\mathbf{X}}^2$ is the variance in the data observed over all measurement points for the training dataset, that is:

$$\sigma_{\mathbf{X}}^2 = \frac{1}{mv}||\mathbf{X} - \overline{\mathbf{X}}||_F^2, \quad \text{where} \ \ \overline{x}_{ij} = \frac{1}{mv}\sum_{ij} x_{ij}, \tag{7.5}$$

and x_{ij} and $\overline{x}_{ij}$ denote the elements of $\mathbf{X}$ and $\overline{\mathbf{X}}$, respectively.

The choice of k is a trade-off between metrology cycle-time and spatial visibility (as reflected in E_R) and is typically chosen to achieve a specified reconstruction accuracy.

The site sampling interval τ_i is the interval between successive sampling of site i. The Maximum Site Sampling Interval (MSSI) [114] is then defined as:

$$T = \max_{i \in \mathcal{I}_V} \tau_i. \tag{7.6}$$

For static sampling plans the MSSI in infinite, which leads to a risk of the appearance of previously unseen process behaviour (at unmeasured sites) going undetected. Dynamic sampling strategies seek to address this deficiency with static sampling by changing the sites measured for each unit in a production sequence, such that all locations are visited periodically. In temporally bounded dynamic sampling an upper limit is placed on the acceptable MSSI, that is, we require that $T \leq T_{max}$ for a given measurement plan $\mathcal{P}$. If we set $M = T_{max}$ then this corresponds to requiring $\bigcup_{i=1}^{M} \mathcal{P}_i = \mathcal{I}_V$. Thus, the temporally bounded dynamic sampling problem can be formulated as:

$$\mathcal{P}^* = \arg\min_{\mathcal{P}} \mathbb{E}\left[\frac{1}{M}\sum_{i=1}^{M} E_R(\mathbf{X}, \mathcal{P}_i)\right] \tag{7.7}$$

subject to $|\mathcal{P}_i| = k \ \forall i$, and $\bigcup_{i=1}^{M} \mathcal{P}_i = \mathcal{I}_V$. Note that for (7.7) to be solvable k and M must be selected such that $kM \geq |\mathcal{I}_V|$.

7.2.2 Forward Selection Component Analysis (FSCA)

The state-of-the-art algorithm for metrology optimization is called Forward Selection Component Analysis (FSCA) [129], [131]. It is a simple greedy algorithm which provides an ordering of the sites from the one with maximum contribution to the variation observed in the dataset to the one with minimum contribution. Specifically, this algorithm iteratively repeats two steps: a minimization step and a deflation step. In the first step the algorithm selects the site i^* that minimizes the reconstruction error over the dataset. The unmeasured sites values are estimated as

$$\hat{\mathbf{X}}(\tilde{\mathbf{x}}_{i^*}) = \frac{\tilde{\mathbf{x}}_{i^*}\tilde{\mathbf{x}}_{i^*}^\top}{\tilde{\mathbf{x}}_{i^*}^\top\tilde{\mathbf{x}}_{i^*}}\tilde{\mathbf{X}},$$

where $\hat{\mathbf{X}}(\tilde{\mathbf{x}}_{i^*})$ estimates $\tilde{\mathbf{X}}$ by regressing on $\tilde{\mathbf{x}}_{i^*}$. The deflation step (line 7 in algorithm 4) is the projection of the matrix $\tilde{\mathbf{X}}$ on the space orthogonal to $\mathbf{x}_{i^*}$.

In algorithm 4 we introduce the pseudocode for a generalized version of FSCA, which extends the original algorithm to allow for a finer control over the selection of sites. In particular, the user can specify a set of preselected sites $\mathcal{I}_0$, which will be included in the final set of selected sites, leaving the algorithm to select only the remaining $k - |\mathcal{I}_0|$ sites from the set of candidate sites $\mathcal{I}_C$. This functionality is useful for the dynamic sampling algorithms presented later in this chapter. The original FSCA algorithm, as presented in [129], corresponds to the case where $\mathcal{I}_0 = \emptyset$ (no preselected sites) and $\mathcal{I}_C = \mathcal{I}_V$ (the set of all available sites). Recently a *lazy* implementation of FSCA has been developed that has similar performance to FSCA, but can be an order of magnitude faster to compute [181].

Algorithm 4: Generalized FSCA (GenFSCA)

```
Input: $\mathbf{X}, k, \mathcal{I}_0, \mathcal{I}_C$
1: $\mathcal{I}_S \leftarrow \mathcal{I}_0$
2: $q = |\mathcal{I}_0|$
3: $\tilde{\mathbf{X}} \leftarrow \mathbf{X} - \hat{\mathbf{X}}(\mathbf{X}_{\mathcal{I}_S})$
4: for $i \leftarrow 1$ to $k - q$ do
5:     $i^* \leftarrow \arg\min_{i \in \mathcal{I}_C} ||\tilde{\mathbf{X}} - \hat{\mathbf{X}}(\tilde{\mathbf{x}}_i)||_F^2$
6:     $\mathcal{I}_S \leftarrow \mathcal{I}_S \cup i^*$
7:     $\tilde{\mathbf{X}} = \tilde{\mathbf{X}} - \hat{\mathbf{X}}(\tilde{\mathbf{x}}_{i^*})$
8: end for
9: return $\mathcal{I}_S$
```

7.2.3 Sequential Dynamic Sampling (SDS)

The first dynamic sampling algorithm presented is called Sequential Dynamic Sampling (SDS). It was developed in [114] and is heavily reliant on the effectiveness of FSCA. The pseudocode for this algorithm is shown in algorithm 5. SDS proceeds in the following way: it computes the k best static measurement sites with FSCA, and then uses these locations as the centers of k clusters $\mathcal{C} = \{\mathcal{C}_1, \mathcal{C}_2, ..., \mathcal{C}_k\}$ to cluster the remaining sites. Each site is assigned to the cluster corresponding to the nearest FSCA selected site. The measurement plan is then built by taking sequentially one site from each the k clusters. Whenever a cluster is completed it restarts again from the first element in the cluster. This proceeds until all elements of the largest cluster have been used. Of course, this means that sites in small clusters will be measured multiple times. The choice of the distance function for clustering is not trivial. In fact it has been shown that a data-driven correlation function leads to overall better results (in terms of E_R) than the classic euclidean function [114]. In any case, the MSSI for this algorithm is equal to the cardinality of the largest cluster among the k clusters, that is

$$T_{SDS} = \max_{i=1,...,k} |\mathcal{C}_i|. \tag{7.8}$$

7.2.4 Induced Start Dynamic Sampling (ISDS)

The current state-of-the-art algorithm yielding the best spatial visibility performance (i.e., minimum E_R) for dynamic sampling is the Induced Start Dynamic Sampling (ISDS) algorithm introduced in [158]. The key idea in ISDS is to preselect at each iteration an initial subset of q sites at random from the set of currently unmeasured sites $\mathcal{I}_U$ and then to use FSCA to select the remaining $k - q$ sites. The pseudocode for ISDS is given in algorithm 6. Here, the $\mathcal{I}_0$ parameter of the generalized FSCA algorithm (algorithm 4), called at line 5, is used to force the inclusion of q previously unmeasured sites. Parameter q is a user-defined tuning parameter. A low value of q leads to low E_R but high MSSI,

Algorithm 5: SDS

```
Input: $\mathbf{X}, k$
1: $\mathcal{I}_{FSC} \leftarrow$ GenFSCA$(\mathbf{X}, k, \emptyset, \mathcal{I}_V)$
2: $\{\mathcal{C}_1, \mathcal{C}_2, ..., \mathcal{C}_k\} \leftarrow$ CLUSTER$(\mathbf{X}, \mathcal{I}_{FSC})$
3: for $t \leftarrow 1$ to $\max_{i=1,...,k} |\mathcal{C}_i|$ do
4:     for $i \leftarrow 1$ to $k$ do
5:         $\mathcal{P}_{t,i} \leftarrow \mathcal{C}_{i,(1+t \bmod |\mathcal{C}_i|)}$
6:     end for
7: end for
8: return $\mathcal{P}$
```

while a larger value increases the E_R but reduces the MSSI. Choosing $q = 1$ is the best choice for minimum E_R, but results in the largest finite MSSI, while setting $q = k$ yields the minimum MSSI and the largest E_R. This trade-off cannot be avoided for fixed k, but ISDS allows it to be adapted to user need. While the MSSI for ISDS cannot be determined *a priori* due to the potential for FSCA to select unmeasured sites at each iteration, it follows that it is upper bounded by $\lceil v/q \rceil$, that is:

$$T_{ISDS} \leq \left\lceil \frac{v}{q} \right\rceil . \tag{7.9}$$

Algorithm 6: ISDS

```
Input: $\mathbf{X}, k, q, \mathcal{I}_V$
1: $\mathcal{I}_U \leftarrow \mathcal{I}_V$
2: $t \leftarrow 1$
3: while $\mathcal{I}_U \neq \phi$ do
4:     $\mathcal{I}_{start} \leftarrow$ select a subset of $q$ random elements from $\mathcal{I}_U$
5:     $\mathcal{P}_t \leftarrow$ GenFSCA$(\mathbf{X}, k, \mathcal{I}_{start}, \mathcal{I}_V)$
6:     $\mathcal{I}_U \leftarrow \mathcal{I}_U \setminus \mathcal{P}_t$
7:     $t \leftarrow t + 1$
8: end while
9: return $\mathcal{P}$
```

7.3 Proposed Methods

While SDS and ISDS provide some level of control over the MSSI, they do not optimize E_R over a specified temporal horizon and therefore do not address the temporally bounded dynamic sampling problem, as encapsulated in equation (7.7). In this section we present a number of novel temporally bounded algorithms that seek to leverage the extra flexibility offered by having a MSSI budget to maximize E_R performance. The underlying assumption is that the MSSI bound T_{max} is sufficiently large to provide flexibility in site selection, or more precisely, that $T_{max} \cdot k \gg v$.

7.3.1 Constrained Induced Start Dynamic Sampling (CISDS)

The majority of algorithms developed are based on the most effective algorithm in the literature, ISDS. This can be constrained to make sure that no more than T_{max} iterations are required by selecting the value of the q parameter as follows:

$$q(t) = \begin{cases} 0, & \text{if } t = 1 \\ \min\left\{ \left\lceil \frac{v-k}{T_{max}-1} \right\rceil, |\mathcal{I}_U| \right\}, & \text{if } t > 1 \end{cases} \tag{7.10}$$

where t is the iteration counter, v is the total number of sites, k is the number of sites measured per unit, and $\mathcal{I}_U$ is the set of currently unmeasured sites. Here, q is set to zero in the first iteration ($t = 1$) since initially there are no previously measured sites and FSCA is guaranteed to only select unmeasured sites. Thus after the first iteration we are left with $v - k$ sites and $T_{max} - 1$ iterations to visit them. To guarantee that the remaining sites are visited at least $\left\lceil \frac{v-k}{T_{max}-1} \right\rceil$ previously unmeasured sites must be included at each iteration, hence q is set accordingly for $t > 1$. Since FSCA may also potentially select unmeasured sites at each iteration $\mathcal{I}_U$ may become empty before reaching T_{max} iterations. To account for this case we set $q = |\mathcal{I}_U|$ when $|\mathcal{I}_U| < \left\lceil \frac{v-k}{T_{max}-1} \right\rceil$. The resulting pseudocode, presented in Algorithm 7, is a minor variation on Algorithm 6.

Algorithm 7: CISDS

```
Input: $\mathbf{X}, k, \mathcal{I}_V, T_{max}$
1: $\mathcal{P}_1 \leftarrow$ GenFSCA($\mathbf{X}, k, \emptyset, \mathcal{I}_V$)
2: $\mathcal{I}_U \leftarrow \mathcal{I}_V \setminus \mathcal{P}_1$
3: $t \leftarrow 2$
4: while $\mathcal{I}_U \neq \phi$ and $t \leq T_{max}$ do
5:     $q \leftarrow \min(\lceil \frac{|\mathcal{I}_V|-k}{T_{max}-1} \rceil, |\mathcal{I}_U|)$
6:     $\mathcal{I}_{init} \leftarrow$ select a subset of $q$ random elements from $\mathcal{I}_U$
7:     $\mathcal{P}_t \leftarrow$ GenFSCA($\mathbf{X}, k, \mathcal{I}_{init}, \mathcal{I}_V$)
8:     $\mathcal{I}_U \leftarrow \mathcal{I}_U \setminus \mathcal{P}_t$
9:     $t \leftarrow t + 1$
10: end while
11: return $\mathcal{P}$
```

To assist with the understanding and reproducibility of this method and the following ones, we report some example simulations of sequences of the q parameter produced using the various methods here presented. In all the examples we assume that $k = 6$, $v = 50$ and $T_{max} = 22$. By applying the formula in equation (7.10) we get $q = 3$ for all iterations after the first one, as shown in Figure 7.1. The upper plot shows the evolution of q in the worst case scenario where there are no *"lucky choices"*, that is, where FSCA does not select any additional sites from $\mathcal{I}_U$ at each iteration. After 16 iterations all sites have been measured and q drops to 0. This is much less than the specified T_{max} due to the impact of rounding which is dependent on the choice of input parameters. For example, if instead T_{max} is chosen to be 23 then $q = 2$ for all 22 iterations following the initial full FSCA iteration.

The lower plot in Figure 7.1 shows an example where we FSCA selects an additional previously unmeasured site at iterations $t = [4, 8, 12]$. As a result the number of elements in $\mathcal{I}_U$ decreases faster than the minimum rate to satisfy the MSSI constraint with all sites measured after 15 iterations.

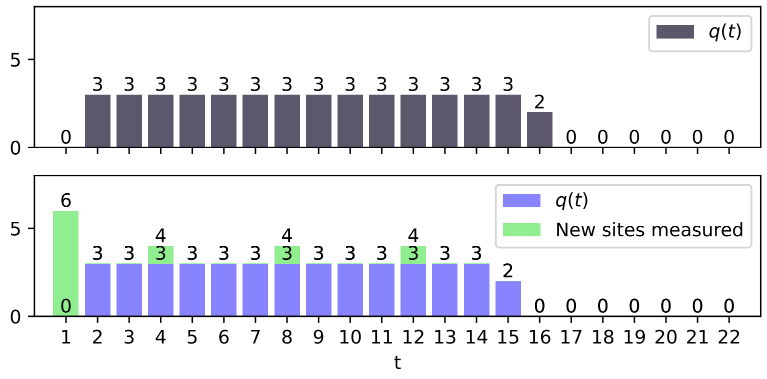


Figure 7.1: Evolution of $q(t)$ in CISDS with and without additional unmeasured sites selected by FSCA.

