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

ADVANCED COMPUTING TECHNIQUES FOR ONLINE AND OFFLINE CMS DATA PROCESSING IN THE HL-LHC ERA

Presentata da: Simone Rossi Tisbeni

Coordinatore Dottorato

Daniele Bonacorsi

Supervisore

Daniele Bonacorsi

Co-supervisore

Felice Pantaleo

Esame finale anno 2026

ALMA MATER STUDIORUM
UNIVERSITY OF BOLOGNA

PHD PROGRAM IN
DATA SCIENCE AND COMPUTATION
37th Cycle

Advanced Computing Techniques for Online and Offline CMS Data Processing in the HL-LHC Era

Presented by: Simone Rossi Tisbeni

Coordinator
Prof. Daniele Bonacorsi

Supervisor
Prof. Daniele Bonacorsi

Co-supervisor
Dr. Felice Pantaleo

Final exam year 2026

Abstract

The discovery of the Higgs boson in 2012 by the CMS and ATLAS collaborations at the Large Hadron Collider (LHC) at CERN marked a landmark achievement for research in particle physics. While the Standard Model has proven successful in describing the fundamental particles and forces of nature, many questions remain unanswered.

The High-Luminosity LHC (HL-LHC) represents the next major upgrade of LHC, scheduled to begin operations in 2030. The HL-LHC will deliver approximately ten times more integrated luminosity over its operational lifetime, with a dramatic increase in collision rate. The goal is to extend the reach of the experiments and explore data that could provide insight into new theoretical models. The challenges posed by the HL-LHC at CMS are high. Event sizes are projected to grow by roughly an order of magnitude, while the trigger acceptance rate is planned to increase to 750 kHz. Without significant advances in data acquisition, reconstruction algorithms, and computing infrastructure, these conditions would exceed the current storage and processing capabilities by orders of magnitude.

In this context, the Next Generation Triggers (NextGen) project was established. In a five-year collaboration between CERN and the ATLAS and CMS experiments, it aims at addressing these challenges with innovations in trigger and Data Acquisition systems, to increase the acceptance rate and efficiency.

This thesis presents my contributions to the NextGen project within Work Package 3, which focuses on rethinking CMS real-time data processing. My work covers three parallel topics: The development of a tool for automated parameter optimization for track reconstruction algorithms, using metaheuristics for multi-objective optimization; the benchmark of lossless and lossy data compression strategies for reducing the size of RAW data; and the deployment and maintenance of an automatic framework for the orchestration of detector calibration workflows.

The techniques presented in this thesis contribute to the broader computational efforts of the Phase-2 upgrade of the CMS experiment, aiming at preparing the experiment for future studies in the physics beyond the Standard Model.

Sommario

La scoperta del bosone di Higgs nel 2012 da parte di CMS e ATLAS al Large Hadron Collider (LHC) del CERN ha rappresentato un traguardo fondamentale per la ricerca in fisica delle particelle. Sebbene il Modello Standard descriva con successo le particelle fondamentali e le forze della natura, molte domande rimangono ancora senza risposta.

L'High-Luminosity LHC (HL-LHC) rappresenta il prossimo grande upgrade di LHC, con entrata in funzione prevista per il 2030. L'HL-LHC fornirà nel corso della sua vita una luminosità integrata circa dieci volte superiore, con un notevole aumento della frequenza delle collisioni. L'obiettivo è ampliare le capacità degli esperimenti e analizzare dati che potrebbero offrire nuove prospettive sui modelli teorici oltre il Modello Standard. Le sfide poste dall'HL-LHC a CMS sono considerevoli. La dimensione media degli eventi crescerà di circa un ordine di grandezza, mentre la frequenza di accettazione del trigger aumenterà fino a 750 kHz. Senza progressi significativi nell'acquisizione dati, negli algoritmi di ricostruzione e nelle infrastrutture di calcolo, queste condizioni supererebbero di diversi ordini di grandezza le capacità attuali di storage e calcolo.

In questo contesto è nato il progetto *Next Generation Triggers* (NextGen). In una collaborazione quinquennale tra il CERN e gli esperimenti ATLAS e CMS, il progetto mira ad affrontare tali sfide introducendo innovazioni nei sistemi di trigger e di acquisizione dati, per incrementare il tasso di accettazione e l'efficienza complessiva.

Questa tesi presenta i miei contributi al progetto NextGen nell'ambito del Work Package 3, dedicato alla riprogettazione del sistema di elaborazione dati in tempo reale di CMS. Il mio lavoro si articola su tre temi principali: lo sviluppo di uno strumento per l'ottimizzazione automatica dei parametri negli algoritmi di ricostruzione delle tracce, basato su metaeuristiche di ottimizzazione multi-obiettivo; la valutazione di strategie di compressione dati, sia lossless che lossy, per ridurre la dimensione dei dati RAW; e la realizzazione e manutenzione di un framework automatico per l'orchestrazione dei flussi di lavoro di calibrazione del rivelatore.

Le tecniche presentate in questa tesi contribuiscono agli sviluppi computazionali previsti con l'aggiornamento a Phase-2 dell'esperimento CMS, con l'obiettivo di preparare l'esperimento alle future indagini sulla fisica oltre il Modello Standard.

Contents

List of Figures

List of Tables

Introduction

The discovery of the Higgs boson in 2012 by the CMS (Compact Muon Solenoid) and ATLAS (A Toroidal LHC ApparatuS) collaborations at the Large Hadron Collider at CERN marked a landmark achievement for research in particle physics [1, 2]. While the Standard Model has proven successful in describing the fundamental particles and forces of nature, many questions remain unanswered. These include the nature of dark matter, dark energy, and the matter-antimatter asymmetry, which motivate the search for physics beyond the Standard Model.

The High-Luminosity Large Hadron Collider (HL-LHC) represents the next major upgrade of LHC, scheduled to begin operations in 2030 following the third Long Shutdown [3]. The HL-LHC will deliver approximately ten times more integrated luminosity over its operational lifetime, with a dramatic increase in collision rate. The goal is to extend the reach of the experiments and explore data that could provide insight into new theoretical models.

The challenges posed by the HL-LHC at CMS are high. The average number of simultaneous proton-proton interactions per bunch crossing will rise from approximately 60 in Run 3 to 200 in the HL-LHC era. Event sizes are projected to grow by roughly an order of magnitude to around 10 MB per event, while the trigger acceptance rate is planned to increase to 750 kHz, compared to the current 100 kHz. Without significant advances in data acquisition, reconstruction algorithms, and computing infrastructure, these conditions would generate approximately 100 exabytes of data per year, exceeding current storage and processing capabilities by orders of magnitude. Simply scaling existing solutions is impossible [4].

In this context, the Next Generation Triggers (NextGen) project was established. In a five-year collaboration between CERN and the ATLAS and CMS experiments, it aims at addressing these challenges with innovations in trigger and Data Acquisition (DAQ) systems [5]. The project explores advanced computing techniques, such as the use of heterogeneous computing architectures and Machine Learning (ML), to increase the trigger acceptance rate and efficiency.

This thesis presents my contributions to the NextGen project within work package 3 (WP3), which focuses on rethinking CMS real-time data processing for the HL-LHC

era. My work covers three parallel topics: automated parameter optimization for event reconstruction algorithms, data compression strategies for RAW data, and automatic workflow orchestration for detector calibration.

The thesis is organized into three parts, each dedicated to one of these technical challenges, preceded by an overview of the CMS experiment and the NextGen project.
Chapter 1 provides the experimental context, introducing the Standard Model of particle physics, the Large Hadron Collider, and the CMS detector. It describes the detector subsystems, the trigger architecture, the planned Phase-2 upgrades, the distributed computing model, and the software framework. This background establishes the technical requirements and constraints that motivate the developments presented in the thesis.
Chapter 2 presents the NextGen project, outlining its organizational structure, scientific objectives, and technical work packages. Particular emphasis is placed on the four tasks of WP3: accelerating reconstruction through heterogeneous computing and optimized data structures (Task 3.1), developing the CMS software in a distributed application (Task 3.2), reducing RAW data size (Task 3.3), and implementing fast calibration workflows (Task 3.4).
Part I presents the challenge of optimizing numerous parameters of different types in complex reconstruction algorithms, and the solution developed.
Chapter 3 introduces the theoretical foundations of multi-objective optimization, with particular focus on heuristic methods and the Multi-Objective Particle Swarm Optimization algorithm (MOPSO).
Chapter 4 describes the design and implementation of `patatune`, a Python framework for multi-objective optimization in high-energy physics. The chapter details the software architecture, the implementation of MOPSO, the performance metrics used for algorithm evaluation, and the interactive visualization tools developed for exploring the Pareto-optimal solutions. Benchmark results on standard test functions validate the implementation.
Chapter 5 presents the application of `patatune` to optimize pixel track reconstruction parameters in CMS. The chapter also describes the successful adaptation of the CMS reconstruction algorithm to a different detector geometry. The results illustrate the portability of the optimization framework when tuning reconstruction on different detector geometries, without requiring changes to the reconstruction algorithms.
Part II presents the storage challenges posed by Phase-2 data rates, and compares compression strategies to utilize the storage resources available optimally.
Chapter 6 provides the theoretical background on data compression, covering fundamental concepts from information theory. It describes entropy coding, dictionary-based methods, and block sorting techniques that form the basis for the compressors used in the benchmarks.

