ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

DOTTORATO DI RICERCA IN

DATA SCIENCE AND COMPUTATION

Ciclo 37

Settore Concorsuale: 02/A1 - FISICA SPERIMENTALE DELLE INTERAZIONI FONDAMENTALI

Settore Scientifico Disciplinare: FIS/01 FISICA SPERIMENTALE

A MACHINE LEARNING-DRIVEN FRAMEWORK FOR ERROR DETECTION AND OPERATIONAL MONITORING OF ATLAS JOB REPORTS INSIDE INFN-CNAF BIG DATA PLATFORM

Presentata da: Giacomo Levrini

Coordinatore Dottorato

Daniele Bonacorsi

Supervisore

Alessandro Gabrielli

Co-supervisore

Antonio Zoccoli

Esame finale anno 2026

To the most splendid person
I will meet in a lifetime,
The most precious of all,
To my sweet, Valentine.

Had to take your hand, and feel your breath
For fear this someday will be over
I pull you close, so much to lose
Knowing that nothing lasts forever
I didn't care, before you were here
I danced in laughter, with the ever after,
but all things change, let this remain.
[Sirens - Pearl Jam]

Abstract

Modern data centers provide the IT infrastructure required to efficiently deliver resources, services, monitoring systems, and data to users efficiently. Over time, they have evolved to accommodate increasing resources demands and ever-growing diverse use cases.

At INFN's CNAF, a Big Data Platform (BDP) has been developed to collect and index log reports from CNAF facilities. This ongoing project serves the Italian groups participating in high-energy physics experiments. Within this framework, for the first time, a dedicated data pipeline was implemented for the ATLAS experiment, sourcing input from the ATLAS Distributed Computing system (PanDA) and collecting reports of computational jobs executed on the INFN Tier-1 farm, thus establishing a first cooperation between the two infrastructures.

The received operational logs were stored in the BDP, forming a comprehensive database of job-related information. Once a sufficiently large and consistent set of logs had been accumulated, the next step was to develop a machine learning framework, embedding two different machine learning models, the first able to perform binary classification of job outcomes (successful or failed), and the second aimed at categorizing the various types of errors which occurred during the execution of failed jobs.

Contents

Introduction

Every second, the Large Hadron Collider (LHC) at CERN produces billions of proton-proton collisions, from which billions of high energy secondary particles are produced. The energy and the information brought by the resulting particles are captured by four major experiments installed along the accelerator ring. Among them, the ATLAS experiment stands as one of the largest and most complex multipurpose particle detectors ever built. Its mission is to record, reconstruct and interpret traces and energy signatures left by the particles crossing the layers of the experiment, allowing a deep investigation into the fundamental laws of nature.

The immense volume of data produced by ATLAS is handled by a huge trigger and data acquisition (TDAQ) system, an online active system which allows selection of specific "events" produced in the collisions. This live selection also allows the data collected to be greatly reduced, despite reaching a huge production of raw data per day, later stored offline in a large storage system (Tier-0). All the extracted data inside the storage servers cannot be handled by a single computing centre.

In fact, a global network of computing centres was created to provide the sufficient computational resources to perform later analysis on raw data, the Worldwide LHC Computing Grid (WLCG). Organized in 'tiers', starting from the Tier-0, it spreads globally into several Tier-1 sites. Among these, the INFN-CNAF in Bologna plays a key role as Italy's Tier-1 facility, ensuring that the flow of data continues smoothly from the detector to physicists around the world for analysis. It is within this context of large-scale data management and distributed computing that the work presented in this thesis takes shape.

To guide the reader through this journey from experiment to data analysis, the manuscript is divided into four chapters:

The first chapter is an overall introduction to the Large Hadron Collider (LHC) and the ATLAS experiment, which is one of the four largest experiments mounted around the LHC ring.

The second chapter is a brief introduction to the INFN, the Italian National Institute of Nuclear Physics, its Bologna Division and the INFN-CNAF national institute, the National Centre for Research and Development on Information and Communication Technologies, its role as a Tier-1 computing centre for the World LHC Computing Grid (WLCG) and their Big Data Processing Infrastructure (BDP).

The third chapter explains in detail the design and implementation of a data pipeline started from the ATLAS' PanDA workload management system. Starting from the single configuration of the various nodes of the platform,

it shows how data flows through the system reaching at the end the data monitoring and management system of the BDP.

The fourth chapter describes the workflow adopted to prepare the data stored in the BDP, the dataset used in the preparation of the machine learning models and eventually the results obtained in the training, testing and evaluation of the various models developed.

Chapter 1

LHC and the ATLAS experiment

1.1 The Large Hadron Collider

The Large Hadron Collider (LHC) is currently the biggest circular hadron collider in the world, with a circumference of 27 km; its purpose is to accelerate protons and heavy ions which will collide and consecutively allow the study of the high-energy particle interactions. It is situated nearby the city of Geneva, between the French and the Swiss border, 100 m underground. It is located in the old Large Electron-Positron (LEP) collider location, and a collaboration of 22 nations known as CERN (Conseil Européen pour la Recherche Nucléaire) works together at high-energy physic experiment. In Figure 1.1 is shown the LHC underground complex.

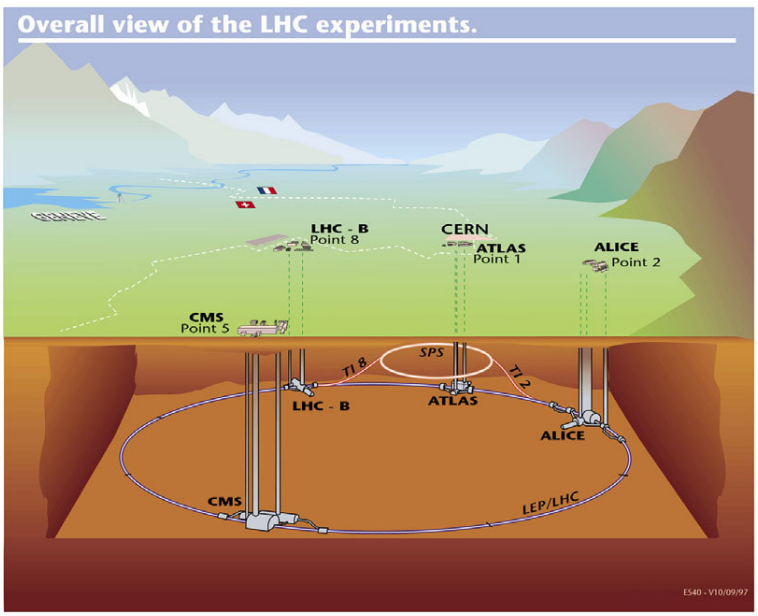


Figure 1.1: *The LHC underground complex below the French-Swiss border.*

Unlike previous particle-antiparticle colliders, in which both beams share the same phase space in a single ring, the LHC machine is based on a proton-proton

collision. The two beams of protons travel in two counter-rotating rings, crossing in eight different locations (so-called interaction points) along the ring.

The two-ring superconducting collider is composed of a particle source and a series of accelerators which together allow proton beams collisions with a total center of mass energy of 14 TeV, that will be reached in the next Run upgrade. Currently, the maximum energy reached is 13 TeV, 6.5 TeV for each proton. The proton source is a hydrogen container where the particles are split in their fundamental components, a proton and an electron; then the protons are collected and sent to the first stage of the acceleration of LHC, LINAC2, a linear accelerator where the protons reach the energy of 50 MeV. Then the protons are accelerated by 3 synchrotron accelerators before reaching the last accelerator (LHC): PSB (Proton Synchrotron Booster) where the protons reach 1.4 GeV, PS (Proton Synchrotron) where the protons reach 25 GeV, and SPS (Super Proton Synchrotron) which accelerates protons at 450 GeV. At the end, LHC, with radio frequency cavities working at 400 MHz, pushes the proton beams, at 6.5 TeV each, in the beam pipes.

The protons' acceleration chain up to the LHC is shown in Figure 1.2.

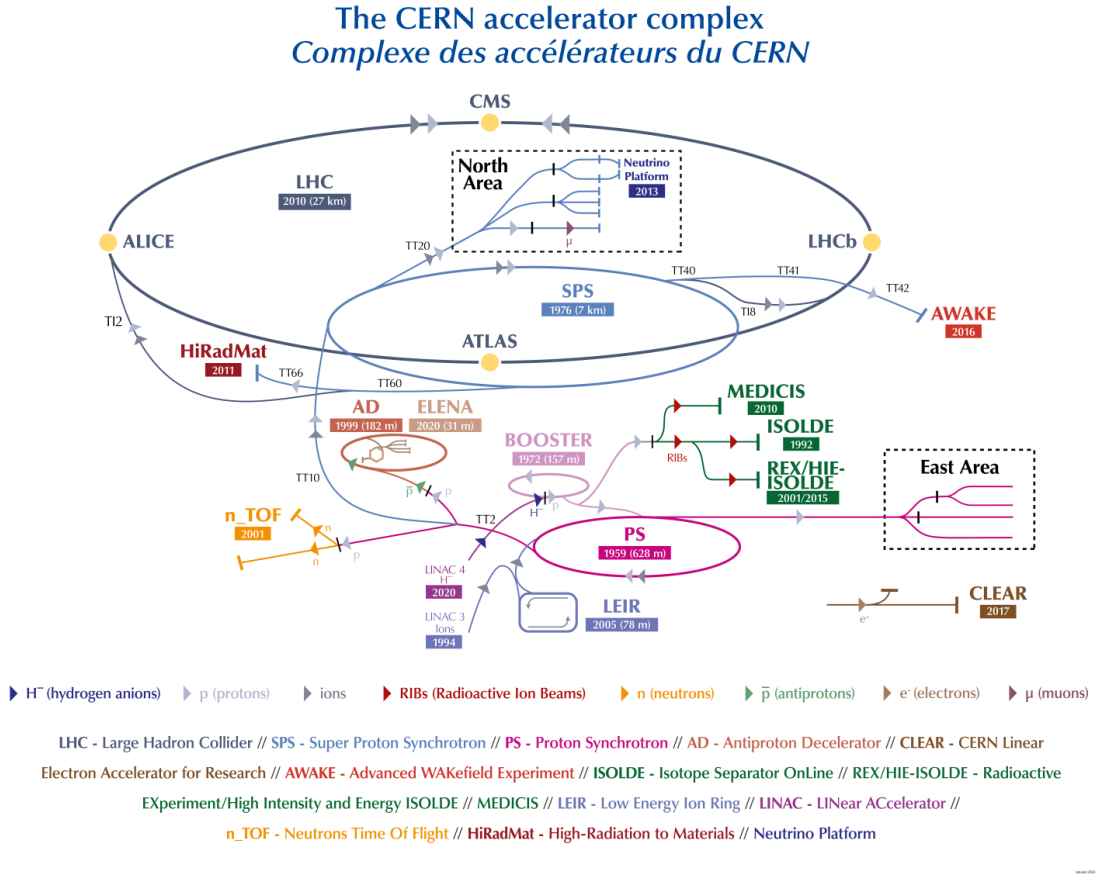


Figure 1.2: *The LHC accelerators complex. The figure also shows the injection chain of the beams: it starts with the linear accelerator LINAC, followed by the Proton Synchrotron Booster (PSB). The Proton Synchrotron (PS) further increases the beam energy and feeds the particle packets to the Super Proton Synchrotron (SPS). After the SPS the beams are injected directly into the LHC.*

The proton beams are kept in the 2 beam pipes and far from its walls by a complex system of magnets, precisely with 1232 superconducting dipole magnets, which keep the beam in the beam pipes, and with 392 quadruple magnets for the beams focusing. After the acceleration, the beams collide in the 4 collision points where the 4 experiments now active at CERN are situated.

The final beam consists of 2808 bunches of protons, formed by the radio frequency cavities, with $\approx 1.2 \times 10^{11}$ protons per bunch. The collisions allow reaching an instantaneous luminosity with a peak of $1034\,\mathrm{cm}^{-2}\,\mathrm{s}^{-1}$.

1.1.1 The LHC experiments

The bunch collisions occur in correspondence of the 4 experiments currently active at LHC, ATLAS, CMS, ALICE, LHCb; each one of them has a specific motivation and task:

- ATLAS (*A Toroidal Large hadron collider ApparatuS*) is a multipurpose detector. It is composed of a series of sub-detectors which surround the beam pipe: an Inner Detector (ID) for the particle tracking, a solenoid magnet to measure the momentum of the particles, an electromagnetic calorimeter to measure the energy of electromagnetic interactive particles, a hadronic calorimeter which measures the energy of particles which interact by strong interaction, a muon spectrometer to detect muons tracks and momenta;

- CMS (*Compact Muon Solenoid*) is another multipurpose detector built with different technologies but with the same layout and purposes of ATLAS;

- LHCb (*Large Hadron Collider beauty*) is a specific apparatus for proton-proton collisions. Its purpose is to investigate the physics of the quark b, in particular the CP-violation of the B meson. Unlike the other experiments, it has a fixed target morphology: the apparatus is composed of a tracker around the region of proton interaction followed by a ring imaging Cherenkov detector (RICH), a series of other trackers, another RICH, an electromagnetic calorimeter, a hadronic calorimeter, and at the end a muon detector;

- ALICE (*A Large Ion Collider Experiment*) is an apparatus built to study Pb-Pb collision, where each couple of colliding nucleons reach an energy of 2.76 TeV. It studies concerns mainly QCD theory, in particular studying the condition of high temperature and high energy density. It is composed of 18 detectors surrounding the collision point, including: a time projection chamber (TPC), a transition radiation chamber, a "time of flight" detector, electromagnetic and hadron calorimeters, and a muon spectrometer.

1.2 The ATLAS detector

The multipurpose detector ATLAS is 46 m long and with a diameter of 26 m. The goals of this apparatus are the conformation and improvement of the values of the Standard Model (SM) and the study of the beyond SM theories. The experiment involves over 3000 physicists and researchers from over 175 institutes.

The general multipurpose design of the ATLAS detector and its detecting features have put it on forefront in the research for the study of the Standard Model. For instance, the detection of the Higgs boson ($m_{\mathrm{H}} = 124.98 \pm 0.98\,\mathrm{GeV}$) opened a new chapter in the particle physics research. The study of all its decay channels, in particular the ones involving the b quark, brought and improvement in the knowledge of the characteristics of this boson. Another relevant area of studies at ATLAS regards the top quark: the improvement of the knowledge about it leads to reach the Standard Model limits, finding new decay processes at lower cross-sections and hints of new physics. At last but not least, the investigation of new physics, the Super-symmetric Model, is performed at the ATLAS experiment . The latter, at this point, has not been proved yet to be the natural extension of the Standard Model, together with other research like the extra-dimension and many other theories of new physics, that at the moment did not give any evidences.

1.2.1 ATLAS coordinate system

In the ATLAS experiment, the interaction point is considered to be in the coordinate $(0, 0, 0)$ in a 3-D Cartesian reference frame, with coordinates (x, y, z). The coordinate z is along the beam line, and the x-y plane is perpendicular to the beam line with the positive x-axis points to the centre of the LHC ring and the positive y-axis points upward to the sky (see Figure 1.3).

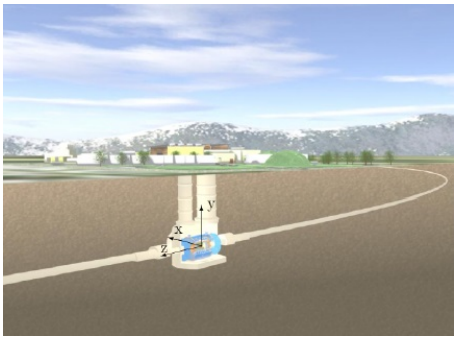


Figure 1.3: *The ATLAS coordinate "real" frame on the left, and the schematics of the frame and the transverse plane on the right.*

In the x-y transverse plane the position coordinates of a particle can be described in polar coordinates: R is the distance from the centre of the plane and ϕ is the azimuthal angle in the transverse plane, from the x-axis around the z-axis. The momentum measured in the transverse plane is called transverse momentum $p_{\mathrm{T}} = \sqrt{p_x^2 + p_y^2}$. The polar angle θ is in the z-y plane from the positive z-axis. Using this coordinate representation it is possible to define Lorentz-invariant variables, such as

$$y = \frac{1}{2} \ln \left[\frac{E + p_z}{E - p_z} \right], \tag{1.1}$$

which is the rapidity, a Lorentz-invariant for transformations along the z-axis. For relativistic particles (i.e., for particles traveling close to the speed of light), this

variable can be reduced to

$$\eta = -\ln \tan\left(\frac{\theta}{2}\right), \tag{1.2}$$

which is the pseudorapidity, a function of the angular position of the particle, not considering its nature and energy. It is therefore possible to measure the position of a particle in the new Lorentz-invariant coordinate system, composed by (η, ϕ, z), and it is possible to represent a difference in distance, ΔR, between two particles by the formulation

$$\Delta R = \sqrt{(\Delta\eta^2 + \Delta\phi^2)}, \tag{1.3}$$

where $\Delta\eta$ and $\Delta\phi$ are respectively difference in pseudorapidity and azimuthal angle of the particles.

The pseudorapidity ranges from 0, alongside the y-axis, to infinity, alongside the z-axis. The high energies involved in the proton collisions makes the partons of the protons collide (each parton carries a fraction of the proton energy and momentum). For the analyses of the collisions many quantities are used: the transverse momentum p_{T}, the transverse energy E_{T} and the transverse missing energy $E_{\mathrm{T}}^{\mathrm{miss}}$.

1.3 Detector composition

ATLAS apparatus is composed by different detectors, where each of them covers an η range and has a specific purpose. In Figure 1.4 is shown the latest proposal for the ATLAS layout for the High Luminosity phase.

Immediately after the proton-proton collision, the first detector which the produced particles traverse is the Inner Detector (ID), a tracking apparatus formed by 3 detectors surrounding the beam line and covering the region $|\eta| < 2.5$. Here, the particle tracking precision is very high, with an intrinsic accuracy varying approximately from 10 to 100 µm. Right after the ID, there is the first stage of the Magnet System, a central solenoid which provides a 2 T magnetic field and allows momentum measurements of p_{T} with a resolution of $\sigma_{p_{\mathrm{T}}}/p_{\mathrm{T}} = (4.83 \pm 0.16) \times 10^{-4}\,\mathrm{GeV}^{-1} \times p_{\mathrm{T}}$.

In this detector section, only charged particles can be detected. After this stage, the following detector is the electromagnetic calorimeter, where electrons, positrons and photons are detected. Through electromagnetic interaction, electromagnetic showers are produced for each mentioned particle, inside which other particles are produced and thereafter detected. The energy and the track of the particles in the shower can be measured. Furthermore, the more energetic particles survived from the previous steps encounter the hadronic calorimeter. Thanks to the strong interaction, the hadrons, formed in the proton-proton collisions or from secondary decays, develop hadronic showers. They are detected with an energy resolution which ranges from 0.13 to 0.06 when jet transverse momentum p_{T} increases. The calorimeter stage covers an angle up to $|\eta| = 4.9$.

Eventually, only the particles with a very low cross-section survived, mainly muons and neutrinos. The latter can not be detected directly by ATLAS, thus they are studied with the missing energy technique. The muons instead can be detected

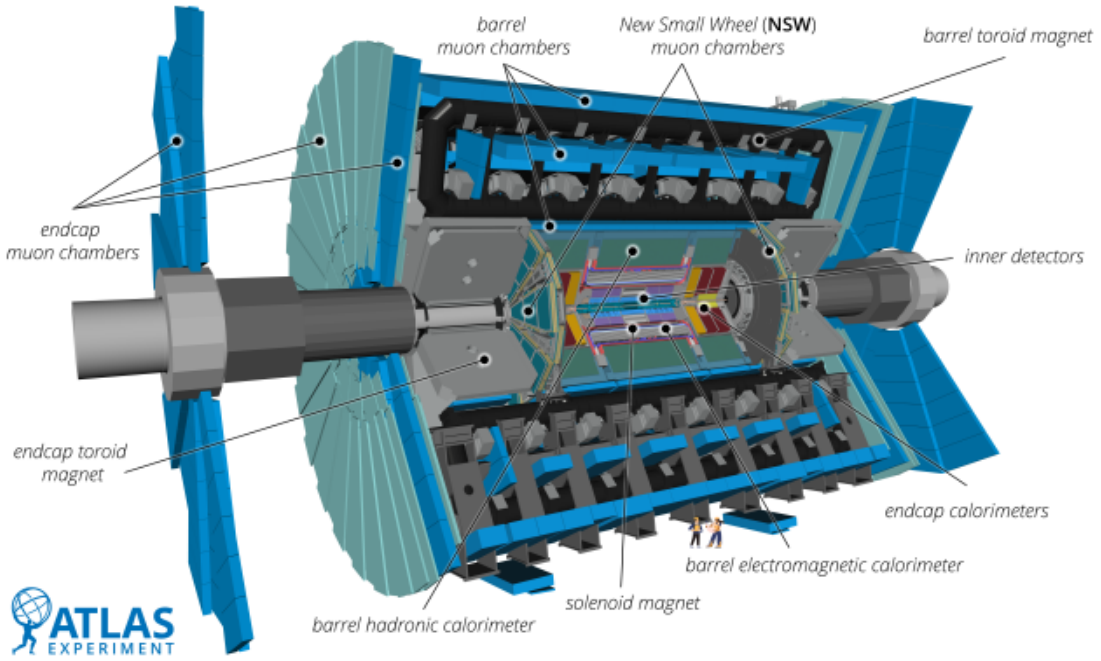


Figure 1.4: *The ATLAS experiment layout section designed for the High Luminosity phase, with all its components. The system is built with cylindrical symmetry with the beam pipe as the axis. From the innermost (closer to the beam pipe) to the outermost, all the sub-detectors of ATLAS are shown.*

in the muon spectrometer, a very large detector, composed of 2 tracking chambers and 2 trigger chambers. It has a coverage $|\eta| < 2.7$.

1.3.1 Inner Detector

The Inner Detector is a 6.2 m long apparatus with a diameter of 2.1 m, placed around the beam line, with a coverage of $|\eta| < 2.5$. Built for the early tracking stage of ATLAS, it is composed the Pixel Detector, the Semiconductor Tracking and the Transition Radiation Tracker. The technologies and components of all these detectors must be the most radiation hard possible, since they are the closest to the beam line.

The Pixel Detector (PD) is the second tracker encountered by a particle produced in the proton-proton interaction. It is a silicon based detector which uses the pixel technology; it has the highest granularity in all ATLAS, and it consists of 4 barrels: Insertable B-Layer (see later), B-Layer, Layer1, Layer 2 and 3 disks for each side.

The Semi-Conductor Tracker (SCT) is 4 layers a Silicon microstrip detector. Each layer is formed by modules composed by two microstrip detector bounded together and glued with a 40 mrad angle of their planes, layout used to obtain a better z-measurement. The two microstrip detectors of a single module are glued with the angle between the two microstrip shifted of 90°. In the barrel region the plane of the microstrip detector is parallel to the beam line, while in the end-cap region is perpendicular.

The Transition Radiation Tracker (TRT) is the largest track detector of ID

and surrounds the previous two. It consists of a large number, about 5×10^4, of straw tubes, that are cylindrical tubes with one positive wire in their inside and the internal wall at negative voltage. The straws all together contribute to the measurement of the particle momentum thanks to the high number of hits, and each one is filled with a mixture of Xenon (70%), CO_2 (27%), and O_2 (3%). In the barrel region the tubes are parallel to the beam line, while in the end-cap region are perpendicular.

All of them are placed both in the barrel and in the end-cap region of ID. The Figure 1.5 shows the ID configuration, dimension and coverage, while the general characteristics of each subcomponent are described in the Table 1.1.

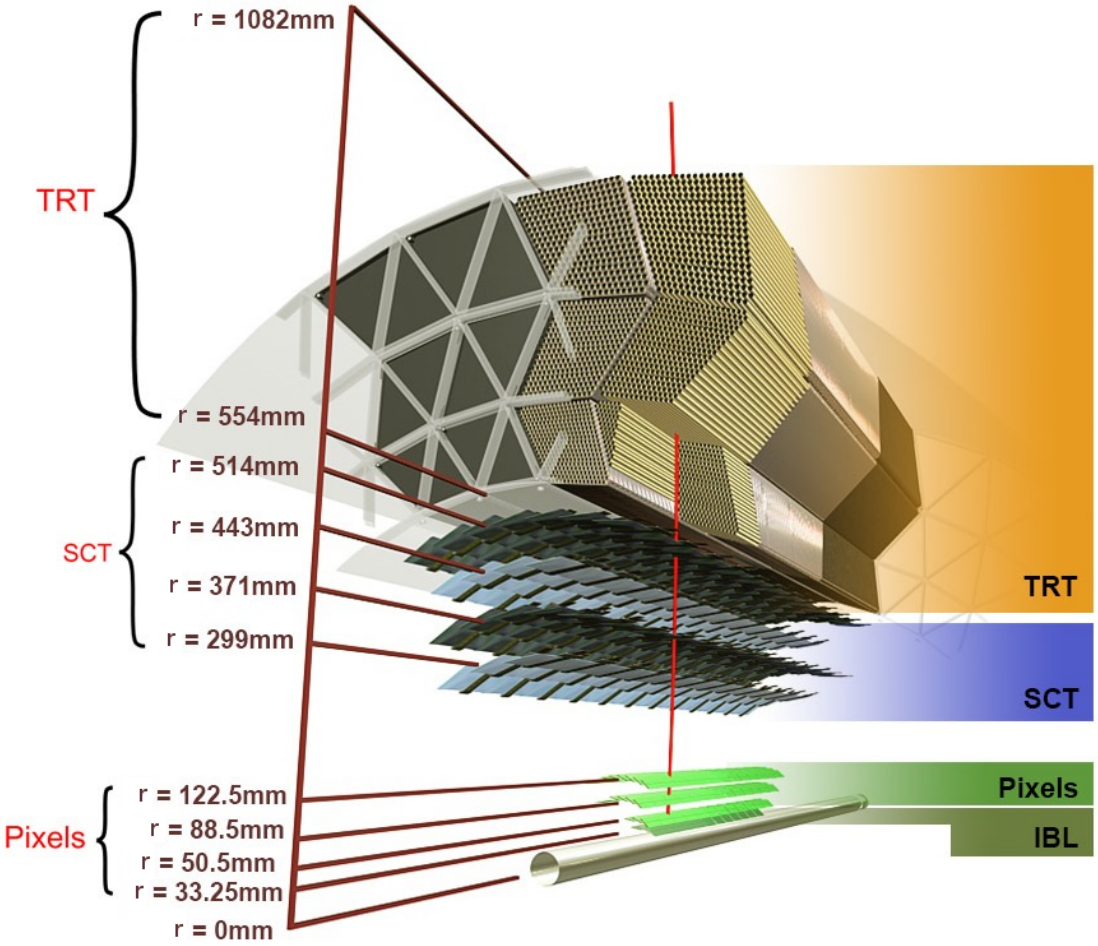


Figure 1.5: *Section of the ID subdivided detectors system, with the relative distances and distributions from the beam line. From the closest one to the further one the subsystems are: the Pixel Detector, the Semi-Conductor Tracker and the Transition Radiation Tracker.*

All these specifics are needed to provide a high precision measurement of the particles track, which may be reconstructed by two different algorithms:

- the inside-out algorithm, where primary tracks of charged particles born from primary interactions are reconstructed using 3 seeds in the silicon

Table 1.1: *Main characteristics of the ID's detector.*

Detector	Hits Tracks	Element Size	Hits Resolution (µm)
PD, $\|\eta\| < 2.5$			
4 barrel layers	3	$50 \times 400\,\mu m^2$	10(R-ϕ)-115(z)
3×2 lateral disks	3	$50 \times 400\,\mu m^2$	10(R-ϕ)-115(R)
SCT, $\|\eta\| < 2.5$			
4 barrel layers	8	50 µm	17(R-ϕ)-580(z)
9×2 end-cap disks	8	50 µm	17(R-ϕ)-580(z)
TRD, $\|\eta\| < 2.0$			
73 barrel tubes	30	$d = 4$ mm, $l = 144$ mm	130/straw
9×2 end-cap disks	30	$d = 4$ mm, $l = 37$ cm	130/straw

detectors (SCT and PD) and subsequently the following hits are added by a combinatorial Kalman-filter algorithm;

- the outside-in algorithm, where the hits in TRD of secondary charged particles, formed by decays from primary particles or other secondary particles, are used to reconstruct the tracks adding the Silicon hits, always with the combinatorial Kalman-fitter algorithm, if there are. The efficiency of track reconstruction is measured by simulated events, and it varies as a function of p_T and η.

Insertable B-Layer

In 2015, it was added a new innermost tracking detector, the Insertable B-Layer, in order to improve the ATLAS tracking performances during the Phase-I LHC operation. The Insertable B-Layer (IBL) is the first detector of the ATLAS setup and the latest upgrade of the pixel detector. The detector is based on 2 different technologies of the silicon sensors:

- the planar sensors, with a "typical" silicon layout, similar to the ones used in the B-layer, but slightly different. In fact, the inactive border passed from the previous 1mm to 450 µm, and from studies on the B-layer, the irradiated sensors increase the collected charge if the thickness is reduced;

- the 3D sensors, with a different geometrical arrangement of the silicon junctions. The signal is collected simultaneously by two different electrodes, since the number of produced charges is lower. Moreover, the active surface of these sensors extends much more, reducing the not sensible volume, and they are closer to each other in the arrangement of the detector, which means a reduction of the applied voltage and the leakage current contribution.

Thus, the new technology of IBL makes the detector itself more radiation hard and with a higher surface coverage, with the usage of the FE-I4 chip.