7.3.2 Greedy CISDS

While analyzing how CISDS is built, it is evident that with FSCA free to choose new sites both from $\mathcal{I}_S$ and $\mathcal{I}_U$, at each iteration the total number of samples drawn from $\mathcal{I}_U$ may be greater than q. When this happens, the constraint for the next iterations can be relaxed. The goal is to keep the total number of preselected 'forced' sites as low as possible to increase E_R performance. This can be achieved by recalculating q at each iteration as follows:

$$q(t) = \begin{cases} 0, & \text{if } t = 1 \\ \left\lceil \frac{|\mathcal{I}_U|}{T_{max} - t + 1} \right\rceil, & \text{if } t > 1 \end{cases} \tag{7.11}$$

While in equation (7.10) q depends only on the number of unmeasured sites after the first iteration, here the parameter depends on the actual number of unmeasured sites after each iteration. The CISDS pseudocode has already been reported in Algorithm 7. The only adaptation required for Greedy CISDS is the adjustment of q in line 5 in accordance with equation (7.11).

Figure 7.2 shows the sequence of q values with Greedy CISDS for the same simulation settings as used with CISDS in Figure 7.1. While initially $q = 3$, as in CISDS, it drops to two after the third iteration resulting in a more even distribution of forced sites across the measurement plan. Since there are unmeasured sites until the last iterations, FSCA has more opportunities to select sites from $\mathcal{I}_U$ (i.e., at $t = [16, 20]$) and thus the total number of forced sites is lower than in CISDS.

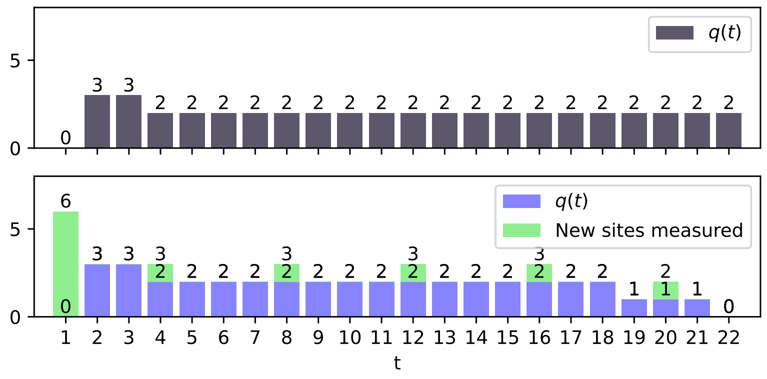


Figure 7.2: Evolution of $q(t)$ in Greedy CISDS with and without additional unmeasured sites selected by FSCA.

7.3.3 Step Up CISDS

The previous approach recalculates q to take into account additional sites drawn from $\mathcal{I}_U$ in earlier iterations, but is conservative with respect to future iterations. It acts as if the following iterations would draw exactly q sites from $\mathcal{I}_U$. It might be better to start optimistically, setting q to a lower value than what is needed, and hoping that sites chosen by FSCA will also be from $\mathcal{I}_U$. Then, after a number of iterations, we start increasing q to be sure to satisfy the MSSI constraint. The idea is to delay the "forced site selection" as long as possible, so that in the end the total number of forced sites will hopefully be significantly lower than with Greedy CISDS.

Specifically, we chose to compute q given k, the iteration t and the number of sites yet to be measured r as:

$$q = f(t, r) = \begin{cases} 0, & \text{if } t = 1 \\ 1, & \text{if } t < T_{max} - \lceil \frac{r}{k} \rceil + 1 \\ \text{k}, & \text{if } t \geq T_{max} - \lfloor \frac{r}{k} \rfloor + 1 \\ r \bmod k, & \text{otherwise} \end{cases} \tag{7.12}$$

Initially the function always returns 1, as there are enough iterations in the future to enable all sites to be visited, so that FSCA has the opportunity to select as many sites from $\mathcal{I}_U$ as possible. In the end as iterations are running out the function switches to selecting k sites per iteration to comply with the MSSI constraint. If r is not a multiple of k it may be necessary to select $q = r \bmod k$ sites during one iteration. The algorithm for this method follows directly from the previous ones.

The operation of Step up CISDS is illustrated in Figure 7.3. In the worst case scenario (upper plot) the value of q remains at 1 for most of the measurement plan and then steps up to 6 for the final four iterations. When FSCA selected unmeasured sites at $t = [4, 8, 12, 16]$ are assumed (lower plot) the step up period reduces to the last three iterations. The drawback of this method is that if insufficient additional unmeasured sites are selected by FSCA a larger number of the final iterations will be composed only of forced sites, resulting in suboptimal E_R performance.

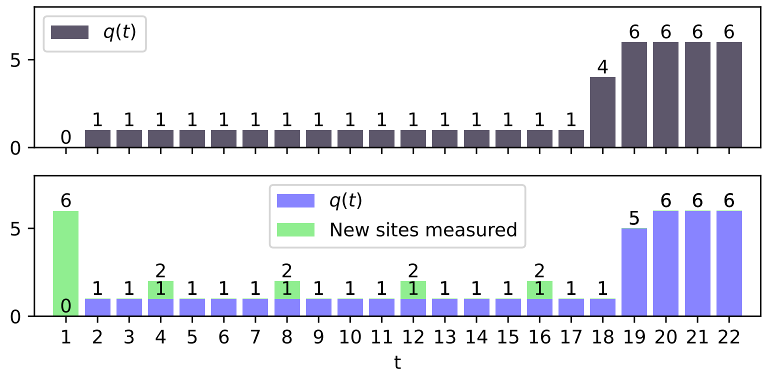


Figure 7.3: Evolution of $q(t)$ in Step up CISDS with and without additional unmeasured sites selected by FSCA.

7.3.4 Ramp Up CISDS

One could argue that in the Step Up CISDS method the transition from $q = 1$ to $q = k$ sites preselected from $\mathcal{I}_U$ per iteration occurs too quickly. In order to study the behaviour of the algorithm with a softer transition, q can be increased gradually by performing a "ramp". This can be achieved by developing a function similar to equation (7.12), where q increases from 1 to k in step increments, while guaranteeing the MSSI constraint is satisfied. Algorithm 8 can be used to generate such ramp behaviour. This takes as its input arguments the current iteration $t \in [1, T_{max}]$, the maximum number of measured sites per iteration k, the number of remaining sites to measure r, the remaining iterations $u = T_{max} - t + 1$ and the width of the ramp steps α. The notation "$RAMP_{-1}$" indicates the last element of the RAMP array. This algorithm simulates the current and all the future iterations as if no sites are selected from $\mathcal{I}_U$ by FSCA and forces the simulation to be a ramp. Then, a new simulation is run at each iteration to take into account the actual number of remaining sites. The algorithm creates an array of ones with its length equal to the number of remaining iterations. This array represents the expected q value at each iteration, so initially a single forced site is selected per iterations. After that, starting from the end of the array more values are added to each cell until all the remaining sites are included in the measurement plan. The last iterations will have up to k forced site measurements (line 13), while iterations are filled with values from a ramp (line 15).

Algorithm 8: Ramp Up Function

```
Input: $t, k, r, u, \alpha$
 1: $Q \leftarrow (1, 1, ..., 1) \in \mathbb{R}^u$
 2: $r \leftarrow r - u$
 3: if $r < 0$ or $t = 1$ then
 4:     return 0
 5: end if
 6: $RAMP \leftarrow RampGenerator(1, k-2, \alpha)$
 7: $i \leftarrow 1$; $j \leftarrow 1$
 8: while $r > 0$ do
 9:     $a \leftarrow 0$
10:     $\mu \leftarrow \max(0, |RAMP| - 1 - u + i)$
11:     if $r > \sum_{x=\mu}^{k} RAMP_x$ then
12:         $a \leftarrow k - 1$
13:     else
14:         $a \leftarrow \min(r, RAMP_{-j})$
15:         $j \leftarrow j + 1$
16:     end if
17:     $Q_{-i} \leftarrow Q_{-i} + a$; $r \leftarrow r - a$; $i \leftarrow i + 1$
18: end while
19: return $Q_0$
```

Algorithm 9: Ramp Generator

```
Input: $from, to, \alpha$
 1: $RAMP \leftarrow \phi$
 2: for $i \leftarrow from$ to $to$ do
 3:     for $j \leftarrow 1$ to $\alpha$ do
 4:         $RAMP$.append($i$)
 5:     end for
 6: end for
 7: return RAMP
```

The ramp is an array with values from 1 to $k-2$ included, which can be generated with Algorithm 9. For example for $k = 6$ and $\alpha = 1$ we have $[1, 2, 3, 4]$. Since this ramp will be added to the array of ones, the values in Q will range from 2 to $k - 1$. The first index (from the end of the array) to be filled with the ramp elements is selected by taking into account the sum of the values in the ramp. Since the ramp could be cut at the beginning of the array, μ is the first index of the ramp that fits inside Q and is used to sum only the values actually fitting into Q. Finally, when the simulation is complete, the first element of Q represents the q value to be used for the current iteration. This algorithm should be called on line 5 of Algorithm 7 to set the q value, similarly to the previous methods. The basic approach is to use a ramp with $\alpha = 1$, so that $RAMP_i = RAMP_{i-1} + 1$. This may be the optimum choice for some cases, but for others the ramp rate may be too high. Ideally we would like to distribute the forced site selection as evenly as possible across the measurement plan to reduce the number of iterations where $q = k$. The slope of the ramp can be reduced by increasing the α parameter, which indicates the number of iterations to wait before increasing the value of the ramp. For example, if $\alpha = 2$ the ramp will be $[1, 1, 2, 2, 3, 3, 4, 4]$. The optimal α parameter can be tuned by finding the value which minimizes the resulting reconstruction error, E_R. Note that while the Q array is simulated for all the remaining iterations, it is important to recalculate q with a new simulation of the full Q array every iteration since the number of remaining sites to select r depends both on the previous q and on the outcome of FSCA.

The results of applying the ramp algorithm to the aforementioned example problem are shown in Figure 7.4 for $\alpha \in \{1, 2, 3\}$. The α parameter is the *width* of the ramp steps, meaning that a bigger α value corresponds to a slower increase of the q value. Then, to guarantee that the total number of sites sums to $v - k$, some values of the ramp may be cut at the beginning. For example, in the third subplot, the value 3 is missing from the ramp, while in the third subplot, the value 3 occurs only once even though $\alpha = 3$. In these cases a lower value of q is enough to guarantee that all unmeasured sites were covered by the end of the iterations.

As expected, q increases more gradually than in the Step Up method, and the total number of forced sites is lower. In fact, the last few iterations are less likely to saturate to k forced sites, creating the possibility of unforced unmeasured site selection by FSCA. Furthermore, increasing α reduces the slope of the ramp further and the final iterations do not saturate. Additional FSCA site selection was not considered in these plots, but if this were to occur, it would further delay the onset and/or reduce the slope of the ramp, reducing the risk of saturation towards the end of the measurement plan.

To further aid in the understanding of the different methods for choosing q in Figure 7.5 we present a comparison between the cumulative value of $q(t)$ produced by each method over the temporal window, T_{max} for the sample problem introduced previously and assuming worst case conditions, that is, no additional unmeasured site selection by FSCA. Note that at the first iteration ($t = 1$) FSCA selects k unmeasured sites, so $q(1) = 0$, while at the end of the measurement plan all v sites must be visited, hence, $\sum q = v - k$.

The plot includes two extremes, *early limit* which represents the earliest possible selection of all sites, and *delay limit* which represents the latest possible selection of all sites. In the former, the first iterations are all saturated at $q = k$ (all sites are forced) until all sites are visited at least once, while in the later unmeasured sites selection is left to the very end, so that the last iterations all have $q = k$ and all preceding iterations

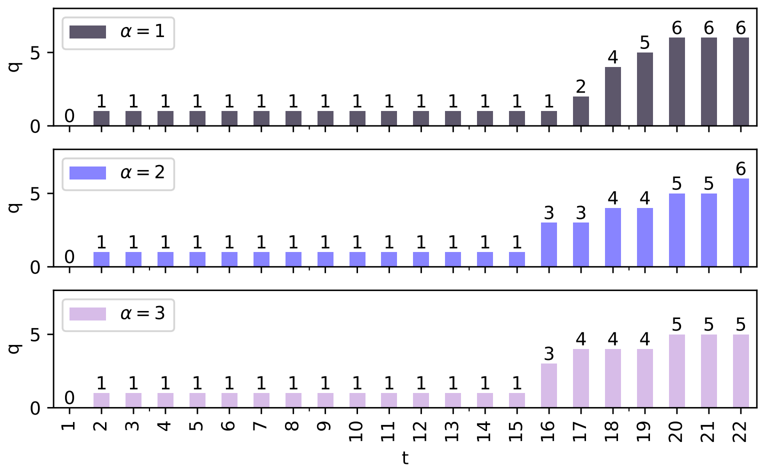


Figure 7.4: Ramp Up CISDS selection of the q parameter, with different α values and no additional unmeasured site selection by FSCA.

have $q = 0$. While these methods are not considered in the paper, they bound the site selection behaviors that can be achieved with the proposed methods. It can be observed that CISDS and Greedy CISDS have more forced sites early in the measurement plan, while Step Up and Ramp Up (with different α values) delay the selection of forced sites to later in the plan.

7.3.5 Double FSCA (DFSCA-Greedy)

In CISDS (Algorithm 7), and its variants introduced thus far, the q preselected elements are drawn randomly from $\mathcal{I}_U$. This may not be the most effective way to select the forced sites. To achieve an optimized selection, the q preselected sites can be chosen using the FSCA algorithm, but forcing it to choose only among sites in $\mathcal{I}_U$ that maximize process visibility (i.e. minimize E_R). The remaining $k - q$ sites are then computed with FSCA from the full set of candidate sites, as in the original CISDS implementation. The resulting Double FSCA (DFSCA) process is applicable to all CISDS variants. The pseudocode for DFSCA applied to the Greedy CISDS variant (equation (7.10)) is given in Algorithm 10. Here, the first q sites are selected from $\mathcal{I}_U$ on line 7 and the remaining $k - q$ sites are selected from $\mathcal{I}_V$ on line 8.

7.3.6 Ramp Up CISDS with DFSCA (DFSCA-Ramp)

As a further test, DFSCA is also implemented with the most advanced technique for selecting the q value, the Ramp Up CISDS algorithm. This implementation simply requires calling Algorithm 8 on line 6 of Algorithm 10 to select the q previously unmeasured sites at each iteration.