Chapter 7 characterizes the RAW event size from Run 3 data and Phase-2 Monte Carlo simulations, providing a detailed breakdown of the contributions from each subdetector. This analysis assesses the main contributors to the event size and defines the baseline measurements for the compression benchmarks. The chapter also discusses the potential for data reduction by storing low-level reconstructed objects rather than RAW detector data.
Chapter 8 presents comprehensive benchmarks of lossless compression algorithms on ROOT file formats. The study extends to GPU-accelerated compression and concludes with an analysis of lossy compression strategies.
Part III addresses the need for fast and reliable calibration for physics object reconstruction.
Chapter 9 describes the architecture and implementation of the Automation Framework, a tool that extends CMS's Prompt Calibration Loop to handle workflows that require data from multiple runs, utilize different data streams, or employ custom analysis tools beyond the standard CMS software framework. The chapter investigates the migration from Jenkins to GitLab Continuous Integration to streamline operations and reduce maintenance overhead.
Chapter 10 presents the Zero Bias dataset production workflow for muon background studies as a use case of the Automation Framework. This workflow automatically processes unbiased collision data to produce files tailored for the analysis of the muon detector backgrounds.

Together, these three parts provide practical solutions to the computational challenges CMS is facing as it prepares for the HL-LHC era: a flexible optimization framework, quantitative compression benchmarks, and a robust automation system. These establish methodologies and infrastructure that will support the evolution of CMS computing throughout Run 4 and beyond.

The thesis concludes with Appendix A, which presents all the results for the test functions used to validate `patatune`'s implementation, and Appendix B, which collects the tables breaking down the contributions of individual subdetectors to the RAW data size in Run 3 data and Phase-2 simulations.

Author's contributions

The author's contribution to Part I,Part II and Part III of this thesis are outlined in the following.

For Part I, focusing on `patatune`, the author contributed from the inception of the project. He participated in the development of the prototype during the 13th Patatrack Hackathon. He contributed to the deployment of the tool for tuning the Patatrack pixel track reconstruction and presented the results at the 27th Conference on Computing in High Energy Physics on behalf of the CMS collaboration. From the initial prototype, he has been the main developer of the tool, handling the entire software stack, the documentation, and the release of the package. He supervised the development of the web interface for the visualization of the results through the 2025 Open Lab summer student project. He also assisted in the development of the prototype of the Reinforcement Learning extension presented at the end of Chapter 4. Motivated by the results obtained with the tool on CMS pixel track reconstruction, he independently extended the benchmarks to the $\mathcal{ZDT}$ family of functions and the TrackML use case.

In Part II, the author contributed to the NextGen 3.3 Task across multiple facets. He developed the extension of the `edmEventSize` program and the visualization tool. He handled the measurement and characterization of the RAW files for Run 3 and Phase-2, and contributed to the writing of the 2024 Task Report. He performed the benchmarks for lossless data compression, presenting the results at the 23rd International Workshop on Advanced Computing and Analysis Techniques in Physics Research on behalf of the CMS collaboration.

As per Part III, the author is the current main maintainer and developer of the Automation Framework. He assisted in its deployment for the specific use case of the Muon's Detector Performance Group from the beginning, and developed most of the automated workflow of the group, including the automation of Zero Bias dataset production described in Chapter 10. He contributed to the extension of the tool throughout 2025 and independently started the feasibility study for the migration to Gitlab CI/CD.

Chapter 1

The Compact Muon Solenoid experiment

The Compact Muon Solenoid (CMS) experiment at CERN's Large Hadron Collider (LHC) is one of the most important experiments in high-energy physics. Building upon the theoretical framework of the Standard Model of particle physics, CMS investigates elementary particles and their interactions through high-energy proton-proton collisions. This chapter provides a comprehensive overview of the CMS experiment, from its historical context to its future upgrades, highlighting the data challenges that motivate the computational approaches developed in this thesis. The chapter begins with an introduction to high-energy physics and the Standard Model. Section 1.2 presents the experimental infrastructure that makes these studies possible, starting with CERN's accelerator complex and the LHC, its operational timeline, and the transition to the High-Luminosity LHC (HL-LHC) era. The following section is a detailed description of the CMS detector, including its subdetectors, triggering system, and proposed upgrades for Phase-2. The chapter concludes with an overview of CMS's computing model and data flow, and how the experiment's distributed computing infrastructure handles petabytes of collision data. This discussion leads to the data challenges that form the core of this thesis and the goal of the Next Generation Triggers (NextGen) project presented in Chapter 2.

The HL-LHC will present unprecedented challenges in data acquisition and real-time processing, requiring significant advancements in trigger systems to maintain physics performance [6]. The NextGen project is as a collaborative initiative to push beyond the planned Phase-2 upgrades, integrating cutting-edge computing techniques to enhance event selection, reconstruction, and data handling [5].

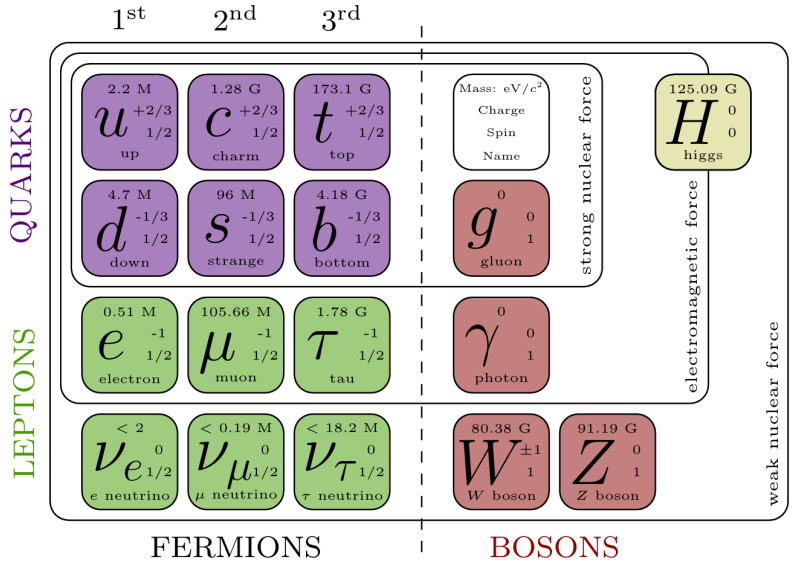

Figure 1.1: Chart of the Standard Model of particle physics [8].

1.1 The Standard Model of High Energy Physics

High-Energy Physics (HEP) studies elementary particles and the fundamental forces by which they interact [7, 8]. The study of elementary particle physics began with the discovery of the electron by J. J. Thomson in 1897, aiming to identify the fundamental constituents of the atom. From then on, the effort was to define a theoretical framework capable of justifying the interaction of energy and matter in physics and develop experiments that could confirm the existence of the theorized particles and forces. Throughout the 20th century, this framework was developed, thoroughly tested, and refined to reach the current formulation of the Standard Model of particle physics.

In the model, all matter is made of two classes of particles: *fermions*, themselves divided into *leptons* and *quarks*; and *bosons*, the mediators of the interactions between particles.

Leptons are classified based on charge (Q) and lepton number (L_e, L_μ, L_τ): the *electron* (e^-) with electron number $L_e = 1$, *muon* (μ^-) with muon number $L_\mu = 1$, and *tau* (τ^-) with tau number $L_\tau = 1$, all have electric charge $Q = -1$; while the corresponding *neutrinos* ν_e, ν_μ, ν_τ have charge $Q = 0$. For each of these lepton flavors, there is a corresponding *antiparticle*, with equal property numbers but opposite sign. The *antielectron* (e^+), for example, has a charge of $Q = 1$ and an electron number of $L_e = -1$. There exists an antiparticle for every neutrino as well, denoted with a bar over the symbol (i.e., $\bar{\nu}_e$ for the electron-antineutrino). This brings the total to 12 leptons.

Similarly, there are six flavors of quarks: *up* (u), *down* (d), *charm* (c), *strange* (s), *top* (t), and *bottom* (b). These have fractional charges, u, c, t with $Q = 2/3$ and d, s, b with $Q = -1/3$. Additionally, quarks have a property called color (*red*, *green*, and *blue*) that determines their interaction with the strong force. Together with their *antiquark* (denoted with a bar over the sign), each quark comes in all three colors, bringing the total to 36.