1.3.2 Calorimeter

The ATLAS calorimeter system is composed by sampling calorimeters whose scope is to perform the energy reconstruction of a crossing particle. The entire sub-detector is divided into multiple parts, each of which dedicated to a different type of particle. Each calorimeter consists of four parts, to which correspond a region of coverage: a barrel part, an extended barrel part, an end-cap part and a forward part. The whole system covers a pseudorapidity up to η=4.9 and a complete ϕ coverage. The ATLAS calorimeter setup is presented in Figure 1.6. The

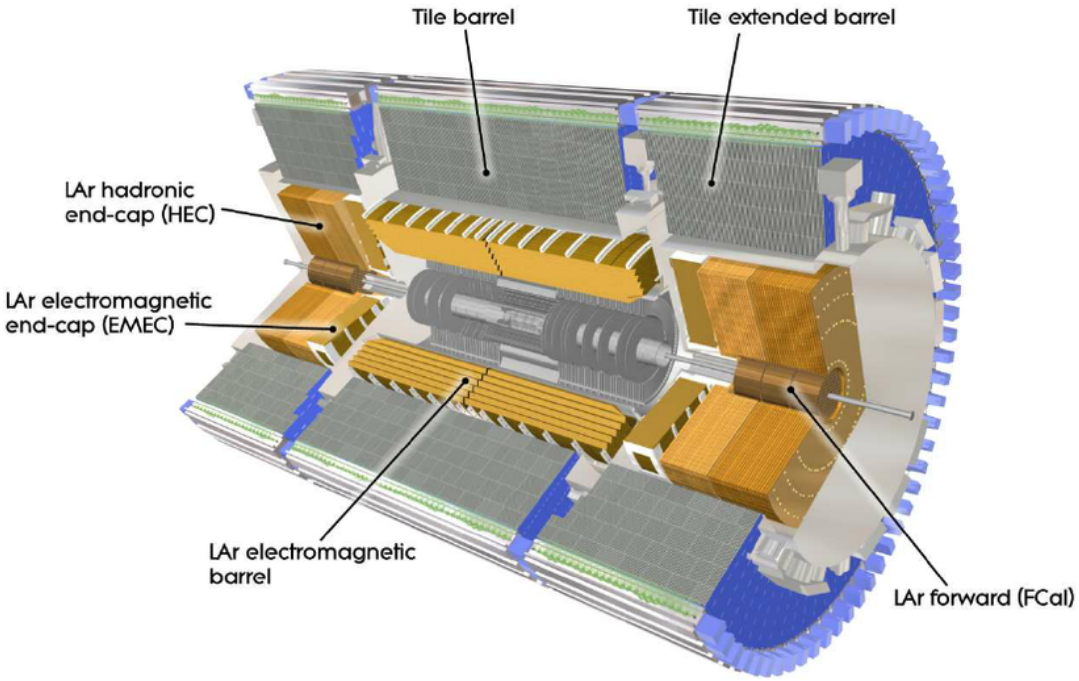


Figure 1.6: *The ATLAS calorimeters system setup. It consists of a Liquid Argon (LAr) electromagnetic calorimeter and a Hadronic Calorimeter. Interactions in the absorbers transform the energy into a "shower" of particles that are detected by the sensing elements.*

whole calorimetric system consists of:

- a Liquid Argon (LAr) electromagnetic calorimeter, focused on the measure of electron, positrons and photons in the pseudorapidity range $|\eta| < 3.2$, with a high granularity. It is divided into a barrel and two end-cap. These calorimeters are based on a lead-LAr detector technology, the lead is a good absorber, while the liquid argon is a good active medium, since its radiation hardness and its good energy resolution;
- a Hadronic Calorimeter (HCAL), focused on the energy measurement of jets originated from hadronic processes, as well as the determination of the missing transverse momentum. It is formed by the Hadronic Tile Calorimeters (HTC), a scintillator-tile calorimeter, by the Hadronic End-Caps Calorimeters (HEC) and the Forward Calorimeter (FCAL) which are both LAr calorimeters.

In terms of energy, the resolution of a sampling calorimeter can be written as

$$\frac{\sigma_E}{E} = \frac{a}{\sqrt{E}} \oplus \frac{b}{E} \oplus c, \tag{1.4}$$

where the term $\oplus$ refers to the sum in quadrature of the terms. Also, a is the stochastic term and comes from intrinsic fluctuations of the development of the shower due to their statistical behavior; b is the term due to the electronic noise of the read-out channels; c is a constant that considers the temperature, the aging of the detector, the radiation damage, and other factors.

The energy resolutions are different depending on the calorimeter composition: $\frac{\sigma_E}{E} = \frac{10\%}{E} + (1.2 \pm 0.1^{+0.5}_{-0.6}\%)$ for the electromagnetic calorimeter in the barrel region, from 0.13 to 0.06 (when the transverse momentum increase) for the hadronic one in the barrel and in the end-cap region, $\frac{\sigma_E}{E} = \frac{10\%}{E} + (2.5 \pm 0.4^{+1.0}_{-1.5}\%)$ for the forward electromagnetic calorimeter.

1.3.3 Muon Spectrometer

In Figure 1.7 is shown the scheme of the ATLAS Muon Spectrometer. High p_{T} muons provide signatures for many physics processes studied in ATLAS and that is the reason why the muon spectrometer has a key role in particle identification. It is designed in order to reach high precision and resolution in the measurement of high p_{T} muons, and it also provides an independent muon trigger from the rest of the detector.

The measurement is based on the magnetic deflection of muon tracks in the large superconducting air-core toroid magnets (the magnetic system is presented in Section 1.3.4). Over the range $|\eta|<1.4$, magnetic bending is provided by the large barrel toroid, while for $1.6<|\eta|<2.7$,muon tracks are bent by two smaller end-cap magnets inserted into both ends of the barrel toroid. Over $1.4<|\eta|<1.6$ (transition region) a combination of barrel and end-cap fields provides magnetic deflection.

The detector is divided in barrel and end-cap region, in which are placed toroid magnets and the system is divided in two different groups of sub-detectors, composed by 4 different detector technologies: 2 of them are the Precision Chambers (Monitored Drift Tubes and Cathode Strip Chambers), which are focused on precision measurement of muon momentum and the other 2 are the Trigger Chambers (Thin Gap Chambers (TGC) and Resistive Plate Chambers (RPC)), which provide online trigger.

The Monitored Drift Tube (MDT) chambers are drift chambers of two multilayer drift tubes which are focused on precise measurement of the z coordinate in the barrel region, in the region $|\eta|<2$. By measuring the drift time in single tubes it is possible to reconstruct the hit position of the particle providing the measurement of momentum and track in the barrel and end-cap regions and

The Cathode Strip Chambers (CSC), are multi-wire chambers with strip cathodes for the measurement of muon momenta in the pseodurapidity region $1.0<|\eta|<2.7$. The CSC wires are composed of parallel anodes which are perpendicular to 1 mm large strips of opposite polarity. They are positioned close to the beam pipe in the innermost layer of the end-cap.

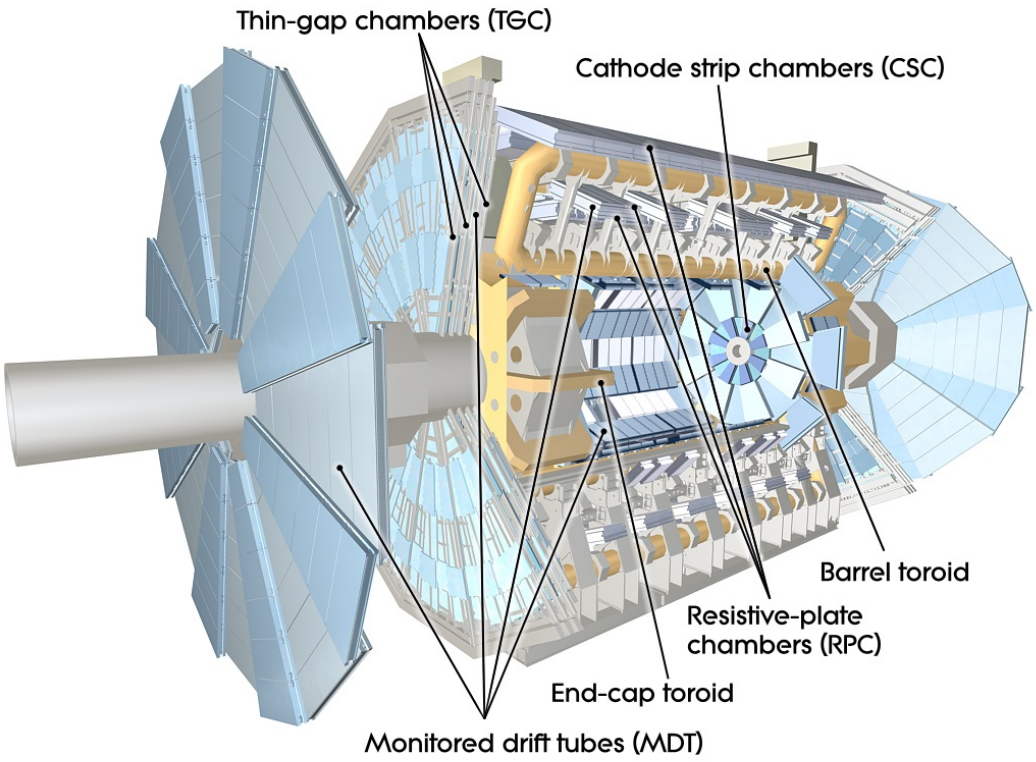


Figure 1.7: *The ATLAS muon spectrometer morphology. The system is divided in two different groups of sub-detectors, the Precision Chambers (Monitored Drift Tubes and Cathode Strip Chambers) for muon moment and Trigger Chambers (Thin Gap Chambers, Resistive Plate Chambers) for online data-acquisition triggering.*

Thin Gap Chambers (TGC) in the end-cap region is a very thin multi-wire chambers. The anode-cathode spacing is smaller than the anode-anode spacing, leading to a very short drift time ($< 20, \text{ns}$). The spatial resolution of these detectors is 4 mm in the radial direction and 5 mm in the ϕ coordinate. The TGC are also used to improve the measurements along the ϕ coordinate obtained from the precision chambers.

The Resistive Plate Chambers (RPC) in the barrel region are gaseous parallel electrode-plate detectors, with a spatial resolution of 1 mm in two coordinates and an excellent time resolution of $1, \text{ns}$. This sub-detector works in the avalanche regime: when a charged particle passes inside the chamber, the primary ionization electrons are multiplied into avalanches by a high electric field,

This whole system identifies muons for which $|\eta < 2.7|$ with a threshold of $p_{\text{T}} > 3\,\text{GeV}/c$, since muons with lower energy are completely absorbed (they lose their whole energy) before reaching the muon spectrometer. The measurements' resolution of p_{T} is about 20% at 1 TeV.

1.3.4 Magnetic System

ATLAS uses a system of superconducting magnets (shown in Figure 1.8) for the measurement of the charged particles momenta.

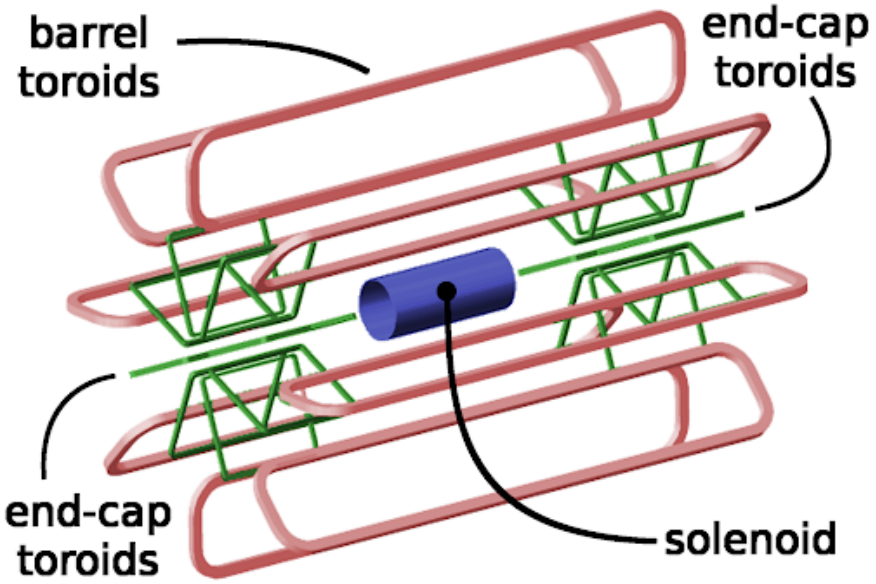


Figure 1.8: *The ATLAS Magnetic system schematics.*

The system is composed by a Central Solenoid (CS) surrounding the Inner Detector, and by a system of 3 large air-core toroids (1 in barrel and 2 in end-cap) generating the magnetic field of the muon spectrometer, with a dimension of 26 m in length and 20 m in diameter. The CS, for the momentum measurement of ID, has a magnetic field of 2 T, and it points in the positive z-axis direction, while the toroids magnet emits a magnetic field of 3.9 T (barrel) and 4.1 T (end-cap). The entire system work at 4.7 K of temperature. The most important parameters for momentum measurements are the field integrals over the track length inside the tracking volume:

$$I_1 = \frac{0.3}{p_{\mathrm{T}}} \int_0^l B \sin(\theta)_{(\mathrm{d}\vec{\mathbf{l}},\vec{\mathbf{B}})}\, \mathrm{d}l$$

and

$$I_2 = \frac{0.3}{p_{\mathrm{T}}} \int_0^{l\sin(\theta)} \int_0^{\frac{r}{\sin(\theta)}} B \sin(\theta)_{(\mathrm{d}\vec{\mathbf{l}},\vec{\mathbf{B}})}\, \mathrm{d}l\, \mathrm{d}r$$

where I_1 is the measurement of bending power field (p_{T}=q $\times$bending power=q $\times(B \times L)$) and I_2 represents the total transverse deflection of the particle from its initial path. θ is the longitudinal component of the angle between the track and the magnetic field and the integrals are calculated on the azimuthal direction of the particle ($l = r/\sin(\theta)$) and on its radial trajectory.

1.3.5 Trigger and Data Acquisition system

Thanks to the high luminosity reached in the LHC ring, in 1 second are produced an order of magnitude of 10^8 proton-proton processes. Despite that, the read-out

electronics can reach at maximum a rate for recording speed of 300 Hz and for this reason the ATLAS experiment has a system of multi-level trigger, composed by a series of three different trigger levels: Level-1 (L1), Level-2 (L2) and Event-Filter (EF). In Figure 1.9 a scheme of the current Trigger and Data Acquisition (TDAQ) system of ATLAS is presented.

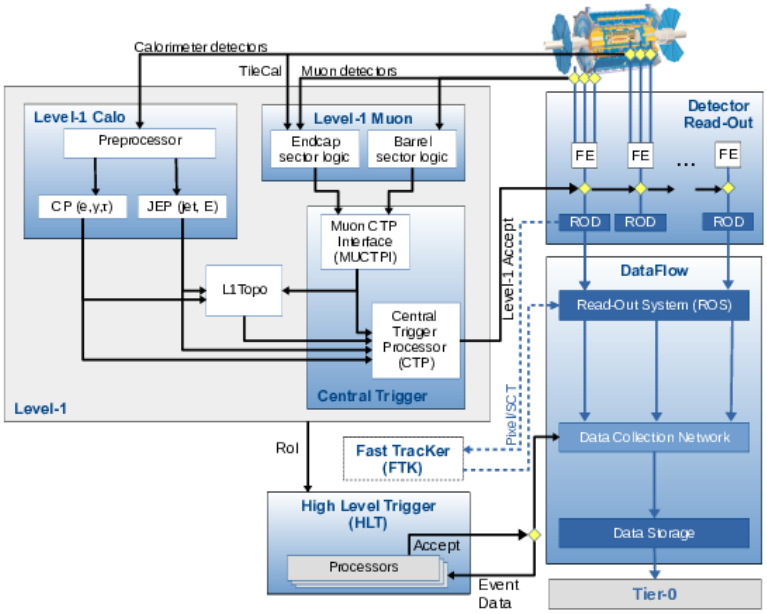


Figure 1.9: *The ATLAS current Trigger and Data Acquisition (TDAQ) system. The Level-1 trigger uses information from multiple subsystems to quickly identify events of interest, while the more refined High-Level Trigger exploits information from all the subsystems. Accepted data is then sent to the Read-Out system to be stored.*

The L1 trigger is a hardware-based trigger which splits data read by high p_{T} leptons, photons, jets, large missing transverse momentum and total transverse energy. It reduces the rate of the dataflow down to approximately 50 kHz, with a decision time for each collision of 2 µs from the collision itself. The data for the trigger comes from the calorimeters and from the Muon Spectrometer (MS), in particular from the RPC and the TGC chambers. The L1 trigger defines the regions in η and ϕ coordinates where the other subsequent triggers will have to start their own works, called Regions of Interest (RoI).

Furthermore, L1 muon trigger searches for coincidences of hits in different trigger stations within a road pointing to the interaction point, because the width of this road is correlated with the transverse momentum. There are six muon p_{T} threshold ruled by the hardware-programmable coincidence logic for this part of the trigger, three for the $6-9$ GeV (low p_{T}) and three for the $9-35$ GeV (high p_{T}).

The L2 trigger instead is a software based trigger which starts from RoIs defined in L1 and uses all the detector information in these regions in its trigger algorithms. It allows reaching less than 5 kHz in less than 50 ms. Eventually, the Event Filter is the final stage of the trigger chain, and reaching the rate of approximately 30 Hz in less than 4 s. This time does not come by algorithms, instead it is the standard time of the off-line event reconstruction of ATLAS TDAQ.

At the end of the entire TDAQ, filtered data reach the permanent storage of the CERN collaboration, which is actually part of the Tier-0 computational center, the main core of the computational power of the more complex network presented in the last section of this chapter.

1.4 LHC data production

As already mentioned in section 1.1, the core of the data production inside LHC relies on the collision of the proton beams in the interaction points. In terms of measurable physics, it may not be intuitive to convert them into an intelligible quantity, but it is possible to go through the entire process of data production, starting directly from the collision itself reaching eventually the final memory storage.

For the conditions at which the LHC accelerates protons, when the beams are actively running, the average number of collision produced is 600 millions per second, so roughly 10^9 collisions per second. Each collision produces in most cases a huge number of particles (in function of the center of mass energy), which are eventually detected by the large ad dedicated detectors mounted around the interaction points. For a single collision, the amount of energy released in a detector allows to collect a converted amount of data of around 1 MB, thus this implies:

$$\text{total data} = 10^9 \ \ \text{collisions/s} \cdot 1 \ \text{MB/collision} = 10^{15} \ \text{PB/s} \tag{1.5}$$

The resulting amount of raw data for a single collision is not even close to be managed by the DAQ systems of all the detectors, which is why are all designed with a trigger system, capable of rejecting the majority of the non-interesting events. As an example, the ATLAS trigger system is designed to collect around 200 events per second, which means

$$\text{filtered data} = 200 \ \ \text{events/s} \cdot 1 \ \text{MB/event} = 200 \ \text{MB/s} \tag{1.6}$$

and considering the working time of the LHC per day, which is around 10 hours per 2 shifts, running abount 300 days per year, the total amount of data collected in a year in a single experiment is

$$\text{yearly data} = 200 \ \text{MB/s} \cdot 3600 \cdot 2 \cdot 10 \cdot 300 \approx 4 \ \text{PB/year} \tag{1.7}$$

which means that collectively the 4 experiments at LHC produce around 15 PB/year of raw data which has to be stored to be later analyzed [30].

1.4.1 From CERN storage to the WLCG

The large amount of data produced yearly in all the LHC facilities presents a great challenge both in therms of storage and later analysis. As a matter of fact, in order to collect all the data produced by the experiments, CERN has built a large storage system, the CERN Data Centre [5]. The entire infrastructure is composed of around 11 000 servers, counting roughly 470 000 processor cores, besides several storage units, divided in 110 000 disks and 30 000 tape cartridges. The whole datacenter had at June 2022 of approximately 1.0 EB of storage capacity occupied [5], in both disks and tapes.

Despite its impressive scale and computational power, the CERN Data Centre alone cannot provide sufficient resources to accommodate and analyze the enormous amount of data produced. Moreover, enabling thousands of researchers worldwide to access and analyze this data in a timely manner requires a distributed infrastructure that extends far beyond CERN's physical boundaries.

In order to grant data access worldwide, CERN acts as the central Tier-0 hub of the Worldwide LHC Computing Grid (WLCG) [22]. After initial data acquisition and prompt reconstruction at the Data Centre, selected datasets are distributed to Tier-1 data centers located in partner institutions around the globe. The scheme of this distributed partners is shown in tiers in Figure 1.10.

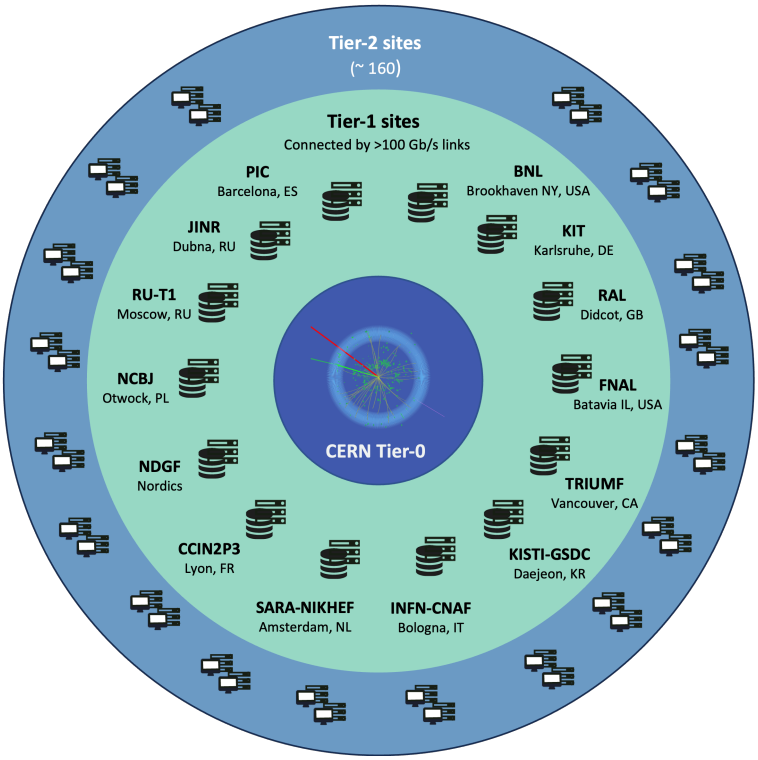


Figure 1.10: *The WLCG Tiers. The Tier-0 is the CERN Data Centre. Tier-1 sites are the main data centers that receive directly data from Tier-0 and provide long-term storage and reprocessing capabilities.*

The Tier-1 sites are the primary connected data centers in the WLCG, which receive data directly from the CERN Data Centre. They provide long-term storage,

data reprocessing capabilities, and serve as a distribution hubs for the Tier-2 and Tier-3 sites. The WLCG thus formed a hierarchical, federated system that bounded a collaboration between more than 170 computing facilities in over 40 countries [22]. In this scenario, the INFN-CNAF (Istituto Nazionale di Fisica Nucleare) computing center, covers the role of Tier-1 site. A detailed description of the INFN-CNAF computing center and its role in the WLCG is presented in the following chapter.

Chapter 2

The Big Data Platform of the INFN-CNAF

2.1 The "Istituto Nazionale di Fisica Nucleare" (INFN)

The "Istituto Nazionale di Fisica Nucleare" (i.e., INFN, translated as the National Institute of Nuclear Physics), is an Italian public body of research, dedicated to the study of the elementary constituents of matter and the fundamental laws of the universe. Since its birth in 1951, the INFN scientific mission has been "to promote, coordinate, and undertake scientific research in the field of nuclear, subnuclear, astroparticle and fundamental interactions' physics, which also involve technological research and development relevant to the activities in these sectors [26]".

Born to take up the legacy of the Italian physicists who contributed to the birth of nuclear physics in the 1930s (i.e., Enrico Fermi and the "Via Panisperna Boys"), the INFN designed in the first years of its activity the first Italian particle accelerator, the *Electron Synchrotron* and the first accumulation ring for matter and antimatter collisions, the *AdA* (i.e., Anello di Accumulo), which was the first accelerator in the world to collide electrons and positrons. In the same decade, INFN started to participate in the CERN research activities, a collaboration that has continued to present days. Over more than seventy years of scientific activities at the frontier of knowledge, INFN has been able to create a great school that trained thousands of researchers who stood out for their skills and expertise in the main international physics laboratories and research centres.

Today, INFN is a community of more than six thousand people that decisively contributes to the most important physics projects in Italy and the world and which has turned fundamental research into a national strength. Inside the national territory, the INFN carries out theoretical and experimental research in the fields of nuclear, elementary particle and astroparticle physics, working in close collaboration with the Italian main universities, and abroad with the major scientific collaborations active worldwide.

In addition, INFN's work is carried out across the whole country with two types of complementary research facilities: the *Divisions*, which are situated in universities' physics departments, for a total of 20 in the whole national territory, and the *National Laboratories*, which are located in specific sites, for a total of 4 in

the whole national territory. In details, the national laboratories are: the LNF (i.e., the National Laboratory of Frascati), situated in Frascati (Rome); the LNL (i.e., the National Laboratory of Legnaro), situated in Legnaro (Padua); the LNGS (i.e., the National Laboratory of Gran Sasso), situated near the Gran Sasso mountain, in Assergi (L'Aquila); and finally, the LNS (i.e., the National Laboratory of Catania), situated in Catania [29]. Moreover, the institute has 3 specialist national centres, the CNAF (i.e. the National Centre for Research and Development on Information and Communication Technologies) located in Bologna; the TIFPA (i.e. the Trento Institute for Fundamental Physics and Applications) located in Trento; and GGI (Galileo Galilei Institute for Theoretical Physics) in Florence. An overall view of all the presented facilities is shown in Figure 2.1.

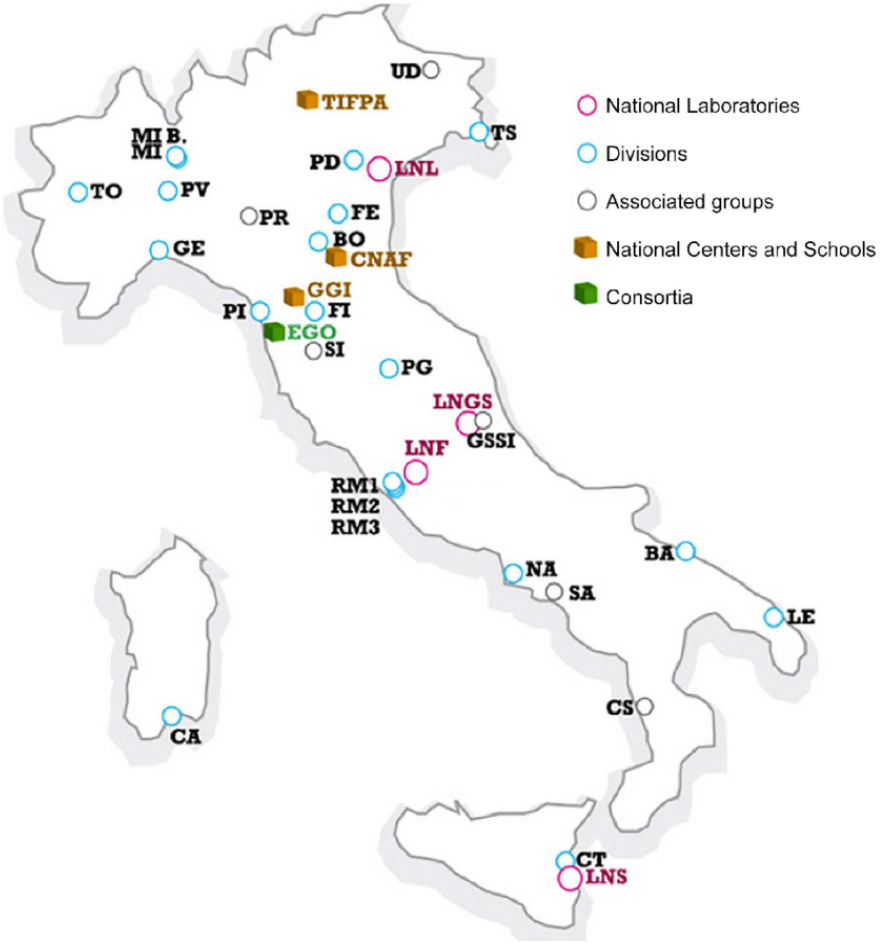


Figure 2.1: *The geographical distribution of the INFN's divisions, facilities, associated groups and national centres in Italy.*

The entire INFN community keeps working on the most important international projects, without forgetting its main mission, the pursuit of fundamental scientific research, as a public national institute. Science and knowledge are not only professional activities, but also a cultural value that INFN promotes to the public, in order to raise awareness of the importance of scientific research and its impact on

society. In order to give a closer look and enter the details of the INFN's researches, the following section will present the INFN Bologna division, which is the one where the research presented in this thesis has been possible to be carried out.

2.1.1 The INFN Bologna division

The history of the Bologna Division of the INFN started 5 years after INFN had been founded in 1951, with a Decree from the President of the National Research Council (CNR). Since its birth, the Division has collaborated with the physics department of the University of Bologna, and since 1953 it has been an "associate group" to the Division of Padua, one of the original four founding division of Rome, Milan and Turin.

During the first years of activity, thanks to first elected Director, Giampietro Puppi, the Division was able to establish itself as a reference point for the experimental physics in Bologna, with the first activities focused on the construction of a large liquid hydrogen bubble chamber [27], a device which was later used in the CERN laboratories for particle detection between 1960 and 1961. This achievement led to a jump in the quality of experimental physics in Bologna and to the local infrastructure, including the mechanical workshop. Since then the groups in Bologna have worked at the forefront of particle physics, contributing to the construction of cutting-edge technology equipments: worth recalling in particular the construction of gas detectors (limited streamer tubes, drift chambers, resistive plate chambers, multi-gap resistive plate chambers), scintillation and Cherenkov detectors, electronic scanning systems and the development of analog and digital electronics.

Subsequently, during the second half of the 1960s, the Division proposed thanks to the idea of its third Director, Antonino Zichichi, the construction of a large underground set of laboratories in the Gran Sasso massif. The proposal led to the creation of the Gran Sasso National Laboratories, a unique world infrastructure for research requiring a low environmental radioactivity background. At the end of the 1980s the Division started to work with more interest on the astroparticle physics field, reaching an important role in the international scientific community, which continues to this day.

In the 2000s, thanks to a renewed integration within the Division and the Bologna's regional fabric, the INFN Bologna Division had the opportunity to participate actively in the super-computing project led by CINECA, one of the largest Italian computing centres globally recognized in the High Performance Computing (HPC). The conclusion of the project led to the creation of the Bologna "Technopole", which is currently hosting one of the most powerful supercomputers in the world, the "Leonardo" supercomputer, which is part of the network of the European High Performance Computing Joint Undertaking (EuroHPC).

At present, the groups of the INFN Bologna Division are involved in a wide range of research activities spanning in particle, nuclear, astroparticle and theoretical physics, nonetheless promoting and advancing in technological challenges to be faced in all modern physics experiments. In particular, in the thematic area of high energy physics, the Division participates actively in all four main LHC experiments (subsection 1.1.1), with key contributions to the realization of experimental devices

to be mounted inside the detectors, as well as in the data analysis.

As a side support to all the groups collaborating to the various researchers' groups of the Division, the specialized INFN-CNAF national centre is actively collaborating with the Division, whose work and research activities will be detailed in the further sections.

2.2 The INFN-CNAF

"INFN-CNAF" is an acronym nowadays associated to the national centre of the INFN whose work is entirely dedicated to the "Research and Development on Information and Communication Technologies" [28]. As already mentioned in section 2.1, CNAF is one of the 3 national centres at service of the INFN, geographically located in Bologna, but a central facility collaborating with all the INFN Divisions and Laboratories in the whole national territory.

Historically, the CNAF was born in 1962, founded by the INFN as a technological centre dedicated to the analysis and high precision measurements of the bubble chamber photographic films. In fact, the original meaning of the acronym CNAF was, in Italian, "Centro Nazionale di Analisi Fotogrammi" (lit., National Centre for Frame Analysis) [25], the primary objective of the institute was the study and the recognition of the elementary particles interacting in the bubble chambers, used at the time to detect particles in nuclear physics laboratories. An example of a bubble chamber frame, in an enhanced version, is shown in Figure 2.2.

Another original mission of the CNAF was to start a centralization in a single centre the big facilities used at the time for bubble chamber analysis, such as the optical and electronic devices or the computers used for real-time data acquisition and data preprocessing. Since its birth, the CNAF objectives have actually evolved as the technological developments progressed, but the main goal of the institute practically never changed.