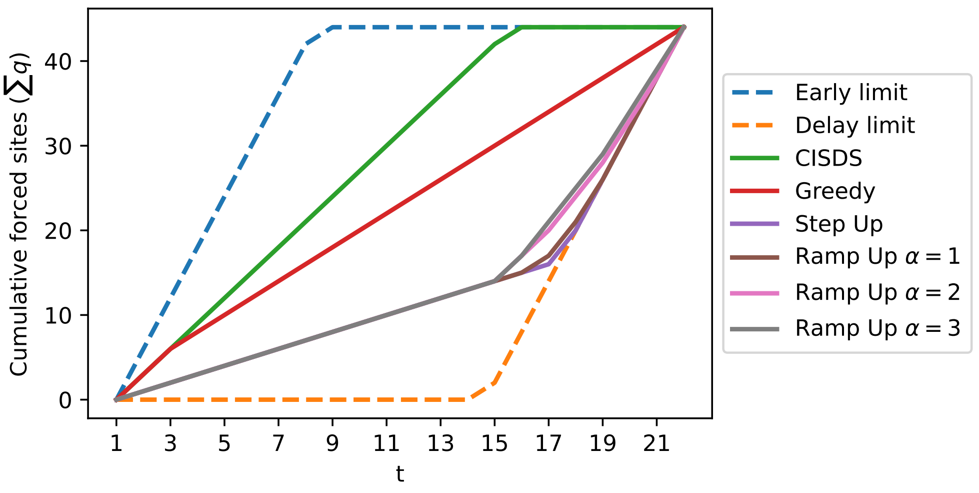


Figure 7.5: A comparison of the unmeasured site selection profiles obtained with various temporally constrained dynamics sampling algorithms

7.3.7 Recursive FSCA (RFSCA)

The previous algorithms are based on different strategies for forcing the selection of q unmeasured sites at each iteration, while accounting for the possibility that FSCA will also select some previously unmeasured sites. However, FSCA is not restricted to selecting sites from $\mathcal{I}_U$, so it may happen that the $k - q$ sites are all selected from previously selected sites. To avoid this, we can force FSCA to select all k sites directly from the set of unmeasured sites $\mathcal{I}_U$ at each iteration. This can be done by simply setting $\mathcal{I}_C = \mathcal{I}_U$ in Algorithm 4. After a number of iterations, $\mathcal{I}_U$ will contain less than k sites (eventually zero), so we can switch to the standard FSCA algorithm. The pseudocode for the resulting algorithm, which we refer to as Recursive FSCA (RFSCA), is reported in Algorithm 11. Note that RFSCA is an exemplar of the *early limit* site selection strategy, as plotted in Figure 7.5.

7.3.8 Constrained Clustering SDS (CCSDS)

As noted earlier the MSSI of SDS is determined by the cardinality of the largest cluster generated. As such, a temporally bounded implementation of SDS can be obtained if the cardinality of the clusters can be upper bounded. In [49] the authors showed how it is possible to obtain good clustering performance, while having a bounded maximum cluster cardinality, using a modified version of Lloyd's algorithm [97]. Lloyd argued that constrained clustering can even outperform classical k-means if you have a good initial guess for the locations of the centroids. In our setting, we can use FSCA to select the initial centroids. Other than the constrained clustering modification all other aspects of SDS (already detailed in Algorithm 5) remain unchanged. The pseudocode for the

Algorithm 10: DFSCA-Greedy

```
    Input: $\mathbf{X}, k, \mathcal{I}_V, T_{max}$
 1: $\mathcal{P}_1 \leftarrow \text{GenFSCA}(\mathbf{X}, k, \emptyset, \mathcal{I}_V)$
 2: $\mathcal{I}_S \leftarrow \mathcal{P}_1$
 3: $\mathcal{I}_U \leftarrow \mathcal{I}_V \setminus \mathcal{I}_S$
 4: $t \leftarrow 2$
 5: while $\mathcal{I}_U \neq \phi$ and $t \leq T_{max}$ do
 6:     $q = \lceil \frac{|\mathcal{I}_U|}{T_{max}-t+1} \rceil$
 7:     $\mathcal{I}_{init} \leftarrow \text{GenFSCA}(\mathbf{X}, q, \emptyset, \mathcal{I}_U)$
 8:     $\mathcal{P}_t \leftarrow \text{GenFSCA}(\mathbf{X}, k, \mathcal{I}_{init}, \mathcal{I}_V)$
 9:     $\mathcal{I}_S \leftarrow \mathcal{I}_S \cup \mathcal{P}_t$
10:     $\mathcal{I}_U \leftarrow \mathcal{I}_V \setminus \mathcal{I}_S$
11:     $t \leftarrow t + 1$
12: end while
13: return $\mathcal{P}$
```

constrained clustering algorithm is presented in Algorithm 12. Here, correlation is used as the distance function, and $\mathcal{I}_{init}$ is the set of initial cluster centroids.

7.4 Results

7.4.1 Case studies and Experimental setup

In this section we evaluate the performance of the different temporally constrained dynamics sampling algorithms previously introduced for two case studies, both representative of spatial metrology applications in semiconductor manufacturing. The first is an *industrial* dataset consisting of wafer metrology data for a process used in read–write head formation within disk drive semiconductor manufacturing. A static measurement plan with 50 sites ($v = 50$) has been used to measure 316 wafer profiles from a single production tool. For confidentiality reasons the data is normalized and cannot be shared, but the results can be compared with [114], which uses the same dataset. The second case study (the *RBF* dataset), first introduced in [114], is a synthetic dataset consisting of wafer height data at 100 measurement points ($v = 100$) on 1257 simulated wafer profiles generated from the sums of randomly generated Gaussian Radial Basis Functions (RBF). The wafer height $z(x, y)$ at position (x, y) on the wafer surface is:

$$z(x,y) = \sum_{i=1}^{N_g} h_i \, e^{\left(\frac{(x-c_{x_i})^2+(y-c_{y_i})^2}{S_f^2}\right)} + \epsilon, \tag{7.13}$$

where $h_i \sim \mathcal{N}(0, 1)$, $c_{x_i}, c_{y_i} \sim U(-1, 1)$ and $\epsilon \sim \mathcal{N}(0, 0.02)$. The wafers are defined on a unit radius disk centered at the origin (i.e. at $(x, y) = (0, 0)$) and the measurements are subject to Gaussian measurement noise (ϵ). Equation (7.13) has two user defined

Algorithm 11: RFSCA

```
Input: $\mathbf{X}, k, \mathcal{I}_V, T_{max}$
1: $\mathcal{I}_U \leftarrow \mathcal{I}_V$
2: $t \leftarrow 1$
3: while $t \leq T_{max}$ do
4:    if $|\mathcal{I}_U| < k$ then
5:       $\mathcal{P}_t \leftarrow$ GenFSCA$(\mathbf{X}, k, \mathcal{I}_U, \mathcal{I}_V)$
6:    else
7:       $\mathcal{P}_t \leftarrow$ GenFSCA$(\mathbf{X}, k, \emptyset, \mathcal{I}_U)$
8:    end if
9:    $\mathcal{I}_U \leftarrow \mathcal{I}_U \setminus \mathcal{P}_t$
10:   $t \leftarrow t + 1$
11: end while
12: return $\mathcal{P}$
```

Algorithm 12: Constrained Clustering

```
Input: $\mathbf{X}, k, \mathcal{I}_{init}, \mathcal{I}_V, T_{max}$
1: for $i \leftarrow 1$ to $k$ do
2:    $\mathbf{c}_i \leftarrow \mathbf{X}_{\mathcal{I}_{init}[i]}$
3:    $\mathcal{C}_i \leftarrow \emptyset$
4: end for
5: while not converged do
6:    for all $j \in \mathcal{I}_V$ do
7:       $i^* \leftarrow \underset{i=1,\dots,k}{\arg\min}\{\text{distance}(\mathbf{c}_i, \mathbf{x}_j) \text{ s.t. } |\mathcal{C}_i| < T_{max}\}$
8:       $\mathcal{C}_{i^*} \leftarrow \mathcal{C}_{i^*} \cup \mathbf{x}_j$
9:    end for
10:   for $i \leftarrow 1$ to $k$ do
11:      $\mathbf{c}_i \leftarrow$ centroid$(\mathcal{C}_i)$
12:   end for
13: end while
14: return $\mathcal{C}$
```

parameters, the number of basis functions, N_g, and the spread factor S_f which characterize the process spatial variability. Following [158] we set $N_g = 100$ and $S_f = 0.6$. The candidate measurement sites are taken from a uniform grid with a resolution of 0.05.

Since the performance of a dynamic sampling algorithm depends on the number of measured sites, k, per unit and the specified MSSI limit, T_{max}, we evaluate the performance of our algorithms for different values of k and different MSSI limits. The minimum MSSI achievable is inversely proportional to k, specifically it is computed as $\lceil \frac{v}{k} \rceil$, where v is the total number of sites. To account for this, we evaluated algorithm performance for MSSI values that were multiples of $\frac{v}{k}$, i.e. $T_{max} = \lceil \beta \frac{v}{k} \rceil$ with $\beta = 1.3$ and 1.7. Larger values of β were not considered as the greater flexibility this provides in choosing q results in algorithm performances approaching that of unconstrained ISDS which forces $q = 1$ unmeasured sites to be selected at each iteration.

The presented results are based on the statistics of the reconstruction error, E_R as defined equation (7.4), computed over 100 Monte Carlo cross validation simulations. In each simulation 67% of the data is randomly selected for training and the remaining 33% is used for testing. In the case of Ramp Up and DFSCA-Ramp results are presented for the best performing $\alpha \in \{1, 2, 3\}$. The optimal α values are noted in the tables.

We also report the performance of the unconstrained ISDS algorithm ($q = 1$) as this represents a lower bound on the E_R that can be achieved with temporally constrained algorithms.

7.4.2 Performance Comparison

The mean and standard deviation of the reconstruction error E_R (over 100 Monte Carlo cross validation simulations) achieved by each method are reported in Tables 7.1 to 7.4 for $k \in \{4, 5, 6, 7, 8\}$. Tables 7.1 and 7.2 are the results for the industrial dataset for $\beta = 1.3$ and 1.7, respectively, and Tables 7.3 and 7.4 are the corresponding results for the RBF dataset. For each value of k, the best performing method is highlighted in bold font and the second best method is highlighted in blue. A method is deemed to be better than the next best method if the difference in their mean reconstruction errors is statistically significant ($p = 0.05$). Otherwise they are considered equivalent. Consequently, for some results presented there are ties for first place (e.g. for $k = 5$ in Table 7.1) and second place (e.g. for $k = 8$ in Table 7.3).

To visualize the difference in performance between methods boxplots of the distribution of E_R over the 100 Monte Carlo simulations for each method are presented in Figure 7.8 for selected k values for both the industrial and RBF datasets.

The results show that across all parameter combinations considered Constrained Clustering SDS (CCSDS) is consistently the worst performing method, while DFSCA-Greedy is consistently the best method. The other methods show more variability in performance, with Ramp Up and DFSCA-Ramp Up in second place or joint first place for the industrial dataset and DFSCA-Ramp and Greedy in second place for the RBF dataset. The performance of Step Up is generally poor, and, as expected, is outperformed by the Ramp

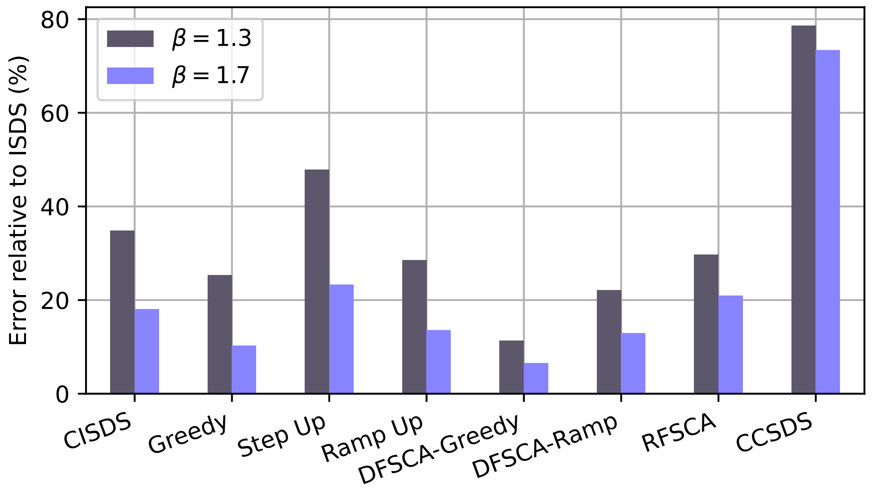


Figure 7.6: Performance of each temporally constrained dynamic sampling algorithm relative to ISDS.

Up and DFSCA-Ramp enhancements. RFSCA works well for low values of k but its performance deteriorates as k increases. This is because RFSCA is an early limit method, and as such the forced sites are distributed more equally when k is low. However, as k increases the forced sites are concentrated in the early units, and the performance of RFSCA deteriorates.

To enable aggregation of results across k values and datasets we define the relative performance of each method with respect to ISDS as

$$\mathcal{E}_{ISDS} = \frac{E_R - E_R^{ISDS}}{E_R^{ISDS}} \times 100\% \tag{7.14}$$

where E_R^{ISDS} denotes the reconstruction error with ISDS.

Figure 7.6 shows the $\mathcal{E}_{ISDS}$ for each method, averaged over both the Industrial and RBF datasets and all values of k, for $\beta = 1.3$ and 1.7 (i.e., for $T_{max} = \lceil 1.3 \cdot \frac{v}{k} \rceil$ and $\lceil 1.7 \cdot \frac{v}{k} \rceil$). As expected, the relative performance of each method improves when β is increased, with $\mathcal{E}_{ISDS}$ reducing by between 40% and 60% for most methods. The exceptions are RFSCA and CCSDS with reductions of 29% and 6.6%, respectively, as these methods do not explicitly take advantage of the extra flexibility afforded by the relaxation of the temporal constraint.

Overall, the results clearly validate the assumptions made in Section 7.2: the best performing methods are those that distribute the forced sites as evenly as possible across the measurement plan, and those that employ FSCA selection of the forced sites from I_U rather than random selection. These assumptions are validated by the fact that DFSCA-Greedy is the best performing method, since it incorporates both the Greedy distribution of forced sites, which yields the most even distribution among the methods investigated, and the FSCA selection of the q sites from I_U. In fact, DFSCA-Greedy achieves an overall relative error of 9%, which is half that of Greedy and DFSCA-Ramp, the second best performing methods ($\mathcal{E}_{ISDS} = 18\%$). Ramp Up is in fourth place with $\mathcal{E}_{ISDS} = 21\%$. The values for RFSCA, CISDS, Step Up and CCSDS are 25%, 27%, 36% and 76%, respectively.