Just four fundamental forces govern the interaction between particles in nature: *strong*, *electromagnetic*, *weak*, and *gravitational*. Each of these forces is linked to a physical theory: Newton's law of universal gravitation and Einstein's general theory of relativity of the gravitational force; *electrodynamics*, in its classical notation by Maxwell and its quantum theory by Tomonaga, Feynman, and Schwinger, for the electromagnetic force; *electroweak theory* for the weak force, first presented in the relativistic quantum formulation by Fermi; and *chromodynamics* for the strong force, developed by Fritzsch, Leutwyler, and Gell-Mann. These fundamental forces are mediated by the exchange of bosons: *photons* (γ), carriers of the electromagnetic fields; *gluons* (g), carriers that mediate the strong force; neutral weak bosons (Z) and charged weak bosons ($W^{\pm}$), carriers that mediate the weak force.

The term High Energy reflects that the particles and interactions described by the Standard Model manifest most clearly at extremely high energies. Many elementary particles, such as the $W^{\pm}$ and Z bosons or the t quark, have masses that require significant energy to produce. These particles, according to quantum field theory and the energy thresholds set by Einstein's relation $E = mc^2$, are not commonly observed. To study them directly, physicists employ high-energy particle collisions, where kinetic energy is converted into the mass of new particles. Large HEP experiments enable collisions that recreate conditions similar to those of the early universe, when all Standard Model particles were abundantly present, allowing precise measurements of particle properties and interactions.

In 2012, the CMS and ATLAS collaborations at CERN (see Section 1.2) confirmed the existence of a fifth boson in the Standard Model: the *Higgs boson* (H) [1, 2]. This particle contributes to the origin of the mass of the other particles. Its discovery provided experimental validation of the theory proposed nearly five decades earlier by Higgs, Englert, and Brout [9, 10].

The elementary particles and their properties, as outlined in the current formulation of the Standard Model and confirmed by experiments, are summarized in the chart in Figure 1.1.

Despite the success of the Standard Model, several open questions remain. No direct experimental evidence supports any particular theory of physics beyond the Standard Model; however, various theoretical frameworks have been proposed. These include Grand Unifying Theories, which would unify the strong, weak, and electromagnetic

interactions, as well as Supersymmetry (SUSY), which proposes a symmetry between fermions and bosons and could explain dark matter and the hierarchy problem [11]. While these approaches remain unconfirmed by experimental data, they continue to motivate searches at current and future particle colliders.

The absence of clear signals for specific theories in recent years has shifted experimental strategies toward more model-independent approaches. ATLAS and CMS continue their mission through two complementary ways: precision measurements of Standard Model parameters and properties, and the search for phenomena that could reveal physics beyond the Standard Model. Precision measurements test the internal consistency of the theory and may reveal subtle deviations pointing to new physics, while direct searches look for rare or unexpected signatures that cannot be explained within the current theoretical framework [12]. Projects like the Next Generation Triggers initiative (Chapter 2) aim to enhance the experiments' sensitivity to rare or unexpected signatures, ensuring that even unanticipated forms of new physics can be discovered if they exist within the accessible energy range [13, 14].

1.2 The Large Hadron Collider

The Large Hadron Collider of the European Center for Research in Particle Physics (CERN) accelerator complex is the largest and latest addition among the high-energy physics experiments. It was designed from the start to probe the high-energy scales necessary for investigating fundamental particles. It started operations in 2008 in the same tunnel that housed the Large Electron-Positron (LEP) collider [15].

Built in a 27 km long circular tunnel positioned 100 m underground, LEP ran from 1989 to 2000, colliding beams of leptons. When bent on a circular orbit, these particles emit light via synchrotron radiation, naturally collimating in very intense beams. However, the energy lost by the particles with each revolution necessitates constant acceleration of the beam, which limits the machine's maximum energy. For these reasons, after reaching an energy of 104.5 GeV per beam and producing an enormous amount of precision data, it was dismantled to make way for the LHC.

The LHC is a particle accelerator and collider that can accelerate beams of protons and heavy ions up to 7 TeV of energy and collide them at four different points of its circle. Inside the accelerator, two separate beam pipes in an ultra-high vacuum hold the two high-energy particle beams traveling in opposite directions at nearly the speed of light (Figure 1.2).

The beams are guided by strong superconducting magnets and accelerated by radiofrequency cavities through electrical impulses. More than one thousand dipole magnets bend the path of the particles. At the same time, quadrupoles arranged symmetri-

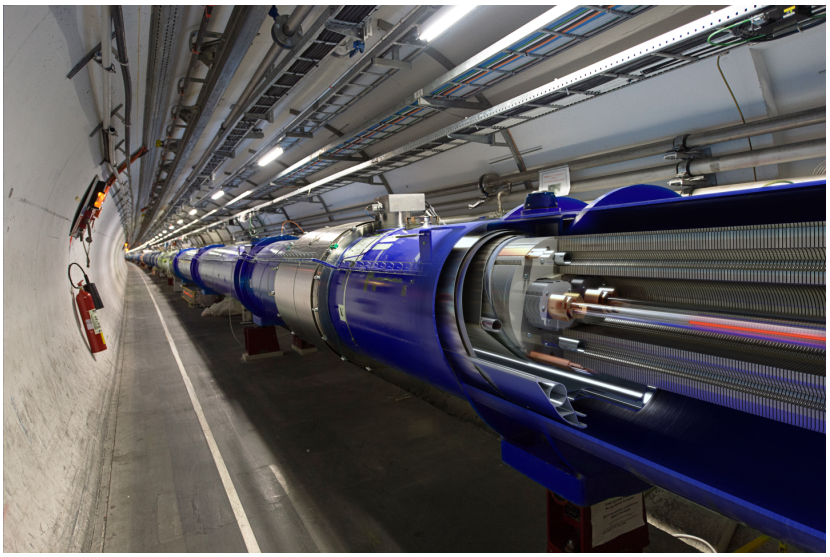

Figure 1.2: 3D cut of the LHC dipole (Image: CERN [16]).

cally around the beam pipe squeeze the beam both vertically and horizontally, keeping the particles in a tight beam.

To reduce the project's cost, the already existing infrastructure of LEP was employed for the LHC. The constraints of building the LHC inside the old tunnel posed challenges due to its length and diameter. The 27 km length limited the maximum attainable energy, requiring the design of novel superconducting magnets. Furthermore, the narrow tunnel required a new vacuum vessel design capable of containing both beams in a single volume. Finally, to further reduce the price, previous CERN experiments were utilized as part of the chain of stages to produce the beams that collide in the LHC. Although many of the steps have been upgraded over the years, some stages still utilize infrastructure dating back more than 50 years.

Figure 1.3 shows the CERN Accelerator complex and the full injection chain of LHC. Hydrogen ions, extracted from a bottle of hydrogen at the beginning of the injection chain, are accelerated to 160 MeV through the Linear Accelerator 4 (LINAC 4). During injection into the next stage, the Proton Synchrotron Booster (PSB), they are stripped of their two electrons, leaving only protons. In this stage, the protons are then accelerated to 1.4 GeV for injection into the Proton Synchrotron (PS), where the beam is further accelerated to 26 GeV. Additionally, the beam is grouped in bunches of one hundred billion protons, about 1.2 m long and separated by 7 m, and maintained this way throughout the remaining stages to the LHC. The following stage is the Super Proton Synchrotron (SPS). With its 1100 m radius, it is the second-largest machine in CERN's accelerator complex. It takes the bunches produced by the PS and accelerates them to 450 GeV, finally injecting them into the LHC in both rings.

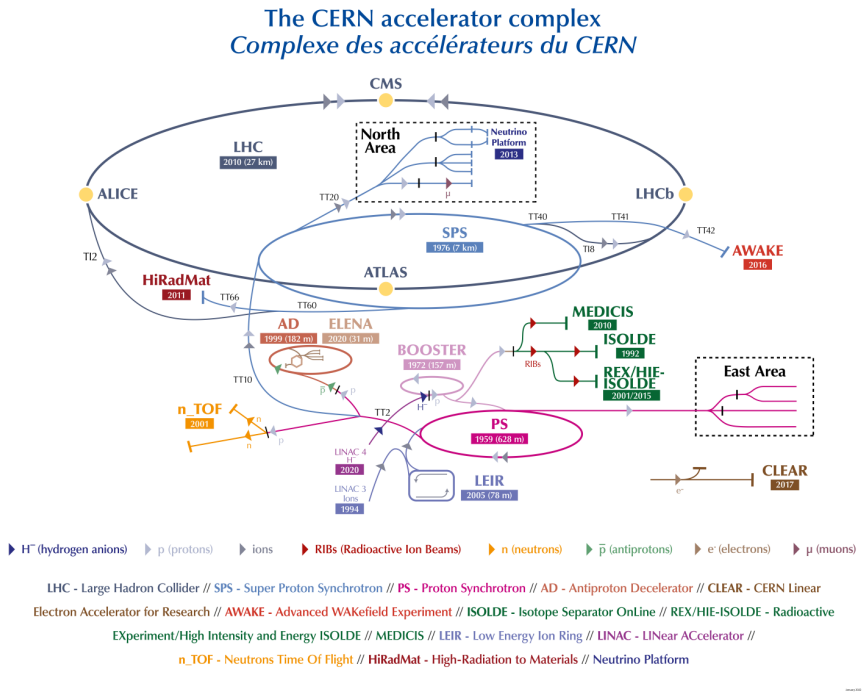

Figure 1.3: The CERN Accelerator complex, including the LHC and the entire LHC injection chain. Protons are accelerated in stages through four machines before reaching the largest ring (Image: CERN [17]).