Figure 2.2: *An enhanced version of a frame film, coming from a bubble chamber.*

In general, the CNAF main objective may be interpreted nowadays as the main centre for the development of evolving technologies in the field of particle physics, from the detectors' infrastructure to the digital technologies used in modern high energy experiments. Moreover, the modern objectives of the CNAF institute has been moved to the field of communication technologies, with the aim to provide and operate the INFN's computing facilities. Nevertheless, CNAF provides technical, scientific and operational support to the whole INFN community, exploiting information, communication technologies and infrastructures. In fact, since the early 1990s, CNAF was invested with the task to implement, manage and

coordinate in time informatics services at service of the INFN community. Up to present days, the CNAF main activities are divided into several areas of interest:

- **Software development and Distributed Systems (SDDS)**: a functional unit tasked with the study, development and support of distributes software technologies and infrastructures, fitting perfectly the strategic vision of computing in the INFN;
- **National Services**: responsible for the management and development of the national services like General ITC services (i.e., national email, domain, Wi-Fi infrastructure, etc.), Management of national contracts (i.e., licenses management in terms of hardware and software management) and INFN Information system services;
- **Information System**: founded in 2001 to digitize and manage all the administrative and accounting processes of the institute, carrying out a gradual transition from paper to digital;
- **Technological transfer Lab (TTLab)**: the INFN research laboratory in Emilia-Romagna, which collects multidisciplinary knowledge from different branches in the region (from Bologna, Ferrara, CNAF);
- **EPIC Cloud**: the cloud service developed and managed by CNAF to fulfill the requirements of projects and experiments dealing with clinical, biomedical and genomic data;
- **World LHC Computing Grid (WLCG)**: the main INFN computing centre, connected directly to CERN data centre, being a primary part of the WLCG.

Regarding the last point, it is worth taking a close look at the Tier-1 computing centre hosted by the CNAF in Bologna. A detailed overview is provided in the next section.

2.2.1 The INFN-CNAF Tier-1 computing centre

Since 2003, CNAF hosts the main computing centre at service of the institute, the *Tier-1*. The Tiers' hierarchy was already presented in subsection 1.4.1, but it is worth recalling and extend more details about the Tier-1 computing centre in Bologna. A data centre of this scale is a highly specialized facility designed to host large numbers of servers, storage systems, and networking equipment. It provides the critical infrastructure needed for processing, storing, and transferring vast volumes of scientific data, with features such as redundant power supplies, cooling systems, and high-bandwidth network connections to ensure continuous operation.

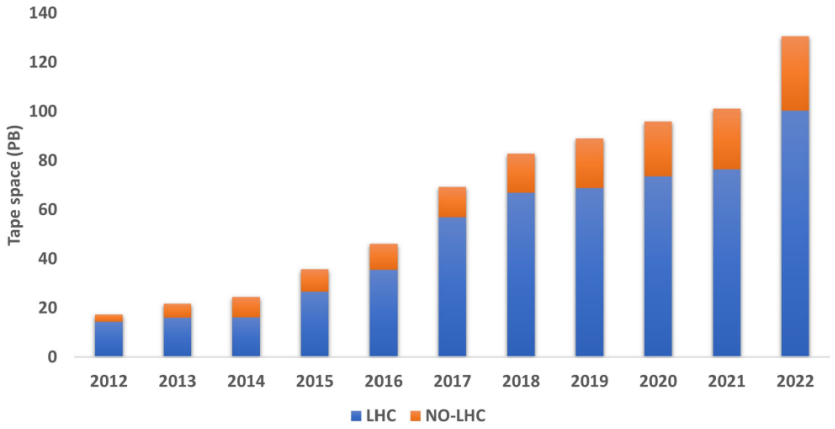
The Tier-1 computing centre at CNAF fits this model, acting as one of the major nodes of the Worldwide LHC Computing Grid (WLCG). It ensures the long-term storage and reprocessing of LHC data, supports data replication across the grid, and provides computational resources to thousands of researchers worldwide. CNAF's

Tier-1 infrastructure combines petabyte-scale disk arrays, extensive tape libraries, and high-performance compute clusters, all interconnected via high-speed networks linking CERN's Tier-0 and other Tier-1 sites.

Besides the Tier-1, the data centre hosts also other computing facilities, such as a Tier-2 for the LHCb experiment, a small HPC cluster and a Tier-3 computing facility co-managed by the INFN Bologna Division and the services for the Grid infrastructure as well as the INFN national services managed by other groups than CNAF.

In general, one of the most challenging aspects of the large scale computing, relies on the data storage. In fact, the evolution of the data storage systems capacity and the relative occupancy the Tier-1 have had along the years, is shown in Figure 2.3. As the plots show, the storage system is equipped with both disk and tape, providing a robust solution for managing the vast amounts of data generated by the LHC experiments and all the other collaborating ones.

(a) Relative disk storage occupancy at CNAF Tier-1 for LHC and non-LHC experiments (until 2022).



(b) Relative tape storage occupancy at CNAF Tier-1 for LHC and non-LHC experiments (until 2022).

Figure 2.3: *The evolution of the storage utilization at CNAF Tier-1 computing centre, from 2012 to 2022, for both the disk and tape memories.*

Both the disk and tape storage systems are organized as a hierarchical mass storage system, managed by the Grid Enable Mass Storage System (GEMSS), a

homemade integration of the IBM General Parallel File System (GPFS), a high-performance clustered file system for rapid and scalable data access to large datasets, and IBM Tivoli Storage Manager (TSM) software, a data protection platform used for backup and data recovery.

The disk storage system is based on a distributed file system, which allows for high-speed access to data and provides redundancy and fault tolerance. At the moment, the disk storage system has a total occupied capacity of about 41 PB, while the tape storage system reaches up to 98 PB (the actual total capacity of the tape storage is about 116 PB) [12]. The storage infrastructure at CNAF is based on industry standards, both for physical inter-connection level using Storage Area Network based on Fiber Channel protocol, and for logical data access level using high-performance clustered parallel file systems (typically one per major experiment).

The adopted solution for the storage has allowed over the years the implementation of a high performance data access system, completely redundant and independent of the hardware point of view, internally operated and maintained by the *Storage* group of the Tier-1 datacenter unit.

As a matter of fact, having at disposal a high-performance storage system, the datacenter mainly supports the LHC experiments, since they are responsible for the majority of the data stored in the CNAF Tier-1, and they all heavily use grid technologies within their working environments. Nevertheless, CNAF provides also computational resources to all the other experiments and projects of the INFN, in the field of high energy physics, astrophysics and astroparticle domains, granting also access to the Bologna research groups specialized in theoretical physics and astronomy, supporting the growing computing communities in other physics experiments. Moreover, also some external research groups can access the Tier-1 resources, belonging to the research area of Computational Chemistry and Biomedical domains.

As a partner of several national and international collaborations, another key aspect of a modern computing centre like the one at CNAF relies on the data transfer capabilities, essentials for the data traffic in and out of the datacenter. Having at disposal a huge number of CPU cores and a large storage capacity, led CNAF *Networking* group to design a high-performance Local Area Network (LAN) able to provide a data access bandwidth of the order of hundreds of gigabits per second. Each process performed must be able to effectively access to data with an access rate of the order of the main scientific experiments which are using the Computing Resources, having at disposal a minimum access bandwidth per job of 5 MB/s [11]. The LAN cable connections are Ethernet-based, ranging in the possibilities of 1/10/40/100 GE (i.e., Gigabit Ethernet). The highest speed is set for the core switches of the Tier-1, which are redundant modular devices with the scalability of more than 500×100 Gbit/s per port on each single device, which in the LAN of the Tier-1 facility lead to the current total capacity installed in the core switches of about 18 Tbit/s.

Being inside an international network for computational, brought the need to have a high-performance connection network in a Wide Area Network (WAN), which is actually set in different ways. In fact, the CNAF Tier-1 WAN have 2×100 Gbit/s links dedicated to the scientific and research networks of the LHCOPN (i.e., LHC

Open Private Network), which is the connection established between the CERN Tier-0 and the Tier-1 sites, and the LHCONE (i.e., LHC Open Network Environment), which is the connection established between the Tier-1 sites and the Tier-2 centres. In parallel, 2 different link connections both of 2×10 Gbit/s are established, one for the general access to Internet, and one dedicated to the IT distributed applications and cloud environment [11].

Another important aspect of networking regards a small extension of the LAN, which was implemented by CNAF in collaboration with CINECA, establishing a dedicated network connection towards one of the most powerful high performance computing centres in Europe, capable of providing a 400 Gbit/s rate on a total available bandwidth of 1.2 Tbit/s. In addition, the latency within the two centres is very low, also thanks to the use of dedicated Data Centre Interconnect (i.e., DCI) technologies, capable of a round trip time of about 1 ms. Having a high bandwidth and low latency connection between the two centres, allows the CNAF to provide a massive usage of multiple computing nodes physically installed in CINECA, without the penalty of the latency in the data transfer (which still is in performed in a physical distance of 20 km).

Eventually, regarding the capabilities in terms of computing, the CNAF cluster supports local, Grid and Cloud resource requests on both local and remote environments. The management of the computing resources like worker nodes, user interfaces and grid services are all handled in bulk by the CNAF *Farming* group. The computing power of the cluster relies on 38000 cores, reaching a whole computational power of 400 kHS06.[1] Even its great potential, all the power is not entirely coming from CNAF, around half of the computing resources are leased from CINECA computing nodes (connected to the cluster via the network already described). Jobs submitted to the farm have direct access to the files in the storage system, and for each experiment exists a dedicated queue to access resources, even the entirety of the cluster resources is handled by a single batch system. The resource allocation is based on a hierarchical fair-share scheduling, allowing to maximize the CPU usage, leading the maximum usage of the Tier-1 farm most of the time, reaching around 100000 batch jobs are executed per day.

Inside this huge working environment, with a lot of active projects in both national and international collaborations, a parallel computing framework was designed and implemented in the cluster, built to be used locally at CNAF. As a core part of this paper work, next section will present in details the architecture and the components of the framework from which the research started its development.

2.3 The Big Data Platform

In the environment of the CNAF Tier-1 computing centre, a local software infrastructure has been designed and implemented: a Big Data Platform (BDP, or equally a Big Data Processing Infrastructure), a suite for the management and the analysis of heterogeneous data, at service of the admins group part of CNAF and all the physics researchers actively collaborating with it. Having at disposal a large

[1] HS06 (HEP-SPEC06) is a benchmark unit derived from the SPEC CPU2006 suite, adapted for high-energy physics applications to quantify computing power in terms of standardized workloads.

computing power and a huge storage capacity, the BDP is a framework designed to provide a set of tools to index data into dedicated databases, allowing to collect logs from several sources. In fact, one of the main goals of the BDP is to assist the researchers working in high energy physics experiments, granting them access to the job reports submitted and executed in the Tier-1 cluster, as well as to the logs generated administrated by the CNAF groups mentioned in subsection 2.2.1.

The BDP architecture is based on a set of components, each one dedicated to a specific task, which are all integrated together to provide a complete solution for the data management and analysis. The currently updated architecture is shown in Figure 2.4.

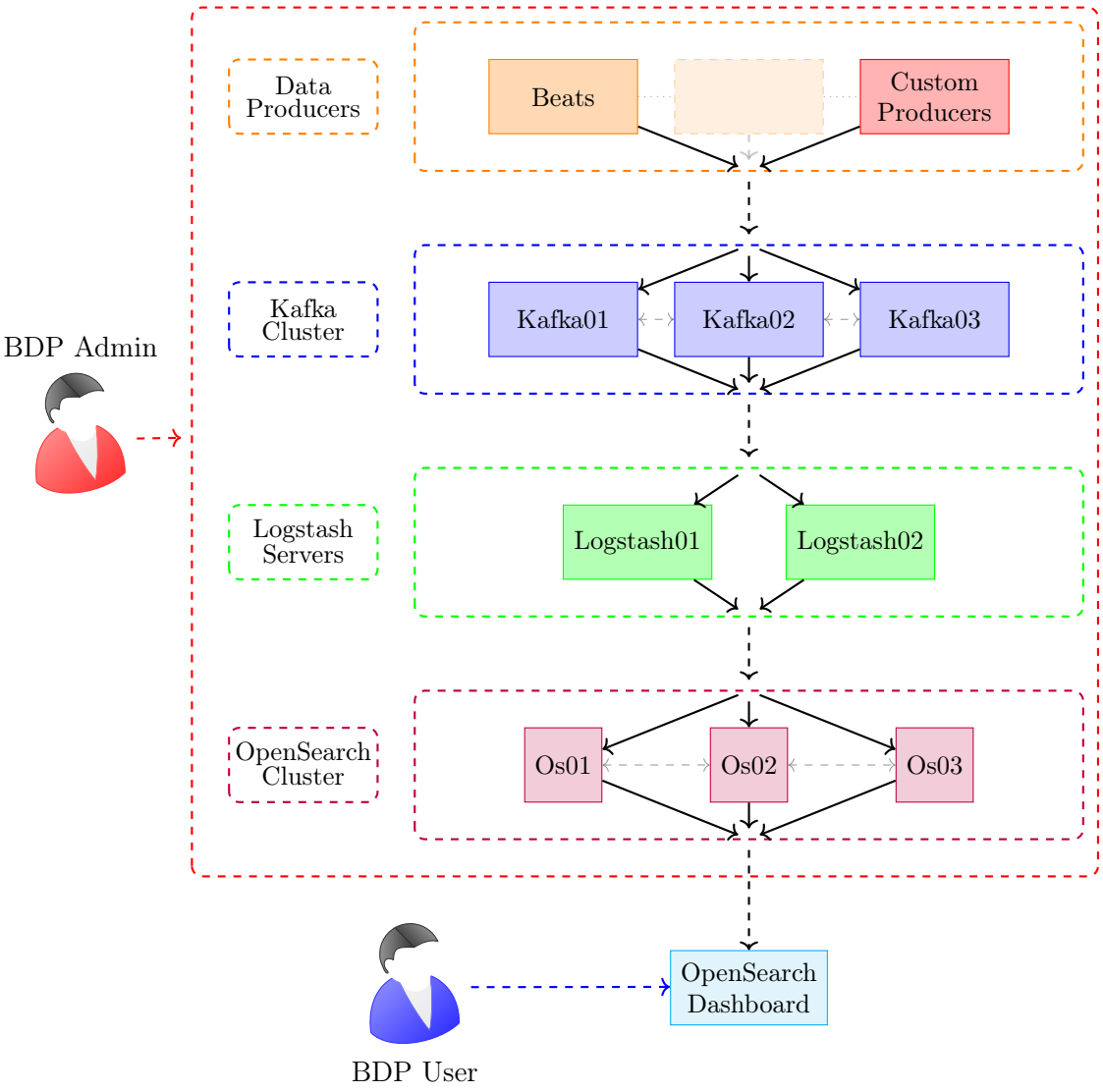


Figure 2.4: *The current software architecture of the Big Data Platform (BDP) at CNAF. Servers are organized into clusters, starting from producers, moving through the broker and the consumers, and finally reaching the storage, able to display all data received into online dashboards. Data flow follows the black arrows, while the gray dashed lines indicate the connections within a cluster.*

The adopted software design suite was developed to be modular and extensible,

allowing the setup of new servers and components, only with the agreement of the facility administrators, since they are the only ones able to manually enter inside the virtual machine inside which each component software is running. User access is granted only to the OpenSearch Dashboard interface, only accessible via the LAN connection of the INFN Bologna division, or either via a VPN connection. Moreover, the choice to adopt each single component rely on the fact that every single component is almost smoothly able to be connected with the others after the installation, allowing a fast and easy setup of the whole platform.

2.3.1 The BDP components

As the figure show, the platform is divided into four smaller clusters, each of which is mounted on a different server. It is pretty trivial to see how the data flow is set. Data is injected in the platform by the producers, which send data to the message broker, the Kafka cluster. The latter is responsible for handling the incoming data streams, allowing efficient queuing and forwarding the data to the consumers, the Logstash servers. In here data is transformed and ultimately processed, ready to be sent to the data nodes of the OpenSearch Cluster.

The clusters tasks may be summarized as follows:

- **Data Producers** are responsible for collecting and sending data to the platform, which can be either Beats (i.e., lightweight data shippers provided by Filebeat) or custom producers developed by the users (e.g., Python scripts);
- **Kafka Cluster** acts as a message broker, handling the data streams between producers and consumers. All the 3 servers may receive data from the producers, and they are all connected to each other, allowing a distributed and fault-tolerant architecture;
- **Logstash Servers** processes and transforms the data before it is stored, digesting the shipped logs, applying filters configured by the users and configuring the output ready to be read by the data cluster;
- **OpenSearch Cluster** provides the storage and search capabilities for the data, organizing data into indices with sharded and replicated data, allowing for safe data retrieval, and easy to check by the users with the OpenSearch Dashboard web interfaces.

In order to provide a detailed overview, it is worth taking a closer look at each components' features and functionalities.

For instance, the main solution chosen for the data producers in the utilization of the Filebeat software, a lightweight shipper software for forwarding and centralizing log data [17]. It is installed as an agent on pre-configurable servers connected locally to the BDP, inside which it is able to search for log files in specific directories inside the machines and send them to the message broker. The majority of the logs collected are provided by the CNAF administrators and working groups, but also with a submission request from asking users, it is possible to submit logs from topics not regarding the CNAF activities, allowing the users to have a complete overview of the logs generated by their jobs (which is what will be explained in

details in chapter 3). Apart from beats generated with Filebeat, user may build custom producers (the easiest to write are Python Scripts) to send data to the Kafka Cluster, but forwarding them only at given conditions. As already mentioned, the producers have to be installed in servers connected to the LAN in which also the BDP is connected, nonetheless, the producers are allowed to send data to the cluster only with a user-password authentication, mediated via SSL/TLS connection[2] (configured by the administrators). Lastly, data arriving at the Kafka cluster has to be organized into *topics*, which are logical channels that allow the producers to send data to the cluster, and the consumers to read a specific one to take data from it. Topic has to be set at producers' level, since the platform is pre-configured to receive data from topics manually added by the admins.

Independently of the producers, if the requirements to send data inside the BDP, data reaches the Kafka cluster, the message broker of the platform. Apache Kafka is an open source event streaming platform, designed to handle real-time data feeds into queues, administrating data injection and message forwarding [19]. Inside the Kafka cluster, three different brokers are installed, each in communication with the others, avoiding the risk of data loss and traffic congestion. An example of the interaction between the producers and the Kafka cluster is shown in Figure 2.5. The

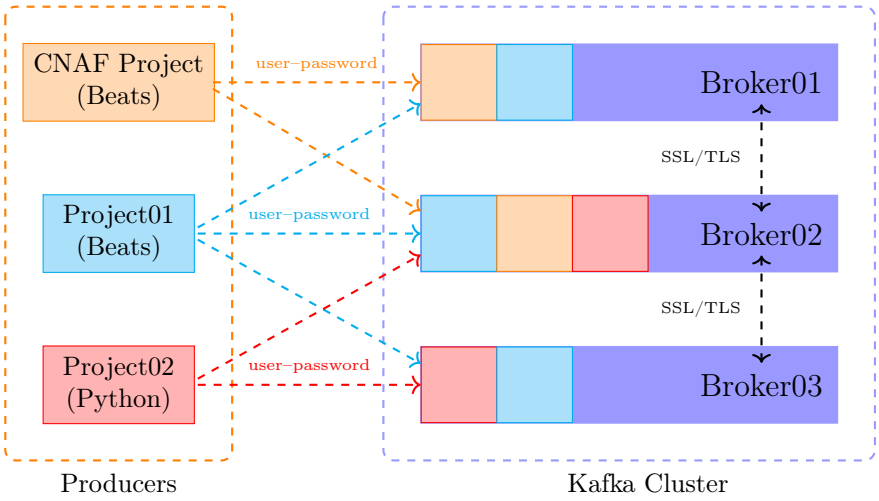


Figure 2.5: *An example of the producers and broker interaction. Producers send logs to the cluster thanks to a user-password authentication, mediated by SSL/TLS connection. The Kafka cluster is composed of 3 brokers in communication with each other, and each broker builds its own queue of messages in accordance to the incoming topics.*

queues are built in accordance to the logs incoming from the producers, arranging them taking into account the topic name, which practically is the logical channel through which the consumers will digest data. In fact, apart from the user-password authentication (which is mandatory for both producers and consumers), Kafka enforce an Access Control Lists (ACLs) defined on the *topics*. An ACL is a set o rules which specifies the possible operations which authenticated principals (i.e.,

[2]SSL/TLS (i.e., Secure Sockets Layer/Transport Layer Security) is a standard security technology for establishing an encrypted link between a server and a client.

users or groups) can perform on the topics, such as reading, writing, or managing the topic itself. Moreover, at the level of the broker, the *topics* may receive a first level of splitting: here some topics are split into *partitions*, which are divisions of a topic, sharded and already set to be allocated is a specific data node later in the pipeline.

The Kafka cluster is able to work and redirect the data sending even if one of the nodes is down, granting a smooth and continuous data flow. The cluster is able to perform so also because the traffic is protected via SSL/TLS connection, which ensures first of all data encryption during communication, but also data integrity and certificate authentication in order to both detect data tampering and verify the communication parties' identities. The SSL connection is established both at the end-to-end level (i.e., between producers/consumers and brokers) and at the broker-to-broker level, allowing a secure communication in a locally connected environment.

Once the data is sent to the broker, simultaneously the data is read by the consumer servers, inside which a Logstash instance is running. It is actually possible to use also consumer customized by the users, but the most suitable solution for the BDP is the Logstash software, which is an open source server side data processing pipeline, able to data from multiple sources, transform it, and send it to a specified destination [18]. Apart from that, Logstash is optimal for its possibility to filter the incoming data, allowing the user to set how data has to be organized when stored: filters may be applied directly on logs files, either on the fields of the logs, providing a very useful tool for data cleaning and at the same time for the data indexing. It is in fact at this level that topics are converted into indices, ready to be sent in the OpenSearch Cluster. A schematic example of the interaction between the Logstash instances and the OpenSearch cluster is shown in Figure 2.6.

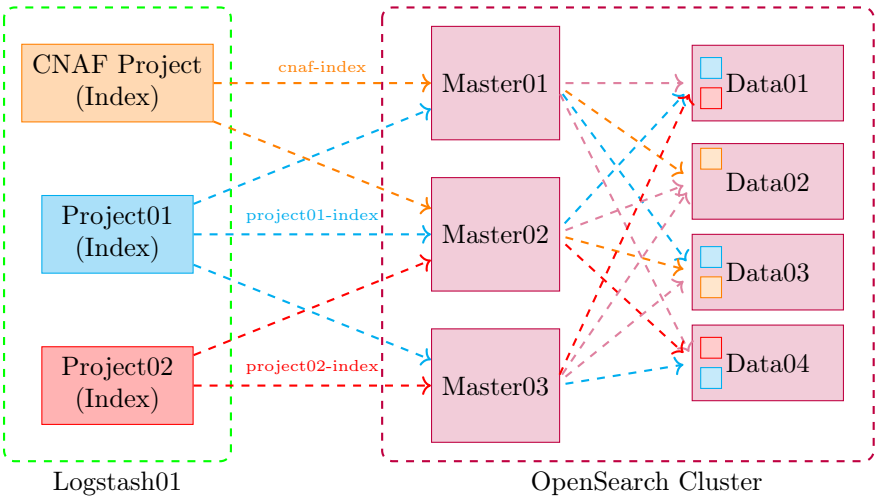


Figure 2.6: *A schematic example of the Logstash and OpenSearch interaction. Logstash sends indices to the OpenSearch cluster, a server composed of 3 master nodes and 4 data nodes. Here indices are split according to their topics' partitioning, sharded into different nodes according to the rules applied to their assigned index.*

It is now important to mention that, in its initial configuration, the BDP hosted

only a single general instance of Logstash, which included all the projects' topics specifications, but as the Figure 2.4 shown, the platform was extended to have two different consumer servers. In fact, the initial goal of the BDP project was to extend the consumer cluster to a single consumer server dedicated to each collaborating experiment, and the current architecture is a first step towards this goal, since the first ever addition to the consumer cluster was done during the development of the work presented in this thesis (more details will be given in the following chapter).

Apart from this change, the interaction between Logstash and OpenSearch is pretty straightforward: Logstash sends the indices to the OpenSearch cluster, which is composed of three master nodes and four data nodes, each one of which is able to store the indices in a sharded and replicated way. The master nodes are responsible for the cluster's data reception and addressing into the data storage nodes. Once data is sharded and indexed, it is practically impossible to read it manually inside the data nodes, due to the data distribution into unreadable binary files. Even though, the data is still marked with an index associated to a specific experiment or project, which is kept even if the data is sharded into different data nodes (like in Figure 2.6). Admins may access the master nodes to apply policies or specify particular data retentions rules, applying those rules only for a specific index. As an example, it is possible to configure the number of replicas for a specific index, or set a rule to check the occupancy of the shards of a specific index to close it or delete it as soon as it reaches a certain threshold.

Even if the data managing may be performed directly inside the master nodes, it is practically impossible to read the data once accessed manually into the servers. In order to see data inside the BDP, another approach must be taken. Since the data passing through the cluster are set to be compact JSON files, the information contained inside a single file is basically contained in all the fields written inside a specific JSON (including the index field). The fields' information is recallable thanks to the utilization of the LUCENE library provided by APACHE [20]. LUCENE is a high-performance engine library, capable of performing full-text search, nearest-neighbor search, and other advanced search capabilities on large indexed datasets. Inside the library, a research engine allows searching inside the data nodes with index specifications, allowing a background search and retrieval of the sharded data across the storage. This kind of research may be performed indirectly on the OpenSearch cluster, since it is performable to the front-end visualization provided by one of its features, the web Dashboards.

The OpenSearch Dashboard is a web-based interface that allows users to visualize and interact with the data. A variety of plots and graphs may be created by the users, with the restriction for each user to the assigned "tenant" role directly assigned by the admins. Each role has a granted access only to specific topics, including for the admins the access to all the index for internal management and monitor. The web interface is accessible as all the other components of the cluster via a direct connection with the LAN of CNAF, and access is performed for each user through a username-password authentication, but also via Indigo-IAM[3] recognition system. The OpenSearch Dashboard is the only component of the BDP that is directly

[3]Indigo Identity and Access Management, an open source or managing digital identities and control access to distributed resources.

accessible by the users, and it is the only way to visualize the data stored inside the BDP. More details on the dashboards are going to be presented in chapter 3, where the data collection will be described in its entirety.

Now, in order to summarize the whole BDP structure and components, and also to make a recap of all the processes involved inside the various components' interactions, an overall descriptive diagram is provided in Figure 2.7, showing the actual passage of a data stream from the initial steps up to the data storage.

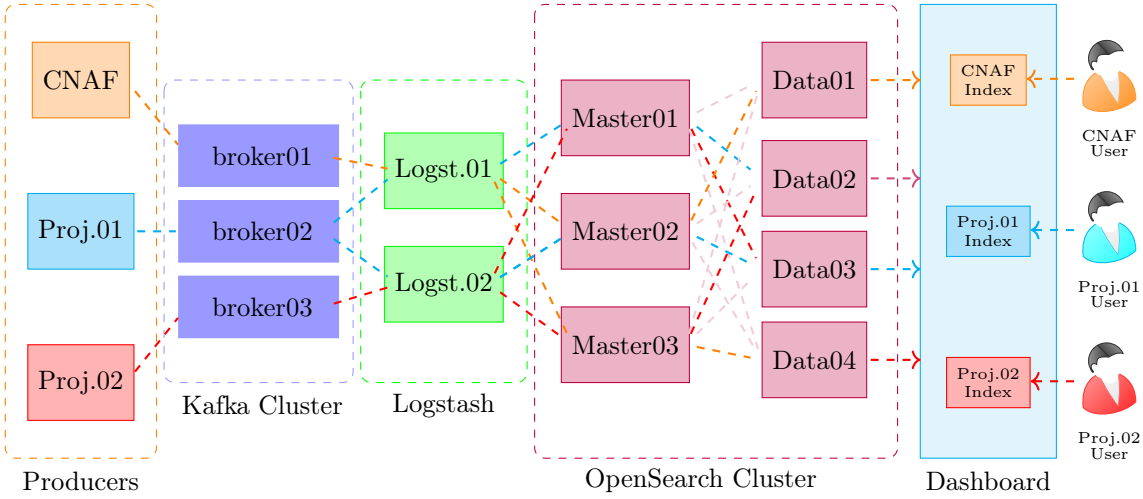


Figure 2.7: *Overview of the BDP infrastructure and its schematic working pipeline, from the producer to the visualization in the Dashboards. User access is only granted to the last part of the BDP, while admins have free access to the whole set of servers inside the architecture.*

As the figure shows, each user of the BDP has dedicated access to the topic inside the dashboard, ensuring limited access to external users to the data. On the other hand, admins have free access to the whole infrastructure, since they are responsible for the overall management of the platform. Notably, the infrastructure allows also the admins to have a full control on the incoming data from the producers, since the SSL bridge can be established only in agreement with CNAF. As a matter of fact, the further parts of this thesis are going to focus on the instantiation of a new data pipeline built inside the BDP, for this reason it was necessary to introduce in details the major features and the components used in the architecture.

In conclusion, the BDP is a powerful and flexible framework for the processing and management of data, ensuring both system administrators and users smooth access to the logs ingested in the infrastructure and enabling efficient operational monitoring. Its architecture is designed to be scalable and adaptable, allowing the integration of new data sources and processing components as needed. Thanks to these features, the BDP represented the fundamental element through which the injection of data logs recalled from the ATLAS experiment database (PanDA) was made possible.

In summary, this chapter has outlined the institutional context of INFN and CNAF, the technical role of the Tier-1 data centre, and the architecture of the Big Data Platform that supports large-scale data management and monitoring. These

foundations are essential to frame the environment in which the research presented in this thesis has been carried out. Building on them, the next chapter will focus on the design and implementation of a dedicated pipeline that connects the ATLAS PanDA workload management system to the CNAF BDP, marking the first step towards the anomaly detection framework developed in the later stages of this work.

Chapter 3

From PanDA to the BDP: data pipeline design and implementation

3.1 A Data Pipeline from CERN to the BDP

Having already introduced the working framework and the resources provided by CNAF, it is possible to proceed with the description of the data source introduced for the purpose of the later analysis. In practical terms, the new established data pipeline starts from the PanDA system at CERN, and reaches the data nodes inside the BDP at CNAF.

Before entering into the full chain of the data processing, it is important to clarify the need of this data moving from the servers at CERN to the ones in Bologna, since one might think that since data is available at CERN, there is no need to move it elsewhere. The reason behind this choice is actually straightforward and regards mostly the fact that inside PanDA, depending on the computing site configuration and the specific requirements of the analysis jobs, logs from the jobs are stored for a limited amount of time, and then deleted. In particular, for the specific case of the Italian sites, at maximum, the data logs are available for 96 hours, but only for the Tier-1 computing site (i.e. CNAF's ones), and 48 hours for all the other lower tiers. Having the logs available for a limited time poses a challenge for the long term analysis which could be performed on them, and for this reason, also to build a long history of the logs internally at CNAF, it was decided to move the data to the BDP.

The data pipeline was designed to be as automated as possible, actually being fully independent of human intervention, apart from initial setup and technical occasional maintenance. As already mentioned, the first chain link to start the pipeline is the PanDA system. PanDA (i.e., the Production and Distributed Analysis system) is ATLAS' workload management system, designed to work at LHC scale, orchestrating both production and distributed analysis workloads [8]. It operates across a vast distributed computing infrastructure of roughly 200 data centres in more than 40 countries. PanDA supports an exabyte-scale data volume, able to manage complex workflows and millions of tasks per day, while ensuring high availability and reliability. The system is built on a modular architecture, made of several key components:

- The **JEDI** (i.e., Job Execution and Data Interface) is an engine designed to optimize workload utilization across several resources. It is the interface responsible for tasks progressing, jobs management and efficient use of computing resources.
- The **PanDA Servers**, which act as the central hub for job management, scheduling, and monitoring. It handles job submission, prioritization, and resource allocation.
- The **Pilots** and **Harvesters**, agents that run on worker nodes within the distributed computing infrastructure. They are responsible for fetching jobs from the PanDA Server, executing them, and reporting back the results.
- The **PanDA Monitor** (i.e., BigPanDAmon), a web-based tool which provides real-time monitoring and visualization of job status, resource usage, and system performance.