7.4.3 Forced sites distribution over the measurement plan

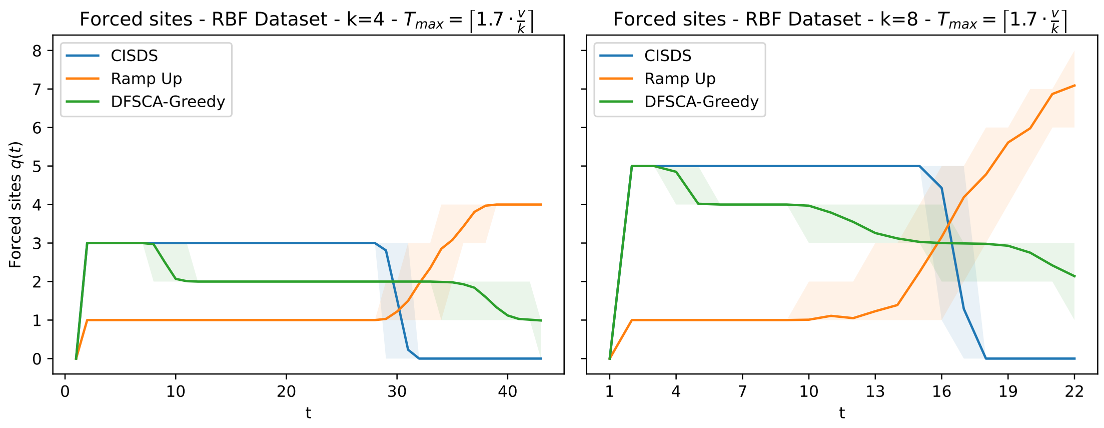


Figure 7.7: Forced sites trends.

Figure 7.7 shows the $q(t)$ trends for the different methods on the RBF dataset, with $k \in \{4, 8\}$ and $T_{max} = \lceil 1.7 \cdot \frac{v}{k} \rceil$. Each line represents the mean $q(t)$ value over 100 cross validation cycles, while the shaded area highlights the minimum and maximum values recorded. We can see that the average $q(t)$ trends are in line with the example trends presented in Section 7.2. CISDS accumulates all the forced sites at the beginning of the measurement plan and the only uncertainty derives from the emptying of the I_U set. The Ramp Up method instead postpones the forced sites selection, and tends to saturate towards the end. In contrast with Greedy selection, as employed in DFSCA-Greedy, the forced sites are generally lower in number and distributed more evenly over the measurement plan, which is linked to its good performance. In general, the plots show that, while the additional unmeasured sites selected by FSCA do influence the $q(t)$ trend, their contribution is limited and the trends are dominated by the characteristics imposed by the $q(t)$ selection method.

7.5 Conclusions

This chapter has considered the problem of temporally constrained spatial dynamic sampling for metrology optimization. We explored different methods of incorporating a constraint on the MSSI into spatial dynamic sampling techniques, providing the user with the ability to specify a desired maximum temporal horizon T_{max} by which all candidate sites must be measured. This is a desirable property for such methods since it guarantees that potential abnormal behaviors can be detected in a limited amount of time.

To satisfy the temporal visibility constraint it is necessary to force some previously unmeasured sites to be visited at each iteration. We have proposed a range of strategies (algorithms) for selecting the number and location of the forced sites on each unit across the measurement plan, and evaluated their performance on real world and simulated semiconductor manufacturing datasets. Our results show that distributing the forced sites evenly over the measurement plan, and employing FSCA to select them from the set of previously unselected sites, achieves the best overall performance. This combination, which is embodied in DFSCA-Greedy, achieves a reconstruction accuracy performance approaching that of the unconstrained ISDS algorithm that serves as a lower bound on the achievable performance with constrained dynamic sampling approaches. On average, over the various parameter and dataset combinations investigated, DFSCA-Greedy achieves measurement reconstruction errors that are within 9% of the ISDS lower bound. This is a factor of two better than the next best algorithm. For appropriately chosen k values and/or more relaxed temporal constraints the difference between DFSCA-Greedy and ISDS is much less. For example, in the Industrial case study, when $k = 8$ and $\beta = 1.7$ the relative error is less than 1%. As such, for practical deployment, DFSCA-Greedy is the recommended approach for temporally constrained dynamic sampling.

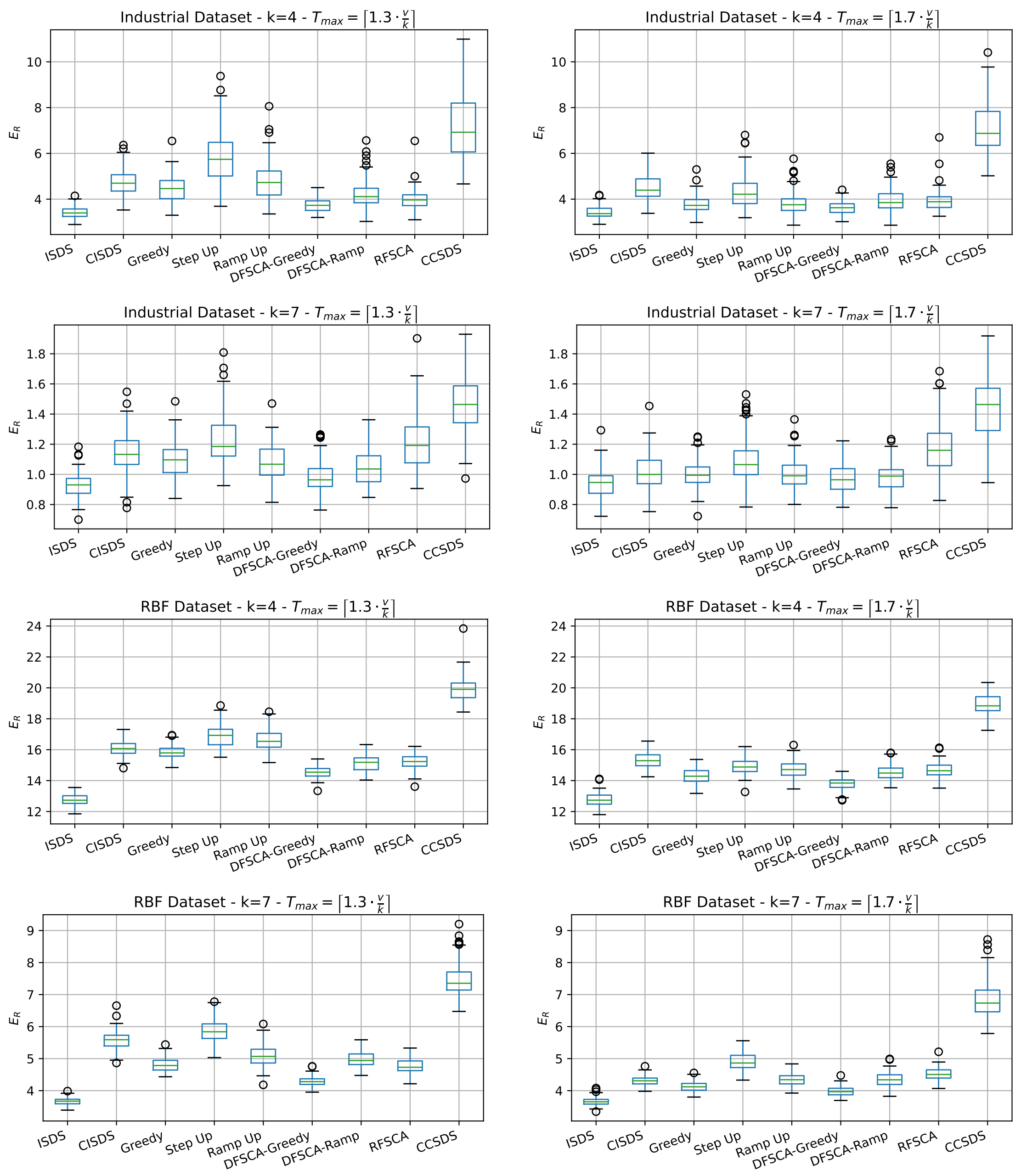


Figure 7.8: Distribution of the normalised mean squared wafer profile reconstruction error (E_R) with each dynamic sampling algorithm for the industrial and RBF datasets when $k = 4$ (top) and $k = 7$ (bottom) and the temporal constraint is $T_{max} = \lceil 1.3 \cdot \frac{v}{k} \rceil$ (left) and $T_{max} = \lceil 1.7 \cdot \frac{v}{k} \rceil$ (right)

.

Table 7.1: E_R as a function of the number of selected sites, k, for the industrial dataset when $T_{max} = \lceil 1.3 \cdot \frac{v}{k} \rceil$.
The values reported are the Mean $\pm$ standard deviation over 100 Monte Carlo simulations. For each value of k, the best performing method is highlighted in **bold** and the second best in blue, assuming a statistical significance level of 95% ($p = 0.05$).

	Normalised Mean Squared Reconstruction Error (%), E_R				
	$k = 4$	$k = 5$	$k = 6$	$k = 7$	$k = 8$
ISDS	3.41 ± 0.25	2.15 ± 0.15	1.38 ± 0.10	0.93 ± 0.08	0.70 ± 0.07
CISDS	4.73 ± 0.58	2.80 ± 0.34	1.86 ± 0.20	1.13 ± 0.14	0.84 ± 0.09
Greedy	4.47 ± 0.56	2.59 ± 0.28	1.69 ± 0.18	1.09 ± 0.11	0.78 ± 0.09
Step Up	5.91 ± 1.22	3.19 ± 0.57	1.96 ± 0.32	1.23 ± 0.18	0.87 ± 0.14
Ramp Up	4.83 ± 0.83	2.49 ± 0.29	1.62 ± 0.17	1.08 ± 0.12	0.78 ± 0.08
	$\alpha = 3$	$\alpha = 3$	$\alpha = 3$	$\alpha = 3$	$\alpha = 2$
DFSCA-Greedy	$\mathbf{3.72 \pm 0.28}$	$\mathbf{2.29 \pm 0.19}$	$\mathbf{1.49 \pm 0.13}$	$\mathbf{0.98 \pm 0.11}$	$\mathbf{0.71 \pm 0.07}$
DFSCA-Ramp	4.23 ± 0.63	2.49 ± 0.37	1.55 ± 0.14	1.05 ± 0.11	0.79 ± 0.09
	$\alpha = 3$	$\alpha = 3$	$\alpha = 3$	$\alpha = 2$	$\alpha = 2$
RFSCA	3.99 ± 0.44	2.75 ± 0.40	1.72 ± 0.18	1.21 ± 0.18	0.83 ± 0.10
CCSDS	7.15 ± 1.39	3.88 ± 0.55	2.09 ± 0.28	1.47 ± 0.18	0.99 ± 0.11

Table 7.2: E_R as a function of the number of selected sites, k, for the Industrial dataset when $T_{max} = \lceil 1.7 \cdot \frac{v}{k} \rceil$. The values reported are the Mean $\pm$ standard deviation over 100 Monte Carlo simulations. For each value of k, the best performing method is highlighted in **bold** and the second best in blue, assuming a statistical significance level of 95% ($p = 0.05$).

	Normalised Mean Squared Reconstruction Error (%), E_R				
	$k = 4$	$k = 5$	$k = 6$	$k = 7$	$k = 8$
ISDS	3.42 ± 0.27	2.11 ± 0.14	1.39 ± 0.12	0.94 ± 0.09	0.70 ± 0.06
CISDS	4.50 ± 0.55	2.39 ± 0.21	1.63 ± 0.17	1.02 ± 0.11	0.77 ± 0.08
Greedy	3.79 ± 0.38	$\mathbf{2.25 \pm 0.19}$	$\mathbf{1.46 \pm 0.11}$	1.00 ± 0.09	0.74 ± 0.07
Step Up	4.34 ± 0.71	2.50 ± 0.41	1.57 ± 0.17	1.09 ± 0.15	0.80 ± 0.10
Ramp Up	3.83 ± 0.49	$\mathbf{2.26 \pm 0.20}$	1.49 ± 0.14	1.00 ± 0.10	0.74 ± 0.08
	$\alpha = 3$	$\alpha = 3$	$\alpha = 2$	$\alpha = 1$	$\alpha = 1$
DFSCA-Greedy	$\mathbf{3.62 \pm 0.28}$	$\mathbf{2.22 \pm 0.17}$	$\mathbf{1.46 \pm 0.13}$	$\mathbf{0.97 \pm 0.09}$	$\mathbf{0.70 \pm 0.06}$
DFSCA-Ramp	3.95 ± 0.49	$\mathbf{2.22 \pm 0.20}$	$\mathbf{1.44 \pm 0.11}$	$\mathbf{0.99 \pm 0.09}$	0.74 ± 0.08
	$\alpha = 3$	$\alpha = 3$	$\alpha = 3$	$\alpha = 3$	$\alpha = 1$
RFSCA	3.92 ± 0.47	2.65 ± 0.36	1.68 ± 0.18	1.17 ± 0.16	0.82 ± 0.11
CCSDS	7.12 ± 1.13	3.86 ± 0.51	2.14 ± 0.26	1.44 ± 0.21	0.95 ± 0.13

Table 7.3: E_R as a function of the number of selected sites, k, for the RBF dataset when $T_{max} = \left\lceil 1.3 \cdot \frac{v}{k} \right\rceil$. The values reported are the Mean $\pm$ standard deviation over 100 Monte Carlo simulations. For each value of k, the best performing method is highlighted in **bold** and the second best in blue, assuming a statistical significance level of 95% ($p = 0.05$).