When the rings are filled, the magnetic field is slowly ramped up, and once the nominal intensity is reached, the beams are steered into collision at four points of the circular path, where four large detectors are situated. A cascade of particles is produced at the vertex of the collision. New particles produced are typically unstable and quickly decay into stable particles. Specific detectors that look for the common signatures of these particles can detect and identify them.

ATLAS and CMS

The two largest detectors, ATLAS and CMS (described in detail in Section 1.3), sit at opposite sides of the circle, with ATLAS in the cavern closest to CERN's main site.

They are general-purpose detectors designed to investigate a wide range of physics [18]. However, while they have the same scientific goal, they use different solutions and infrastructure.

The ATLAS detector, composed of six different layers of subdetectors, records the energy, direction, and momentum of particles thanks to its magnet system. This system, composed of a central solenoid magnet and three toroid magnets (two at the endcaps and a massive one surrounding the center of the experiment), generates a high-intensity magnetic field perpendicular to the beam axis, bending the particles' path and allowing

a precise measurement of their momentum. CMS, instead, employs a superconducting solenoid that generates a magnetic field parallel to the beam axis.

The first layer of subdetectors that the particles coming out of the collision will interact with is the *inner tracker*. Composed of silicon detectors in CMS and silicon and gas for ATLAS, these detectors measure the precise paths of charged particles to reconstruct their trajectories. The curvature of the trajectory in the magnetic fields allows to derive the particles' momentum. The energy of neutral particles can be measured in the next layer of the detector, the *calorimeters*. Electromagnetic calorimeters measure the energy deposited by electrons and photons, while hadronic calorimeters measure the energies of particles that interact mainly by the strong force. Muons are the only particles that penetrate the hadronic calorimeters and are tracked in the outermost detector, the *muon chambers*. The two detectors observe 10^9 proton-proton collisions per second, resulting in a considerable amount of data to be analyzed. To cope with the size, event selection is done quickly in the *trigger* systems, reducing the billions of interactions to a few hundred events for storage and analysis (see Section 1.3.6).

The initial physics goal of the two detectors was to search for the Higgs boson. They are now focused on exploring physics beyond the Standard Model, including the search for supersymmetry, and the investigation of dark matter and dark energy.

LHCb

In contrast to the two general-purpose detectors, the LHCb experiment was built to study a very specific physics phenomenon: the production of a pair of b and anti-b quarks in a direction very close to the beam axis. For this reason, the LHCb detector, instead of having a cylindrical shape, is mainly built in a reclined pyramid shape that stretches out 20 m along the beam pipe with the apex located at the collision point [19].

It consists of a dense configuration of detectors that begins at the collision point and extends outward. At the closest to the apex sits the Vertex Locator (VELO). This detector consists of a series of silicon sensors aligned along the beam direction, which are used to accurately measure the collision vertices. This detector sits extremely close to the beam; therefore, it can be retracted from its rest position during injection, when the beams are not yet stable, to minimize the damage it receives. After the VELO, a dipole magnet provides a bending field for measuring momentum, followed by tracking stations located both upstream and downstream of the magnet to track the particles' paths. Particle identification is achieved through two Ring-Imaging Cherenkov detectors (RICH1 and RICH2), which distinguish between charged hadrons. Electromagnetic and hadronic calorimeters enable the identification of photons, electrons, and hadrons, while muon stations before and after the calorimeters provide efficient muon tracking and momentum determination.

Since its inception, the LHCb's purpose has been to investigate the explanation behind the significant asymmetry between matter and antimatter in the universe and to study weak interactions.

ALICE

Unlike the other experiments, ALICE (A Large Ion Collider Experiment) does not study proton-proton collisions; instead, it serves as a general-purpose detector for heavy ions [20]. It studies collisions between lead ions at the top energies of the LHC. In these collisions, lead nuclei reach extremely high temperatures and energy densities, forming a quark-gluon plasma that mimics conditions that were reached only for a few microseconds after the Big Bang.

The detector employs a central barrel, which is embedded in a solenoidal magnet, and a forward muon spectrometer. It measures an extensive range of observables. A silicon Inner Tracking System detects the vertices of the collisions in the barrel, and the Time Projection Chamber provides precise three-dimensional tracking of particles. A strip array detector and a Transition Radiation Detector are used for particle identification. The calorimeters measure photons, mesons, and the rates of jet production. The forward muon spectrometer is designed to detect muons from the decay of heavy quarks. Several small, specialized detectors are used for triggering or measuring global event characteristics. Following significant upgrades to the detector, a new online-offline computing system (O^2) enables near real-time data reconstruction with accelerators, reducing the data volume readout from the detector [21].

Together, these detectors allow the LHC to function as a multipurpose experiment for particle physics research. Ongoing upgrades to both the accelerator and detectors continue to expand their capabilities and extend the physics reach of the facility.

1.2.1 Data taking plans

After a three-year commissioning period, the LHC began delivering stable physics collisions in 2011, marking the start of its first data-taking. In this initial phase, the energy of the collisions measured at the center of mass was 7 TeV [15]. To quantify the collider's performance, the measure commonly used is the number of potential collisions occurring per unit time and unit area at the interaction point. This measure, called *instantaneous luminosity*, is given by Equation (1.1)

$$\mathcal{L} = \frac{f_{\text{rev}} \cdot n_b \cdot N_1 \cdot N_2}{4\pi\sigma_x\sigma_y} \cdot F \tag{1.1}$$

Where: f_{rev} is the frequency of revolutions of the beam (Hz); n_b is the number of bunches per beam; $N1$ and $N2$ are the number of particles per bunch in each beam; σ_x

and σ_y are the transverse beam sizes at the interaction point; F is a reduction factor that accounts for the crossing angle between the beams. Its unit of measure is $\mathrm{cm}^{-2}\mathrm{s}^{-1}$, with higher luminosity meaning more collisions per second.

To compare data-taking performance over time, the instantaneous luminosity is integrated over the period of stable collisions, yielding the *integrated luminosity*, usually expressed in *inverse femtobarns* (fb^{-1}). This quantity represents the total number of potential proton-proton interactions delivered to an experiment and is defined as:

$$L_{\mathrm{int}} = \int \mathcal{L}\, dt. \tag{1.2}$$

The first data-taking period started with an instantaneous luminosity of 3.8×10^{33} cm^{-2} s^{-1}, achieved using proton bunches spaced 50 ns apart and containing approximately 10^{11} protons per bunch. This configuration resulted in a rate of crossings between bunches of protons of 20 MHz and an average number of proton-proton interactions during a single *bunch crossing*, called *pileup* (PU), of around 25.

Since then, the LHC has operated in a well-defined cycle consisting of running periods (Runs), where the experiments collect data for analysis, and Long Shutdowns (LS), during which the accelerator and the experiments undergo upgrades and maintenance. The Runs typically last three to four years, separated by Shutdowns of various lengths. These Shutdowns are scheduled to allow time to perform the upgrades needed to increase the center-of-mass energy and the instantaneous luminosity in subsequent Runs. At the end of every year of data taking there is a Year End Technical Stop for the essential servicing of the accelerator and detector systems, with a lower impact on the operational cycle. Every few years, this period is extended into an Extended Year-End Technical Stop (EYETS), lasting several months, during which more substantial upgrades, repairs, or infrastructure work are carried out in preparation for the next phase of data-taking. Figure 1.4 shows the LHC schedule of physics Runs, planned Shutdowns from 2011 to 2030 and beyond, and the targeted luminosities and energies [22].

Between Run 1 and the current Run 3, the energy has increased from 7 TeV to 13.6 TeV. At the same time, the instantaneous luminosity has surpassed 2×10^{34} cm^{-2} s^{-1}, with the bunch space reduced to 25 ns, corresponding to a 40 MHz collision rate. As a result, the pileup has grown significantly, reaching an average of up to 60 interactions per bunch crossing. Run 3 is the latest operational phase of the LHC, beginning in 2022 and scheduled to conclude in 2026. A total of 250 fb^{-1} is anticipated to have been delivered to the experiments by the end of Run 3. The data collected during this period is used in part of the analysis and benchmarks presented in this thesis.