The PanDA monitor system is the actual key through which it is possible to access jobs and their relative logs. In fact, like the BDP, the monitor suite provides summary dashboards filterable by tasks, jobs, sites and users (an example dashboard is shown in Figure 3.1). The dashboards access is web based, allowing users to easily

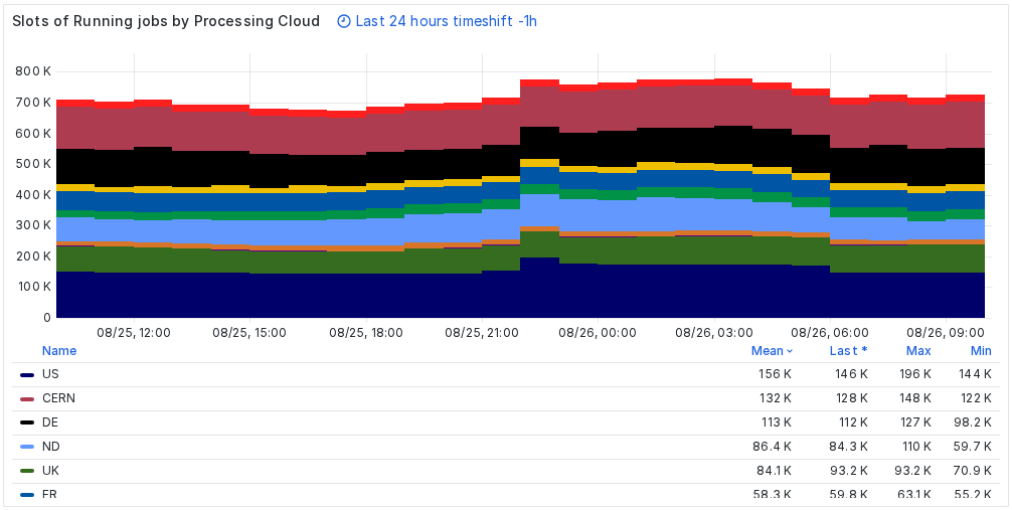


Figure 3.1: *One of the dashboards of the PanDA monitor interface, showing a summary of the jobs executed in the last 24 hours in different computing sites.*

navigate and filter the information they need. Moreover, the monitor system provides a REST API, which allows programmatic access to the data, enabling users to retrieve tasks list and jobs lists by the usage of a URL, in which it is possible to apply specific filters. The API returns the results in JSON format, making it easy to integrate with other tools and systems. For example, the API to retrieve the tasks list is `https://bigpanda.cern.ch/tasks/`, inside which is possible to apply filters like `user`, `date`, `status` and more, in order to get only the tasks of interest. The same applies to the jobs list, but in particular, thanks to the API, it was possible to retrieve jobs reports with the url `https://bigpanda.cern.ch/jobs/?json&hours=1&jobstatus=finished|failed&computingsite=INFN-CNAF|INFN-CNAF_ARM&outputs=`

`jobs`, which redirected the research inside PanDA of all the jobs executed in the last hour, with status finished or failed, and executed exclusively on `INFN-CNAF` and `INFN-CNAF_ARM` computing sites. The output of the API call is a nested JSON file, containing a summary report of multiple jobs, distinguished by a `pandaid`, so basically all the information related to the jobs can be retrieved using this unique identifier. An example of a job report and how it is displayed in JSON format is shown in Listing 3.1.

Listing 3.1: *Example of a simplified JSON response from the PanDA API. Only a subset of fields is shown for clarity, while the full output includes many additional job-related attributes.*

```
{
  "jobs": [
    {
      "pandaid": 1234,
      "jobstatus": "finished",
      "computingsite": "INFN-CNAF",
      "starttime": "2025-08-26T14:29:55",
      "endtime": "2025-08-26T15:02:05",
      ...
    },
    {
      "pandaid": 5678,
      "jobstatus": "failed",
      "computingsite": "INFN-CNAF_ARM",
      "starttime": "2025-08-26T14:30:00",
      "endtime": "2025-08-26T15:02:10",
      ...
    },
    ...
  ]
}
```

It is important to note that direct access to the API is restricted, since it requires proper authentication and authorization. Users must hold an active CERN account, be part of the ATLAS collaboration, and have a valid X.509 certificate[1] installed in their browser to access the dashboards. While this is sufficient for interactive use, it does not allow programmatic retrieval of the data directly from the web interface. To automate the extraction process, a dedicated script was deployed on a virtual machine connected to the BDP, equipped with the necessary credentials. This virtual machine acts as the producer node of the ATLAS pipeline, ensuring continuous ingestion of PanDA job reports into the platform.

[1] An X.509 certificate is a digital certificate issued by a trusted Certification Authority (CA) that proves the identity of the user.

3.2 BDP components setup

It is now time to come back to the BDP, starting to describe its configuration, as already anticipated, it all started with the virtual machine used as a producer node for the ATLAS data pipeline.

3.2.1 Producer node configuration

In a first instance, the virtual machine was configured with the CentOS 07 operative system, but later it was migrated to AlmaLinux 9 (since the CentOS 7 reached its end of life), keeping intact the main producer structure. In details, the machine was configured since its birth to extract jobs reports from PanDA, periodically querying the API and storing temporary results in a small local database in JSON-formatted logs. Since the direct access to the API is restricted, the producer machine must first establish a valid session with CERN's authentication services before executing any scripts to recall data. This authentication procedure requires two consecutive steps:

- Kerberos[2] initialization, since CERN relies on it for authentication, each user must obtain a Kerberos ticket by executing the command `kinit`, which requires valid CERN credentials, and when performed it grants a temporary identity proof to access protected web services;
- PanDA Cookie[3] generation, with a valid Kerberos ticket, it is possible to obtain a PanDA cookie, which is essentially a small file stored locally on the machine, which allows direct communication with the monitor system, practically replacing the manual login, allowing scripts interaction without user intervention.

These two steps together establish a temporary, authenticated session, which allows automated scripts to interact securely with the PanDA API. Once these authentication factors are in place, the producer query can actually start with the operation of `bash` and `python` scripts.

In order to start the first recall of the query, a script called `panda-reports.sh` is executed, requiring the dashboard access to the reports filtered with the URL already shown in section 3.1. In this first part of the access, the download performed saves in the local machine a JSON file, which is nested, including the all the jobs reports retrieved from the API of the hour before the script is launched. This script is set to be executed at the start of each hour, except for the first one of the day, in which it is executed at 00 : 05 after the authentication procedure. Half an hour after the download, a second script, written in python and called `json-arrangement.py`, is executed. The code opens the JSON file downloaded before, and loops inside the field `jobs`, extract each job report individually in a single field JSON file, named with its `pandaid`, and storing temporarily it in a local database. After the writing

[2]Kerberos is a network authentication protocol designed to provide strong authentication for client/server applications through secret-key cryptography.

[3]The PanDA cookie is a session token generated after Kerberos login, stored locally as a file, and reused by scripts to authenticate API requests to the PanDA monitor."

the same script performs a data forwarding operation of the single JSON, by sending it to Kafka, since in Python is possible to include a library from Kafka, allowing to set inside a script a producer object, which is able to send data to a specific topic. In this case, the chosen topic was `atlas`, since as mentioned in subsection 2.2.1, one of the main goals of the BDP is to collect data from all the major experiments at CERN.

The operations of downloading, cleaning and forwarding are performed in loop every hour, every day of the week. At the end of each daily loop, at 23 : 55, a last script called `report-removal.sh` is executed. The script accesses the folder containing the temporary cleaned reports and removes them (specifically the ones collected during the day). The removal is necessary since the virtual machine has a limited memory capacity, and also because, since the data has already been forwarded to the BDP, data is also accessible through the dashboards. In order to give a graphical view of the scheduled loop operations inside the producer machine, a schematic representation is shown in Figure 3.2.

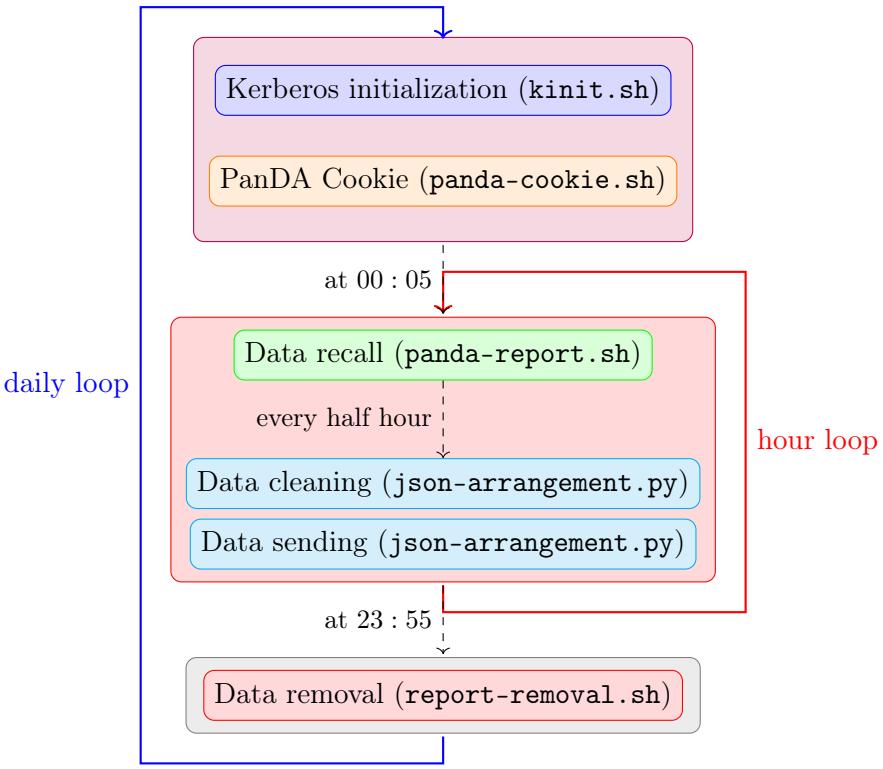


Figure 3.2: *Authentication and automated extraction loop running inside the producer node for the ATLAS pipeline. Each daily cycle: obtain Kerberos ticket, generate PanDA cookie, enter the hour loop, exit at the end of the day and remove already sent reports. Each hour cycle: query PanDA API, clean data, forward to Kafka.*

In order to set these loops automatically, and collect eventual log errors from the launched command, a cronjob schedule is set to be executed inside the machine. In case some issues are encountered in the schedule of the producer, logs are always available to search for specific problems, and moreover, thanks to the fact that the data forwarding is discretized, is even easier to locate specific issue by knowing the

time it happened, since the logs report the forwarding time to Kafka.

The user operating the scripts is the only one configured inside the machine (called as the operative system `almalinux`), and it has all the necessary permissions to perform each required action in the daily `crontab` schedule. An example of the `cronjob` file is shown in Listing 3.2.

Listing 3.2: *Example of the cronjob set in the producer machine.*

```
# Kerberos and PanDA authentication
0 1 * * * almalinux bash $ATLAS_DIR/scripts/kinit.sh >> /tmp/kinit
    .log
5 1 * * * almalinux bash $ATLAS_DIR/scripts/panda-cookie.sh >> /
    tmp/panda_cookie.log
# Data recall
0 * * * * almalinux bash $ATLAS_DIR/scripts/panda-report.sh >> /
    tmp/panda_report.log
# Data cleaning & forwarding
30 * * * * almalinux python3 $ATLAS_DIR/scripts/json-arrangement.
    py >> /tmp/json_arrangement.log
# Daily data removal
55 */2 * * * almalinux bash $ATLAS_DIR/scripts/report-removal.sh
    >> /tmp/report_removal.log
```

A detailed view of the scripts used in this cronjob is provided in Appendix A (not displaying though sensitive date used inside them).

After the full configuration of the new producer node, the further step was to set up the broker and the consumer nodes, in order to be able to receive, filter and send to storage data, under the new inserted `atlas` topic.

3.2.2 Broker node configuration

The Kafka broker nodes of the BDP were already active and working at the time of the pipeline instantiation (already set with some CNAF topics). The broker has a key role in the whole BDP infrastructure, since it mediates the data communication between the producers, the consumers and within the brokers themselves. The cruciality of the broker is in ensuring a secure data traffic, with the utilization of either a keystore and a truststore[4] in each machine involved in the Kafka cluster.

Anyway, pretending to have to configure from scratch the broker, the installation procedure requires the machine in which Kafka is installed to have some prerequisites. First, the installation requires to have `Java` installed, since some of its packages are needed to run Kafka. Both the installations may be performed from command line, using the package manager specific to the operating system. After the first installation, the first action required is to enable SSL communication inside the `server.properties` file, which is the main configuration file of the broker. This configuration file grants the secure communication both internally the broker cluster and externally with the clients, and moreover the file allows Kafka to listen on multiple SSL ports, each of which can be configured with its own security settings.

[4] A keystore contains the broker's private key and its signed certificate, proving its identity to others. A truststore contains the list of trusted certification authorities (CAs) and is used to verify the certificates presented by other brokers or clients.

An example of the SSL file configuration, and in particular the parameters involving the SSL communication, is shown in Listing 3.3.

Listing 3.3: *Example of the SSL configuration inside the* `server.properties` *file. The* $*$ *symbol represents a placeholder for the actual port numbers and passwords.*

```
ssl.keystore.location=/root/keystore.jks
ssl.keystore.password=*******
ssl.key.password=*******
ssl.truststore.location=/root/truststore.jks
ssl.truststore.password=*******
```

At this point, the broker is configured to use SSL for secure communication. The further steps regard both the actual creation of the truststore and the keystore. The first one is the container of the CA (Certificate Authority) certificates, which in the case of the BDP is the `Puppet CA`, while the second one is the container of the broker's private key and signed certificate (which is unique for each server connected to the broker). A more detailed view of this process is shown in Appendix A.

These security controls mat be set directly in the Kafka configuration file, where it is possible to specify the locations and passwords of the keystore and truststore. From that point on, all internal traffic between brokers, and external traffic between either producers or consumers and brokers, occurs over an encrypted channel. A schematic representation of the SSL/TLS authentication interaction in Kafka is shown in Figure 3.3.

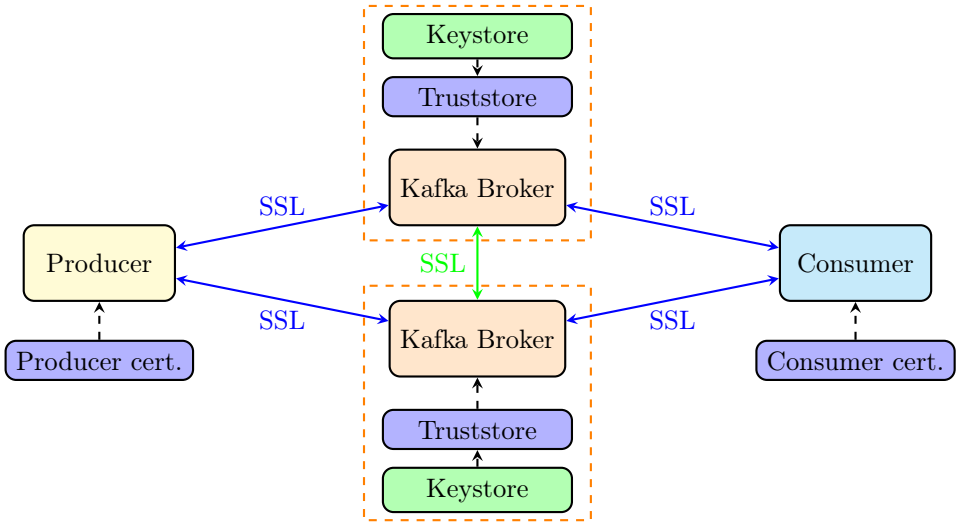


Figure 3.3: *Schematics of SSL/TLS communication. Each broker uses a keystore, its private key and signed certificate, to communicate internally with other brokers, and a truststore, where all certificates for secure communication are stored. Producers and consumers similarly have their certificate to communicate with the brokers. Communication happens over an encrypted channel, while Access Control Lists (ACLs) inside the brokers regulate access to topics.*

Those security connections allow secure data forwarding between the broker's cluster and the clients, while the internal communication channel is also secured, granting an internal coordination within the cluster's element, avoiding the reception

of connection requests from unauthorized or untrusted brokers. The authentication of clients may also be reinforced with SASL (i.e., simple authentication and security layer) mechanisms, which in particular for the broker inside the BDP is set with a username-password access over SSL. On the other hand, after the establishment of the connection, the various authorizations to access data are handled via Access Control Lists (ACLs), which define users or services allowed to perform operations (e.g, read, write or manage) on specific Kafka topics.

As mentioned in subsection 2.3.1, it is at this level of the infrastructure that the data are organized into *topics*, which are basically logical streams of messages through which data flows. It is possible to subdivide them into partitions, distributed inside the cluster, enabling horizontal scalability (by spread of data loads) and fault tolerance (by data replication in different servers). For the purpose of the work, a new topic was introduced in Kafka, categorizing the incoming data from the producer node with the topic `atlas`. The topic was created with a script like the one presented in Listing 3.4, in which are specified the configuration command, the active Kafka servers, the topic name, number of partitions and replication factor.

Listing 3.4: *An example of the command to create a Kafka topic. The* `amdinclient-config` *file is required to perform inside the cluster operations with admin priviledges*

```
kafka-topics.sh \
  --command-config adminclient-config.conf \
  --bootstrap-server kafka01.cnaf.infn.it:****,\
                     kafka02.cnaf.infn.it:****,\
                     kafka03.cnaf.infn.it:****  \
  --create --topic atlas \
  --partitions 3 \
  --replication-factor 3
```

After the creation, a set of commands allow to list the whole set of topics available inside the cluster and their information, but also it is possible to perform direct actions on them, like authorization modifications or even deletion of a topic itself. More details on these operations can be found in the Appendix A. Once a topic starts to be populated by a producer, the consumers authenticate with Kafka and subscribe to a relevant topic of interest, and retrieve the incoming data stream in real time. In order to explain the process to enable authorization in a consumer node, and more generally to configure from scratch a consumer server, it is possible to enter a detailed description on the installation and setup of a Logstash server, the data shipper bridge between Kafka and the OpenSearch storage.

3.2.3 Consumer node configuration

The main solution adopted for a consumer node inside the BDP is the implementation of a Logstash instance inside a server to be connected the BDP, its node is responsible for data recalling from the broker according to a specific topic, and forwarding it to the storage, indexing it in OpenSearch. As anticipated in subsection 2.3.1, the initial configuration of the BDP included only a single Logstash instance as a main consumer for all the active topics, but at the time of the ATLAS pipeline instantiation, the need to separate the external data sources

from the CNAF internal ones became a priority. For this reason, a new Logstash instance was introduced as an external machine, to be eventually connected to independently in the BDP architecture. The new instance was configured to include the `atlas` topic, and probably also other new data entries from other sources will be instantiated inside this node.

With respect to the previous implementations, for both the producer and the broker, it actually required more passages to implement the necessary version of the Logstash software, since the downloadable version of any system package manager did not include a necessary plugin responsible for the output communication with the OpenSearch nodes. Instead, Logstash was installed and configured manually, working as a system service (actionable via `systemd`) always running. The version installed in the server is the build 8.9.0, since it was the last inside which the building files for the OpenSearch plugin were included in the Logstash distribution folder.

The manual installation requires manual download of the correct package with `wget`, the unpacking of the package with `tar` and lastly the setup of the file `logstash.service` in the operative system. The last operation is required to prepare in advance the Logstash instance, since it allows later to perform via it the command line to install the OpenSearch plugin. Once this first part is completed, is possible to check if the installation was successful by running a simple command from the terminal. The whole process to reach this point, is presented in more details in Appendix A.

After the installation, the further step is to modify and configure the Logstash configuration files, in order to make it communicate with the other components of the BDP. The files responsible for the Logstash configuration are typically located in the `/etc/logstash/config` directory, and the ones to be modified are three, each of which is responsible for a specific task:

- `input.conf`, which specifies the input data source, to which the consumer search for the topics, including also the reference for the SSL truststore file for authentication;
- `output.conf`, which redirects now locally saved data from Kafka to the OpenSearch cluster, using the same credentials created for the Kafka broker;
- `filters.conf`, which applies various additions or cuts in the JSON to be sent to the storage, in order to add searchable metadata and standardize the data format.

The default versions of those files are usually provided with a normal installation, but in case they are missing, they can be added manually in the Logstash configuration folder. The three files are written in a specific syntax, provided by Logstash, and a partial example of each of them is shown in Appendix A.

Thus, this completes the consumer setup, completing entirely the pipeline: starting from automated log extraction at the producer side, through secure storage and management in Kafka brokers, reaching the consumer for a final data cleaning and reshaping, to an eventual ingestion into OpenSearch for real-time monitoring and long-term storage.

3.3 ATLAS Data inside the BDP

After the completion of the full data pipeline setup, the ATLAS data began to flow into the BDP architecture, being actually visible and accessible through the OpenSearch Dashboards. The data ingestion started in July 2024, after some preliminary tests, all the components were tuned to work in a stable way, and the data started to be collected continuously. The OpenSearch cluster is the final destination of the data, where it is eventually stored and indexed for future analysis. The indexing is received by the master nodes directly from the Logstash consumer, addressing the sharded data into the four data nodes.

The data visualization is performed through the OpenSearch Dashboards, which is the user interface of the BDP, a tool which has direct access to the data nodes of the cluster. The web interface provides a user-friendly way to explore, visualize, and analyze the ingested data. The dashboards are customizable, allowing users to create visualizations, charts, and graphs based on their specific needs. Moreover, the dashboards includes a set of specific "tenants" (i.e., workspaces), each of which is dedicated to a specific set of indices to which is possible to operate queries and analysis. Those tenants are used both by the administrators, in order to have a general overview of BDP's health and data flow, and by the users, who can create their own tenants to perform analysis on specific data, or require data to be extracted in a specific format.

After having access to the page of the dashboards, it is possible to select one of the user's available tenants, and start to explore the data. For instance, the tenant created and dedicate to ATLAS can be selected from the list of topics available, in order to show the incoming data flow from the `atlas` topic in function of the injection time. An example of the OpenSearch Dashboards interface, called *Discover* dashboard, is shown in Figure 3.4.

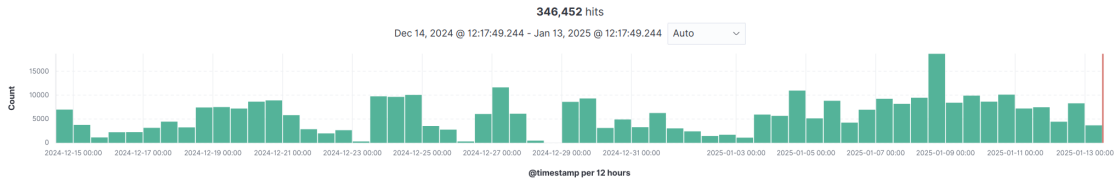


Figure 3.4: *The Discovery dashboard interface, showing all the incoming logs for the topic* `atlas` *in function of the timestamp.*

Having at disposal a tool like the dashboards, it was possible to create custom dashboards, in order to have a detailed monitoring of the ATLAS jobs executed in the Italian computing sites. In the further section, some example dashboards are presented, showcasing the various visualizations and insights that can be derived from the ATLAS data.

3.3.1 Dashboards view of ATLAS data

The dashboards which can be recreated inside the OpenSearch interface are multiple, and at service of the user. One of the most powerful features of the dashboards is the possibility to select different time ranges or periods (from a

minimum of one second up to several years), allowing users to focus on specific timeframes of interest. For the purpose of this section, the data displayed from now on will be referred to the period between July 2024 and May 2025, which is the time period in which the data collected up to that moment were downloaded in a single large dataset, in order to perform later analysis outside the BDP (specifically in section 4.1 the dataset recall and preprocessing will be described in more details). Moreover, the data displayed will have a small gap in the end of October 2024 until the beginning of November 2024, since in that period a small maintenance was made to the BDP infrastructure (i.e., physical movement of some machines), and also because after that period the pipeline broke down, and in few days was fixed again.

One of the main objectives of ATLAS data collection was to have a long-term monitoring of the jobs executed in the Italian computing sites, but also to have a control environment on distributions of jobs inside the computing machines of the Tier-1 site. For this reason, a first dashboard was created to monitor the jobs executed in the CNAF computing machines, both for daily distributions or for long periods. An example of a reconstruction of this dashboard is shown in Figure 3.5, in which are displayed the number of jobs executed in the CNAF computing machines over time.

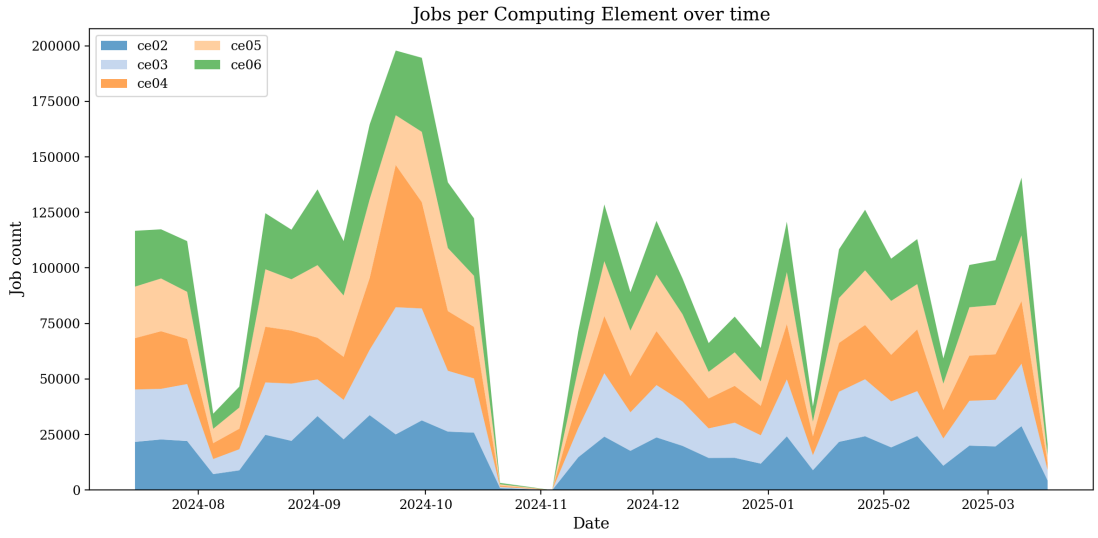


Figure 3.5: *The computing distribution dashboard, showing the number of jobs executed in the CNAF computing machines over time from ATLAS jobs. The different colors represent the computing elements inside the CNAF site.*

Having at glance the distribution of jobs over time, it is possible to monitor the activity of the site, and also to identify which is having the most activity or either some issues in the functioning. In fact, possible issues may be detected directly from the dashboard, since setting a specific time range and a filter on the field `jobstatus`, allows highlighting the jobs that failed during their execution. In particular, a plot including the ratio between failed and finished successful jobs was created, in order to have a direct view of the percentage of failed jobs inside the selected dataset to be used for later analysis. The plot is shown in Figure 3.6,

where the time series is identical in shape to the one shown in Figure 3.5.

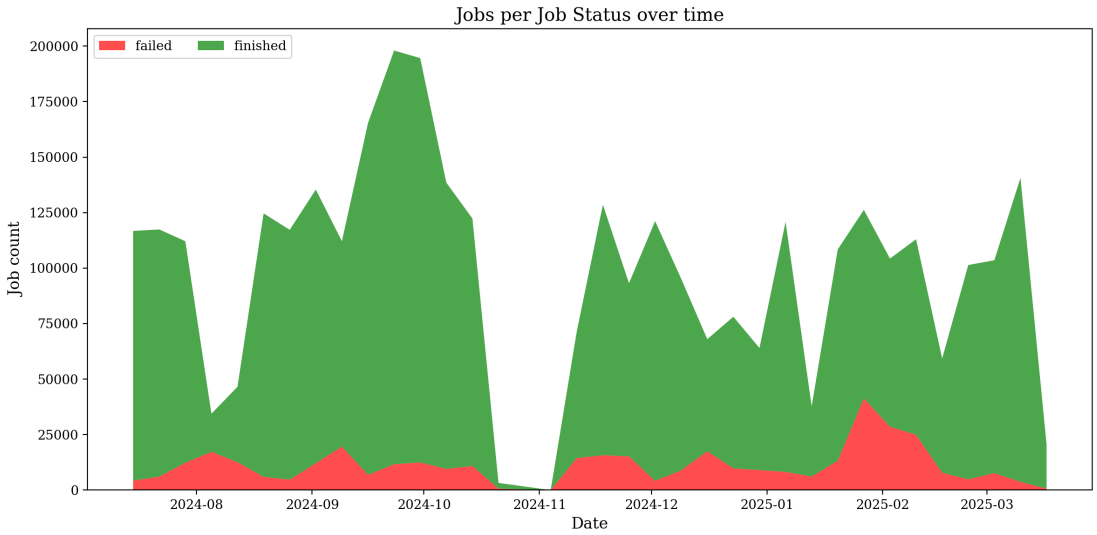


Figure 3.6: *The time-series showing the ratio between failed and finished jobs executed in the CNAF computing machines over time from ATLAS jobs.*

An important aspect of the possibility to filter the data in the dashboards, is the capability to select specific fields to be displayed. In this particular case, the field `jobstatus` was used to plot the status comparison time-series, but also to start to select and display the actual error messages of all the failed jobs. A recreation of the dashboard created to monitor the error messages is shown in Figure 3.7, in which the top 20 most recurrent error messages are displayed.