	Normalised Mean Squared Reconstruction Error (%), E_R				
	$k = 4$	$k = 5$	$k = 6$	$k = 7$	$k = 8$
ISDS	12.77 ± 0.34	8.09 ± 0.26	5.40 ± 0.18	3.67 ± 0.11	2.54 ± 0.08
CISDS	16.09 ± 0.52	11.24 ± 0.46	8.01 ± 0.38	5.57 ± 0.28	3.49 ± 0.17
Greedy	15.85 ± 0.43	10.72 ± 0.41	7.20 ± 0.29	4.81 ± 0.21	3.28 ± 0.14
Step Up	16.89 ± 0.69	11.91 ± 0.61	8.45 ± 0.58	5.86 ± 0.37	4.08 ± 0.40
Ramp Up	16.61 ± 0.64	11.37 ± 0.59	7.67 ± 0.43	5.08 ± 0.35	3.36 ± 0.16
	$\alpha = 3$	$\alpha = 3$	$\alpha = 3$	$\alpha = 3$	$\alpha = 3$
DFSCA-Greedy	**14.55 ± 0.38**	**9.42 ± 0.36**	**6.36 ± 0.25**	**4.29 ± 0.16**	**2.97 ± 0.13**
DFSCA-Ramp	15.15 ± 0.49	10.08 ± 0.38	6.96 ± 0.32	4.99 ± 0.23	3.45 ± 0.20
	$\alpha = 1$	$\alpha = 3$	$\alpha = 2$	$\alpha = 1$	$\alpha = 3$
RFSCA	15.24 ± 0.48	10.22 ± 0.35	7.08 ± 0.22	4.79 ± 0.24	3.30 ± 0.17
CCSDS	19.89 ± 0.75	14.83 ± 0.90	10.28 ± 0.67	7.48 ± 0.53	5.37 ± 0.41

Table 7.4: E_R as a function of the number of selected sites, k, for the RBF dataset when $T_{max} = \left\lceil 1.7 \cdot \frac{v}{k} \right\rceil$. The values reported are the Mean $\pm$ standard deviation over 100 Monte Carlo simulations. For each value of k, the best performing method is highlighted in **bold** and the second best in blue, assuming a statistical significance level of 95% ($p = 0.05$).

	Normalised Mean Squared Reconstruction Error (%), E_R				
	$k = 4$	$k = 5$	$k = 6$	$k = 7$	$k = 8$
ISDS	12.78 ± 0.40	8.08 ± 0.25	5.38 ± 0.18	3.67 ± 0.13	2.54 ± 0.08
CISDS	15.30 ± 0.50	9.56 ± 0.36	6.74 ± 0.25	4.30 ± 0.16	3.05 ± 0.13
Greedy	14.30 ± 0.45	9.35 ± 0.31	6.17 ± 0.24	4.12 ± 0.16	2.88 ± 0.12
Step Up	14.91 ± 0.51	10.21 ± 0.41	7.14 ± 0.36	4.90 ± 0.28	3.44 ± 0.22
Ramp Up	14.71 ± 0.56	9.83 ± 0.44	6.59 ± 0.32	4.34 ± 0.19	3.02 ± 0.16
	$\alpha = 3$	$\alpha = 3$	$\alpha = 3$	$\alpha = 3$	$\alpha = 2$
DFSCA-Greedy	**13.81 ± 0.39**	**8.81 ± 0.30**	**5.87 ± 0.20**	**3.98 ± 0.14**	**2.81 ± 0.11**
DFSCA-Ramp	14.53 ± 0.47	9.51 ± 0.34	6.46 ± 0.28	4.34 ± 0.22	3.17 ± 0.15
	$\alpha = 2$	$\alpha = 3$	$\alpha = 3$	$\alpha = 3$	$\alpha = 3$
RFSCA	14.68 ± 0.49	9.63 ± 0.38	6.72 ± 0.26	4.52 ± 0.20	3.15 ± 0.16
CCSDS	18.93 ± 0.63	13.82 ± 1.40	10.55 ± 1.09	6.85 ± 0.56	5.04 ± 0.34

Chapter 8

Conclusions and Future Perspectives

This thesis and the Ph.D. research projects focused on the application of innovative ML and DL models to the control systems of particle accelerators. The main goal was to improve the performance of the control systems, in terms of reliability, efficiency, and cost-effectiveness. The research was carried out in collaboration with the INFN at the LNL. Furthermore, we explored the use of data-driven algorithms to optimized measurement plans in industrial processes.

First we explored the use of Anomaly Detection models to predict faults in the machine. The RF control system of the ALPI accelerator was used as a case-study given the availability of a recent dataset containing runtime faults. Two different approaches have been studied and evaluated: the first is based on simple unsupervised AD models which calculate an outlier score from a static dataset composed of features extracted from sliding windows of the original data. In the second method instead, the outlier score is obtained as the prediction loss of a deep learning time-series forecasting model. In this case the model does not require defining custom features but is able to learn complex patterns from the raw data. Then, the F-score is used to select an optimal threshold on the outlier score to distinguish between faults and normal data points.

The second area of research focused on Reinforcement Learning models applied to the optimization of beam dynamics, which is a fundamental task in the setup of a run of a particle accelerator. The goal is to find the optimal set of control system parameters, which maximize the beam quality or to minimize the beam losses. We tested both the REINFORCE and PPO algorithms, both policy gradient methods, to train a model on the ADIGE beam line which could iteratively reduce the beam emittance by correcting the configuration of an electrostatic multipole with 48 parameters. This single element is used as an example in place of a more complete beam line where a high number of parameters have to be tuned together to achieve the best beam dynamics, often requiring online adjustments.

Finally, we studied the problem of constrained dynamic sampling for quality assessment in industrial processes, which output a certain product at the end of a production line. In these cases, the quality of the product is assessed by measuring a set of features at different sites of the product, with the assumption that such measures are spatially correlated. Data-driven strategies are employed to minimize the number of measures required to achieve a certain level of accuracy. In particular, we want to guarantee that all measurement sites are visited at least once in a certain number of iterations of the process, while retaining the best possible measurement accuracy. This is useful to avoid undetected defects on the production line resulting in anomalous measurement values in any given site. We proposed and evaluated different strategies to force new sites to be visited along the measurement plan, in order to fulfill the constraint. Distributing such forced sites evenly across the measurement plan and choosing the actual sites with the FSCA algorithm resulted in the best performance.

When discussing the results of the various methods presented in this thesis we highlighted the strengths and weaknesses of each approach. In particular, the results obtained from the two AD approaches are comparable, with the simpler and lighter ML model slightly outperforming the DL one. This indicates that in some cases deep domain knowledge and good feature engineering, especially when combined with feature selection techniques like feature importance, can lead to excellent results without requiring complex DL models. On the other hand, this approach is not very scalable and requires a lot of time and effort to be applied to different systems. The flexibility offered by DL models is therefore a great advantage, especially when the data is not well understood and the features are not trivial to extract. Furthermore, while a large amount of data was available, the number of faults was very limited, which means that it was difficult to build a reliable statistics on the results. By collecting new datasets from future run of the machine it will be possible to improve the performance of the models and to better understand their limitations. In fact, different types of faults can be present in the data, with some of them being extremely rare. The more variability and different scenarios are included in the data, the more the model will be able to learn the underlying patterns. In this case, the advantage of more complex DL models will become more evident.

However, this does not come without a cost. The computational requirements of dealing with larger and larger datasets can easily become a limiting factor. In fact, once the data can no longer fit in the GPU memory, the training process has to be updated to streaming solutions, as we did in chapter 5. To speed up the training one can consider scaling to multiple GPUs on the same machine. Then, when even the host memory or processing power is not enough, the whole computing model has to be rethought. In this case, the use of cloud computing platforms or large computing clusters becomes necessary. This is a very common problem in the fields of DL and big data, and requires the use of specific software frameworks like Spark or Hadoop, where the computation is distributed across multiple machines. Thus, high performance computing infrastructures are required, along with high quality, production-grade software to exploit it. This is not always the case in research environments, where the focus is on the scientific results and not on the software engineering, and it's an area where experience from the industry can be very useful. Nevertheless, this could easily become a limiting factor or a hurdle to research in a world where models are becoming more and more complex, with trillions of parameters. Researches should be able to focus on the scientific aspects of their work, having easy access to the computational resources they need. For this reason cloud computing platforms are becoming more and more popular, as they allow to scale up and down the computational resources as needed, without having to worry about the underlying hardware and provide high-level easy to use access to the resources. The experimental physics research community has a long history of developing and maintaining large computing infrastructures, without relying on expensive third-parties, and should leverage it to provide modern solutions for the new computation paradigms. In this sense initiatives like INFN-Cloud are very promising and should be further developed and supported.

The work on Reinforcement Learning to optimize beam dynamics is a paradigmatic example of these requirements. Even on a small scale beam line the simulation computing time became the limiting factor, with limited options for scaling. On one side research on sample efficient algorithms should be pursued, but on the other side even experimenting with different algorithms requires a lot of computing resources. Thus, providing an effective platform open to researchers could enable better research and finally better models. Nevertheless, results presented in chapter 6 showed that combining physics constraints and the RL framework it's possible to learn an optimization model for the beam dynamics. This means that the model, once trained mainly offline, can be used online on the accelerator to optimize the beam transport by tuning all the control system parameters. In linear accelerators this is useful on the beam preparation phase, while synchrotrons can use these techniques for continuous orbit correction. The advantage over traditional methods comes from the knowledge learned during training, which results in faster and more accurate corrections, thus reaching the optimal objective function value in few iterations.

The models presented in this thesis can be already tested on the real machine as aids to the human operators, such as to indicate how well the RF control system is working based on the online outlier score produced by the AD models. The RL model can be used to optimize the beam dynamics in the ADIGE beam line, and the results can be compared with the current manual procedure or other online optimization algorithms. However, ML techniques like these have the potential to enable future control systems to run completely autonomously, with all the beam transport and acceleration parameters automatically tuned. This means that the operations of large facilities like particle accelerators or complex industrial plants can become much more efficient, thus maximizing the scientific output or the production yield. Both theoretical and applied research is needed to achieve this goal, and a strong community of researchers and engineers is required to develop the necessary tools and infrastructures. The interest for ML techniques inside the particle accelerator community is growing as demonstrated by the many papers published in recent years [76, 146, 98, 50, 99], and we hope that the work presented in this thesis will further raise interest and awareness on the topic, finally contributing to the development of the field.

Appendix A

Source Code

```
# Features definition
features = {
'mean' : (['FrwwRd', 'RflwRd', 'NetwRd', 'PresRd'],
          ['FrwwAv', 'RflwAv', 'NetwAv', 'PresAv']),
'std'  : (['FrwwRd', 'RflwRd', 'NetwRd', 'PresRd'],
          ['FrwwStd', 'RflwStd', 'NetwStd', 'PresStd']),
'delta' : (['FrwwRd', 'NetwRd', 'PresRd'],
           ['FrwwDt', 'NetwDt', 'PresDt']),
'sum' : (['Motr.MOVN'], ['MotrMov']),
'rising' : (['Motr.TDIR'], ['MotrRev'])}

# Output result
feature_df = pd.DataFrame()
# Resample to one point per second
data_by_step = data.resample(step, label='right').mean().ffill()
data_roll = data_by_step.rolling(window)
for func, io in features.items():
    # Calc feature values
    if func == 'delta':
        feature_df[io[1]] = data_by_step[io[0]].diff(periods = (window/
step)).bfill()
    elif func == 'rising':
        feature_df[io[1]] = (data[io[0]].diff() > 0).resample(step, label
='right').sum().rolling(window).sum()
    else:
        feature_df[io[1]] = getattr(data_roll[io[0]], func)().bfill()
```

Source Code A.1: Fast feature calculation for classical ML models.

```
def calc_f_score(y_pred, y_true, beta=1.0):
    available_events = len(np.unique(y_true[y_true >= 0].dropna()))

    true_pos = len(np.unique(y_true[y_true >=0][y_pred].dropna()))
    false_pos_all = y_true[y_true < 0][y_pred]
    false_pos = false_pos_all.mask((false_pos_all.shift(1).notna())).count
().sum() #remove consecutive duplicates
    false_neg = available_events - true_pos

    prec = true_pos/(true_pos+false_pos) if true_pos else 0.0
    recall = true_pos/(true_pos+false_neg)
    f_score = (1+beta**2) * true_pos / ((1+beta**2) * true_pos + beta**2*
false_neg + false_pos)

    return pd.Series({'f_score':f_score, 'prec':prec,
                      'recall':recall, 'true_pos':true_pos,
                      'false_pos':false_pos, 'false_neg':false_neg})
```

Source Code A.2: F-score, precision and recall metrics implementation.

```
class DeepicsModel(object):
    def __init__(self, model, roll_t=False, beta=1.0):
        self.model = model
        self.thrs = None
        self.roll_t = roll_t
        self.beta = beta

    # get outlier probability, eventually averaged over roll_t seconds
    def _probability(self, X):
        # I tried unify method but it's much worse, using linear
        y_prob = self.model.predict_proba(X)[:, 1]
        y_prob = pd.DataFrame(y_prob, index=X.index)
        if self.roll_t:
            y_prob = y_prob.rolling(self.roll_t).mean()
        return y_prob

    # fit model and choose best thrs based on f_score
    def fit(self, X_train, y_train):
        self.model.fit(X_train)
        y_train_prob = self._probability(X_train)

        head = y_train_prob[y_train >= 0].mean().sum()
        tail = y_train_prob[y_train < 0].mean().sum()

        max_f_score = 0
```

```
            best_scores = None
            for rel_thrs in np.linspace(0.5, 2.5, num=81, endpoint=True):
                thrs = rel_thrs * head
                y_pred = y_train_prob > thrs
                scores = calc_f_score(y_pred, y_train, self.beta)
                if scores[0] > max_f_score:
                    max_f_score = scores[0]
                    best_scores = scores
                    self.thrs = thrs

            return best_scores

        # predict outlier probability of new data
        def predict(self, X_test):
            y_test_prob = self._probability(X_test)
            return y_test_prob > self.thrs

        def score(self, y_pred, y_true):
            return calc_f_score(y_pred, y_true, self.beta)[0]
```