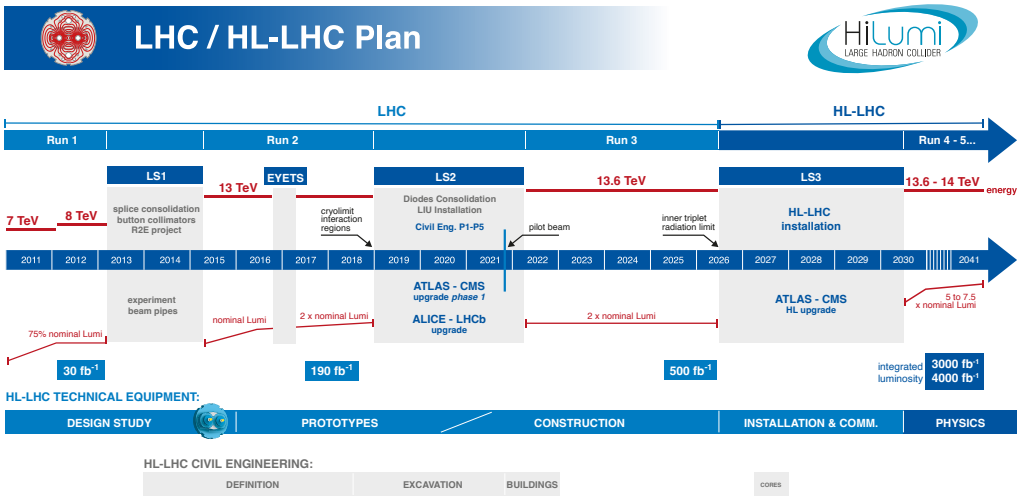

Figure 1.4: The current long-term schedule for the LHC. Approved by the CERN Research Board in September 2024 and revised in November 2024 [22]. The blue arrow shows the schedule from 2011 to 2029 and beyond. The red line at the bottom represents luminosity, while the red line at the top represents energy (Image: HL-LHC [3]).

1.2.2 High-Luminosity LHC

The transition from Run 3 to Run 4 of the LHC marks the beginning of the HL-LHC phase, which will significantly increase the collider's data production capabilities. This transition will be preceded by the third Long Shutdown (LS3), which is now scheduled to begin in the second half of 2026. This represents a delay of approximately seven and a half months from the original plan, extending the overall shutdown duration by around four months [23].

The primary reason for this delay is the challenges faced by the experiment teams during the upgrades, particularly for ATLAS and CMS. At the same time, the HL-LHC accelerator upgrade requires significant civil engineering, which has experienced further delays due to supply chain issues caused by COVID-19 and the sanctions following the Russian invasion of Ukraine. Operations are expected to resume in 2028, with full HL-LHC beam commissioning targeted for 2030.

LS3 will also include a broad program of maintenance, consolidation, and infrastructure upgrades across the entire accelerator chain [24]. The result of these activities will be to prepare the accelerator complex and detectors for the higher instantaneous and integrated luminosities of Run 4 and beyond.

HL-LHC aims to reach instantaneous luminosities 5 to 7.5 times the nominal value of LHC. This will allow the experiments to increase their data sample size by a factor of ten over the 12 years of HL-LHC operation compared to the LHC baseline program,

with the number of interactions per bunch crossing rising to a pileup of about 200 [24, 25].

The High-Luminosity upgrade will involve developing innovative superconducting magnets, vacuum, cryogenics, and machine protection to surpass the technical limits of the LHC hardware. It will also require new concepts for collimation and diagnostics, advanced modeling for the intense beams, and novel beam crossing schemes to maximize the physics output of the collisions [24].

A significant upgrade of the LHC injection system and the four large detectors is in production alongside the LHC upgrade.

The CMS Phase-2 Upgrade projects will replace or improve the CMS detector's subsystems to provide the necessary physics performance under the challenging conditions at the HL-LHC [26]. Installation of the upgraded detector systems started in LS2 and is planned to be completed in LS3. A significant experimental requirement for the upgrades will be differentiating hard proton collisions from the number of off-center collisions that occur during each beam crossing at high pileup. This will require a significant upgrade in the detectors' resolution to distinguish tracks originating from the collision vertex. This will be achieved by introducing additional channels in the Silicon Tracker and new detectors that will provide information on particles' 3D topology and time of flight. Additional details on the Phase-2 upgrade of the CMS detector will be presented in Section 1.3.7.

With the HL-LHC, it is estimated that ATLAS and CMS will collect approximately thirty times more data than what has been produced by the collider to date [27]. This has significant consequences for the detectors' operation and the performance of the reconstruction software. The two experiments will upgrade their trigger systems (see Section 1.3.6) to record 5–10 times as many events as they do today. It is anticipated that HL-LHC will deliver about 300 fb^{-1} of data each year [24]. Given that the total volume of data already collected by the LHC is nearing an exabyte, it becomes evident that the challenges to be addressed necessitate methodologies that extend beyond the scaling of existing solutions.

Simulations of Phase-2 data are used in part of the data analysis and benchmarks presented in this thesis, and most of the software developed has been designed to meet the upgrade requirements and the high-luminosity environment.

1.3 Compact Muon Solenoid

CMS is the second general-purpose detector among the LHC experiments. Weighing approximately 14000 t, it is 15 m high and 28.7 m long. The experiment takes its name from its compactness, given the density of the detector and sensor it contains. It has a very accurate and redundant muon system for detecting these heavy particles and, in

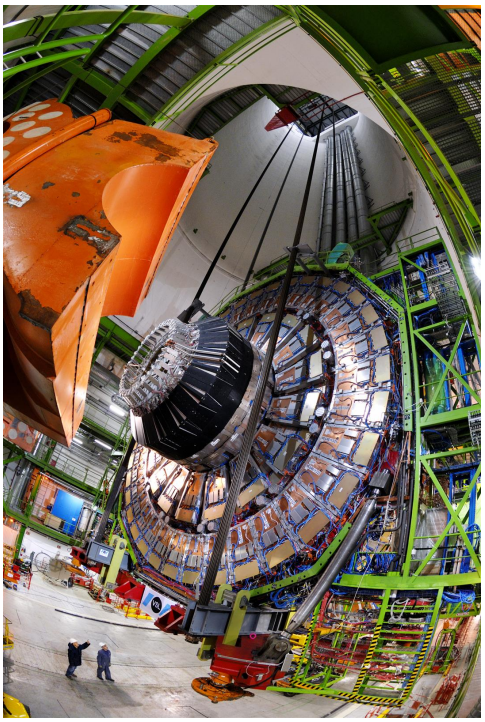

Figure 1.5: The lowering of the first CMS endcap in the experimental cavern (Image: CERN [28]).

contrast with ATLAS, it generates the magnetic field necessary to bend the trajectory of charged particles using a superconducting solenoid magnet. The magnet is the most powerful of its kind, capable of generating a magnetic field of 3.8 T [29].

CMS was built in fifteen "slices" on the surface, each lowered into the underground cavern, where it was finally assembled. This allowed each slice to be tested on the surface before assembling, and each subdetector to be designed with easy maintenance access in mind. This resulted in having to transport and lower the huge and delicate pieces of the detector, the heaviest of which weighed as much as 2000 t and took almost 12 hours to travel down the 100 m shaft to the cavern (Figure 1.5).

This construction approach allowed CMS to proceed independently of the excavation, which faced challenging local geology and delays. During the excavation, the engineers found the archaeological remains of a Roman villa, including pottery, tiles, and coins. They also had to solve the challenge of digging through underground water tables, which had to be frozen around the shaft to continue the excavation.

The final element of the detector was lowered into the cavern in January 2008, after eight years of production and assembly. It received its first beam in September of the same year and registered its first collision a few months later [15].

Figure 1.6 shows a 3D rendering of the CMS detector and its subsystems. The detector is built as a cylinder along the beam axis. It comprises different concentric

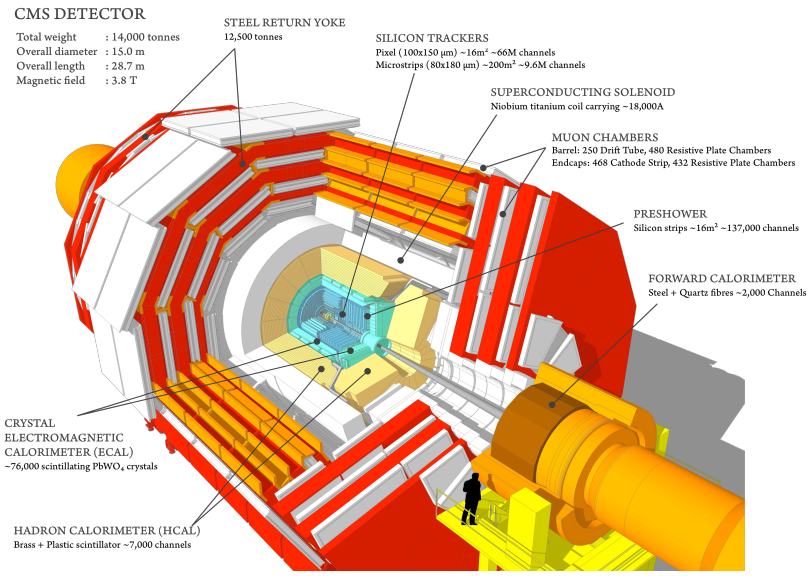

Figure 1.6: Cutaway view of the CMS detector [30].

layers, each containing the various subsystems and detectors. The innermost layer is the silicon tracker, which measures particle tracks to determine their momentum. The second layer is the electromagnetic calorimeter, which measures particle energy by fully absorbing them, and the pre-shower detectors, which are placed in front of the calorimeters in the endcaps. The third layer is the hadronic calorimeter, which measures hadron energy by sampling the secondary particles produced when hadrons interact with the absorber. The fourth layer is the superconducting solenoid, which also serves as the mounting point for the rest of the detector. Finally, the fifth exterior layer consists of muon chambers, designed to stop the most energetic muons and track their positions. Surrounding much of the CMS detector is the steel return yoke, which serves a dual function: it returns the magnetic flux generated by the magnet, helping to contain the magnetic field within the detector, and it acts as the mechanical support structure for the muon chambers. The forward calorimeter is located at the very ends of the CMS detector. It covers the regions close to the beam line and measures the energy of particles that emerge at shallow angles. The following section describes the coordinate system used in the detector; the subdetectors of the experiments are described in more detail in subsequent sections.