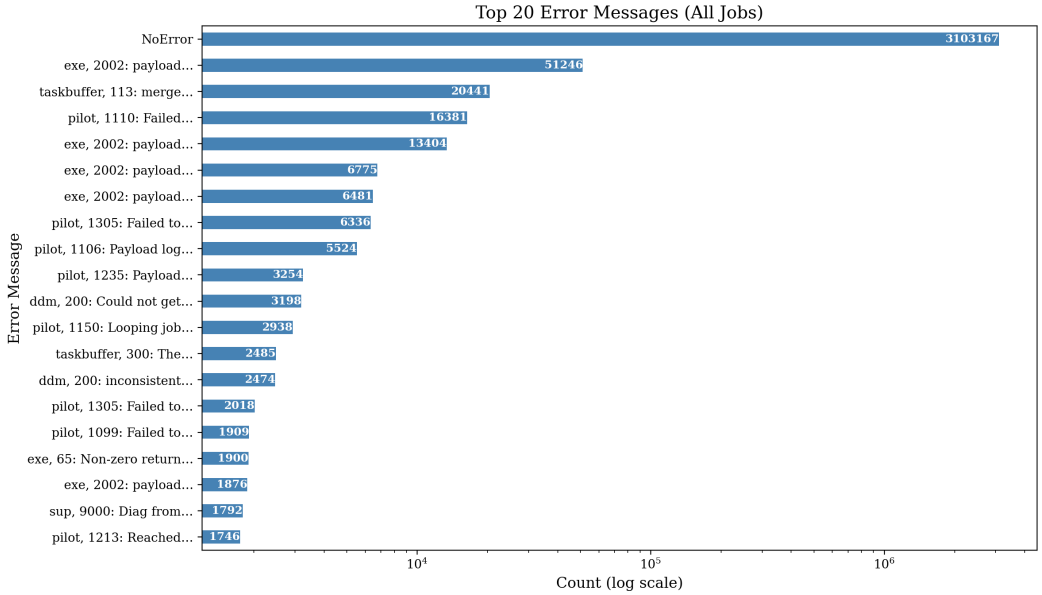


Figure 3.7: *The top 20 most recurrent error messages encountered during the job execution in the CNAF computing machines. The insights of this plot enhanced the imbalance between error and non-error messages.*

The plot was created thanks to the selection of the field `errorinfo`, which gives a general description of the error encountered during the job execution, including though when present a very long string of text, not easy readable from the dashboards. As it was already clear for the previous plot, the logs containing error messages different from `noError` are a minority in the dataset used, since the most populated error message is one order of magnitude lower than the logs with no errors (a more detailed discussion on the dataset handle and preparation will be given in section 4.1). Despite that, the construction of this dashboard was the first step that led to the subsequent analysis, since a more detailed focus on the errors was posed. Removing the `noError` entries, it was possible to have a more detailed view of the order of magnitude of the other error messages, and to have a clearer view of the most recurrent issues encountered inside the dataset. A recreation of this second dashboard is shown in Figure 3.8.

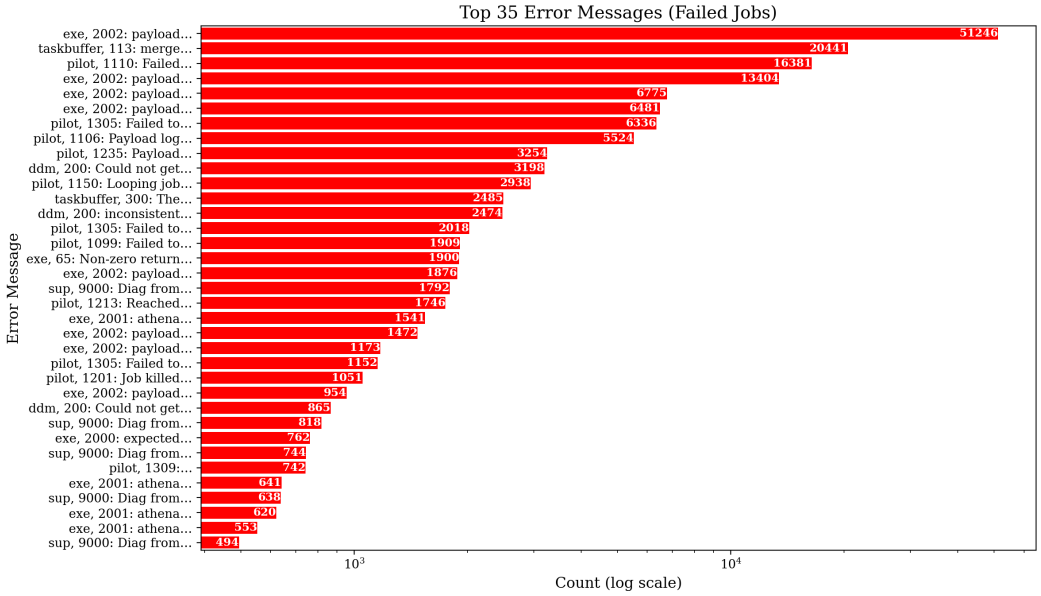


Figure 3.8: *The top* 30 *most recurrent error messages excluding the ones without errors. The error logs are in minority with respect to the non-error messages, but with a high multiplicity of different error types.*

It was actually from the last plot that the most interesting information emerged, since the multiplicity of different error messages was, in theory, very high, but most of them were almost identical, differing only in minor details (e.g, different file names or paths, different serial numbers). Eventually, the ultimate idea was to extract those messages from the dashboards, leading to subsequent data preparation as a base dataset for the machine learning algorithms whose scope is to classify the error messages received.

The last development of work presented in this thesis is presented in the next chapter, starting from the data extraction into a unique database and preprocessing, reaching the training, testing and evaluation of all the models developed to reach an automated error classification.

Chapter 4

A framework for anomaly detection

4.1 Data retrieval and preprocessing

The design and implementation of the ATLAS pipeline connecting to the BDP, as described in chapter 3, lead to an accumulation of significant amounts of job report over time. This growing dataset of logs was almost constantly fed with new incoming reports, creating a valuable resource for analysis, all rearranged and indexed as a JSON file in the BDP. One of the first tasks was to retrieve locally the dataset, in particular in order to create a single `.csv` file as the main dataset for the machine learning model. In order to do so, a custom Python script was created to query the BDP API, which by selecting a required topic, provides the possibility to download all the entries of the chosen topic in a single file instance.

Apart from a minor issue related to some nested fields inside the JSON structure, which required a simple flattening procedure, the data retrieval was quite straightforward to build up the main dataset.

As already mentioned, the reports stored and used as the base dataset were collected in a time range from July 2024 to May 2025, resulting in a total of approximately 3.5 million job reports. Also, it is important to recall that each report regards only jobs executed directly on the INFN-CNAF Tier-1 farm, via a redirect managed by PanDa, as specified in subsection 3.2.1.

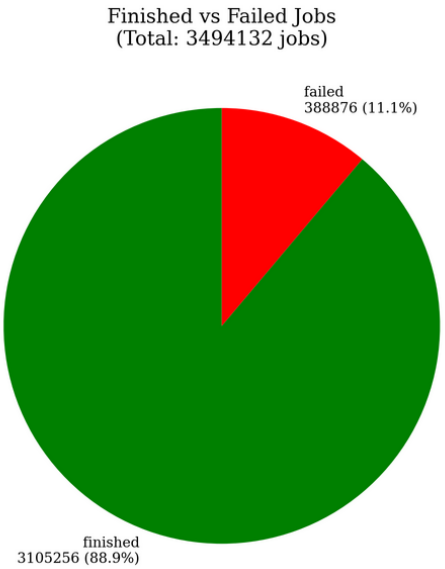


Figure 4.1: *Pie chart showing the percentage of finished and failed jobs in the main dataset.*

Inside those recalled files, the overall job status chart is shown in Figure 4.1. It is clear that the obtained dataset is highly imbalanced, with the majority of jobs being successful without error messages, being roughly 90% of the total reports. The remaining 10% of the logs contain explicit error messages, with a high apparent multiplicity mostly due to serial number differences inside the error

string or smaller text differences representing the same errors. The disparity inside the dataset could be in principle a source of problems regarding the actual training of the algorithms, but it actually led to a preparation of several datasets all randomly extracted from the main database, in order to train the same algorithms with different data allowing a performance comparison.

Although the data retrieval and preprocessing pipeline described in the previous section ensures consistency and scalability, some intrinsic limitations of the dataset must be acknowledged. The job reports originate from a specific operational context, namely the ATLAS workloads executed at the CNAF Tier-1, and therefore reflect the characteristics, scheduling policies, and infrastructure conditions of this environment. As a consequence, the dataset may not fully capture a high variability across the different sources.

Furthermore, the natural class imbalance between successful and failed jobs represents an inherent bias of the operational data, which may still influence the learning process. Additional limitations arise from the heterogeneous quality of the error-related fields, such as the `errorinfo` attribute, which can be incomplete, noisy, or inconsistently populated depending on the failure mode. These aspects are intrinsic to large-scale distributed systems and must be taken into account when interpreting the results of the machine learning models presented in the following sections.

Being aware of the limitations related to the original dataset, the strategy adopted to perform data preparation was split into two phases:

- the first configuration regarded the preparation of three different general datasets, extracted entirely from the main database, in order to later train a decision tree to classify job reports according to the parameter `jobstatus`;
- in a second instance, only the jobs with error messages (i.e., `failed`) were extracted from the main dataset in order to create a base for two multi classifier algorithms.

The decision to split the main dataset into different configurations was driven to evaluate the performance of the models in different scenarios, especially considering the class imbalance. For the binary case, it was actually possible to modulate the ratio inside the three preprocessed datasets, being the main dataset highly populated. Instead, for the multi-classifier case, the limitation in the number of entries with error messages led to the creation of only two separated datasets, acting directly not on the ratio, but rather in the multiplicity and diversity of the error messages.

Before entering into further details on the preprocessing, to eventually reach the development of the machine learning algorithms and their actual performances, the preliminary configuration of all the dataset is presented in the next section.

4.1.1 Binary datasets preparation

The first step to move while building machine learning algorithms is actually the preparation of a proper dataset to feed to the learning model. In the case of the first dataset configuration since the first algorithm task is the binary classification

of the `jobstatus` parameter, the manipulation and preparation of the datasets occurred as shown in Figure 4.2.

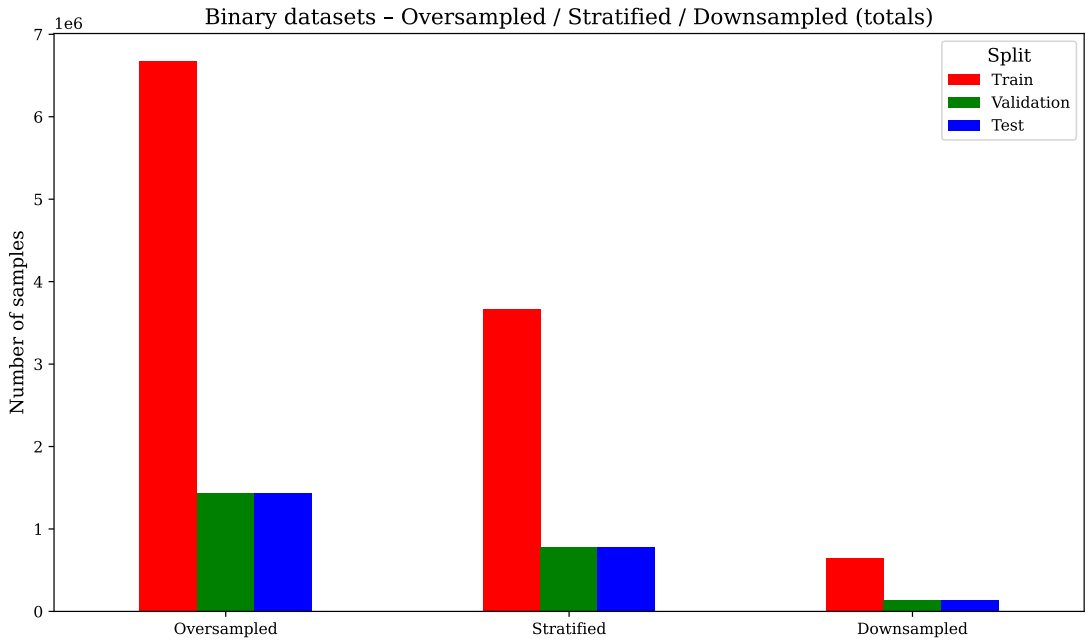


Figure 4.2: *The comparison of the various datasets built to train the binary classifier: over-sampled, stratified and down-sampled. The three schemes differ in the ratio of* `failed` *jobs over* `finished` *jobs.*

As already mentioned, having at disposal a very large dataset with a huge discrepancy between the number of finished and failed jobs, led to a preparation of three different subsets. Each of them was built in accordance with peculiar parameter, which could be set thanks to the high number of entries of the main dataset: the three subsets differs from each other in the ration of failed over finished jobs. This choice was driven by the will to evaluate the performance of the model into three different case scenarios, still using as main data source the entries of the original dataset.

For each of the splits, a train, validation and test set were created, by sampling randomly from the main dataset. The three sub-splits are composed as it follow:

- **Over-sampled datasets**, where the number of failed jobs was artificially increased by randomly duplicating reports, reaching a highly populates set with a $1:1$ ratio between failed and finished jobs;
- **Stratified datasets**, a simple stochastic extraction of reports from the main dataset, keeping the original ratio of roughly $1:10$ between failed and finished jobs;
- **Down-sampled datasets**, where the number of finished jobs was artificially reduced by randomly erasing reports, reaching a smaller but balanced set with a $1:1$ ratio between failed and finished jobs.

In order to show the results of the prepared datasets, the actual entries of each dataset of each split are reported in Table 4.1.

Table 4.1: *Binary classification datasets composition with relative ratio of* `failed` *jobs over* `finished` *jobs.*

(a) *Over-sampled datasets and actual job reports counts.*

Over-sampled	Job Reports			Ratio
Dataset	Total	Finished	Failed	$\frac{\text{Failed}}{\text{Finished}}$
Train	6 675 895	3 337 948	3 337 947	1.00
Validation	1 430 549	715 274	715 275	1.00
Test	1 430 550	715 275	715 275	1.00

(b) *Stratified datasets and actual job reports counts.*

Stratified	Job Reports			Ratio
Dataset	Total	Finished	Failed	$\frac{\text{Failed}}{\text{Finished}}$
Train	3 661 559	3 337 948	323 611	0.10
Validation	784 620	715 274	69 346	0.10
Test	784 620	715 275	69 345	0.10

(c) *Down-sampled datasets and actual job reports counts.*

Down-Sampled	Job Reports			Ratio
Dataset	Total	Finished	Failed	$\frac{\text{Failed}}{\text{Finished}}$
Train	647 222	323 611	323 611	1.00
Validation	138 691	69 346	69 345	1.00
Test	138 691	69 345	69 346	1.00

The difference in the configuration was done in order to evaluate the performance of the models in three different scenarios, one with a realistic but imbalanced dataset, one with a balanced but smaller dataset, and one with a balanced and larger dataset. The evaluation of the models and their performances will be later described in section 4.2.

4.1.2 Multi-classifier datasets preparation

Instead, the second configuration of the datasets regarded only the failed jobs, in order train a classifier to recognize the different types of errors. In this case, the preparation of the datasets was more simple, since only 2 different splits were created, still composed of a training, validation and test set. The actual entries of each dataset of each split are reported in Table 4.2.

The difference of the datasets regards the presence of duplicate error messages, leading to a creation of a dataset with only unique errors, and a second with all the real entries, including duplicates. The relative ration of unique over duplicate error messages is roughly $1 : 2$, but it is not sufficient to describe the real difference between the two datasets. The actual main work for the preparation for these

Table 4.2: *Multi-classifier datasets summary. The ratio between unique and duplicate error messages is roughly* $1:2$.

	Failed Job Reports		
Dataset	Unique	Duplicates	$\frac{\text{Unique}}{\text{Duplicates}}$
Train	141 205	272 211	0.52
Validation	30 258	58 331	0.52
Test	30 259	58 331	0.52

datasets regarded the text processing of the error messages, in particular the creation of a vocabulary to convert complex text strings into first numerical vectors, and then associate them to a specific error category. The extracted categories are the following:

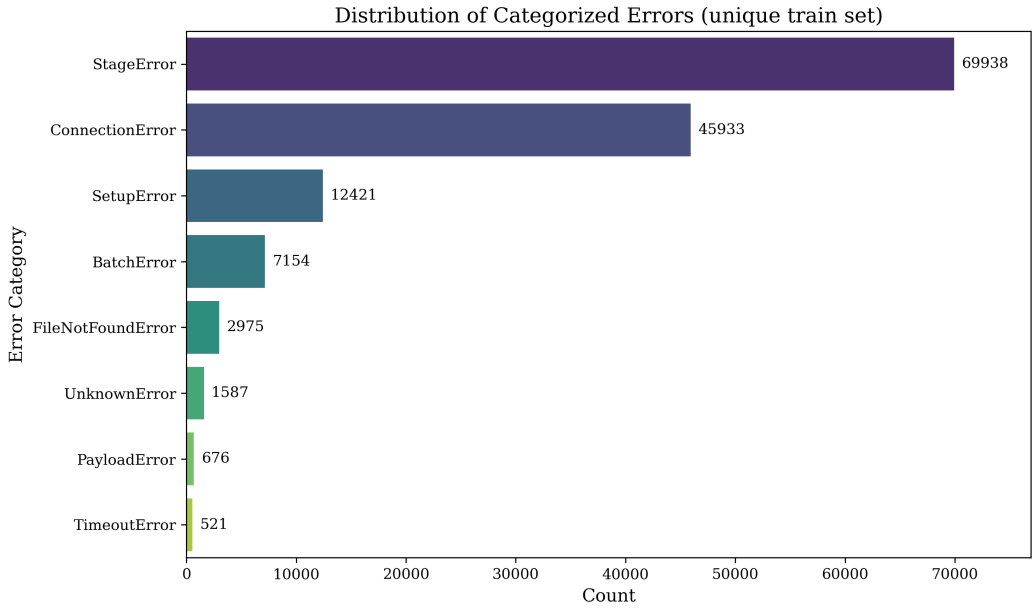
- `StageError`: errors occurring during the staging phase, related to preparation or movement of input/output data and resources before execution;
- `ConnectionError`: network or connectivity issues, such as socket failures, timeouts, or lost connections during job execution;
- `SetupError`: problems during the initialization or configuration phase of the environment;
- `BatchError`: failures linked to the submission or scheduling system, including batch queue management issues;
- `FileNotFoundError`: missing input or output files required by the job for correct execution;
- `ImageCompatibilityError`: errors due to incompatibilities between the job's execution environment and the required software image or container;
- `PayloadError`: errors arising from within the job's own application payload or user code;
- `TimeoutError`: jobs that exceeded the allocated runtime or stalled waiting for resources;
- `UnknownError`: uncategorized or unrecognized failures not falling into the defined categories.

It is now clear that the two dataset have a very different overall distribution of the error categories. For instance, the unique dataset was built with the aim to have a more balanced distribution, but also to enhance the error diversity more than error multiplicity. On the other hand, the duplicate dataset is more similar to a real case scenario, where error messages are often repeated several times, thus leading to a more imbalanced and biased distribution.

In particular, as the two plots show, some categories are very well populated in both the datasets, like `SetupError` and `ConnectionError`, while others are only appearing once in the duplicate dataset, since their diversity in therms of non repetition was too low to generate a well populated category of unique entries, like `ImageCompatibilityError`. Moreover, for the same reason, a category has a very different weight in the two datasets, like for example the `PayloadError` category, which is the second most populated in the duplicate dataset, while it is one of the least populated in the unique dataset. Nonetheless, the difference in the

distribution of the dataset is a good test to later evaluate the performance of the models in two different scenarios. The actual distribution of the various categories in the training sets of the two datasets is shown in Figure 4.3.

(a) Distribution of the dictionary categories inside the training set of the unique error dataset.



(b) Distribution of the dictionary categories inside the training set of the duplicate error dataset.

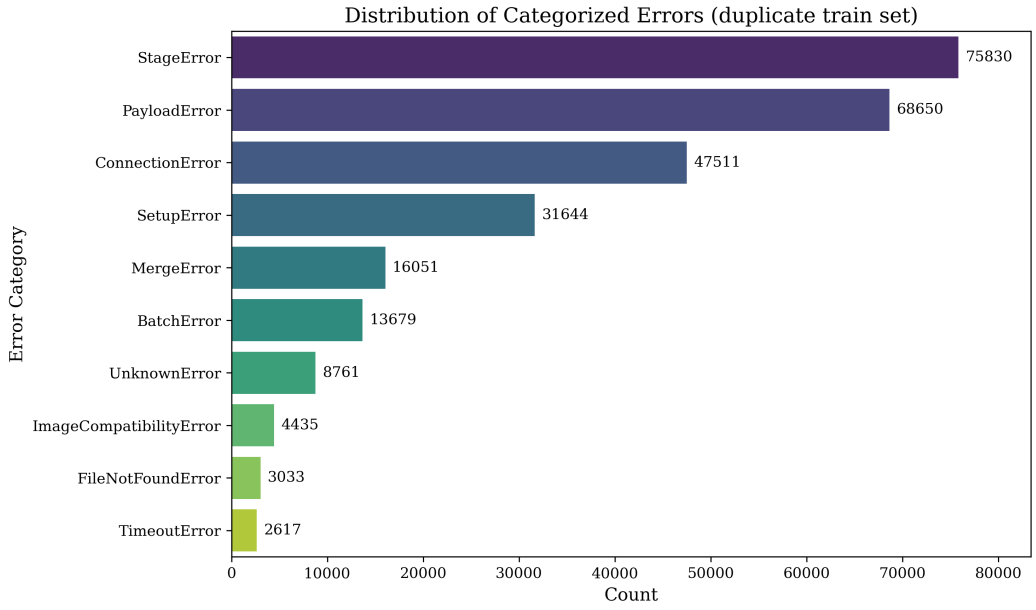


Figure 4.3: *Comparison of the error category distributions inside the training sets of the two multi-classifier datasets.*

Eventually, even if the entries of the dataset were actually in minority with respect to sets used for binary classification, the multi-classifier datasets were still sufficiently large to train complex models with a good number of examples for each category. In order to display the actual results of the trained models, the next sections will describe the built algorithms and their performances, starting from the binary classifier and then moving to the multi-classifier models.

4.2 Binary Classifier: Random Forest

The first machine learning model built for the working framework was a binary classifier, whose goal was to classify the job reports according to the value of the `jobstatus` parameter, i.e., `finished` or `failed`, assigned respectively to a numerical class 0 or 1. The algorithm chosen for this task was a Random Forest classifier, which is an ensemble learning method that operates by constructing multiple decision trees during training and outputting the mode of the classes (classification) of the individual trees. The model is particularly well-suited for handling large datasets keeping a good robustness to over-fitting, especially when dealing with imbalanced classes, as in this case. The Random Forest model was implemented using the `RandomForestClassifier` class from the `sklearn.ensemble` module in Python (an actual mathematical description of this model is presented later in subsection B.2.1). The model was trained and evaluated on the three different scenarios previously described in subsection 4.1.1, keeping constant the number of estimator trees to 100, while letting free the maximum depth of the trees to grow without any limit. The actual training results are summarized in Table 4.3.

Table 4.3: *Random Forest trees (100 estimators) trained on the three dataset schemes. Report of the minimum, mean, and maximum tree depth and number of leaves.*

	Tree Depth			Leaves		
Dataset	Min	Mean	Max	Min	Mean	Max
Oversampled	46	69.9	96	211	394.2	777
Stratified	55	74.9	98	253	495.7	984
Downsampled	32	52.9	73	112	185.1	320

The table summarizes the characteristics of the trained trees, in particular their depth and number of leaves, for each of the three different dataset schemes. Having left free the maximum depth constraint, the trees were able to grow quite deep, ranging from a minimum of 32 levels for the down-sampled dataset, to a maximum of 98 levels for the stratified dataset. Also, the number of leaves was on average quite high, with the highest average of approximately 496 leaves for the stratified dataset, but in all the scenarios the order of magnitude for the number of leaves was around a few hundreds. In general, the variability of the trees describes perfectly on how the model could adapt to different datasets: deeper and more populated trees for the more populated datasets, and shallower and less populated trees for the smaller datasets.

4.2.1 Model evaluation and results

The evaluation of the Random Forest model was performed on each of the three different test sets described in subsection 4.1.1, in order to compare the performances of algorithm working with different datasets. A first performance comparison is presented in Figure 4.4.

From each test set, the evaluation of the prediction was portrayed in a confusion matrix, which is a specific table layout that allows the visualization of the perfor-

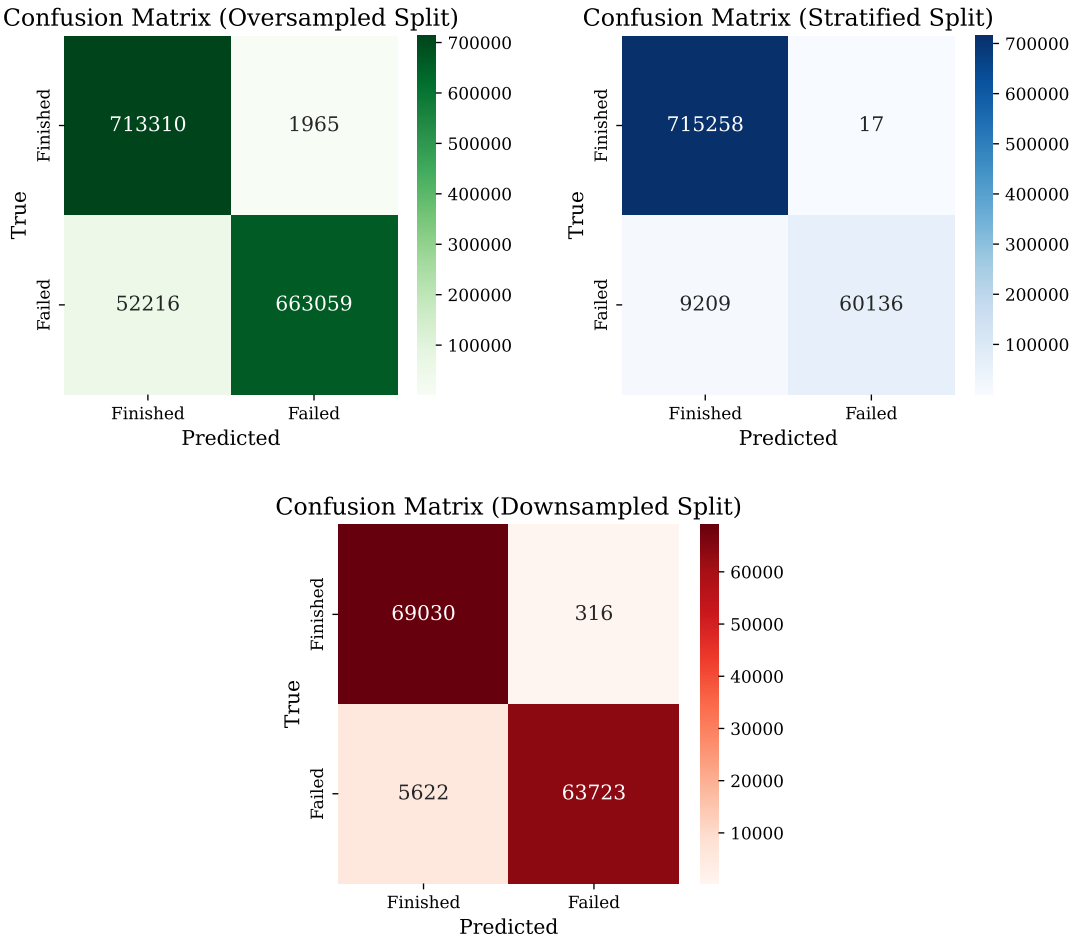


Figure 4.4: *Confusion matrices for the Random Forest model evaluated on the three different test sets. In green the over-sampled dataset, in blue the stratified one and in red the down-sampled one.*

mance of the model across different classes. In the case of a binary classifier, the confusion matrix is a 2×2 matrix, where the rows represent the actual classes and the columns represent the predicted classes. In general, the model is performing good when the matrix entries along the diagonal are high with respect to the off-diagonal entries.

As the three plots show, the model performed quite well in all the scenarios. In particular, with the over-sampled dataset (the green matrix), the model achieved a good recognition of both classes, though paying a relatively high number of false positives (i.e., failed jobs misclassified as finished). The stratified dataset (the blue matrix), which reflects the $1:10$ imbalance of failed jobs, led to an excellent accuracy, but many failed jobs were incorrectly classified, confirming the challenges posed by imbalanced stratified datasets. On the contrary, the down-sampled dataset (the red matrix), provided a balanced evaluation, with a very good capability of recognizing the two classes with comparable accuracies, although the expense of the smallest prepared dataset, thus not quite close to a real case scenario.

Another way to evaluate the performance of the model is to look at the Receiver Operating Characteristic (ROC) curve, which for the three case scenarios, the ROC

curves are displayed in Figure 4.5. The ROC curve plot shows the True Positive

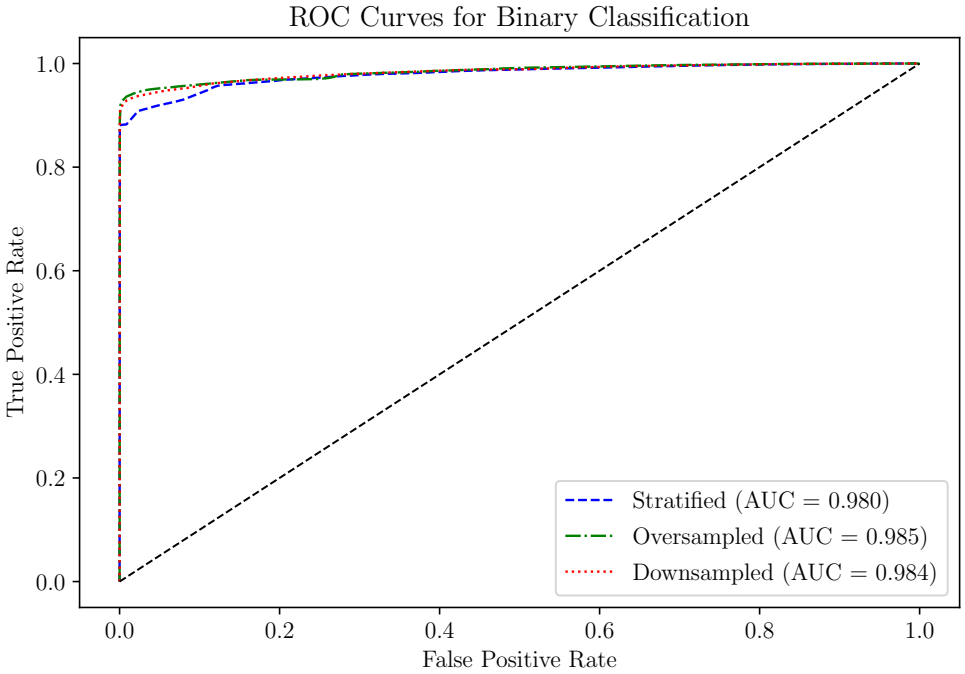


Figure 4.5: *ROC curves for the Random Forest model evaluated on the three different test sets. The color pattern is the same as in Figure 4.4.*

Rate (TPR, or Recall) against the False Positive Rate (FPR) at various threshold settings. A model with no discrimination ability would result as a diagonal line from the bottom left to the top right, while a perfect model would reach the top left corner. The Area Under the Curve (AUC) provides a scalar value to summarize the actual performance of the model, meaning that the closer the AUC is to 1, the better the model is at distinguishing the classes. In the case of the Random Forest model, the AUC values are quite high in all the three scenarios, ranging from 0.980 to 0.985. All the models performed very well, confirming that all the models were able to distinguish between the classes with excellent accuracy.