Source Code A.3: DeepicsModel class implementation. This wraps a pyod model to extend the fit method to search for the best threshold value.

```
    DRIFT 1e−008 150 0 0 0
    DRIFT 214 150 0 0 0
    DRIFT 112 150 0 0 0
    AD.SO.01 : SOLENOID 320 0.375536 65
    DRIFT 425 150 0 0 0
    DRIFT 308.5 150 0 0 0
    AD.SO.02 : SOLENOID 320 0.134319 65
    DRIFT 88.5 150 0 0 0
    DRIFT 190.5 150 0 0 0

    DRIFT 200 150 0 0 0
    AD.AT.01 : ELECTROSTA_ACC 120000 500 0.817421 50
    DRIFT 200 150 0 0 0
    DRIFT 42.5 150 0 0 0
    DRIFT 42.5 150 0 0 0

    Dia : DRIFT 1e−008 150 0 0 0
    DRIFT 150 150 0 0 0
    DRIFT 100 150 0 0 0
    AD.ST.04 : THIN_STEERING 0 0 150 0
    DRIFT 150 150 0 0 0
```

```
AD.1EQ.01 : QUAD_ELE 200 -9530.98 50 0 0 0 0 0 0
DRIFT 513.209 150 0 0 0
AD.1EQ.02 : QUAD_ELE 200 -2262.8 50 0 0 0 0 0 0
DRIFT 155.77 350 0 0 0
SUPERPOSE_MAP_OUT 1050 1050.0 0.0  0 0 -90
AD.D.02 : FIELD_MAP 70 1400 0 550 -0.200708 0 0 0 sd1b

DRIFT 4 350 0 0 0
superpose_map 0
AD.EM : FIELD_MAP 7 1200 0 150 0 -1230.72 0 0 MpoloV6b
superpose_map 0
AD.EM : FIELD_MAP 7 1200 0 150 0 -24.4137 0 0 MpoloV8b
superpose_map 0
AD.EM : FIELD_MAP 7 1200 0 150 0 -1.99093 0 0 MpoloV10b
superpose_map 0
AD.EM : FIELD_MAP 7 1200 0 150 0 -83.4286 0 0 MpoloV12b
DRIFT 4 350 0 0 0

SUPERPOSE_MAP_OUT 1050 1050.0 0.0  0 0 -90
AD.D.03 : FIELD_MAP 70 1400 0 550 0.200708 0 0 0 sd2b
DRIFT 4 350 0 0 0
DRIFT 151.77 350 0 0 0
AD.1EQ.03 : QUAD_ELE 200 -2262.8 50 0 0 0 0 0 0
DRIFT 513.209 150 0 0 0
AD.1EQ.04 : QUAD_ELE 200 -9530.98 50 0 0 0 0 0 0
DRIFT 50 150 0 0 0
DRIFT 50 150 0 0 0
DRIFT 50 150 0 0 0
AD.ST.05 : THIN_STEERING 0 0 150 0
DRIFT 50 150 0 0 0
DRIFT 50 150 0 0 0
DRIFT 50 150 0 0 0
DRIFT 100 150 0 0 0
dia : DRIFT 1e-008 150 150 0 0
DIAG_TWISS 22 0.0001 0.3 0.0001 0.3
DIAG_WAIST 20 0.001
DIAG_EMIT 30 0.1
end

```

Source Code A.4: ADIGE beam line definition for the TraceWin simulator.

```
class TraceWinEnv():
    def _perform_actions(self, actions):
        actions = np.array(actions)
        decim = np.array([50]*len(self.params))
        self.params[actions>0] = self.params[actions>0] - decim[actions>0]
        self.params[actions<=0] = self.params[actions<=0] + decim[actions<=0]
        return self._params_to_args(self.params)

    def _observation(self):
        # Parse resulting files
        curr_run = self.get_current_run()

        # Calculate reward
        res = curr_run.results
        ex = res[res["##"]==48]["ex"].to_numpy()
        ex_diff = (self.last_ex - ex)*10
        self.last_ex = ex

        # Calculate new state
        distr = curr_run.dst
        H, xedges, yedges = np.histogram2d(distr["x"]*1000, distr["xp"]*1000, bins=9, range=[[-10, 10], [-20, 20]], normed=False)
        state = H.T/(H.max()+0.0001)

        # Calculate done
        done = ex < 0.15 or ex > 0.7 or self.n_steps > 10
        return state, ex_diff, done, ex # observation, reward, done, info

    def reset(self, size=1, params=None):
        self.n_steps = 0
        self.params = params
        if self.params is None:
            self.params = np.random.randint(-1700, -700, size=size).astype("float64")
        state, _, _, info = self.step(np.random.randint(1))
        return state, info

    def step(self, actions): # {0: UP 1: DOWN}
        self.current_run = None
        self.n_steps +=1

        # Decide new params given the actions
        args = self._perform_actions(actions)

        # RUN simulation
        self.tracewin.run(args)

        # Return an observation
        return self._observation()
```

Source Code A.5: TraceWin environment for the REINFORCE algorithm.

```
policy = PolicyNet()
env = TraceWinEnv("ramdisk/MRMS_config/CB_newMRMS_RFQ_Fields_1.ini")
optimizer = torch.optim.Adam(policy.parameters(), lr=0.01)
n_episode = 1

try:
    while True:
        rewards = []
        actions = []
        states  = []

        # reset environment
        state, ex = env.reset()

        while True:
            inps = torch.tensor(state).unsqueeze(0).unsqueeze(0).float()
            probs = policy(inps)
            sampler = Categorical(probs)
            action = sampler.sample()

            # use that action in the environment
            new_state, reward, done, info = env.step(action)

            # store state, action and reward
            states.append(state)
            actions.append(action)
            rewards.append(reward)

            state = new_state
            if done:
                break

        # preprocess rewards
        rewards = np.array(rewards)
        R = torch.sum(torch.tensor(rewards))
```

```

            # preprocess states and actions
            states = torch.tensor(np.array(states)).unsqueeze(1).float()
            actions = torch.tensor(actions)

            # calculate gradient
            probs = policy(states)
            sampler = Categorical(probs)
            log_probs = -sampler.log_prob(actions)
            pseudo_loss = torch.sum(log_probs * R)

            # update policy weights
            optimizer.zero_grad()
            pseudo_loss.backward()
            optimizer.step()

            n_episode += 1

    except KeyboardInterrupt:
        pass
```

Source Code A.6: Training loop for the REINFORCE algorithm.

Appendix B

Acronyms

LNL	Legnaro National Laboratories
ALPI	*Acceleratore Lineare per Ioni*
PIAVE	*Positive Ion Accelerator for VEry low velocity ions*
INFN	Istituto Nazionale di Fisica Nucleare
SPES	*Selective Production of Exotic Species*
ADIGE	*Acceleratore Di Ioni a Grande Carica Esotici*
MRMS	Medium Resolution Mass Separator
TIS	Target Ion Source
LHC	Large Hadron Collider
ML	Machine Learning
RF	Radio Frequency
ECR	Electron Cyclotron Resonance
RFQ	Radio Frequency Quadrupole
HV	High Voltage
FC	Faraday Cup
He	Helium
N	Nitrogen
Cu	Copper
Nb	Niobium
QWR	Quarter Wave Resonator
EPICS	*Experimental Physics and Industrial Control System*
IOC	Input Output Controller

CA	Channel Access
PV	Process Variable
GUI	Graphical User Interface
STSC	*Subsequence time-series Clustering*
LOF	Local Outlier Factor
KNN	K-Nearest Neighbour
LRD	*Local Reachability Density*
CBLOF	Cluster Based Local Outlier Factor
NN	Neural Network
CNN	Convolutional Neural Network
RNN	Recurrent Neural Network
LSTM	Long Short Term Memory
CSV	Comma Separated Values
TP	True Positive
TN	True Negative
FP	False Positive
FN	False Negative
AD	Anomaly Detection
DL	Deep Learning
SVM	Support Vector Machine
MSE	Mean Squared Error
SGD	Stochastic Gradient Descent
GPU	Graphical Processing Unit
TCN	Temporal Convolutional Network
RL	Reinforcement Learning
MDP	Markov Decision Process
POMDP	Partially Observable Markov Decision Process
A2C	Advantage Actor-Critic
PPO	Proximal Policy Optimization
LEB	Low Energy Buncher
ESS	European Spallation Source
MSSI	Maximum Site Sampling Interval

FSCA	Forward Selection Component Analysis
SDS	Sequential Dynamic Sampling
ISDS	Induced Start Dynamic Sampling
CISDS	Constrained Induced Start Dynamic Sampling
DFSCA	Double FSCA
CCSDS	Constrained Clustering SDS
RBF	Gaussian Radial Basis Functions

Bibliography

[1] K. Agari et al. An Application of Machine Learning for the Analysis of Temperature Rise on the Production Target in Hadron Experimental Facility at J-PARC. In *Proc. ICALEPCS'19*, pages 992–996, 08 2020.

[2] Andreetto, Paolo, Costa, Fulvia, Crescente, Alberto, Fantinel, Sergio, Fanzago, Federica, Mazzon, Paolo Emilio, Menguzzato, Matteo, Sella, Gianpietro, Sgaravatto, Massimo, Traldi, Sergio, Verlato, Marco, Zanetti, Marco, and Zangrando, Lisa. Evolution of the cloudveneto.it private cloud to support research and innovation. *EPJ Web Conf.*, 245:07013, 2020.

[3] F. Angiulli and C. Pizzuti. Fast outlier detection in high dimensional spaces. In *European conference on principles of data mining and knowledge discovery*, pages 15–27. Springer, 2002.

[4] M. Antonini, M. Vecchio, F. Antonelli, P. Ducange, and C. Perera. Smart audio sensors in the internet of things edge for anomaly detection. *IEEE Access*, 6:67594–67610, 2018.

[5] R. Arboretti, R. Ceccato, L. Pegoraro, and L. Salmaso. Design of experiments and machine learning for product innovation: A systematic literature review. *Quality and Reliability Engineering International*, 38(2):1131–1156, 2022.

[6] P. Arpaia et al. Machine learning for beam dynamics studies at the cern large hadron collider. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 985:164652, 2021.

[7] S. Aryal, K. Santosh, and R. Dazeley. usfad: a robust anomaly detector based on unsupervised stochastic forest. *International Journal of Machine Learning and Cybernetics*, pages 1–14, 2020.

[8] S. Aryal, K. M. Ting, J. R. Wells, and T. Washio. Improving iforest with relative mass. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 510–521. Springer, 2014.

[9] P. Baldi, P. Sadowski, and D. Whiteson. Enhanced higgs boson to $\tau^+\tau^-$ search with deep learning. *Phys. Rev. Lett.*, 114:111801, Mar 2015.

[10] T. Barbariol, F. D. Chiara, D. Marcato, and G. A. Susto. A review of tree-based approaches for anomaly detection. In *Control Charts and Machine Learning for Anomaly Detection in Manufacturing*, pages 149–185. Springer, 2022.

[11] L. Bellan et al. New techniques for the lnl superconductive linac alpi beam dynamics simulations and commissioning. In *Proc. IPAC'23*, number 14 in IPAC'23 - 14th International Particle Accelerator Conference, pages 1289–1292. JACoW Publishing, Geneva, Switzerland, 05 2023.

[12] I. Bird. Computing for the large hadron collider. *Annual Review of Nuclear and Particle Science*, 61(1):99–118, 2011.

[13] E. Boldsaikhan, M. Milhon, S. Fukada, M. Fujimoto, and K. Kamimuki. Metrology of sheet metal distortion and effects of spot-welding sequences on sheet metal distortion. *Journal of Manufacturing and Materials Processing*, 7(3), 2023.

[14] M. Braei and S. Wagner. Anomaly detection in univariate time-series: A survey on the state-of-the-art. *arXiv preprint arXiv:2004.00433*, 2020.

[15] L. Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[16] M. M. Breunig et al. Lof: Identifying density-based local outliers. *SIGMOD Rec.*, 29(2):93–104, May 2000.

[17] K. Brown and S. Biedron. Summary of the 3rd icfa beam dynamics mini-workshop on machine learning applications for particle accelerators. In *Proc. IPAC'23*, number 14 in IPAC'23 - 14th International Particle Accelerator Conference, pages 4389–4392. JACoW Publishing, Geneva, Switzerland, 05 2023.

[18] M. Carletti, C. Masiero, A. Beghi, and G. A. Susto. Explainable machine learning in industry 4.0: Evaluating feature importance in anomaly detection to enable root cause analysis. In *2019 IEEE International Conference on Systems, Man and Cybernetics (SMC)*, pages 21–26. IEEE, 2019.

[19] M. Carletti, C. Masiero, A. Beghi, and G. A. Susto. Explainable machine learning in industry 4.0: Evaluating feature importance in anomaly detection to enable root cause analysis. In *2019 IEEE International Conference on Systems, Man and Cybernetics (SMC)*, pages 21–26, 2019.

[20] M. Carletti, M. Terzi, and G. A. Susto. Interpretable anomaly detection with diffi: Depth-based feature importance for the isolation forest. *arXiv preprint arXiv:2007.11117*, 2020.

[21] P. F. Carmona et al. Introducing big data analysis in a proton therapy facility to reduce technical downtime. *Proceedings of the 17th International Conference on Accelerator and Large Experimental Physics Control Systems*, ICALEPCS2019:USA, 2020.

[22] F. Chollet et al. Keras. `https://keras.io`, 2015.

[23] P.-H. Chou, M.-J. Wu, and K.-K. Chen. Integrating support vector machine and genetic algorithm to implement dynamic wafer quality prediction system. *Expert Systems with Applications*, 37(6):4413–4424, 2010.

[24] L. Clissa, M. Lassnig, and L. Rinaldi. Analyzing WLCG File Transfer Errors Through Machine Learning: An Automatic Pipeline to Support Post-mortem Distributed Data Management. *Comput. Softw. Big Sci.*, 6(1):16, 2022.

[25] M. Comunian, C. Roncolato, E. Fagotti, F. Grespan, and A. Palmieri. Beam dynamics simulations of the piave-alpi linac.

[26] Y. Cui, Z. Liu, and S. Lian. A survey on unsupervised anomaly detection algorithms for industrial images. *IEEE Access*, 11:55297–55315, 2023.

[27] A. Dainelli et al. Commissioning of the alpi post-accelerator. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 382(1):100 – 106, 1996.

[28] D. Dalle Pezze. Methodological advancements in continual learning and industry 4.0 applications. 2022.

[29] M. Das and S. Parthasarathy. Anomaly detection and spatio-temporal analysis of global climate system. In *Proceedings of the third international workshop on knowledge discovery from sensor data*, pages 142–150, 2009.

[30] L. de Ruvo, M. Allegrini, D. Benini, et al. Functional architecture of spes safety system. In *Proc. IPAC'23*, number 14 in IPAC'23 - 14th International Particle Accelerator Conference, pages 4682–4684. JACoW Publishing, Geneva, Switzerland, 05 2023.

[31] Deja, Kamil, Trzcin´ski, Tomasz, and Graczykowski, Lukasz. Generative models for fast cluster simulations in the tpc for the alice experiment. *EPJ Web Conf.*, 214:06003, 2019.

[32] C. Désir, S. Bernard, C. Petitjean, and L. Heutte. One class random forests. *Pattern Recognition*, 46(12):3490–3506, 2013.

[33] T. Dewitte et al. Anomaly detection for cern beam transfer installations using machine learning. In *Proc. ICALEPCS'19*, pages 1066–1070. JACoW, 2019.