1.3.1 Coordinate system

This thesis employs the conventional coordinate system to describe the CMS detector, which is chosen to reflect the cylindrical symmetry of the detector [32]. Figure 1.7 shows a geometrical representation of the coordinate system. The origin is in the center of the detector with the $z-$axis aligned with the direction of the beam. The $y-$axis is

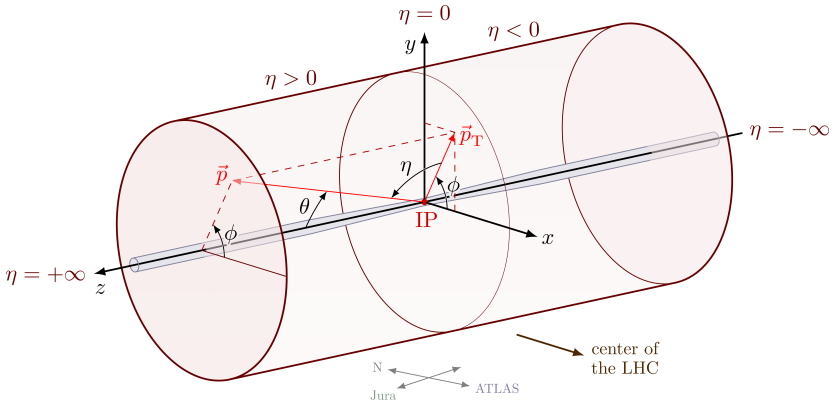

Figure 1.7: Geometric representation of the CMS Coordinate system with the cylindrical detector [31].

orthogonal to the ground, and the $x-$axis is parallel to the ground and points to the center of the LHC. The $z-$axis defines the longitudinal direction, and the $x-y$ plane is the transverse plane. The azimuthal angle $\phi \in [0, 2\pi)$ is the angle in the transverse plane from the positive direction of the $x-$axis. The polar angle $\theta \in [0, \pi)$ is the angle with respect to the z-axis. In describing the kinematics of collision events' objects, a particle's transverse momentum p_{T} is the projection of the momentum $\vec{p}$ on the transverse plane. In contrast, the longitudinal momentum p_z is the momentum projection onto the z-axis. Since in high-energy collisions, the distribution of the collision products is more skewed towards the beam axis, the quantity *pseudorapidity* η (Equation (1.3)) is more commonly used than θ.

$$\eta = -\frac{1}{2} \ln \left(\tan \left(\frac{\theta}{2} \right) \right) = \frac{1}{2} \ln \left(\frac{p + p_z}{p - p_z} \right) \tag{1.3}$$

1.3.2 Solenoid magnet

The superconducting solenoid magnet is cylindrical, located just outside the hadron calorimeter in the barrel, and measures 6 m in diameter and 13 m in length. It generates a powerful, uniform magnetic field of 3.8 T, aligned parallel to the beam axis [29]. This field is generated by circulating high currents through the windings of an Nb-Ti (niobium-titanium) superconducting cable, which is cooled to cryogenic temperatures to maintain its superconductivity. The resulting magnetic flux is returned through the steel return yoke, which surrounds the solenoid and provides a low-resistance path for the magnetic flux to complete its circuit. This design ensures the magnetic field remains strong and stable inside the detector, effectively closing the magnetic loop and preventing the field from spreading uncontrollably into the surrounding environment. The strong solenoidal field bends the paths of charged particles in the plane transverse to the beam, allowing the measurement of their momentum.

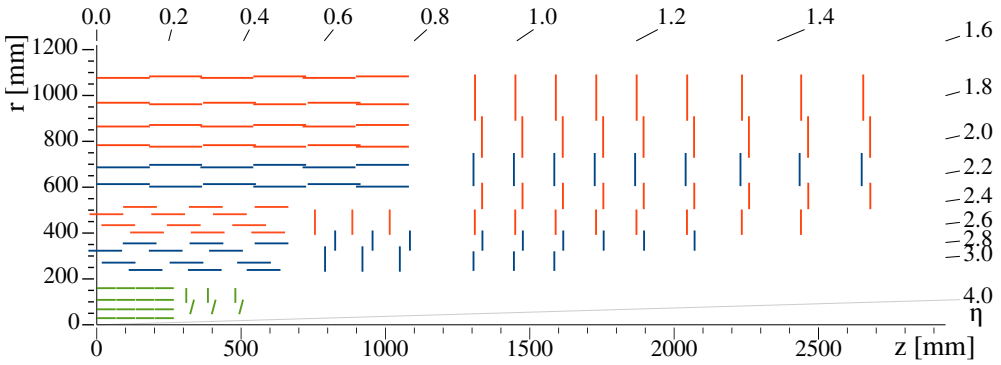

Figure 1.8: Diagram of one quarter of the Run 3 CMS tracking system in $r - z$ coordinates. The pixel detector is shown in green, and the strip modules are shown in red and blue [33].

1.3.3 Trackers

The particles' tracks, bent by the solenoid's magnetic field, are measured in the innermost part of the CMS detector. The tracking system consists of two subdetectors: the silicon pixel tracker close to the interaction point, and the silicon microstrip tracker as the outer part [34]. Figure 1.8 shows the tracker layers and their position in the $r - z$ plane of the detector (the detector coordinate system is described in Section 1.3.1).

The pixel detector contains 124 million pixels, allowing it to track the paths of particles emerging from collisions with extreme accuracy. It is the closest detector to the beam pipe, with cylindrical layers at 3 cm, 7 cm, 11 cm, and 16 cm from the beam, and disks at either end. Each of the four layers comprises silicon modules, split into silicon sensors (pixels) 100 μm by 150 μm. Due to the large number of particles passing through the detector so close to the collision, the pixel detector has been designed to be radiation-resistant yet lightweight, to alter the particles' path as little as possible.

The silicon strip detector is placed outside the pixels. It detects particles that pass the first detector up to a radius of 130 cm. The inner part of the detector, closest to the beam line, is structured in four concentric cylindrical layers, the Tracker Inner Barrel. At each end, they are delimited by three discs, the Tracker Inner Discs. The inner barrel and the discs are surrounded by six additional cylindrical layers, the Tracker Outer Barrel, and are completed by two Tracker Endcaps, achieving full coverage of the particles' paths.

For both the pixel and the strip detectors, the energy of the charged particles passing through the sensors is enough to eject electrons from the silicon atoms. A voltage applied to the sensor allows these charges to be collected as small electric signals, which are then amplified by an electronic readout chip into the detector's output. The signals are stored in the chip's memory for several microseconds and then processed before being converted into infrared pulses and transmitted over fiber-optic cables for

analysis in a radiation-free environment. The front-end electronics then digitize this analog pulse into discrete units: the Analog-to-Digital Converter (ADC) counts. For the silicon strip detector, each strip corresponds to a readout channel The ADC count is the integer value representing the amplitude of the pulse for that strip. Higher ADC counts indicate a stronger signal (more charge collected), up to the saturation limit of the ADC.

Additional information on the pixel tracker is presented in Chapter 5.

1.3.4 Calorimeters

The calorimeters are detectors that collect data on the energies of particles produced in each collision. The Electromagnetic Calorimeter (ECAL) is the inner layer of the system and measures the energy of electrons and photons by completely stopping them [35]. Hadrons, instead, pass through the ECAL and are stopped in the outer layer by the Hadron Calorimeter (HCAL) [36].

ECAL comprises a cylindrical barrel with 36 modules of 1700 crystals (EB), and two endcaps of 15000 crystals (EE). The crystals are made of lead tungstate ($PbWO_4$), a material composed primarily of metal that is heavier than stainless steel but highly transparent. This crystal scintillates when electrons and photons pass through it, producing a shower of photons proportional to the particle's energy. Photodetectors at the back of the crystals detect the scintillation light and convert it into an electronic signal, which is amplified and transmitted through fiber optics. The material is fairly radiation-resistant, but under experimental conditions, it suffers radiation damage that affects the color of the crystal and the light traveling through it. This effect has to be accounted for during the analysis and requires a constant calibration during data taking (see Section 9.1). The preshower endcap detectors (ES) are placed in front of the ECAL in the endcaps. The detector contains two layers of thin lead converters, followed by silicon strip detector planes. The two sampling layers can determine the impact positions and profiles of the electromagnetic showers with fine granularity, compensating for the low resolution of ECAL in the forward region of the endcaps.