Another important evaluation metric for the model is the Precision-Recall (PR) curve, which is particularly useful when dealing with imbalanced datasets. The plot focus is more dedicated to the class in minority, relating the parameters of precision (the fraction of predicted failures that are correct) to recall (the fraction of actual failures that are correctly detected). The PR curves for the three case scenarios are displayed in Figure 4.6. The PR curve plot reports also the Average Precision (AP) score, which summarizes the precision-recall curve for the model in the three scenarios. The overall AP scores are quite high, indicating that the model maintains a good balance, especially the case of over-sampled and down-sampled datasets with the highest values (respectively of 0.989 and 0.988), another confirmation that class balancing allows the model to better detect failed jobs without sacrificing precision. The stratified dataset instead performs a bit worse, with an AP score of 0.940, indicating that the model struggles more to maintain high precision with imbalanced data distribution, reflecting a reduced sensitivity to the minority class. Nevertheless, the model still performs reasonably well even in the more realistic

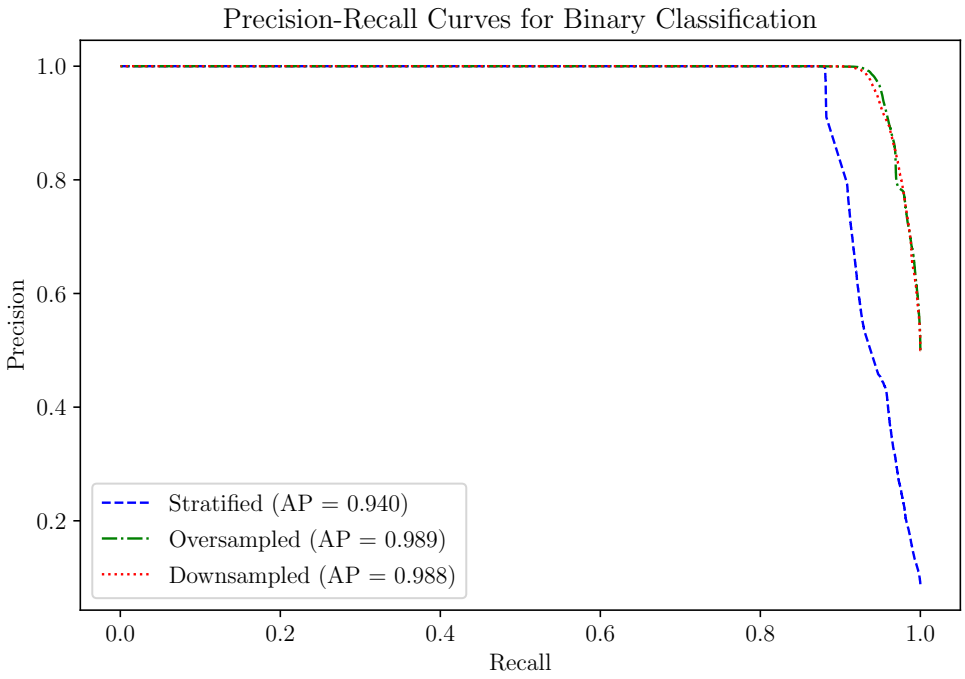


Figure 4.6: *Precision-Recall curves for the Random Forest model evaluated on the three different test sets. The color pattern is the same as in Figure 4.4.*

scenario of the stratified dataset, showing that the generalization capabilities of the Random Forest algorithm are quite good.

In conclusion, the results highlight the differences between dataset preparation strategies: oversampling and downsampling enforce class balance and improve the classifier's sensitivity to failed jobs, while stratification preserves the real-world distribution but leads to poorer recall on the minority class. Moreover, the ROC curve and the PR curve confirm that the model performs better with unrealistic balanced datasets, as shown by the higher AUC values for the over-sampled and down-sampled datasets, but even the stratified dataset achieved good results, despite being a very imbalanced dataset replicating fully the original distribution of the main dataset.

Overall, the Random Forest model had robust performances across all the configurations, delivering 3 different trained models with good capabilities of generalization, thus confirming the choice of the algorithm for the binary classification task. The real challenge was then to move to a smaller dataset to use, composed of only the failed jobs, and build with that a multi-classifier model, able to eventually categorize the various types of error. Next section will describe the adopted models and their performances.

4.3 Multi-classifiers models for anomaly detection

In a first instance, the real challenge of the second part of the machine learning framework was not the choice and the implementation of the model to adopt itself, but rather the preparation of at least a minimum viable dataset to train it.

The preparation of the multi-classifier datasets was at first a bit disarming, since

the actual failed jobs inside the main dataset were only roughly 10% of the total, and among those, the various error messages were apparently very heterogeneous, but in reality many of them were just duplicates or slightly different versions of the same error. Although the challenge, the idea for the new datasets was actually pretty simple: the creation of the two datasets presented in subsection 4.1.2, one with only unique error messages and one with all the duplicate messages.

Having to face this limitation in the dataset, the choice of a single model for the multi classification task was not sufficient anymore. Thus, the decision was to at least try to implement two different but complementary models, in order to compare their performances on the two different datasets. The chosen models were a logistic regression (i.e., `logreg`) and a Neural Network (i.e., `NN`), and their actual development was the following.

The logistic regression was chosen because it is a relatively simple but efficient linear mode, able to reveal through the feature weights the tokens to assign to a specific error class. The model was actually a baseline for multi classification, since it combines two parts for the actual training:

- The first part of the algorithm transforms logs with Term Frequency-Inverse Document Frequency (TF-IDF, used as the vectorizer to convert text features into numerical ones), whose scope is to assign a score to the token according to their frequency (TF) in this case inside the error messages, but also considering their rarity across the entire dataset (IDF). For instance, TF assigns higher weights to more frequent tokens, while IDF assigns higher weights if the tokens are rare inside the whole set of errors.

- The second is the actual `logreg` algorithm, trained using the stochastic gradient `SAGA` solver provided by `sklearn`, designed for large and sparse datasets. The model learns a set of linear decision functions, one per each category, normalizing eventually the output to estimate the class probability outcome.

The model needs a preliminary part managed by the TF-IDF to produce a computable matrix to perform actual predictions, but also to handle class imbalance, by also providing class-balanced weights later applied in the training to enhance underrepresented classes.

On the other hand, the neural network was chosen to have a different handling of the dataset feeding model, since it used a different tokenizer produced from the same datasets. Moreover, this approach keeps the semantic and the distributional patterns, making the assignment to a specific class also possible with words that are actually recognized as synonyms or belonging to the same error context. In this case, the logs were initially tokenized (as integers) into a fixed vocabulary of $10,000$ words, which were then transformed by an embedding layer into dense vectors (16-dimension), thus producing a dense layer from which learn distributional similarities of the actual words included in the training data (i.e., producing semantically related tokens are treated similarly by the model). The scheme of the neural network is presented in Figure 4.7.

The produced vectors are then aggregated and standardize into a fixed length to reproduce a message (via Global Average Pooling). Messages enter the network,

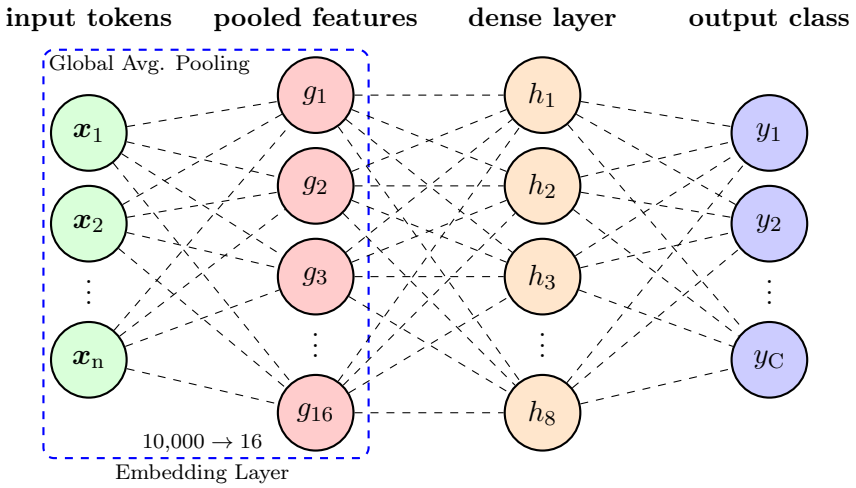


Figure 4.7: *Neural network schematic: Token sequence ($n \approx 100$) is mapped by an Embedding layer (vocabulary 10,000) to 16-d vectors, averaged by Global Average Pooling into pooled features ($g_1, \ldots, g_{16}$), passed to a dense layer, and classified over the resulting C classes.*

activating in order the ReLU (Rectified Linear Unit), the L2 regularization and dropout, to eventually reach the output layer for the actual prediction. The whole training is optimized again with class weights, using a loss function for multi-class classification (i.e., *categorical cross-entropy loss*) and an optimizer to update weights during training (i.e., *ADAM*, Adaptive Moment Estimation). The mathematics behind these models is going to be presented later respectively in subsection B.2.2 and subsection B.2.3. Both the models were independently instantiated, trained and tested, reaching excellent results and performance, which are going to be presented in the last section of this chapter. Before reaching that part, a brief part on the training of the models is going to be faced, showing for both models the actual training, validation and test results extracted before the actual classification.

4.4 Training and validation of non-overfitting

As already presented in subsection 4.1.2, the datasets used to train both the `logreg` and the `NN` are indeed much less populated than the ones for the binary classification. In fact, it could be argued that the `unique` and `duplicate` splits are too little to train a machine learning model without bringing it to overfitting. Nevertheless, in the case scenario proposed by this thesis, the error classes produced in the preprocessing are actually a good generalization of all the entries received by the BDP, which consistently, are the only entries on which the models should perform actual categorization. Despite that, in order to guarantee the non-overfitting of both the models, proving also their generalization capabilities, a set of quantitative and visual checks was performed while the training proceeded. The following subsections will present the analysis performed for both the machine learning models.

4.4.1 Logistic regression training and validation

In order to provide confirmation for the proper training of the logistic regression, some parameters were kept in track during the actual phases of the processing, in

order to later compare them and have a clear view of the overall correct behavior of the model. The metrics collected from the various splits are actually the most common ones which better describes the performance of the model on the input dataset. In details, those metrics are displayed in Table 4.4.

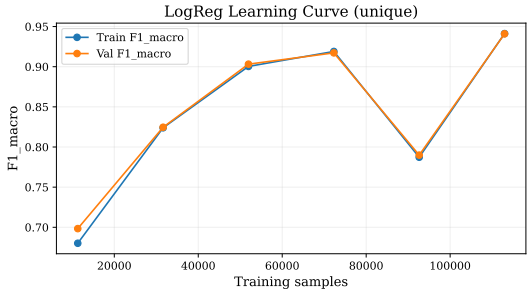
Table 4.4: *Train, validation and test metrics for the logistic regression model.*

`logreg`	Split	$F1_{macro}$	$Precision_{macro}$	$Recall_{macro}$	Accuracy
Unique	Train	0.955	0.934	0.987	0.995
	Validation	0.952	0.929	0.986	0.996
	Test	0.958	0.939	0.986	0.996
Duplicate	Train	0.9988	0.9988	0.9990	0.9993
	Validation	0.9988	0.9985	0.9991	0.9993
	Test	0.9989	0.9989	0.9991	0.9993

As the table shows, the performances were evaluated using standard classification metrics, namely *Accuracy*, *Precision*, *Recall* and *F1-score*. These parameters are actually extracted from elements of the confusion matrix, defining a macroscopic review of the model classification performances (more details on the formal definitions are in section B.2). The obtained results show consistency across the various splits, reaching in some cases differences lower than 1% for the `unique` dataset, and practically for all the values of the `duplicate` dataset (which is why to compare the results the entries have 4 digits after the decimal point). In general this shows a very high degree of generalization and almost absence of memorizing effects (also because the metrics are very close in value).

Another possible verification for the `logreg` is the learning curve, which can be performed both during the training phase and the validation one. The plots extracted during these phases are shown in Figure 4.8. The overall trend for the

(a) *Unique dataset learning curves.*



(b) *Duplicate dataset learning curves.*

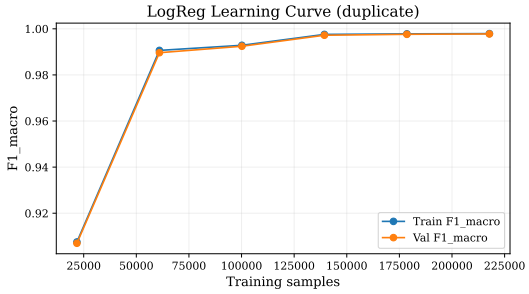
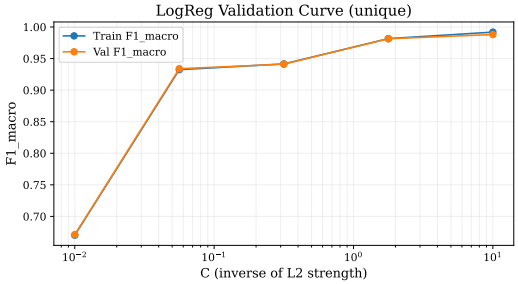


Figure 4.8: *Learning curves of the logistic regression model. In both plots, the blue line represent the training set, while the orange line represent the validation one.*

F1-score shown in the plots is an increase up to almost the maximum value for the `duplicate` datasets, while the `unique` keeps almost the same behavior but with a steep loss at around 90000 samples, but reaching still a very good final F1-score. Apart from that, the generalization capabilities of the model are once again confirmed, since in both plots the curves are almost always overlapping, also a confirmation of the stability of the model in the two scenarios.

Lastly, the concluding test for the logistic model was performed by extracting the validation curves, shown in Figure 4.9. The plots are actually showing the variation

(a) Unique dataset validation curve.



(b) Duplicate dataset validation curve.

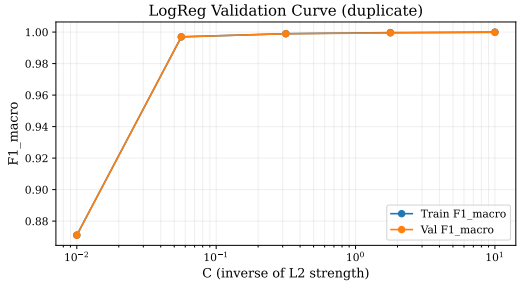


Figure 4.9: *Validation curves of the logistic regression model for different regularization strengths. The dataset color scheme is the same as the learning curve.*

of the training and validation F1 scores in function of the inverse regularization parameter C. In both cases, the trending of the curves are again almost always overlapping, and the curves rise steeply from the factor 10^{-2} to 10^{-1}, reaching a stable and high F1 value. Actually, in the `duplicate` case, the curves are practically on the maximum F1 score, above the order 10^{-1}. Thus, the plots confirm again that the model keeps regularity and stability over a wide rage of different regularization factors.

Eventually, the stability of the metrics across the data splits, the convergence of the training curves and the overlapping behavior in the validation curves proves the good generalization capabilities of the `logreg` model, but most importantly they prove solidly the non-overfitting of the model itself. Before reaching the final classification results, it is necessary to perform the same cross-checks also on the `neural` model.

4.4.2 Neural network training and validation

For the neural model, some verification tests performed were the same as the `logreg` model, while some others a bit different since structure of the training phase is different in the two models. As the previous case, the same metrics were collected (F1-score, precision, recall and accuracy), and are displayed in Table 4.5.

Table 4.5: *Train, validation and test metrics for the neural network model.*

`NN`	Split	$F1_{macro}$	$Precision_{macro}$	$Recall_{macro}$	Accuracy
Unique	Train	0.986	0.978	0.995	0.998
	Validation	0.979	0.967	0.989	0.997
	Test	0.985	0.976	0.994	0.998
Duplicate	Train	0.9984	0.9982	0.9985	0.9993
	Validation	0.9973	0.9972	0.9975	0.9987
	Test	0.9978	0.9975	0.9981	0.9990

Once again the metrics collected during the various phases of the model development were indeed solid. As the previous results, in all the splits the values

are pretty close to each other, with variations below one percentage point in the `unique` dataset, and even lower in the `duplicate` dataset. The results are again a proof of non-overfitting of the `neural` model in both the case study.

The actual difference in the verifications which can be performed in this case relies on the nature of the model itself. The model is trained across so-called "epochs", meaning that the model metrics are building a sort of history of their values, making possible their showing along the development of the training and the validation phases. In particular, it is possible to display the evolution of the accuracy and the loss of the model in the mentioned phases, extracted for both the `unique` a `duplicate` dataset. The results of the training history of the neural model are shown in Figure 4.10 for the unique case and in Figure 4.11 for the duplicate one.

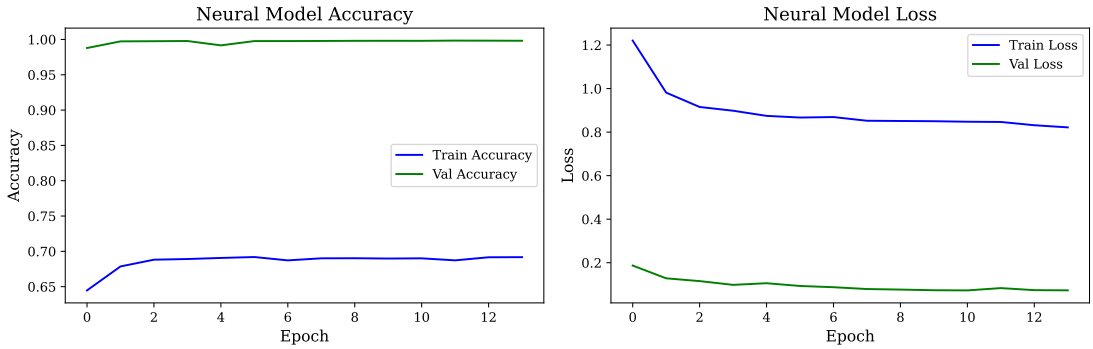


Figure 4.10: *Accuracy and loss history plots for the* `unique` *dataset. The curve in blue represent the training set while the green one the validation set.*

In the `unique` scenario, the accuracy is significantly low in the train set, but along the epochs it is actually pretty stable, having a sort of convergence to a plateau. The validation set instead is almost always at its highest value across, with a small decrease at the fourth epoch. In the loss curves, the behavior is actually showing a tendency to decrease until both curves are actually reaching a stable point, again with a significant discrepancy between train and validation. The results are in any case confirming a non-overfitting behavior of the model, since in both the plots, accuracy and loss curves are converging.

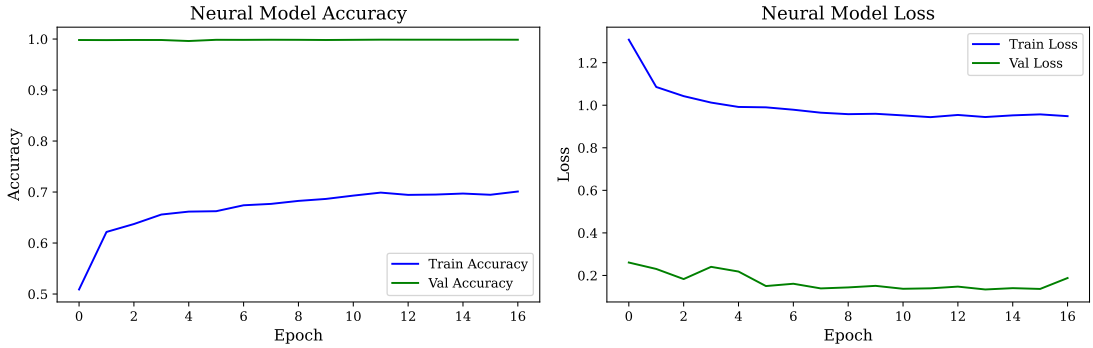


Figure 4.11: *Accuracy and loss history plots for the* `duplicate` *dataset. The curve in blue represent the training set while the green one the validation set.*

In the `duplicate` scenario, the accuracy curves keeps almost the same behavior

of the previous curves, in this case with a perfect alignment of the validation curve and a training curve growing until it reaches stabilization in the later epochs. The loss curves are a bit more interesting in this case, since the train one is very similar to the previous one, but the validation curve has some increase in the loss in some epochs (likely 3, 4 and 16), but still with much lower values with respect to the train curve. Again, the behavior of the curves, both accuracy and loss ones, is confirming the non-overfitting of the model, but it seems with some lack in the loss validation curve.

Summarizing, the metrics results for the neural model are confirming the stability of the performances in all the three different sets prepared to be fed to the model. Moreover, the convergence in the loss and accuracy curves is the ultimate proof of the proper behavior of the neural network in all the studied cases presented to this point. Having provided all the necessary proof to ensure the models' solidity, it is now time to reach the last section of this thesis.

4.5 Multi-classifier models results

Although the expectations on the performance of multi classifier models were somehow influenced by the very low number of entries in the failed job dataset, the models actually performed beyond those expectations.

4.5.1 Unique dataset

Starting from the `logreg` model, the resulting confusion matrix produced from the unique dataset is shown in Figure 4.12. As already mentioned, the performance

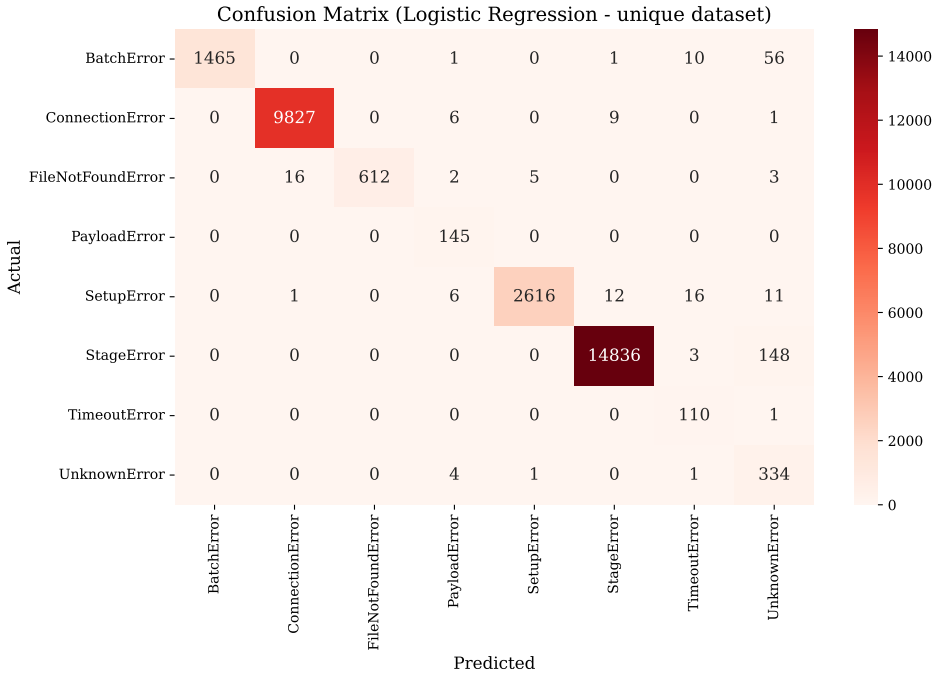


Figure 4.12: *Confusion matrix of the `logreg` mod el trained and evaluated on the unique datasets.*

evaluation inside a confusion matrix are defined in function of the diagonal entries

with respect to the non-diagonal entries (for the multi classification task), and in this case the model performed very well, presenting lots of correct predictions along the various classes. The most populated ones, like `ConnectionError` and `StageError` have a very high accuracy. Despite that, some categories were misclassified, particularly, `SetupError` and `BatchError` are spread into other categories. Probably the linear decision boundaries are not optimally performing into error categories with overlapping distributions.

The neural model on the unique dataset reached a clearer separation of all the classes, as shown in Figure 4.13.

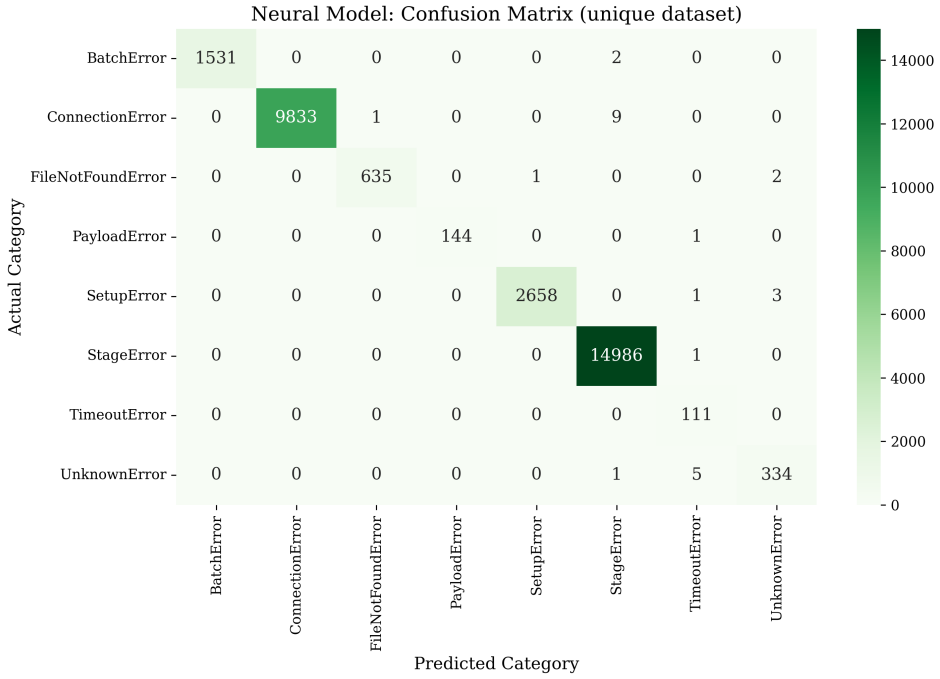


Figure 4.13: *Confusion matrix of the `NN` model trained and evaluated on the unique datasets.*

In the plot, fewer entries are off the diagonal, since most of the misjudged entry of the `UnknownError` were actually missing, and thus properly categorized. Categories such as `BatchError` and `FileNotFoundError` are better captured, while bad assignments to `UnknownError` are reduced. In particular the most visible improvement is in the `SetupError` and `StageError` classes, where the confusion of the Logistic Regression model were almost eliminated.

The overall classification on the unique dataset were indeed very solid. The logistic regression model was optimal in highly populated categories, showing some limitations in the classifications of lower populated classes. Instead, the neural network showed a very high granularity and generalization capabilities, handling almost perfectly all the classes, reducing the category misjudging. The results show that a more expressive model is better performing in dealing with heterogeneous error type data. The comparison highlights the trade-off between model simplicity and performance. Logistic regression offers good classification and acceptable results, while the neural network demonstrates superior accuracy and robustness across all error categories.

Another proof of the difference in the models' performances, is presented in table Table 4.6, summarizing the overall precision, recall and F1-score for every class, displaying also the accuracy, the average and the weighted average for these parameters.

Table 4.6: *Classification performance metrics for the Logistic Regression and Neural Network models on the* ***unique*** *test dataset.*

	`logreg`			`NN`		
Class	Precision	Recall	F1-score	Precision	Recall	F1-score
BatchError	1.000	0.956	0.977	1.000	0.997	0.999
ConnectionError	0.998	0.998	0.998	1.000	0.999	0.999
FileNotFoundError	1.000	0.959	0.979	0.992	0.995	0.994
PayloadError	0.884	1.000	0.939	0.973	0.993	0.983
SetupError	0.998	0.983	0.990	1.000	0.998	0.999
StageError	0.999	0.990	0.994	0.999	0.999	0.999
TimeoutError	0.786	0.991	0.876	0.933	1.000	0.965
UnknownError	0.603	0.982	0.747	0.968	0.979	0.974

The values for the overall accuracy in all the classes are indeed pretty high, having also most of the average and weighted average above 0.99. This confirms in general a good capability and reliability in the distinction of the categories which led to a job failure. Moreover, the neural model consistently achieves the better performances even in the more challenging categories, where the logistic regression is generally struggling. Considering again `UnownError`, the difference in accuracy between the models is larger that the 35%, and similarly the `TimeoutError` shows almost the same trend, but with a lower discrepancy. Again this table shows how the neural model's superior capabilities in enhancing the differences even in the minority cases. An ultimate proof of evaluation of the models results is displayed in the ROC and PR curves, which are all displayed in section B.3.

Eventually, the evaluation of the models on the unique dataset seems to lead to a sure decision in the model adoption, trending for the neural network model. Before reaching a final decision, the results from the duplicate dataset have to be presented and discussed.

4.5.2 Duplicate dataset

The performance on the duplicate dataset were overwhelming superior beyond any expectations, since both the models performed almost perfectly in the error categorization.

Starting again from the logistic regression model, its confusion metrics result from the duplicate test set is shown in Figure 4.14 (the distinction from the previous matrix lies also on the fact that in this dataset an additional error category was added to the class because of its multiplicity: `ImageCompatibilityError`).

In comparison with the previous results, training on the duplicate dataset led the `logreg` model to perform extremely better. All the dominant cate-

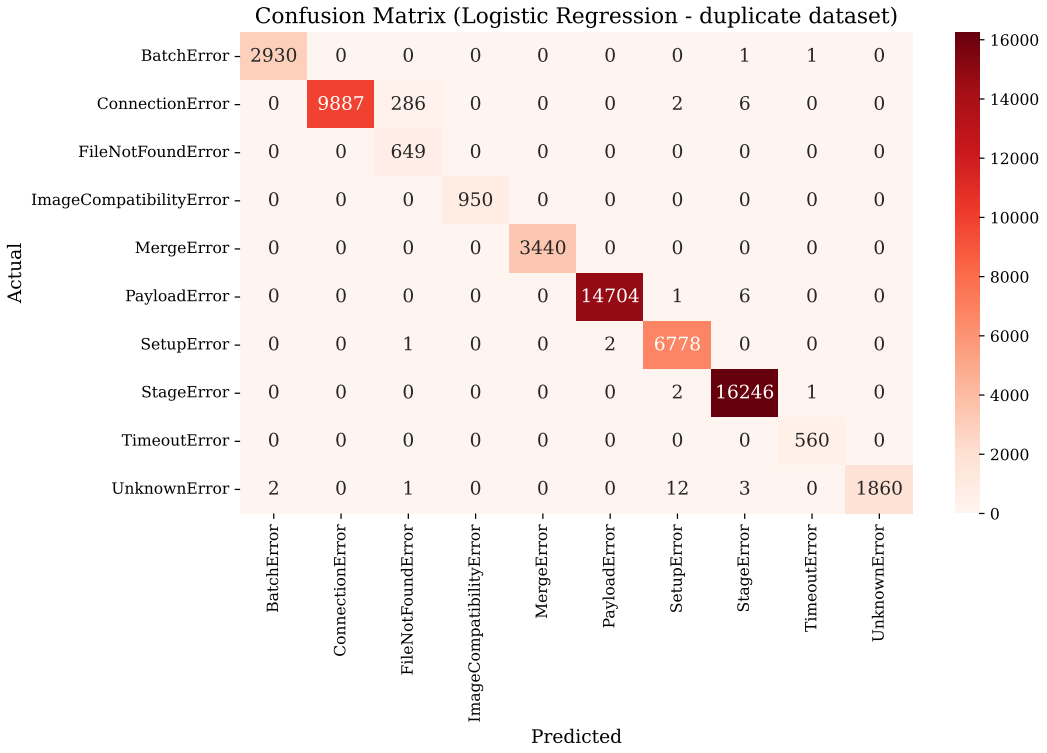


Figure 4.14: *Confusion matrix of the* `logreg` *model trained and evaluated on the duplicate datasets.*

gories (i.e., `StageError` and `PayloadError`) entries were almost perfectly classified. Despite that, also in this case some misclassifications are present, like some `ConnectionError` entries were misclassified, while some entries were again falsely assigned to the `UnknownError` class. Even these failures, the model performed slightly better, also probably because the training set was doubled in dimension, but still it shows that some difficulties are present in handling ambiguous or less frequent classes, even if the preamble of the algorithm included a class weight to try to overcome the class imbalance. As a general consideration, the redundancy of the dataset led to a better performance of the logistic regression algorithm.

In the same way, the training and testing of the neural network led to again extremely optimal results, but with respect to the unique dataset training, the neural network made a bit more mistakes in the assignment of some classes. The confusion matrix resulted from the test of the duplicate dataset on the `NN` model is shown in Figure 4.15.