[34] A. Di Girolamo et al. Preparing Distributed Computing Operations for the HL-LHC Era With Operational Intelligence. *Front. Big Data*, 4:753409, 2022.

[35] A. C. Diebold. *Handbook of silicon semiconductor metrology.* CRC Press, 2001.

[36] Z. Ding and M. Fei. An anomaly detection approach based on isolation forest algorithm for streaming data using sliding window. *IFAC Proceedings Volumes*, 46(20):12–17, 2013.

[37] Z.-G. Ding, D.-J. Du, and M.-R. Fei. An isolation principle based distributed anomaly detection method in wireless sensor networks. *International Journal of Automation and Computing*, 12(4):402–412, 2015.

[38] Y. Donon et al. Extended anomaly detection and breakdown prediction in linac 4's rf power source output. In *2020 International Conference on Information Technology and Nanotechnology (ITNT)*, pages 1–7, 2020.

[39] A. Edelen, S. Biedron, S. Milton, and J. Edelen. First steps toward incorporating image based diagnostics into particle accelerator control systems using convolutional neural networks. *arXiv preprint arXiv:1612.05662*, 2016.

[40] A. Edelen, C. Mayes, D. Bowring, D. Ratner, A. Adelmann, R. Ischebeck, J. Snuverink, I. Agapov, R. Kammering, J. Edelen, I. Bazarov, G. Valentino, and J. Wenninger. Opportunities in machine learning for particle accelerators, 2018.

[41] A. L. Edelen et al. Neural networks for modeling and control of particle accelerators. *IEEE Transactions on Nuclear Science*, 63(2):878–897, apr 2016.

[42] B. H. et al. Machine learning applications for orbit and optics correction at the alternating gradient synchrotron. In *Proc. IPAC'23*, number 14 in IPAC'23 - 14th International Particle Accelerator Conference, pages 4409–4412. JACoW Publishing, Geneva, Switzerland, 05 2023.

[43] C. G. et al. Virtual photon pulse characterisation using machine learning methods. In *Proc. IPAC'23*, number 14 in IPAC'23 - 14th International Particle Accelerator Conference, pages 4417–4419. JACoW Publishing, Geneva, Switzerland, 05 2023.

[44] E. Fagotti et al. Upgrade of the heavy ion accelerator complex at infn-lnl. In *Proc. IPAC'23*, number 14 in IPAC'23 - 14th International Particle Accelerator Conference, pages 2169–2172. JACoW Publishing, Geneva, Switzerland, 05 2023.

[45] L. Felsberger et al. Explainable deep learning for fault prognostics in complex systems: A particle accelerator use-case. In *Machine Learning and Knowledge Extraction*, pages 139–158, Cham, 2020.

[46] A. Franciosi and M. Kiskinova. Elettra-sincrotrone trieste: present and future. *The European Physical Journal Plus*, 138(1):79, 2023.

[47] A. Galatà et al. Progresses in the installation of the spes-charge breeder beam line. *Journal of Instrumentation*, 13(12):C12009, dec 2018.

[48] A. Galatà et al. First beams from the 1+ source of the adige injector for the spes project. *Journal of Physics: Conference Series*, 2244(1):012069, apr 2022.

[49] N. Ganganath, C.-T. Cheng, and C. K. Tse. Data clustering with cluster size constraints using a modified k-means algorithm. *International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery*, pages 158–161, 2014.

[50] Y. Gao. Autoencoder-based anomaly detection in the air conditioning system of brookhaven collider-accelerator complex. In *Proc. IPAC'23*, number 14 in IPAC'23 - 14th International Particle Accelerator Conference, pages 4397–4400. JACoW Publishing, Geneva, Switzerland, 05 2023.

[51] M. Goldstein and A. Dengel. Histogram-based outlier score (hbos): A fast unsupervised anomaly detection algorithm. *KI-2012: Poster and Demo Track*, pages 59–63, 2012.

[52] P. Gopalan, V. Sharan, and U. Wieder. Pidforest: anomaly detection via partial identification. *arXiv preprint arXiv:1912.03582*, 2019.

[53] E. Govorkova, E. Puljak, T. Aarrestad, T. James, V. Loncar, M. Pierini, A. A. Pol, N. Ghielmetti, M. Graczyk, S. Summers, J. Ngadiuba, T. Q. Nguyen, J. Duarte, and Z. Wu. Autoencoders on field-programmable gate arrays for real-time, unsupervised new physics detection at 40 MHz at the large hadron collider. *Nature Machine Intelligence*, 4(2):154–161, feb 2022.

[54] F. Grespan, L. Antoniazzi, A. Baldo, C. Baltador, R. Baron, A. Battistello, L. Bellan, T. Bencivenga, P. Bottin, A. Colombo, M. Comunian, D. Conventi, E. Fagotti, L. Ferrari, B. Jones, P. Mereu, C. Mingioni, M. Nenni, E. Nicoletti, A. Palmieri, R. Panizzolo, A. Pisent, and D. Scarpa. ESS Drift Tube Linac Manufacturing, Assembly and Tuning. In *Proc. IPAC'21*, number 12 in International Particle Accelerator Conference, pages 1797–1800. JACoW Publishing, Geneva, Switzerland, 08 2021. https://doi.org/10.18429/JACoW-IPAC2021-TUPAB173.

[55] F. E. Grubbs. Procedures for detecting outlying observations in samples. *Technometrics*, 11(1):1–21, 1969.

[56] A. Hagerty and I. Rubinov. Global ai ethics: a review of the social impacts and ethical implications of artificial intelligence. *arXiv preprint arXiv:1907.07892*, 2019.

[57] S. Hariri, M. Kind, and R. Brunner. Extended isolation forest. *IEEE Transactions on Knowledge and Data Engineering*, 33(4):1479–1489, 2021. cited By 1.

[58] D. M. Hawkins. *Identification of outliers*, volume 11. Springer, 1980.

[59] Z. He et al. Discovering cluster-based local outliers. *Pattern Recognition Letters*, 24(9):1641 – 1650, 2003.

[60] D. J. Hill and B. S. Minsker. Anomaly detection in streaming environmental sensor data: A data-driven modeling approach. *Environmental Modelling & Software*, 25(9):1014–1022, 2010.

[61] G. Hinton, S. Osindero, and Y.-W. Teh. A fast learning algorithm for deep belief nets. *Neural computation*, 18:1527–54, 08 2006.

[62] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[63] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[64] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989.

[65] F. Huhn. Automating beam dump failure detection using computer vision. In *Proc. IPAC'23*, number 14 in IPAC'23 - 14th International Particle Accelerator Conference, pages 4405–4408. JACoW Publishing, Geneva, Switzerland, 05 2023.

[66] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing in science & engineering*, 9(03):90–95, 2007.

[67] T. Idé. Why does subsequence time-series clustering produce sine waves? In *european conference on principles of data mining and knowledge discovery*, pages 211–222. Springer, 2006.

[68] B. Iglewicz and D. C. Hoaglin. *How to detect and handle outliers*, volume 16. Asq Press, 1993.

[69] H. Ishii, M. Nagase, N. Ikeda, Y. Shiba, Y. Shirai, R. Kuroda, and S. Sugawa. A high sensitivity and compact real time gas concentration sensor for semiconductor and electronic device manufacturing process. *ECS Transactions*, 85(13):1399–1405, 2018.

[70] A. Ivanov and I. Agapov. Physics-based deep neural networks for beam dynamics in charged particle accelerators. *Physical review accelerators and beams*, 23(7):074601, 2020.

[71] X. Jia, Y. Di, J. Feng, Q. Yang, H. Dai, and J. Lee. Adaptive virtual metrology for semiconductor chemical mechanical planarization process using gmdh-type polynomial neural networks. *Journal of Process Control*, 62:44–54, 2018.

[72] S.-y. Jiang and Q.-b. An. Clustering-based outlier detection method. In *2008 Fifth International Conference on Fuzzy Systems and Knowledge Discovery*, volume 2, pages 429–433. IEEE, 2008.

[73] Joblib Development Team. Joblib: running python functions as pipeline jobs.

[74] H. John and S. Naaz. Credit card fraud detection using local outlier factor and isolation forest. *Int. J. Comput. Sci. Eng.*, 7(4):1060–1064, 2019.

[75] K. E. Jr. Overview of client tools. Part of the EPICS "Getting Started" Lecture Series, 2004.

[76] V. Kain, S. Hirlander, B. Goddard, F. M. Velotti, G. Z. Della Porta, N. Bruchon, and G. Valentino. Sample-efficient reinforcement learning for cern accelerator control. *Phys. Rev. Accel. Beams*, 23:124801, Dec 2020.

[77] U. Kamath, J. Liu, and J. Whitaker. *Deep learning for NLP and speech recognition*, volume 84. Springer, 2019.

[78] P. Kang, D. Kim, and S. Cho. Semi-supervised support vector regression based on self-training with label uncertainty: An application to virtual metrology in semiconductor manufacturing. *Expert Systems with Applications*, 51:85–106, 2016.

[79] D. Kim, H. Yang, M. Chung, S. Cho, H. Kim, M. Kim, K. Kim, and E. Kim. Squeezed convolutional variational autoencoder for unsupervised anomaly detection in edge device industrial internet of things. In *2018 international conference on information and computer technologies (icict)*, pages 67–71. IEEE, 2018.

[80] M. Kopp, T. Pevny, and M. Holeňa. Anomaly explanation with random forests. *Expert Systems with Applications*, 149:113187, 2020.

[81] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[82] C. Lea et al. Temporal convolutional networks: A unified approach to action segmentation, 2016.

[83] C. Lea, M. D. Flynn, R. Vidal, A. Reiter, and G. D. Hager. Temporal convolutional networks for action segmentation and detection, 2016.

[84] S. learn developers. Outlier detection with local outlier factor (lof).

[85] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1:541–551, 1989.

[86] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[87] S. Levine, C. Finn, T. Darrell, and P. Abbeel. End-to-end training of deep visuomotor policies, 2016.

[88] R. J. Leão, C. R. Baldo, M. L. C. da Costa Reis, and J. L. A. Trabanco. Engineering survey planning for the alignment of a particle accelerator: part ii. design of a reference network and measurement strategy. *Measurement Science and Technology*, 29(3):034007, feb 2018.

[89] S. Li and A. Adelmann. Review of time series forecasting methods and their applications to particle accelerators. 9 2022.

[90] S. Li et al. A novel approach for classification and forecasting of time series in particle accelerators. *Information*, 12(3), 2021.

[91] F. T. Liu, K. M. Ting, and Z.-H. Zhou. Isolation forest. In *2008 eighth ieee international conference on data mining*, pages 413–422. IEEE, 2008.

[92] F. T. Liu, K. M. Ting, and Z.-H. Zhou. On detecting clustered anomalies using sciforest. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 274–290. Springer, 2010.

[93] F. T. Liu, K. M. Ting, and Z.-H. Zhou. Isolation-based anomaly detection. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 6(1):1–39, 2012.

[94] J. Liu, J. Guo, P. Orlik, M. Shibata, D. Nakahara, S. Mii, and M. Takáč. Anomaly detection in manufacturing systems using structured neural networks. In *2018 13th World Congress on Intelligent Control and Automation (WCICA)*, pages 175–180. IEEE, 2018.

[95] S. Liu et al. Time series anomaly detection with adversarial reconstruction networks. *IEEE Transactions on Knowledge and Data Engineering*, 35(4), 2023.

[96] W. Liu, J. He, S. Han, F. Cai, Z. Yang, and N. Zhu. A method for the detection of fake reviews based on temporal features of reviews and comments. *IEEE Engineering Management Review*, 47(4):67–79, 2019.

[97] S. Lloyd. Least squares quantization in pcm. *IEEE transactions on information theory*, 28(2):129–137, 1982.

[98] I. Lobach. Long short-term memory networks for anomaly detection in storage ring power supplies. In *Proc. IPAC'23*, number 14 in IPAC'23 - 14th International Particle Accelerator Conference, pages 4373–4376. JACoW Publishing, Geneva, Switzerland, 05 2023.

[99] J. Lundquist. Virtual diagnostics for longitudinal phase space imaging. In *Proc. IPAC'23*, number 14 in IPAC'23 - 14th International Particle Accelerator Conference, pages 4420–4423. JACoW Publishing, Geneva, Switzerland, 05 2023.

[100] J. MacQueen et al. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, volume 1, pages 281–297. Oakland, CA, USA, 1967.

[101] M. Maggiore, D. Campo, P. Antonini, A. Lombardi, M. Manzolaro, A. Andrighetto, A. Monetti, D. Scarpa, J. Esposito, and L. Silvestrin. Spes: A new cyclotron-based facility for research and applications with high-intensity beams. *Modern Physics Letters A*, 32:1740010, 05 2017.

[102] M. Maggipinto, A. Beghi, and G. A. Susto. A deep convolutional autoencoder-based approach for anomaly detection with industrial, non-images, 2-dimensional data: A semiconductor manufacturing case study. *IEEE Transactions on Automation Science and Engineering*, 2022.

[103] K. L. Malanchev, A. A. Volnova, M. V. Kornilov, M. V. Pruzhinskaya, E. E. Ishida, F. Mondon, and V. S. Korolev. Use of machine learning for anomaly detection problem in large astronomical databases. In *DAMDID/RCDL*, pages 205–216, 2019.

[104] D. Marcato. Progettazione e sviluppo di un nuovo software per il controllo della radiofrequenza dell'acceleratore alpi., 2017.

[105] D. Marcato, G. Arena, M. Bellato, D. Bortolato, F. Gelain, G. Lilli, V. Martinelli, E. Munaron, M. Roetta, and G. Savarese. Pysmlib: A Python Finite State Machine Library for EPICS. *JACoW*, ICALEPCS2021:TUBL05, 2022.

[106] D. Marcato, G. Arena, D. Bortolato, F. Gelain, V. Martinelli, E. Munaron, M. Roetta, G. Savarese, and G. A. Susto. Machine learning-based anomaly detection for particle accelerators. In *2021 IEEE Conference on Control Technology and Applications (CCTA)*, pages 240–246, 2021.

[107] D. Marcato, D. Bortolato, V. Martinelli, G. Savarese, and G. A. Susto. Time-series deep learning anomaly detection for particle accelerators. In *Proceeding of the 22nd World Congress of the International Federation of Automatic Control (IFAC'23)*, 2023.

[108] D. Marcato et al. Demonstration of beam emittance optimization using reinforcement learning. In *Proc. IPAC'23*, number 14 in IPAC'23 - 14th International Particle Accelerator Conference, pages 2838–2841. JACoW Publishing, Geneva, Switzerland, 05 2023.