The HCAL, the last subdetector inside the solenoid, is situated outside the ECAL layers and measures hadronic energy. It is segmented into different regions to cover the detector geometry hermetically. The barrel section (HB) consists of 36 steel wedges, each weighing about 26 t. It forms the innermost layer of hadronic calorimetry inside the solenoid. Just outside the magnet coil lies the outer barrel (HO), which adds a few extra layers of scintillator to catch any particle that escapes the HB, ensuring full containment of hadronic showers. At either end of the detector, the endcap section (HE) also comprises 36 wedges and measures the energy of particles that exit through the solenoid ends. Extending even further into the forward region, the forward calorimeter (HF) is designed to resist the intense radiation near the beamline and captures particles

emitted at very small angles. The HCAL is a sampling calorimeter, made of alternate layers of metal absorber and plastic scintillators. When hadrons hit the plates, they can interact, producing secondary particles that flow through successive layers of the absorber, producing a shower in the scintillation layers. This leads to the emission of blue-violet light, which is absorbed by wavelength-shifting fibers. These fibers split the green light and carry it to the readout components. The optical signal is then summed into "towers" that cover the particle's path through the HCAL and measure its energy. Silicon Photomultipliers convert the optical signal into an electronic signal and send it out for analysis.

The calorimeters' hermetic coverage not only allows for accurate measurement of the energy of electrons, photons, and hadrons but also provides an indirect measurement of non-interacting, uncharged particles, such as neutrinos, by detecting an imbalance in the energy and momentum of particles emerging from both sides of the detector.

1.3.5 Muon system

The Muon system is the widest part of the detector. Since muons are 200 times more massive than electrons and positrons, they are the only charged particles that can penetrate several meters of material without losing much energy. Therefore, the muon system consists of different tracking layers located on the outer part of the detector, where only muons can leave clear signals.

The muon system is comprised of four different subsystems. 250 Drift Tubes (DT) are arranged concentrically around the beam line in the barrel part of the detector. They consist of a positively-charged wire immersed in a gas mixture of Ar and CO_2 [37]. When a charged particle passes through the volume, it knocks electrons from the gas atoms. The electrons drift towards the wire, generating a signal. The time it takes the electron to reach the wire is proportional to the distance of the muon crossing point, providing an accurate measure of the particle's spatial position.

In the endcaps, 40 cathode strip chambers (CSC) track the particle positions where the particle rate is higher. CSCs consist of arrays of positively charged anode wires crossed with negatively charged copper cathode strips within a gas volume. The electrons of the gas displaced by the muon generate a current in the wires and a charge pulse in the strips. Since the strips and wires are perpendicular to each other, the detector accurately measures the passing particle's spatial position, and the closely spaced wires make the detector fast enough for triggering.

610 resistive plate chambers (RPCs) form a redundant trigger system. RPCs consist of two parallel plastic plates separated by a thin gas volume. When a muon crosses the chamber, it ionizes the gas, triggering an electron avalanche. The signal passes through the transparent plates and is collected by external strips. This process gives a quick measure of the muon's momentum, which the trigger uses to select the data. Square

plates are used in the barrel part of the detector, while trapezoidal-shaped plates are used in the endcaps.

Finally, gas electron multiplier chambers (GEMs) are the latest addition to the muon system. A first layer of 144 chambers has been installed on the endcaps during LS2, and two additional disks are planned for installation during the Phase-2 detector upgrade (see Section 1.3.7). They cover the region close to the beam pipe and have high-rate capabilities in detecting muons. Like the other muon subsystems, GEMs are gaseous detectors. Their core component is a plastic foil coated with copper, perforated with hexagonally arranged microscopic holes, and filled with gas. Muons create electrons that drift toward the foils, where strong electric fields in the holes multiply them. The resulting avalanche induces a signal on the readout strips used in triggering.

The muon subdetectors are sensitive to backgrounds from sources such as natural radioactivity and cosmic rays. However, they are particularly sensitive to charged hadrons that escape the calorimeters and interact with the muon chambers, generating hits. This problem is addressed in more detail in Section 1.3.5.

1.3.6 Trigger system

At peak performance, the CMS detector records approximately one billion proton-proton interactions per second. To manage this amount of data, a dedicated trigger system is used to rapidly identify potentially interesting events. This system reduces the event rate from the initial 40 MHz collision frequency to approximately a few hundreds events per second, a rate compatible with storage and offline analysis capabilities. This task is performed in two separate levels [38].

The Level-1 Trigger (L1T) consists of Field Programmable Gate Array (FPGA) based processors. It uses energy and position measurements from the calorimeters and muon detectors in trigger primitives to select events at a rate of around 100 kHz within a fixed latency of about 4 μs. The L1T global trigger (GT) then makes a trigger decision based on these reconstructed particles' multiplicity and kinematic information, their proximity to each other, timing information, and beam presence. The configuration is implemented in a "menu" with several hundred trigger algorithms, selected based on the physics requirements. Upon a positive GT decision, the full detector data are read out for further filtering in the second-level trigger: the High-Level Trigger (HLT).

The HLT consists of a farm of processors running a version of the full event reconstruction software optimized for fast processing. The software-based system reduces the event rate to around 1 kHz before data storage. Similarly to the L1T, the individual triggers are configured in an HLT menu. The output is divided into several online streams depending on the trigger algorithm and sent to the storage system [39].

Before the start of LHC Run 3, CMS improved many aspects of its trigger system to boost the experiment's physics reach [40]. The L1T project added dedicated hardware

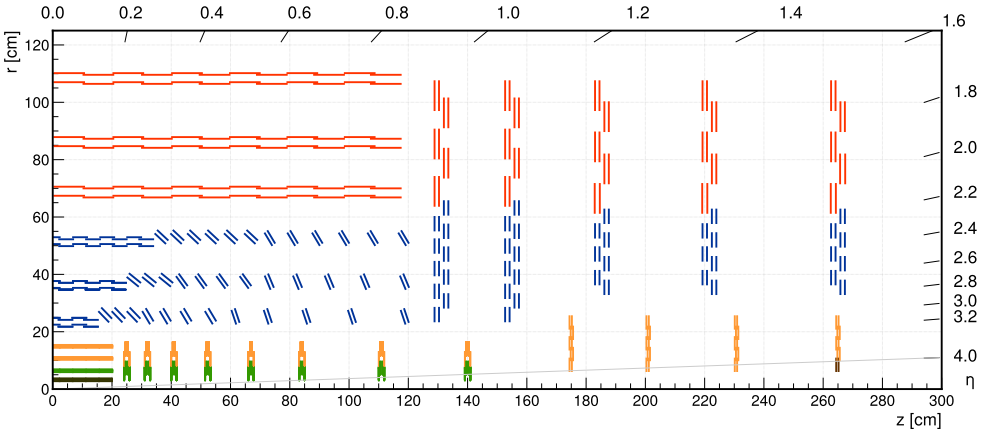

Figure 1.9: Diagram of one quarter of the Phase-2 CMS tracking system in r-z coordinate. The Inner Tracker is shown in green and orange, and the Outer Tracker in blue and red [42].

for the recording of objects reconstructed in the L1T without filtering: the L1T Data Scouting system. The data are stored as objects reconstructed by the L1T system, which allows for the study of signatures that feature an irreducible background and, therefore, are rejected in the regular L1T stream. Since the start of Run 3, the HLT has used graphics processing units (GPUs) in the trigger filter farm [41]. Algorithms implemented to run on both central processing units (CPUs) and GPUs are automatically directed to run on a GPU if one is available. In addition to the improved reconstruction algorithms, the trigger menus have been significantly expanded to explore new and unconventional physics signatures.

1.3.7 Phase-2 upgrade

During the LS3 the CMS experiment will be adapted to the requirements of the high-luminosity environment of HL-LHC. This upgrade concerns most components, including detectors, readout electronics, and the triggering system [26]. Since radiation damage is approximately linear with luminosity, the expected increase in luminosity during the upgrade will lead to a higher demand for radiation hardness in the detector. In addition, the increase in pileup will raise the demand for spatial and temporal resolution of the detectors. The trigger system will also require a substantial upgrade to cope with the increased data requirements and response time in the decision logic [6]. For these reasons, most of CMS's subsystems will require some upgrades.

The existing silicon tracking system, comprising both pixel and strip detectors, is scheduled for a complete replacement. The tracker system will be reorganized in an Inner Tracker made of silicon pixel modules and an Outer Tracker made of silicon modules with strips and macro-pixel sensors [33]. The new tracker layout is shown in Figure 1.9.

The upgraded tracker will offer significantly enhanced capabilities, including extended coverage in the region with high forward pseudorapidity, improved radiation tolerance in the innermost pixel layer closer to the beam, and increased granularity across all tracker regions. Importantly, the new tracker will also deliver tracking information directly to the level-1 trigger (L1T), a functionality currently restricted to the high-level trigger (HLT) [33].