In comparison with the `logreg` model, the plots appear slightly cleaner in the non-diagonal elements of the matrix, including only few cross-class predictions in the minority class. Even if lower, those misclassifications were absent in the matrix in Figure 4.13, displaying a small issue in prediction. The highest misclassified entry is the one for the `FileNotFoundError`, classified 24 times more than the actual entries, which belonged to the `ConnectionError` classe. Overall, the most failed general prediction was the assignment of a specific class to the `UnknownError` one, still keeping a very high accuracy in their global classification (those misclassifications

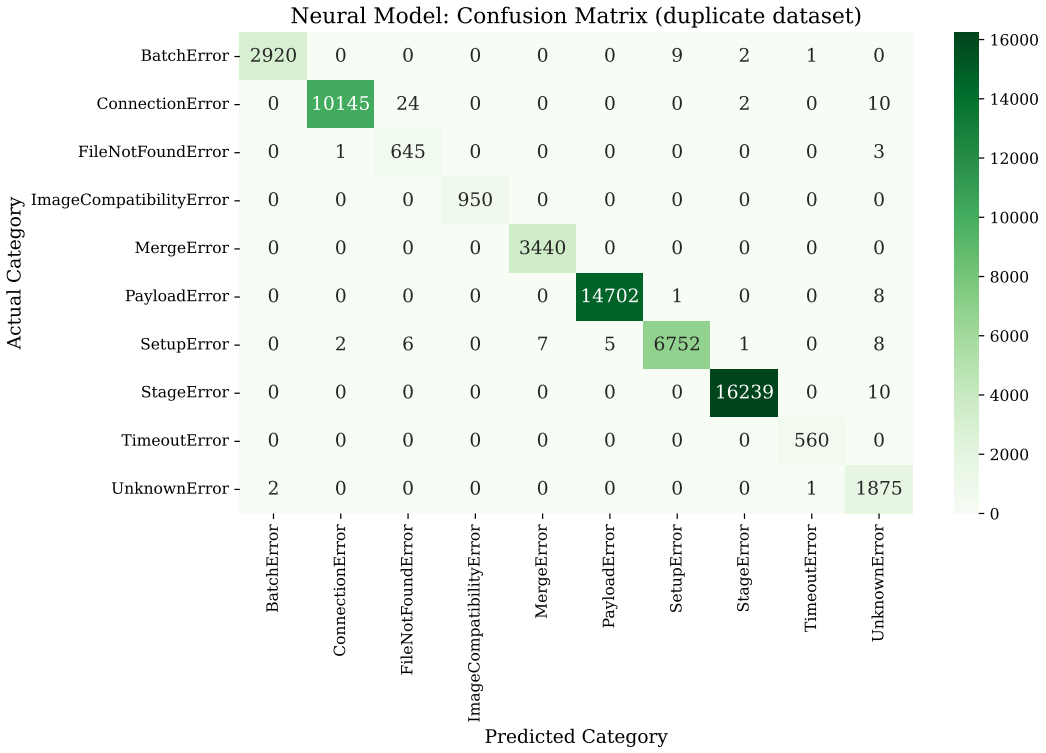


Figure 4.15: *Confusion matrix of the* `NN` *model trained and evaluated on the duplicate datasets.*

led to a loss in accuracy lower than 1). Probably the increase in the entry of the dataset of the duplicates led to a bit more confusion to the model, since its learning was more based on the semantics rather than the word vectorization. Apart for this minor issue, the overall performance in classification of the classes operated by the `NN` model is again excellent.

In the case of the duplicate dataset usage, both models showed an increased ability in the actual classification, on the one hand, the `logreg` with minor classifications mistakes and better performances, on the other hand the neural model with slightly more classification errors, but still reaching almost perfection in most of the classes. Logistic regression improved its performance, but still being less efficient than the neural model in this scenario. Both models excel on the duplicate dataset, but the neural network capabilities in the classification of minor populated class are indeed superior with respect to the logistic regression, in both the unique and duplicate dataset cases.

Another major proof for the eventual comparison is provided by the summary Table 4.7, again comparing the overall trend for the precision, the recall and the F1-score for both the models, and their accuracy, average and weight average.

As anticipated, both the models in most of the cases performed perfectly with some values reaching the maximum approximation possible (i.e., 1.000 cut at three digits). One class is actually weak for the `logreg` model, since the precision of the `FileNotFoundError` is approximately 0.7 with a F1-score of 0.81, suggesting that even with a duplicate dataset the algorithm still overlaps the class with others. This

Table 4.7: *Classification performance metrics for the Logistic Regression and Neural Network models on the* ***duplicate*** *test dataset.*

	logreg			NN		
Class	Precision	Recall	F1	Precision	Recall	F1
BatchError	0.999	0.999	0.999	0.999	0.996	0.998
ConnectionError	1.000	0.971	0.985	1.000	0.999	1.000
FileNotFoundError	0.693	1.000	0.818	0.991	0.991	0.991
ImageCompatibilityError	1.000	1.000	1.000	0.997	1.000	0.998
MergeError	1.000	1.000	1.000	1.000	1.000	1.000
PayloadError	1.000	1.000	1.000	1.000	1.000	1.000
SetupError	0.997	1.000	0.999	0.999	1.000	0.999
StageError	0.999	1.000	0.999	1.000	1.000	1.000
TimeoutError	0.996	1.000	0.998	0.989	0.998	0.994
UnknownError	1.000	0.990	0.995	0.993	0.997	0.995

lack is probably due to the linearity of the model, thus not refining perfectly the boundaries in the feature space. Instead, the general behavior of the NN model is confirming its robustness on lower frequency classes, and keeping an overall optimal accuracy (having as minimum the value of 0.989 for the `TimeoutError`).

The neural model again proved to be elastic in both the prefigured scenarios, but keeping and incredibly high generalization capability. Also in this case an ultimate proof of evaluation of the models results is displayed in the ROC and PR curves, which are all displayed in section B.3.

4.6 Perspectives for operational deployment

The results obtained from the full deployment of the classification models have brought to light several considerations, regarding their performance, robustness and potential for integration into the BDP infrastructure. In this section, a small discussion related to the models' general behavior and the possible developments for an online monitoring is going to be shared, before reaching the summary of the chapter.

4.6.1 Generalization and robustness of the proposed models

In section 4.2 and section 4.5, the achieved results prove that both binary and multi-class classification models are able to effectively capture relevant patterns in inside ATLAS job reports. Nevertheless, their performance must be interpreted, in the context of the characteristics of the dataset used and the operational conditions under which the data were collected.

The models have been trained and evaluated on historical data originating from a specific infrastructure and time window, which may limit their ability to generalize to future workload or to environments characterized by different configurations, software releases or management policies. Also, changes in job submission patters

or computing infrastructure could impact the future performance of the models, requiring periodic monitor of the performances and eventual retraining with renewed data.

The importance of adopting strategies such as periodic retraining, continuous validation and performance monitoring is a key aspect to ensure the long-term effectiveness of the models. Rather than being static tools, these models should be treated as dynamic components of the monitoring system, capable of adapting to evolving conditions, and most importantly, being regularly updated with new data to maintain high accuracy and reliability.

4.6.2 Integration within the BDP environment

From an architectural perspective, the BDP allows in principle an external integration of machine learning models.

Having centralized data collection, managed by the several cooperating components into different data pipelines, it is easy to see that the model could be injected as a part of the ATLAS pipeline, within the layers of the consumer servers and the data storage systems. In a production scenario, the trained models could at this level could receive the data in a JSON format, already including all the entries on which the models have been trained on.

Performing automated classification while data flows in would help the early identification of anomalous failure patterns. Such integration would enhance the current monitoring capabilities by complementing traditional rule-based approaches with data-driven insights, while preserving transparency for system operators.

At the same time, operational deployment introduces new challenges related to model lifecycle management, including retraining policies, versioning, and validation against evolving workloads. Addressing these aspects is essential to ensure long-term robustness and to position machine learning models as decision-support tools within the BDP, assisting human operators rather than replacing existing control mechanisms.

4.7 Final summary

All the results presented in this chapter confirm the feasibility of applying machine learning models for anomaly detection on ATLAS job reports stored within the INFN-CNAF Big Data Platform. For the binary classification task, the Random Forest model demonstrated excellent performance across all dataset preparation strategies, with particularly strong results when the class re-balancing techniques (over-sampling or down-sampling) were applied. Even in the stratified case, which mirrors the real-world imbalance of failed versus successful jobs, the model retained solid predictive power, though at the expense of sensitivity to the minority class.

For the more challenging multi-class classification task, two complementary models were compared: logistic regression and a neural network. Both showed strong capabilities, but the experiments revealed clear differences:

- On the unique dataset, the neural network consistently outperformed logistic regression, especially in minority and heterogeneous classes;

- On the duplicate dataset, both models achieved near-perfect accuracy, though the neural network maintained a slight advantage in robustness and generalization, particularly on classes with limited representation. Logistic regression benefited more directly from the redundancy of duplicated entries, but this highlighted its limitations when facing noisier or less frequent categories.

These findings suggest that the adoption of neural network models within the BDP would provide the most reliable support for operational monitoring, while the logistic regression remain valuable as lightweight baseline, still usable with a lower confidence, but could be used as a second check marker in the job report classification.

In conclusion, all the models are actually validated and tested and ready to be inserted in the BDP infrastructure as an integrative part of the established data pipeline, allowing a preliminary check on the incoming reports with the possibility to send eventual alerts to the system administrator in case of major errors incoming.

Conclusion

Having the possibility to work in collaboration for a period of the PhD with the INFN-CNAF groups and having access to their infrastructures and data led the way for the development of this thesis.

Starting from the basics to enter in contact with the environment of the BDP facility, the first steps were the understanding of the main components of the infrastructure, the acquisition of the date and the instantiation of a data pipeline from an external source to the internal database. The chose source fell on the logs produced by the ATLAS experiment, which were redirected through the WLCG to the Tier-1 computing site hosted by the INFN-CNAF. Thanks to the help of the CNAF system administrators, a first data pipeline was created, from PanDa framework provided by ATLAS to the servers hosting the BDP. After all the verification of the data correctness and integrity, the logs were stored in order to create a dataset to be used for the creation of a framework inside which a series of machine learning models could be trained and tested. The goal was actually reached, since a first binary classifier was created, reaching an excellent accuracy in the classification of the job outcome, and then a multi-classifier was also trained and tested, reaching optimal results in the classification of the various types of error.

The results obtained are only the first step towards the integration of those models inside the BDP facility, ready to be implemented and automatized in order to be an instrument useful for the system administrators to monitor the status of the computing jobs and to predict possible failures. Moreover, the same framework and the same pipeline instantiation could be used by other researchers working for other experiments in collaboration with INFN-CNAF, enlarging the data input and possibly the statistics of the datasets, in order to improve for each of the input topic a specific trained and tested model.

Appendix A

A.1 Producer cronjob files

In this section, details of the producer machine configuration are provided. The schedule of the cronjob presented in subsection 3.2.1 includes the following files:

- `kinit.sh`:

Listing A.1: *Script to initialize Kerberos authentication.*

```
#!/usr/bin/bash
echo <password> | kinit <username>@CERN.CH
```

- `panda-cookie.sh`:

Listing A.2: *Script to download Panda cookie.*

```
#!/usr/bin/bash
ssh -K lxplus "auth-get-sso-cookie -u https://bigpanda.cern.ch
    /oauth/login/cernoidc/ -o bigpanda.cookie.txt;"
```

- `panda-report.sh`:

Listing A.3: *Script to download reports from Panda. label*

```
#!/usr/bin/bash
current_date=$(date +"%d-%m-%Y")
current_time=$(date +"+%T") #date format HH:MM:SS
for computing_site in INFN-CNAF INFN-CNAF_ARM
do
    for job_status in finished failed
    do
        echo "Retrieving job reports on $current_date at
            $current_time of the last hour $job_status jobs at
             $computing_site" >> ~/atlas-data/raw-jobs-reports
            /job_download_status.log
        ssh -K lxplus 'curl -b ~/bigpanda.cookie.txt -H '"'"'
            Accept:application/json'"'"' -H '"'"' Content-type
            :application/json'"'"' "https://bigpanda.cern.ch/
            jobs/?json&hours=1&jobstatus='$job_status'&
            computingsite='$computing_site'&outputs=jobs"' >
            ~/atlas-data/raw-jobs-reports/jobReport-
            $computing_site-$job_status-jobs_raw.json
```

```
    done
done
```

- `report-removal.sh`:

Listing A.4: *Script to daily remove reports.*

```
#!/usr/bin/bash
rm -rf ~/atlas-data/raw-jobs-reports/jobReport-*.json
```

A.2 Kafka configuration files and commands

In this section, a more detailed overview of the configuration for Kafka brokers is provided. All the displayed files and command are intended for reproducibility purposes.

The following commands illustrate how to configure SSL/TLS for a Kafka broker at first installation. The process requires the creation of both a truststore and a keystore, referenced in subsection 3.2.2.

- Creation of a truststore:

Listing A.5: *Command to create the truststore inside the broker machine.*

```
keytool \
  -keystore <path of the truststore> \
  -alias truststore \
  -import \
  -file <path of the CA certificate>
```

- Creation of a keystore:

Listing A.6: *Commands to create the keystore inside the broker machine.*

```
# Starting from a .pem private key generating a .p12 key
openssl \
    pkcs12 -export -in /etc/puppetlabs/puppet/ssl/certs/<
       HOSTNAME>.pem \
    -inkey /etc/puppetlabs/puppet/ssl/private_keys/<HOSTNAME>.
       pem  \
    -name hostcertkey > /root/<HOSTNAME>.p12

# Importing a private .p12 key
keytool \
    -importkeystore \
    -destkeystore <path of the new keystore> \
    -srckeystore /root/<HOSTNAME>.p12
```

- List of truststore and keystore

Listing A.7: *Command to list the contents of the truststore and keystore.*

```
keytool -list -v -keystore <path of the truststore>
keytool -list -v -keystore <path of the keystore>
```

- Remove an alias from the keystore:

Listing A.8: *Command to remove an alias from the keystore.*

```
keytool \
    -delete -alias <alias_name> \
    -keystore /root/keystore.jks
```

Moreover, the management of the *topics* inside Kafka is performed through specific commands, displayed in the following list:

Listing A.9: *Commands to manage Kafka topics.*

```
# List all topics
kafka-topics.sh  \
    --bootstrap-server kafka01.cnaf.infn.it:****,kafka02.cnaf.infn
       .it:****,kafka03.cnaf.infn.it:**** \
    --command-config adminclient-config.conf \
    --list

# Describe a specific topic
kafka-topics.sh  \
    --bootstrap-server kafka01.cnaf.infn.it:****,kafka02.cnaf.infn
       .it:****,kafka03.cnaf.infn.it:**** \
    --command-config adminclient-config.conf \
    --describe --topic <topic_name>

# Edit an already existing topic:
 # 1- Number of partitions
 kafka-topics.sh \
     --bootstrap-server kafka01.cnaf.infn.it:****,kafka02.cnaf.
        infn.it:****,kafka03.cnaf.infn.it:**** \
     --command-config adminclient-config.conf \
     --alter --topic <topic_name> --partitions <num_partitions>

 # 2- Replication factor
 kafka-topics.sh
     --bootstrap-server kafka01.cnaf.infn.it:****,kafka02.cnaf.
        infn.it:****,kafka03.cnaf.infn.it:**** \
     --command-config adminclient-config.conf \
     --reassignment-json-file <path_to_reassignment_json_file> --
        execute

 # 3- Retention time
 kafka-topics.sh \
     --bootstrap-server kafka01.cnaf.infn.it:****,kafka02.cnaf.
        infn.it:****,kafka03.cnaf.infn.it:**** \
     --command-config adminclient-config.conf \
     --alter --entity-type topics --entity-name <topic_name> --add
        -config retention.ms=<retention_time_in_milliseconds>

# Get messages from a topic
kafka-console-consumer.sh \
    --bootstrap-server kafka01.cnaf.infn.it:****,kafka02.cnaf.infn
       .it:****,kafka03.cnaf.infn.it:**** \
    --consumer.config <path_to_consumer_properties_file> \
    --topic <topic_name> \
```

```
    --from-beginning # add this flag to display messages from the
       beginning of the topic

# Delete a topic
kafka-topics.sh \
    --bootstrap-server kafka01.cnaf.infn.it:****,kafka02.cnaf.infn
       .it:****,kafka03.cnaf.infn.it:**** \
    --command-config adminclient-config.conf \
    --delete --topic <topic_name>
```

A.3 Logstash installation and configuration

In this section, the installation and configuration of Logstash is presented in more details. The installation is performed manually on the consumer machine, following the steps below:

- Download and unpacking:

Listing A.10: *Commands to download and unpack Logstash on the consumer machine.*

```
# Download and install Logstash
wget https://artifacts.opensearch.org/logstash/logstash-oss-
   with-opensearch-output-plugin-8.9.0-linux-x64.tar.gz
tar -zxvf logstash-oss-with-opensearch-output-plugin-8.9.0-
   linux-x64.tar.gz
```

- Setup of the `logstash.service` file:

Listing A.11: *Service file for the Logstash instance.*

```
[Unit]
Description=logstash

[Service]
Type=simple
User=root
Group=root
EnvironmentFile=-/etc/default/logstash
EnvironmentFile=-/etc/sysconfig/logstash
ExecStart=/etc/logstash/bin/logstash "--path.settings"
   "/etc/logstash/config"
Restart=always
WorkingDirectory=/
Nice=19
LimitNOFILE=16384
StandardOutput=null
StandardError=syslog

TimeoutStopSec=infinity

[Install]
WantedBy=multi-user.target
```

- Opensearch plugin installation:

Listing A.12: *Commands to install the Opensearch output plugin for Logstash.*

```
/opt/logstash-8.9.0/bin/logstash-plugin install logstash-
   output-opensearch
```

- Test proper installation:

Listing A.13: *Command to test Logstash installation*

```
bin/logstash -e "input { stdin { } } output { stdout { } }
```

- Input configuration file:

Listing A.14: *Example of Logstash input configuration file.*

```
input {
   kafka {
    codec => json
    bootstrap_servers => "kafka01.cnaf.infn.it:****,kafka
       02.cnaf.infn.it:****,kafka03.cnaf.infn.it:****"
    sasl_jaas_config => "org.apache.kafka.common.security
       .plain.PlainLoginModule required username='
       logstash' password='**********';"
    security_protocol => "SASL_SSL"
    sasl_mechanism => "PLAIN"
    ssl_truststore_location => "/root/truststore.jks"
    ssl_truststore_password => "tspass"
    group_id => "logstash"
    auto_offset_reset => "earliest"
    topics => ["<topic_name>"]
    id => "logstash"
   }
}
```

- Output configuration file:

Listing A.15: *Example of Logstash output configuration file.*

```
output {
     if "<topic_name>" in [tags] {
    opensearch {
      hosts => ["https://osmaster01.cnaf.infn.it:****","
         https://osmaster02.cnaf.infn.it:****","https://
         osmaster03.cnaf.infn.it:****"]
      user     => "<configured_user>"
      password => "<configured_password>"
      index    => "%{index_prefix}-%{+YYYY.MM.dd}"
      cacert   => "/etc/puppetlabs/puppet/ssl/certs/ca.
         pem"
    }
  }
```

```
}
```

- Filters configuration file:

Listing A.16: *Example of Logstash filters configuration file.*

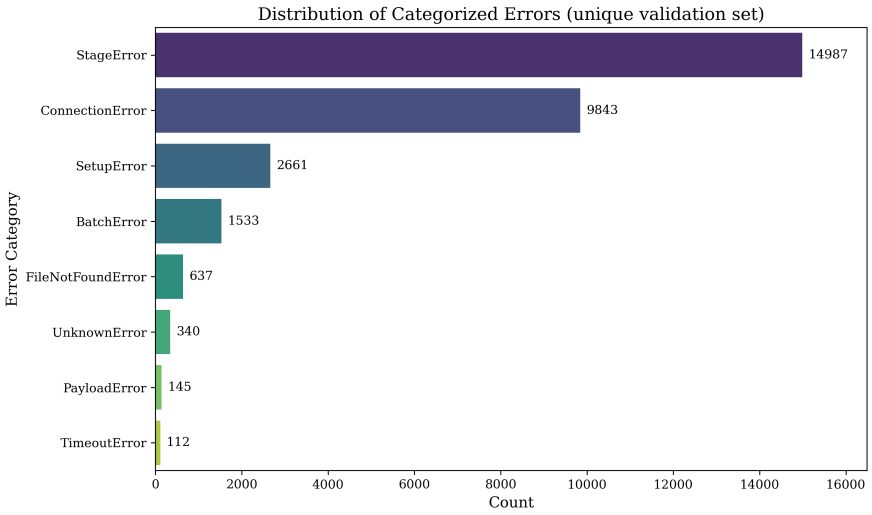
```
filter {
  if "<topic_name>" in [tags] {
    mutate {
      add_field => { "index_prefix" => "logstash" }
    }
  }
}
```

Appendix B

B.1 Validation and Test set distributions for multi-classification

In this brief section are shown also the error category distribution of the validation and test set used for both the `logreg` and neural model (paired with Figure 4.3).

(a) *Distribution of the dictionary categories inside the validation set of the unique error dataset.*



(b) *Distribution of the dictionary categories inside the validation set of the duplicate error dataset.*

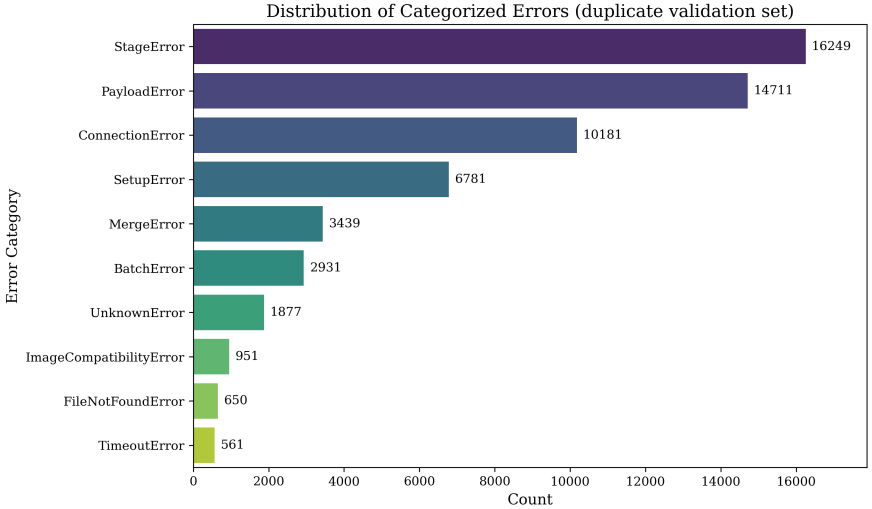
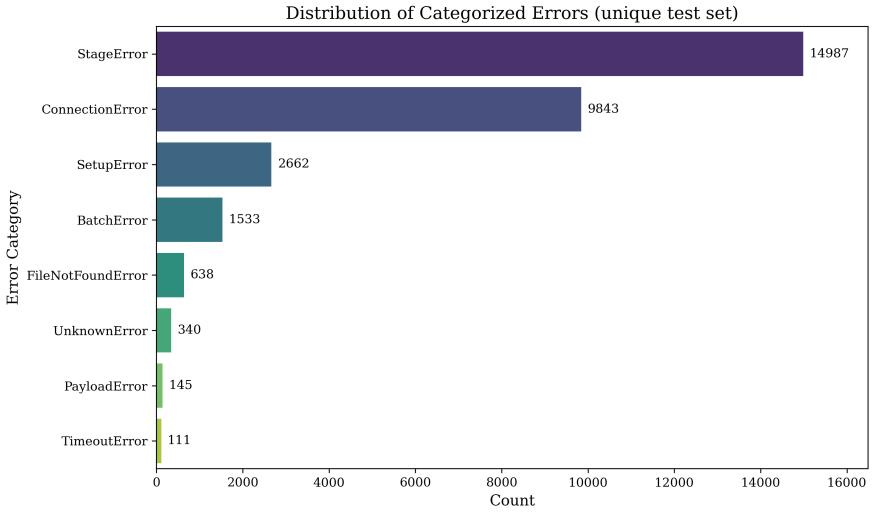


Figure B.1: *Comparison of the error category distributions inside the validation sets of the two multi-classifier datasets.*

The validation sets were used before the actual test sets, to tune the models

parameters. The actual distribution of entries of validation and test sets are shown in Table 4.2

(a) Distribution of the dictionary categories inside the test set of the unique error dataset.



(b) Distribution of the dictionary categories inside the test set of the duplicate error dataset.

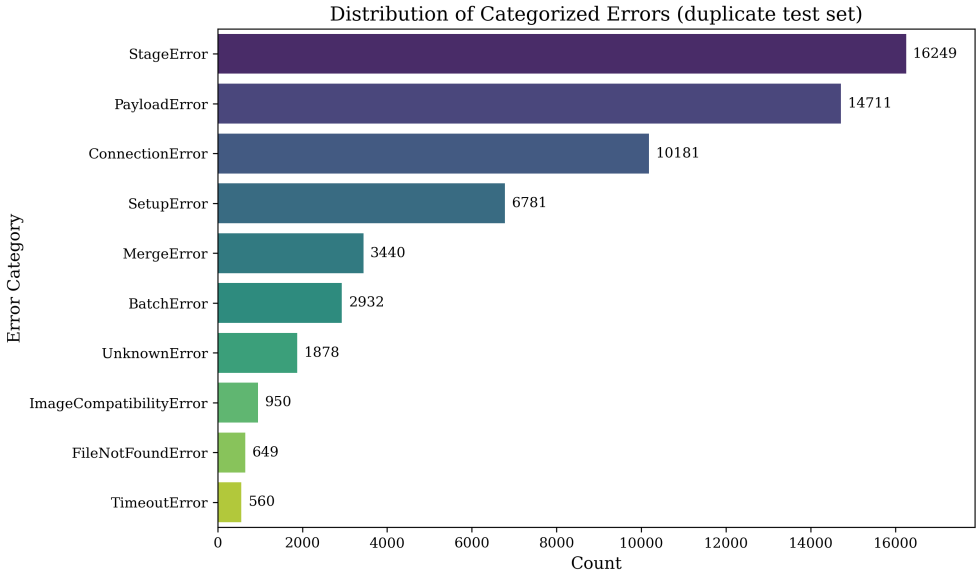


Figure B.2: *Comparison of the error category distributions inside the test sets of the two multi-classifier datasets.*

B.2 Machine learning models and metrics

In this appendix section an explanation of the formal metrics used to monitor the model developments and a brief mathematical description of the adopted model are going to be given. The main feature of each model and their characteristics, giving a small description of each, and also giving a small explanation on the reason of the choice of each model.

B.2.1 Random Forest

A Random Forest is an ensemble learning method which builds several decision trees, using their outcomes to build the output prediction either on a majority vote (for classification) or on average (regression). Each tree is trained on a random bootstrap sample of the dataset and splits are chosen from a random subset of features. This use of both bagging (bootstrap aggregating) and feature randomness reduces overfitting and improves generalization compared to a single decision tree.

The characteristics that define the model are the following:

- Number of trees (T), the more trees are present the better the model performs, at the price of computational resources;
- Maximum depth, allows control on the model complexity (in terms of variance and bias);
- Minimum sample per leaf, i.e., minimum number of samples required to create a leaf node;
- Splitting criterion, controls the rule to split the leaves.

The model performs classification through a majority vote among the whole set of trees T created, using a majority vote system, in order to predict the $\hat{y}$ class

$$\hat{y} = \text{mode}\{h_t(\boldsymbol{x}), t = 1, 2, \dots, T\} \ , \tag{B.1}$$

where $h_t(\boldsymbol{x})$ is the prediction of the t-th tree. At each split, the model minimizes a node impurity measure, typically with the Gini impurity

$$\text{Gini}(S) = 1 - \sum_{k=1}^{K} p_k^2 \ , \tag{B.2}$$

where p_k is the part of samples in class k inside the node S.

A forest model is able to handle non-linear data and complex feature interactions, keeping high robustness to overfitting with respect to a single tree. It works also performs well in large feature datasets and imbalanced datasets. It is though less interpretable than a single decision tree, and it generally requires a higher computing resource for both the training and prediction.

B.2.2 Logistic Regression

Logistic Regression is a linear classification model, able to estimate the probability of belonging to a specific class. It exists in two variants, a binary case, where it applies the logistic function to a linear combination of the input feature, and a multinomial case, where it performs multi classification by assignment of probability distribution over multiple classes. In the specific case of the thesis, the multinomial logistic regression was used, in order to classify text transformed via TF-IDF tokenization, reducing the high data dimensionality and sparseness. In this case, the classification is done through

$$P(y = c|\boldsymbol{x}) = \frac{e^{\boldsymbol{w}_c^{\mathrm{T}} \cdot \boldsymbol{x} + b_c}}{\sum_k e^{\boldsymbol{w}_k^{\mathrm{T}} \cdot \boldsymbol{x} + b_k}} \ , \tag{B.3}$$

where the classification relies on the predicted probability that the input $\boldsymbol{x}$ belongs to the class c, computed and normalized with the exponential of a linear $\boldsymbol{w}_c^{\mathrm{T}} \cdot \boldsymbol{x} + b_c$, which includes the weights and the bias of a specific class c to apply at the input vector. The actual prediction is evaluated with

$$\hat{y} = \arg\max_c P(y = c|\boldsymbol{x}) \ , \tag{B.4}$$

by selecting the maximum probability across the predictions, thus providing also calibrated probabilities in the outputs.

The strength of the model relies on its simpleness of interpretation, keeping a high robustness and efficiency. It is particularly suited for high dimensional sparse data, providing calibrated probabilities in outcome. The model though is limited to linear decision boundaries, and as the case scenario of the thesis, the imbalanced datasets required specific weighting or sampling prerequisites. Despite that, the training is low in computational costs and fast with respect to more complex models.

B.2.3 Neural Network

A neural network is a computational model so called because its structures reminds the neural interconnections inside a human brain. In fact, it consists of layers of interconnected nodes (called neurons, see Figure 4.7), able to transform the input through a series of weighted connections and non-linear activation functions. For the case study of this thesis, the neural model takes as input a vectorized representation of error messages (after a preliminary processing), producing eventually class probabilities with a final softmax layer. The actual mathematical representation for an input $\boldsymbol{x}$ entering a layer is

$$\boldsymbol{h}^{(l)} = \sigma(\boldsymbol{W}^{(l)}\boldsymbol{h}^{(l-1)} + b^{(l)}), \quad \boldsymbol{h}^{(0)} = \boldsymbol{x} \ , \tag{B.5}$$

where $\boldsymbol{W}^{(l)}$ is the matrix in layer l, $b^{(l)}$ the bias vector and $\sigma(\dots)$ the activation function (as mentioned in section 4.3 a ReLU factor). The softmax layer evaluates the classes probabilities of the input $\boldsymbol{x}$ with

$$P(y = c|\boldsymbol{x}) = \frac{e^{z_c}}{\sum_k e^{z_k}} \ , \tag{B.6}$$

so basically as the logistic regression model, but computed with the resulting scalars z_k from the last layer.

In general, neural networks can reproduce complex or non-linear models with good approximation, thanks to their flexibility in the architecture (basically controlling the neuron per layer and the number of layers), thus making them suitable for large and sparse datasets. On the other hand, the amount of data to train them must be high, and the costs to train them may be very high (depending on the architecture dimensions), and the operational working of a neural network due to its non-linearity is less interpretable. The most relevant feature for which the neural model was chosen relies on its possibility to capture semantic information form textual datasets, thus considering eventual synonyms or correlated words.