[109] D. Marcato et al. Upgrade of the alpi low and medium beta rf control system. In *Proc. IPAC'23*, number 14 in IPAC'23 - 14th International Particle Accelerator Conference, pages 4154–4157. JACoW Publishing, Geneva, Switzerland, 05 2023.

[110] V. Martinelli, L. Bellan, D. Bortolato, M. Comunian, E. Fagotti, P. Francescon, A. Galatà, D. Marcato, and G. Savarese. BOLINA, a Suite for High Level Beam Optimization: First Experimental Results on the Adige Injection Beamline of SPES. In *Proc. IPAC'22*, number 13 in International Particle Accelerator Conference, pages 933–936. JACoW Publishing, Geneva, Switzerland, 07 2022.

[111] B. Maschler and M. Weyrich. Deep transfer learning for industrial automation: A review and discussion of new techniques for data-driven machine learning. *IEEE Industrial Electronics Magazine*, 15(2):65–75, 2021.

[112] mc.ai. My notes on neural networks.

[113] W. Mcculloch and W. Pitts. A logical calculus of ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:127–147, 1943.

[114] S. McLoone, A. Johnston, and G. A. Susto. A methodology for efficient dynamic spatial sampling and reconstruction of wafer profiles. *IEEE TRANSACTIONS ON AUTOMATION SCIENCE AND ENGINEERING*, 15(14):1692–1703, October 2018.

[115] L. Meneghetti, M. Terzi, S. Del Favero, G. A. Susto, and C. Cobelli. Data-driven anomaly recognition for unsupervised model-free fault detection in artificial pancreas. *IEEE Transactions on Control Systems Technology*, 28(1):33–47, 2018.

[116] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. Asynchronous methods for deep reinforcement learning, 2016.

[117] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

[118] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

[119] J. Molleda, R. Usamentiaga, A. F. Millara, D. F. García, P. Manso, C. M. Suárez, and I. García. A profile measurement system for rail quality assessment during manufacturing. *IEEE Transactions on Industry Applications*, 52(3):2684–2692, 2016.

[120] J. Moody and M. Saffell. Learning to trade via direct reinforcement. *IEEE Transactions on Neural Networks*, 12(4):875–889, 2001.

[121] Z. Obermeyer, B. Powers, C. Vogeli, and S. Mullainathan. Dissecting racial bias in an algorithm used to manage the health of populations. *Science*, 366(6464):447–453, 2019.

[122] S. Oehmcke, O. Zielinski, and O. Kramer. Event detection in marine time series data. In *KI 2015: Advances in Artificial Intelligence: 38th Annual German Conference on AI, Dresden, Germany, September 21-25, 2015, Proceedings 38*, pages 279–286. Springer, 2015.

[123] M. Paganini, L. de Oliveira, and B. Nachman. CaloGAN : Simulating 3D high energy particle showers in multilayer electromagnetic calorimeters with generative adversarial networks. *Phys. Rev. D*, 97(1):014021, 2018.

[124] T. Pan, B. Sheng, D. S. Wong, and S. Jang. A virtual metrology system for predicting end-of-line electrical properties using a mancova model with tools clustering. *IEEE Transactions on Industrial Informatics*, 7(2):187–195, 2011.

[125] R. Pascanu, T. Mikolov, and Y. Bengio. On the difficulty of training recurrent neural networks, 2013.

[126] T. Pevny. Loda: Lightweight on-line detector of anomalies. *Machine Learning*, 102(2):275–304, 2016.

[127] A. Pisent, G. Bisoffi, A. Lombardi, V. Andreev, G. Bassato, G. Bezzon, S. Canella, F. Cervellera, F. Chiurlotto, M. Comunian, et al. The new lnl injector piave, based on a superconducting rfq. In *Proceedings of the sixth European Particle accelerator Conference Stockholm, Sweden*, pages 758–760, 1998.

[128] A. M. Porcellato. Corso nuovi operatori, 05 2013.

[129] P. Prakash, B. Honari, A. Johnston, and S. McLoone. Optimal wafer site selection using forward selection component analysis. *Proc. Adv. Semiconductor Manuf. Conf. (ASMC)*, pages 91—-96, May 2012.

[130] Psycopg2 Development Team. Psycopg2.

[131] L. Puggini and S. McLoone. Forward selection component analysis: Algorithms and applications. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(12):2395–2408, 2017.

[132] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021.

[133] A. Raghu, M. Komorowski, L. A. Celi, P. Szolovits, and M. Ghassemi. Continuous state-space models for optimal sepsis treatment-a deep reinforcement learning approach. *Proceedings of Machine Learning in Healthcare*, 2017.

[134] D. Ramotsoela, A. Abu-Mahfouz, and G. Hancke. A survey of anomaly detection in industrial wireless sensor networks with critical water system infrastructure as a case study. *Sensors*, 18(8):2491, 2018.

[135] J. Reback et al. pandas-dev/pandas: Pandas 1.2.0, 2020.

[136] V. Rigato. Multidisciplinary physics with mev ion beams at the laboratori nazionali di legnaro using the cn and an2000 accelerators. Technical report, 2022.

[137] J. V. Ringwood, S. Lynn, G. Bacelli, B. Ma, E. Ragnoli, and S. McLoone. Estimation and control in semiconductor etch: Practice and possibilities. *IEEE Transactions on Semiconductor Manufacturing*, 23(1):87–98, 2010.

[138] RL4AA-Collaboration. Collaboration on reinforcement learning for autonomous accelerators `https://rl4aa.github.io/`.

[139] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408, 1958.

[140] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, Oct. 1986.

[141] S. Saadat. Model of a dynamic orbit correction system based on neural network in cls. In *Proc. IPAC'23*, number 14 in IPAC'23 - 14th International Particle Accelerator Conference, pages 2819–2822. JACoW Publishing, Geneva, Switzerland, 05 2023.

[142] G. Savarese, L. Antoniazzi, D. Bortolato, A. Conte, F. Gelain, D. Marcato, and C. Roncolato. Vacuum Control System Upgrade for ALPI Accelerator. In *Proc. IPAC'22*, number 13 in International Particle Accelerator Conference, pages 744–746. JACoW Publishing, Geneva, Switzerland, 07 2022.

[143] G. Savarese, G. Arena, D. Bortolato, F. Gelain, D. Marcato, V. Martinelli, E. Munaron, and M. Roetta. Design and Development of the New Diagnostics Control System for the SPES Project at INFN-LNL. In *Proc. ICALEPCS'21*, number 18 in International Conference on Accelerator and Large Experimental Physics Control Systems, pages 428–432. JACoW Publishing, Geneva, Switzerland, 03 2022.

[144] G. Savarese et al. First installation of the upgraded vacuum control system for alpi accelerator. In *Proc. IPAC'23*, number 14 in IPAC'23 - 14th International Particle Accelerator Conference, pages 840–843. JACoW Publishing, Geneva, Switzerland, 05 2023.

[145] A. Scheinker et al. Applying artificial intelligence to accelerators. *Proceedings of the 9th Int. Particle Accelerator Conf.*, IPAC2018:Canada, 2018.

[146] M. Schenk, E. F. Combarro, M. Grossi, V. Kain, K. S. B. Li, M.-M. Popa, and S. Vallecorsa. Hybrid actor-critic algorithm for quantum reinforcement learning at cern beam lines, 2022.

[147] R. M. Schmidt. Recurrent neural networks (rnns): A gentle introduction and overview, 2019.

[148] J. Schulman et al. Proximal policy optimization algorithms, 2017.

[149] R. Sharankova. Time-drift aware rf optimization with machine learning techniques. In *Proc. IPAC'23*, number 14 in IPAC'23 - 14th International Particle Accelerator Conference, pages 32–35. JACoW Publishing, Geneva, Switzerland, 05 2023.

[150] A. Sherman. Sequential chemical vapor deposition, 6 1999. US Patent 5,916,365.

[151] A. Shukla, S. N. Victoria, and R. Manivannan. A review on chemical mechanical planarization of barrier layer metals. *Key Engineering Materials*, 882:171–180, 2021.

[152] D. Silver, A. Huang, C. Maddison, A. Guez, L. Sifre, G. Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529:484–489, 01 2016.

[153] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

[154] A. Solopova et al. Srf cavity fault classification using machine learning at cebaf. *Proceedings of the 10th Int. Particle Accelerator Conf.*, IPAC2019:Australia, 2019.

[155] L. Stojanovic, M. Dinic, N. Stojanovic, and A. Stojadinovic. Big-data-driven anomaly detection in industry (4.0): An approach and a case study. In *2016 IEEE international conference on big data (big data)*, pages 1647–1652. IEEE, 2016.

[156] A. Sulc et al. A Data-Driven Anomaly Detection on SRF Cavities at the European XFEL. In *Proceedings of IPAC 2022*. JACoW, Jun 2022.

[157] H. Sun, Q. He, K. Liao, T. Sellis, L. Guo, X. Zhang, J. Shen, and F. Chen. Fast anomaly detection in multiple multi-dimensional data streams. In *2019 IEEE International Conference on Big Data (Big Data)*, pages 1218–1223. IEEE, 2019.

[158] G. A. Susto, M. Maggipinto, F. Zocco, and S. McLoone. Induced start dynamic sampling for wafer metrology optimization. *IEEE TRANSACTIONS ON AUTOMATION SCIENCE AND ENGINEERING*, 17(1):418–432, JANUARY 2020.

[159] G. A. Susto, D. Marcato, M. Maggipinto, M. Donà, and S. McLoone. On temporally bounded spatial dynamic sampling for metrology optimization. *submitted to IEEE Transactions on Automation Science and Engineering*, 2023.

[160] G. A. Susto, A. Schirru, S. Pampuri, S. McLoone, and A. Beghi. Machine learning for predictive maintenance: A multiple classifier approach. *IEEE Transactions on Industrial Informatics*, 11(3):812–820, 2015.

[161] M. Szczepanski. Economic impacts of artificial intelligence (ai). 2019.

[162] S. C. Tan, K. M. Ting, and T. F. Liu. Fast anomaly detection for streaming data. In *Twenty-Second International Joint Conference on Artificial Intelligence*, 2011.

[163] K. M. Ting, G.-T. Zhou, F. T. Liu, and J. S. C. Tan. Mass estimation and its applications. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 989–998, 2010.

[164] J. F. Torres, D. Hadjout, A. Sebaa, F. Martínez-Álvarez, and A. Troncoso. Deep learning for time series forecasting: A survey. *Big Data*, 9(1):3–21, 2021. PMID: 33275484.

[165] Y.-L. Tsou, H.-M. Chu, C. Li, and S.-W. Yang. Robust distributed anomaly detection using optimal weighted one-class random forests. In *2018 IEEE International Conference on Data Mining (ICDM)*, pages 1272–1277. IEEE, 2018.

[166] D. Uriot and N. Pichoff. Status of TraceWin Code. In *Proc. 6th International Particle Accelerator Conference (IPAC'15), Richmond, VA, USA, May 3-8, 2015*, number 6 in International Particle Accelerator Conference, pages 92–94, Geneva, Switzerland, June 2015. JACoW. https://doi.org/10.18429/JACoW-IPAC2015-MOPWA008.

[167] D. Uriot and N. Pichoff. Tracewin documentation, 04 2019.

[168] M. Weber, S. Klug, E. Sax, and B. Zimmer. Embedded hybrid anomaly detection for automotive can communication. In *9th European Congress on Embedded Real Time Software and Systems (ERTS 2018)*, 2018.

[169] P. J. Werbos. Applications of advances in nonlinear sensitivity analysis. In *Proceedings of the 10th IFIP Conference, 31.8 - 4.9, NYC*, pages 762–770, 1981.

[170] G. White et al. The EPICS Software Framework Moves from Controls to Physics. In *Proc. 10th International Particle Accelerator Conference (IPAC'19)*, pages 1216–1218, Geneva, Switzerland, Jun. 2019. JACoW Publishing.

[171] Wikipedia. Isolation forest.

[172] J. Williams. Ion implantation of semiconductors. *Materials Science and Engineering: A*, 253(1):8–15, 1998.

[173] R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8:229–256, 1992.

[174] C. Xu. Beam trajectory control with lattice-agnostic reinforcement learning. In *Proc. IPAC'23*, number 14 in IPAC'23 - 14th International Particle Accelerator Conference, pages 4436–4439. JACoW Publishing, Geneva, Switzerland, 05 2023.

[175] C. Xu, R. Roussel, and A. Edelen. Neural network prior mean for particle accelerator injector tuning, 2022.

[176] X. Yang. Accurate prediction of mega-electron-volt electron beam properties from ued using machine learning. In *Proc. IPAC'23*, number 14 in IPAC'23 - 14th International Particle Accelerator Conference, pages 4401–4404. JACoW Publishing, Geneva, Switzerland, 05 2023.

[177] Yaw-Jen Chang, Yuan Kang, Chih-Liang Hsu, Chi-Tim Chang, and Tat Yan Chan. Virtual metrology technique for semiconductor manufacturing. In *The 2006 IEEE International Joint Conference on Neural Network Proceedings*, pages 5289–5293, 2006.

[178] X. Zhang, W. Dou, Q. He, R. Zhou, C. Leckie, R. Kotagiri, and Z. Salcic. Lshiforest: A generic framework for fast tree isolation based ensemble anomaly analysis. In *2017 IEEE 33rd International Conference on Data Engineering (ICDE)*, pages 983–994. IEEE, 2017.

[179] Y. Zhao et al. Pyod: A python toolbox for scalable outlier detection. *Journal of Machine Learning Research*, 20(96):1–7, 2019.

[180] X. Zhou, Y. Hu, W. Liang, J. Ma, and Q. Jin. Variational lstm enhanced anomaly detection for industrial big data. *IEEE Transactions on Industrial Informatics*, 17(5):3469–3477, 2021.

[181] F. Zocco, M. Maggipinto, G. A. Susto, and S. McLoone. Lazy fsca for unsupervised variable selection. *Engineering Applications of Artificial Intelligence*, 124:106624, 2023.

[182] T. Zonta, C. A. da Costa, R. da Rosa Righi, M. J. de Lima, E. S. da Trindade, and G. P. Li. Predictive maintenance in the industry 4.0: A systematic literature review. *Computers & Industrial Engineering*, 150:106889, 2020.

Acknowledgements

I would like to thank my supervisor, Prof. Gian Antonio Susto, for his guidance and support during these years. I would also like to thank my colleagues at INFN, especially D. Bortolato, for the opportunity to pursue this Ph.D..

Finally, I would like to thank my wife Alice for her support and patience.