B.2.4 Classification metrics

In order to assess and judge the performance of the machine learning models able to perform classification, several standard metrics well mathematically define were used, namely Precision, Recall, F1-score and Accuracy. These metrics are actually extracted from the mathematical object called confusion matrix (widely used to show the classification results in chapter 4), a matrix which can be produce in two general and different classification cases: a binary classification, where the classes to be predicted are only 2, a multi classification, where the classes to be classified are several. In order to start to give some mathematical definitions, an example of a binary confusion matrix is shown in Figure B.3.

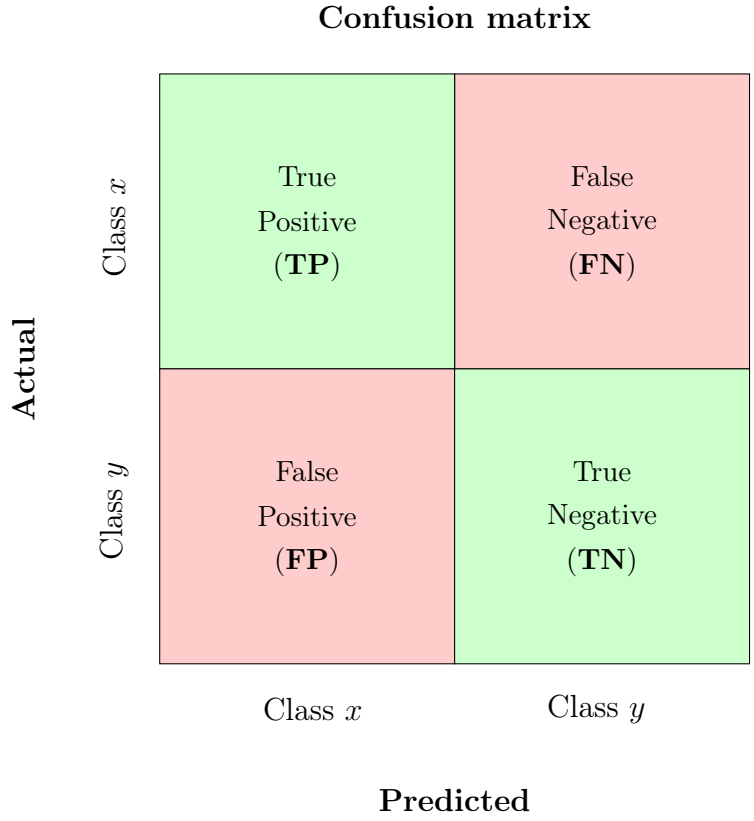


Figure B.3: *A schematic example of the entries of a* 2×2 *confusion matrix.*

Starting from a simple binary classification, all the metrics can be formalized according to the entries of this matrix. For instance, using the same letter scheme as the binary matrix, the *precision* is defined as

$$\text{precision} = \frac{\text{TP}}{\text{TP} + \text{TF}} \ , \tag{B.7}$$

representing the proportion of properly identified positive cases among all predicted positives (having high precision means that when the model predicts a positive case, it is likely to be correct), while the *recall* is defined as

$$\text{recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \ , \tag{B.8}$$

thus it measures how many actual positive samples are correctly identified (high recall means the model misses few positives). The actual overall success of the model is represented by the *accuracy*

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{FP} + \text{TN} + \text{FN}} , \tag{B.9}$$

which as the equation shows, it can suffer from imbalanced datasets since if one class is more frequent than the other, a model will mostly predict that class seeming accurate, bur it could actually perform poorly. Lastly, the *F*1-*score* is

$$\text{F1-score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} , \tag{B.10}$$

which is the combination of precision and recall, providing a balanced score when both FP and FN are high (this parameter is particularly important in imbalanced datasets, since it keeps track of detection and prediction).

For the multi-classification case, the metrics have of course different definitions, being the confusion matrix made of K different classes, thus having dimension $K \times K$, like the one in Figure B.4. In the figure, the rows are again the actual

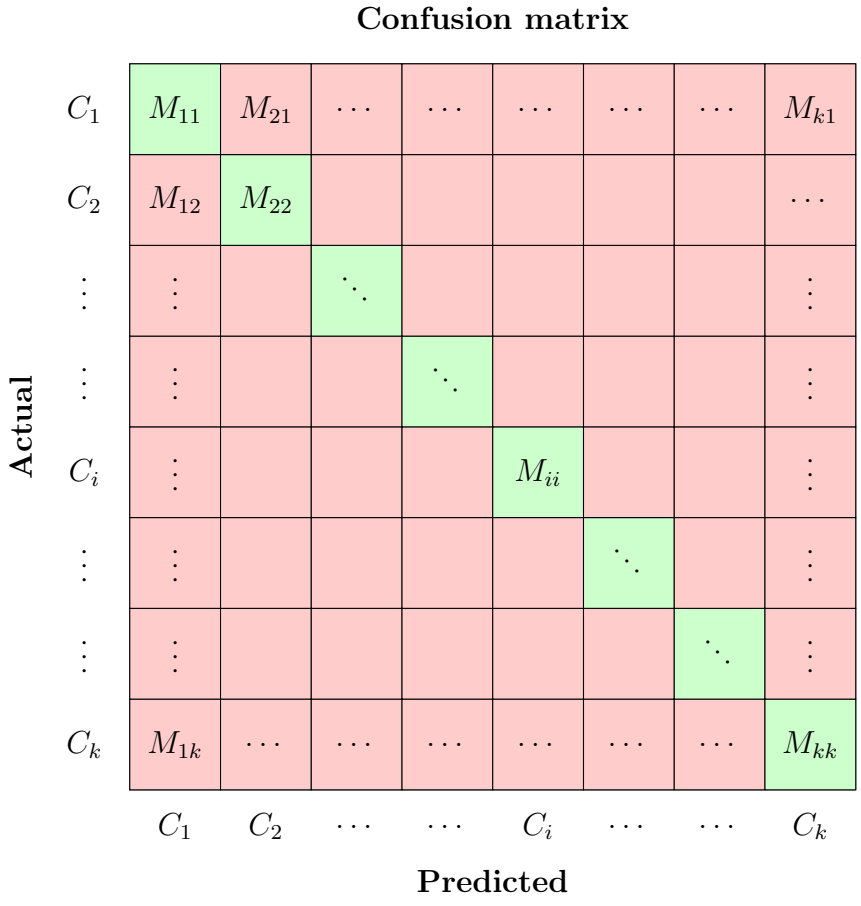


Figure B.4: *A schematic example of a confusion matrix for multi-classification purposes.*

classes, while the columns are the prediction performed on those classes. Thus, the elements on the diagonal M_{ii} are the corrected classified samples, while the

non-diagonal ones M_{ij} are the wrong classified entries for the class i assigned to a different class j. In this case, the accuracy becomes

$$\text{Accuracy} = \frac{\sum_{i=1}^{K} M_{ii}}{\sum_{i=1}^{K}\sum_{j=1}^{K} M_{ij}} \ , \tag{B.11}$$

basically extending the previous definition from the binary case, keeping the issue of not performing well in case of imbalanced datasets. Moreover, also the previous definitions can be extended, by simply treating the matrix as a binary one, by comparing a class C_i with all the others $K-1$ classes as a whole. In this case, we can define the precision as

$$\text{Precision}_i = \frac{\text{TP}_i}{\text{TP}_i + \text{TF}_i} \ , \tag{B.12}$$

which is again the number of samples predicted of the i-th class are correct, the recall is

$$\text{Recall}_i = \frac{\text{TP}_i}{\text{TP}_i + \text{FN}_i} \ , \tag{B.13}$$

again, the actual number of samples of the class i which are correctly identified, and lastly the F1-score

$$\text{F1-score}_i = 2 \cdot \frac{\text{Precision}_i \cdot \text{Recall}_i}{\text{Precision}_i + \text{Recall}_i} \ , \tag{B.14}$$

still the balance between precision and recall.

Eventually, another metric can be used in the multi-classifier case, since the outputs are multiple, a global average can be used to summarize the results, defined as macro average score. For all the previous definitions (apart from the general accuracy), it is possible to actually evaluate these scores with the following equations

$$\text{Precision}_{\text{macro}} = \frac{1}{K}\sum_{i=1}^{K} \text{Precision}_i \ , \tag{B.15}$$

$$\text{Recall}_{\text{macro}} = \frac{1}{K}\sum_{i=1}^{K} \text{Recall}_i \ , \tag{B.16}$$

$$\text{F1}_{\text{macro}} = \frac{1}{K}\sum_{i=1}^{K} \text{F1-score}_i \ , \tag{B.17}$$

in which every equation allows to evaluate the average relative metric (either precision, recall or F1-score) by considering all the classes' contribution equally, without taking into account the actual entries of the classes, thus considering equally all the classes. The macro averaging allows extraction of information even if some of those classes are low populated, since they are all treated with the same importance, despite their rarity across the dataset

In conclusion, all the presented metrics are the most used in the evaluation of the machine learning models, both for the binary classifiers and the multi-classifiers models, since apart from the accuracy which is a good indicator but not able to detect eventual imbalance, the other metrics are able to show the model capabilities on each different sample to be eventually classified.

B.3 ROC and PR plots

In this section the Receiver Operating Characteristic (ROC) curves and the Precision-Recall (PR) curves from the executed test on the unique and duplicate datasets are shown. Starting from the unique dataset results, the ROC and PR curves are shown from the `logreg` to the `NN` model.

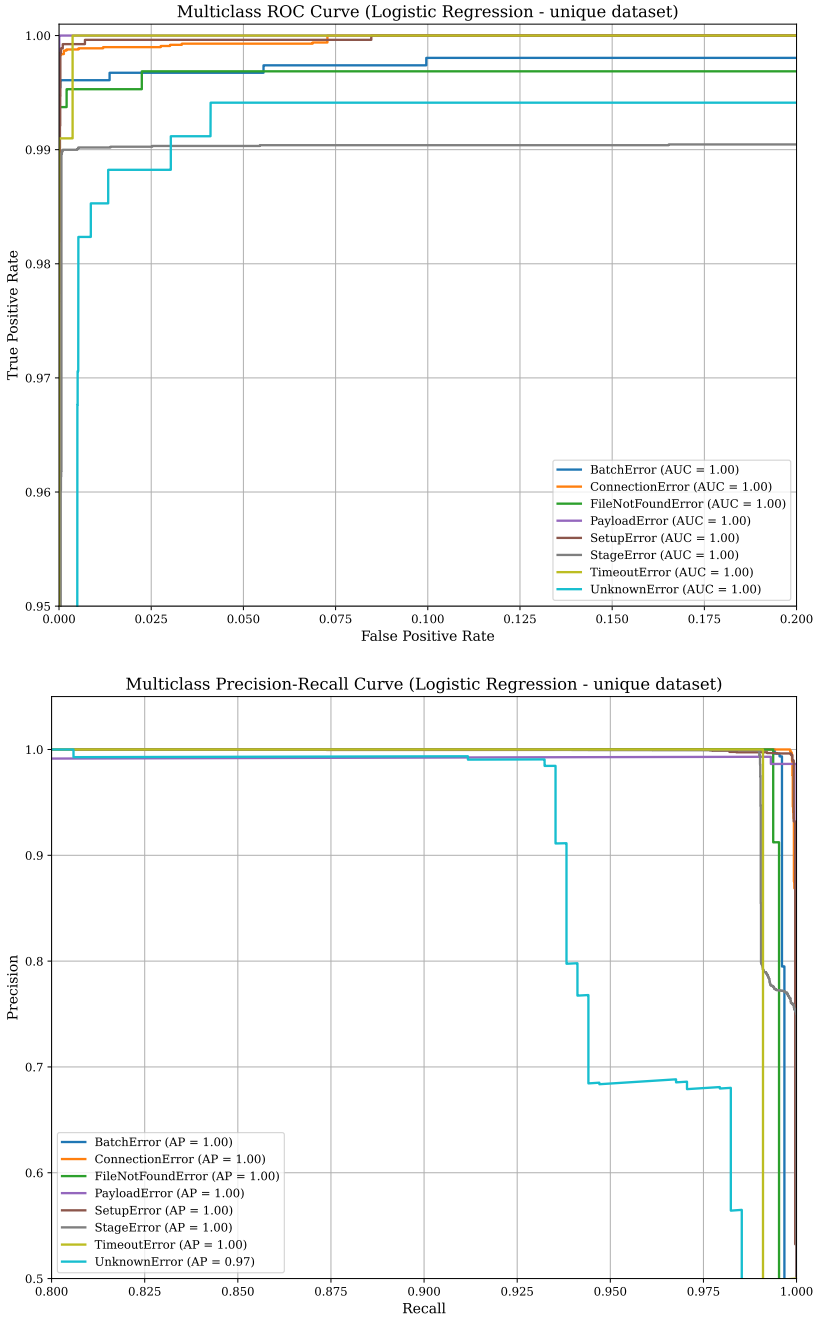


Figure B.5: *ROC and PR of the* `logreg` *model for the unique dataset.*

The plots are actually zoomed in a region where the differences of the various curves are visible (in most of the cases the values are actually approximated at the maximum value of 1). For all the unique plots, the performance were optimal, with some classes with lower performances (like `UnknownError`). Anyway, the

classification performances are again enhanced by these results, leading the model to be almost perfect in class separation with the unique dataset (i.e., the non-realistic one).

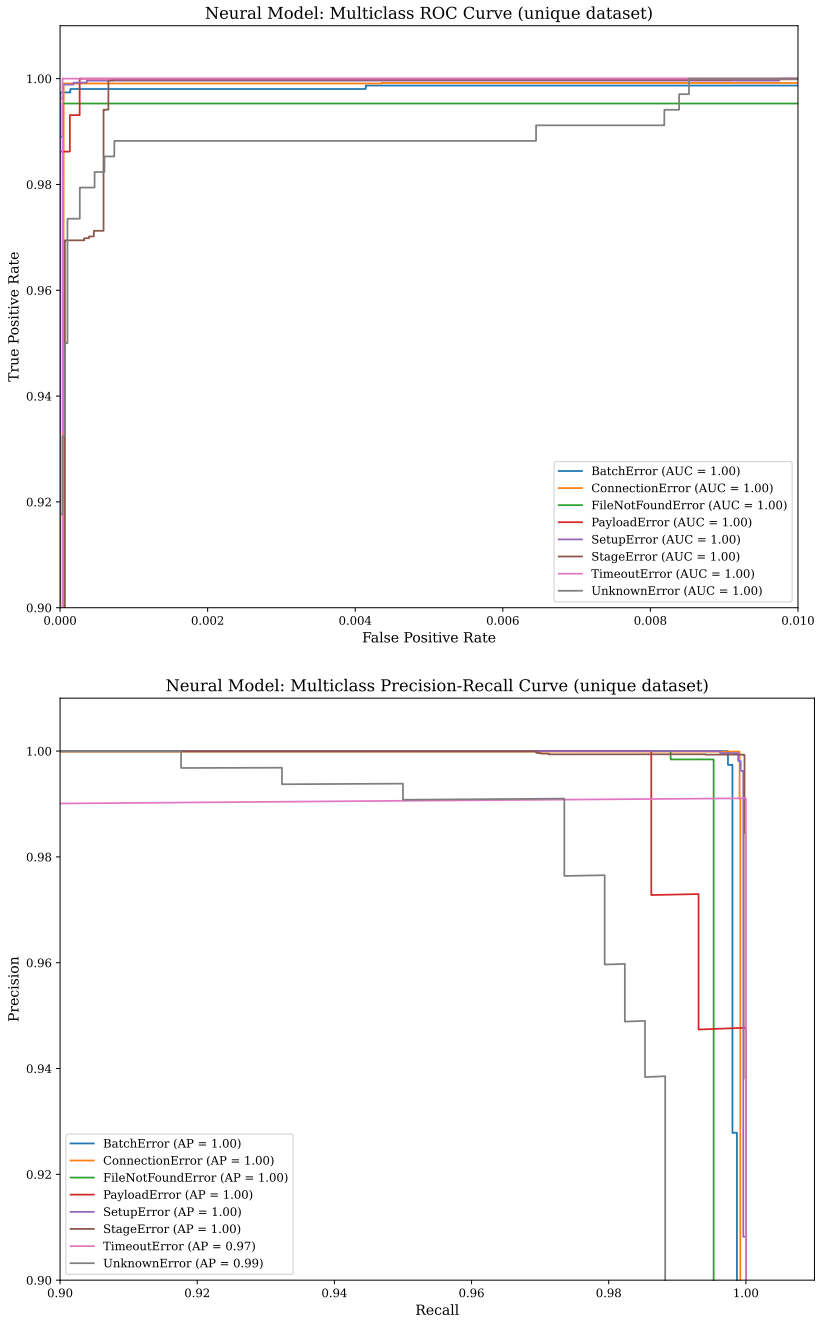


Figure B.6: *ROC and PR Curve of the* `NN` *model for the unique dataset.*

As the previous plots shown, the ranges for the False Positive Rate (FPR) and the Recall are actually zoomed into regions where the comparison could be in principle visible, but also keeping it fixed for each ROC and PR plots of the same model (e.g., the `logreg` FPR ranged in $[0 : 0.200]$).

In the duplicate case scenario, the capabilities of the `logreg` where actually excellent in the various class classifications, since in the ROC and PR curves are

all approximated as the maximum AUC and AP values (in fact, keeping the same intervals on the axis, the curves are actually overlapping in that region of interest).

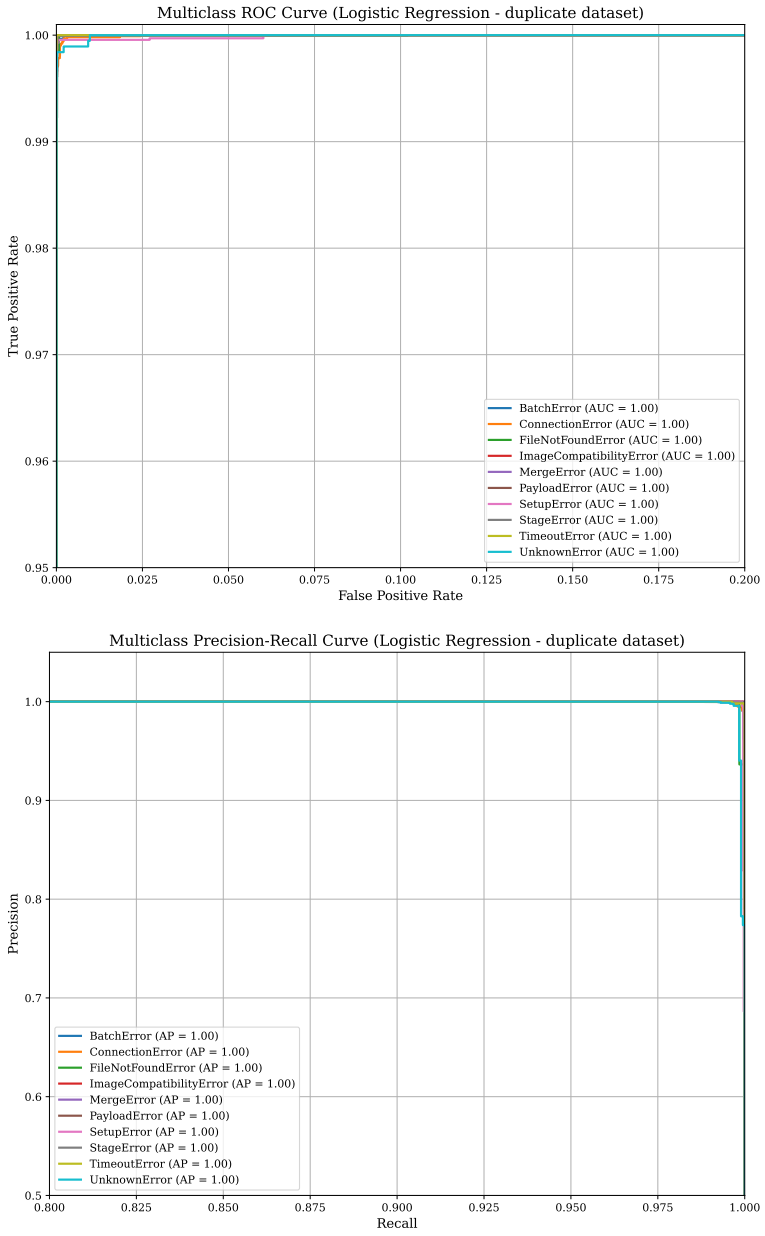


Figure B.7: *ROC and PR curves of the* `logreg` *model for the duplicate dataset.*

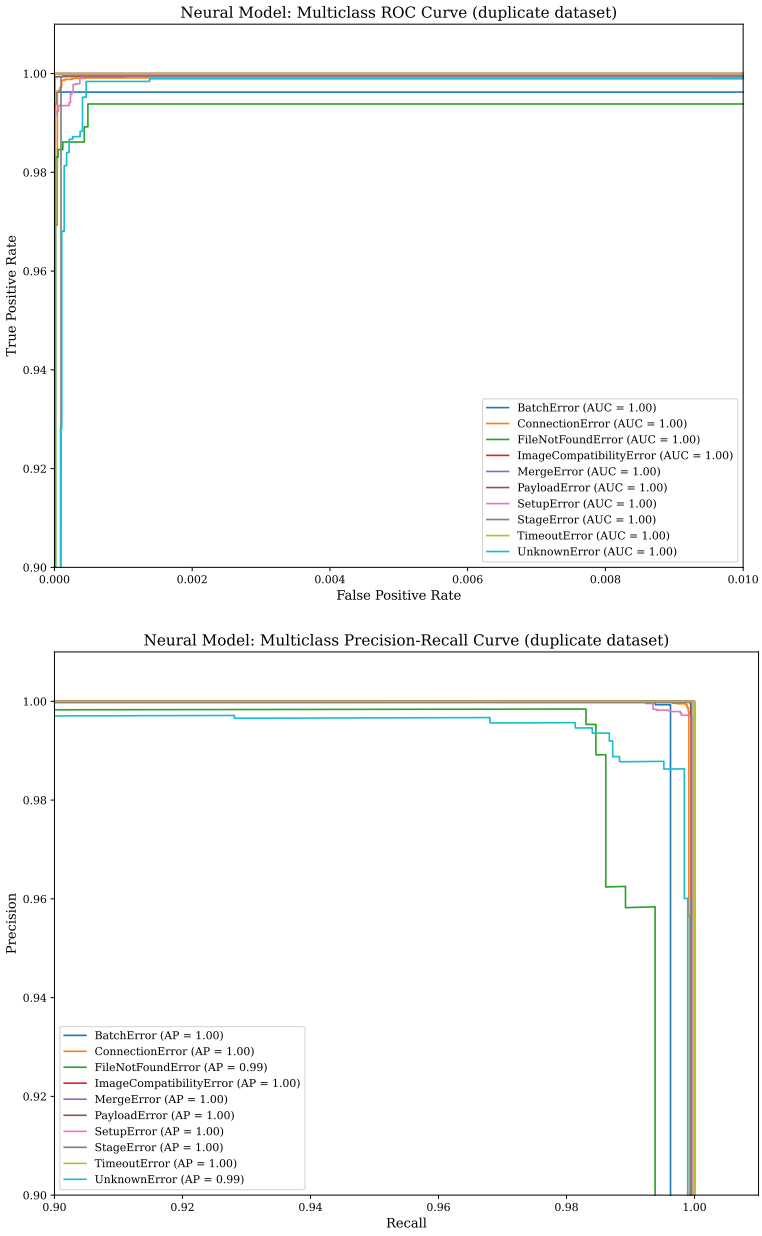


Figure B.8: *ROC and PR Curve for the* `NN` *model for the duplicate dataset.*

Bibliography

[1] Christopher M. Bishop. *Neural Networks for Pattern Recognition*. Oxford, UK: Oxford University Press, 1995. ISBN: 978-0198538646.

[2] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. New York: Springer, 2006. ISBN: 978-0387310732.

[3] T. Boccali, S. Dal Pra, and D. ... Bonacorsi. "Extension of the INFN Tier-1 on a HPC system". In: *EPJ Web Conf., 245 (2020) 09009*. 2020. DOI: 10.1051/epjconf/202024509009.

[4] Leo Breiman. "Random forests". In: *Machine learning* 45.1 (2001), pp. 5–32. DOI: 10.1023/A:1010933404324.

[5] CERN. *CERN Data Centre - Key Information (June 2022)*. 2022. URL: https://information-technology.web.cern.ch/sites/default/files/CERNDataCentre_KeyInformation_June2022V3.pdf.

[6] CERN. *CERN Data Centre - Key Information (Nov 2021)*. 2021. URL: https://information-technology.web.cern.ch/sites/default/files/CERNDataCentre_KeyInformation_Nov2021V1.pdf.

[7] CERN. *Facts and Figures about the LHC*. 2024. URL: https://home.web.cern.ch/resources/faqs/facts-and-figures-about-lhc.

[8] CERN. *PanDA: Production and Distributed Analysis System*. 2025. URL: https://panda-wms.readthedocs.io/en/latest/.

[9] CERN. *Storage at CERN*. 2024. URL: https://home.cern/science/computing/storage.

[10] CNAF (INFN). *Computing*. https://www.cnaf.infn.it/en/core-computing/.

[11] CNAF (INFN). *Networking*. https://www.cnaf.infn.it/en/networking/.

[12] CNAF (INFN). *Storage*. https://www.cnaf.infn.it/en/data-management-and-storage/.

[13] CNAF (INFN). *WLCG Tier-1 Data Center*. https://www.cnaf.infn.it/en/wlcg-tier-1-data-center-en/. Accessed: 2025-07-22.

[14] ATLAS Collaboration. *Open Data for Research*. 2024. URL: https://atlas.cern/Updates/News/Open-Data-Research.

[15] ATLAS Collaboration. "The ATLAS Experiment at the CERN Large Hadron Collider". In: *Journal of Instrumentation* 3.8 (Aug. 2008), S08003. DOI: 10.1088/1748-0221/3/08/S08003.

[16] T. Diotalevi et al. "Collection and harmonization of system logs and prototypal Analytics services with the Elastic (ELK) suite at the INFN-CNAF computing centre". In: *arXiv* (2021).

[17] Elastic. *Filebeat: Lightweight shipper for forwarding and centralizing log data*. 2024. URL: https://www.elastic.co/docs/reference/beats/filebeat.

[18] Elastic. *Logstash: Server-side data processing pipeline*. 2024. URL: https://www.elastic.co/docs/reference/logstash.

[19] Apache Software Foundation. *Apache Kafka*. 2024. URL: https://kafka.apache.org/.

[20] Apache Software Foundation. *Apache Lucene*. 2024. URL: https://lucene.apache.org/.

[21] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. Cambridge, MA: MIT Press, 2016. ISBN: 978-0262035613. URL: https://www.deeplearningbook.org.

[22] Worldwide LHC Computing Grid. *WLCG - The Worldwide LHC Computing Grid*. 2024. URL: https://wlcg.web.cern.ch/.

[23] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. 2nd. New York: Springer, 2009. ISBN: 978-0387848570.

[24] David W. Hosmer, Stanley Lemeshow, and Rodney X. Sturdivant. *Applied Logistic Regression*. 3rd. Hoboken, NJ: Wiley, 2013. ISBN: 978-0470582473.

[25] INFN. *CNAF History*. 2024. URL: https://www.cnaf.infn.it/en/institute/history/.

[26] INFN. *History of the INFN*. 2024. URL: https://www.infn.it/en/infn-institute/history-and-mission/.

[27] INFN. *INFN Bologna*. 2024. URL: https://www.bo.infn.it/en/bologna-division/.

[28] INFN. *INFN-CNAF*. 2024. URL: https://www.cnaf.infn.it/en/institute/.

[29] INFN. *INFN Offices*. 2024. URL: https://www.infn.it/en/infn-offices/.

[30] CERN LHC Closer Look. *LHC Data Analysis*. 2024. URL: https://www.lhc-closer.es/taking_a_closer_look_at_lhc/0.lhc_data_analysis.

[31] L. Morganti et al. "Large Scale Data Handling experience at INFN-CNAF Data Center". In: (2023). DOI: 10.22323/1.414.0205.

[32] Pier Paolo Ricci et al. "The INFN-CNAF Tier-1 GEMSS Mass Storage System and database facility activity". In: (2012).

[33] Wikipedia contributors. *Worldwide LHC Computing Grid*. 2024. URL: https://en.wikipedia.org/wiki/Worldwide_LHC_Computing_Grid.

Ringraziamenti

Scrivere questi ringraziamenti segna la fine di un lungo percorso, dedicato alla ricerca e allo studio, iniziato dieci anni fa. Il mio cammino nel mondo dell'accademia è cominciato con gli studi in Fisica, è proseguito con la Fisica nucleare e subnucleare, per poi approdare alla "scienza dei dati" e alla computazione. Un percorso così lungo ha richiesto impegno e costanza, ma mi ha permesso di crescere sia dal punto di vista personale che professionale. Non sarebbe stato possibile raggiungere questo traguardo senza l'aiuto, la presenza e il supporto di molte persone, che desidero ringraziare con tutto il cuore in questo spazio.

Sento di dover iniziare i miei ringraziamenti da chi mi ha accompagnato in questo percorso da "dentro" l'accademia: il mio supervisore, il Prof. Alessandro Gabrielli, e il mio fidato compagno di ufficio, il Dott. Fabrizio Alfonsi. Senza di loro, probabilmente, il mio percorso si sarebbe interrotto molto tempo fa.

Un grande ringraziamento va a tutti i collaboratori del CNAF, con cui ho avuto il piacere di lavorare in questi anni e dai quali ho avuto la fortuna di apprendere tanto sul mondo del computing e della gestione di grandi infrastrutture di calcolo.

Rimanendo in ambito accademico, ma sconfinando anche nella vita privata, voglio ringraziare tutti i miei vecchi compagni di corso. Un ringraziamento speciale va alla *Quark Family*: Gianluca, Marco, Mattia, Gianfranco e Carmine, che insieme a me rappresentano probabilmente i migliori (e allo stesso tempo i peggiori) fisici sperimentali di tutti i tempi.

Un grazie speciale è dedicato a tutte le persone che ho la fortuna di chiamare amici. Non posso citarvi uno per uno in questo ringraziamento, altrimenti allungherei questa tesi di moltissime pagine, sappiate che vi voglio un mondo di bene.

Dedico solo piccolo spazio a coloro che sono cresciuti insieme a me, da quando ero un ragazzo fino ad oggi: i miei amici un po' *Naïf*: Mattia, Marco, Gianfranco, Ruo, Fabio e, in particolare tra loro, il mentore del mio intero percorso accademico, nonché mio compagno di avventure nella divulgazione scientifica, Daniele.

Infine, un ringraziamento va soprattutto alla mia famiglia. Anzi, per essere precisi, alle mie famiglie. In primis, la mia famiglia di nascita, che fin dall'inizio mi ha dato la possibilità di intraprendere un percorso nel mondo scientifico, sostenendomi in ogni difficoltà. Grazie di cuore a Corrado, Elena e Gabriela.

In ultimo, il grazie più grande va alla mia nuova famiglia, alla persona che ho conosciuto prima di iniziare il mio percorso di dottorato, che mi ha sostenuto durante tutto questo incredibile cammino e che ho avuto la fortuna di sposare prima della fine dei miei studi. Grazie davvero, Valentina, perché *non sono niente se non resti con me.*