A precision Timing Detector sensitive to minimum ionizing particles (MTD) will be installed in the gap between the tracking system and the calorimeters [43]. It is designed to have a high time resolution and provide hermetic coverage. It will consist of two main components: the Barrel Timing Layer (BTL) and the Endcap Timing Layer (ETL).

Improvements are being implemented across the calorimeter systems to handle higher luminosities and increased data rates [44]. The front-end and back-end electronics will be fully replaced in the ECAL barrel. The lead tungstate crystals will be sampled at a higher rate to enhance time resolution and suppress anomalous signals. For the HCAL, including the HB, HO, and HF sections, the focus is on upgrading the back-end electronics to handle increased Level-1 trigger rates. The HB will retain its current front-end system but adopt the same back-end boards as the ECAL. Similarly, the HO and HF will upgrade their back-end electronics for uniformity and higher performance. No significant hardware changes are planned for the detectors themselves.

As part of the upgrade program, a High Granularity Calorimeter (HGCAL) is being proposed to replace the existing endcap calorimeters [45]. This new calorimeter is designed to prevent the degradation of physics performance that current calorimeters would face if pushed beyond their designed integrated luminosity. Inspired by the silicon tracker's radiation hardness, silicon sensors were chosen as the active material for the bulk of the upgraded endcap calorimeters. The HGCAL is composed of two compartments: the electromagnetic (CE-E) and the hadronic (CE-H), both housed within a thermally controlled volume. It has around 6.5 million detector channels, divided into 50 layers. The first 28 layers form the electromagnetic section based on hexagonal silicon sensors divided into hexagonal cells. The following eight layers are similar, forming the front part of the hadronic section of HGCAL, but are single-sided and use a lighter baseplate. In contrast, the final 12 layers incorporate silicon modules and scintillator tiles. Plastic scintillator tiles optimize the overall cost of the HGCAL whilst maintaining excellent long-term performance in the less harsh regions.

Thanks to the already reliable radiation hardness, minimal upgrades are expected in the various subsystems for the muon system. The scope of the Phase-2 DT upgrade includes replacing the on-detector electronics and their receiving back-end electronics, as well as reorganizing the roles of both parts in the trigger and readout chains. For CSC a complete replacement of the back-end electronics in LS3 will support the higher

data rates and provide adequate throughput for L1T. Several upgrades have already been applied to the GEM and RPC systems, and will continue during LS3 such as the additional GEM disks, the improved chambers (iRPC) and front-end electronics' designs [46].

The Phase-2 L1T is designed to maintain or improve the CMS acceptance for all physics objects under the 200 PU conditions [47, 48]. These improvements profit from the upgrade of the readout systems of the subdetectors. The GT will make selection decisions based on trigger paths and neural network inference on the reconstructed objects from the upstream subsystems. The L1T Data Scouting system, as demonstrated at the end of Run 3, will be renovated with the Phase-2 upgrade to produce physics objects of comparable quality to those present in the HLT.

The upgrade of the Data Acquisition (DAQ) and HLT systems will need to handle a much higher event rate and significantly more complex events [6]. To achieve higher performance, the upgrade will focus on implementing a simplified HLT menu with improved timing while maintaining performance close to that of Phase-1. It will also adopt additional heterogeneous hardware and expand the code that utilizes accelerators. Both the data formats and the algorithm design must be adapted to leverage the capabilities of the heterogeneous hardware.

Additional details on the planned upgrade of the trigger system are discussed in Chapter 2.

1.4 CMS computing model

The experiment presents significant challenges not only in terms of detector design and physics goals, but also for computing and data handling. Due to technical and financial constraints, meeting these computing and storage requirements at a single location, such as CERN, would be impractical. Moreover, most CMS collaborators are based at institutions worldwide, and having access to non-CERN computing resources is advantageous for CMS computing.

To address this, the CERN experiments have adopted a distributed computing model: the Worldwide LHC Computing Grid (WLCG). This global net enables scientists to access computing, storage, and network resources independently of their physical location, while ensuring fair usage across the collaboration. These resources are used for critical and heavy tasks such as data processing, Monte Carlo simulation, data archiving, and user analysis.

As the experiment prepares for the increased luminosity of the HL-LHC era, the CMS computing model must also evolve. The projected rise in pileup increases the computational load, making it essential to explore more efficient software solutions and to integrate high-performance computing (HPC) facilities and hardware accelerators into

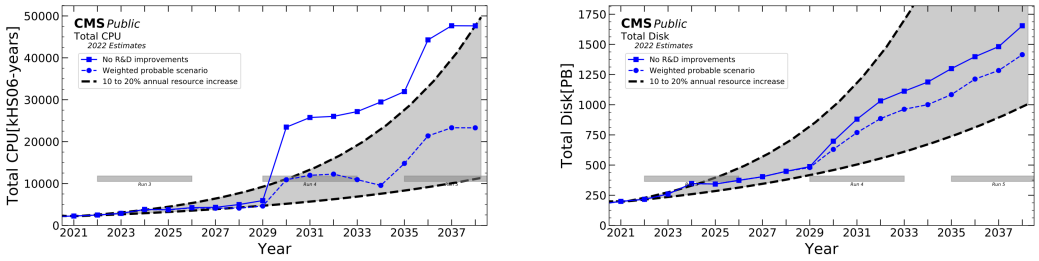

Figure 1.10: The CPU time (on the left) and disk space (on the right) projected requirements estimated for the annual CMS processing and analysis needs [4].

the processing workflow [4]. Figure 1.10 illustrates the expected growth in computing and storage demands.

1.4.1 Tier system and data flow

WLCG links national and international Grids to build and maintain a distributed computing infrastructure comprising around 160 computing centers in more than 40 countries. The network is arranged in *Tiers*, giving near real-time access to LHC data. WLCG provides seamless access to a wide range of computing resources, including data storage capacity, processing power, sensors, and visualization tools, among others. Users can submit jobs, which in turn are made of a complex set of atomic requests (i.e., requests for data in input, requests to apply algorithms and hence need for processing power, and requests to write and save the outcome of such processing), and this can happen from one of the many entry points into the system. The Grid systems check the credentials of the user, and search for available sites that can provide resources adequate to satisfy the user's requests, without any need for the user to have a deep technical knowledge of the available computing resources and systems that are serving their needs behind the scenes [49].

The Tier structure was formalized in the work done by the "Models of Networked Analysis at Regional Centers for LHC Experiments" team, which produced a model known as "the MONARC model" [50]. Following this model, the Tier levels are labeled with numbers from 0 to 3, with the number intended to indicate the level and complexity of services offered by the center: namely, the lower the number, the higher the level and quantity of services.

The experiment's computing models regulate the use of the Tiers' resources.

- Tier-0 is responsible for the safe-keeping of the RAW data (first copy), first pass reconstruction, distribution of RAW data and reconstruction output to the Tier-

1s, and data reprocessing during LHC down-times. The Tier-0 center for LHC is located at the CERN laboratory in Geneva.

The integrity and availability of the data are guaranteed by a redundancy mechanism that stores the data (at least) in two copies: one at CERN (the so-called "cold" copy) and (at least) one distributed to Tier-1s (the so-called "hot" copy).

Despite all the data from LHC passing through this central hub, Tier-0 provides a relatively low fraction of the total Grid computing capacity: less than 20%.

- Tier-1 centers are thirteen large computer centers, spread all around the globe, with sufficient storage capacity to store LHC data, and round-the-clock support for the Grid. They are responsible for storing a proportional share of RAW and reconstructed data on both disk caches (for high-performance access) and tapes (for long-term archival storage). They manage large-scale reprocessing and storage of the corresponding output, data distribution to Tier-2s, and safekeeping of a share of simulated data.

- Tier-2s are typically universities and other scientific Institutes, which can store sufficient data and provide adequate computing power for analysis tasks. They are used in central processing for CMS, especially for Monte Carlo simulation, totaling to around 50% of CMS computing resources [51].

- Tier-3 centers typically refer to local computing resources through which individual scientists access the WLCG facilities. Unlike Tier-2 centers, there is no formal engagement between WLCG and Tier-3 resources.

A schematic representation of the Tier levels, as defined by the MONARC model, is shown in Figure 1.11.

The CMS Computing Model has continuously evolved since it was first formalized in the Computing Technical Design Report (TDR) in 2005 [51, 53]. More flexibility is allowed in moving data between sites, with Tier-2 sites being able to transfer data to and from any Tier-1 site, and Tier-2 sites serving as sources of data for other Tier-2 sites. For the workflows, Tier-0 performs both prompt reconstruction and calibration. Tier-1 sites perform prompt and offline reconstruction of data and simulation. Tier-2 sites provide the bulk of the analysis computing and generate simulated events.

This data federation allows applications to read data from remote storage transparently to the user and enables processors across multiple sites to execute a single workflow using storage at one location.

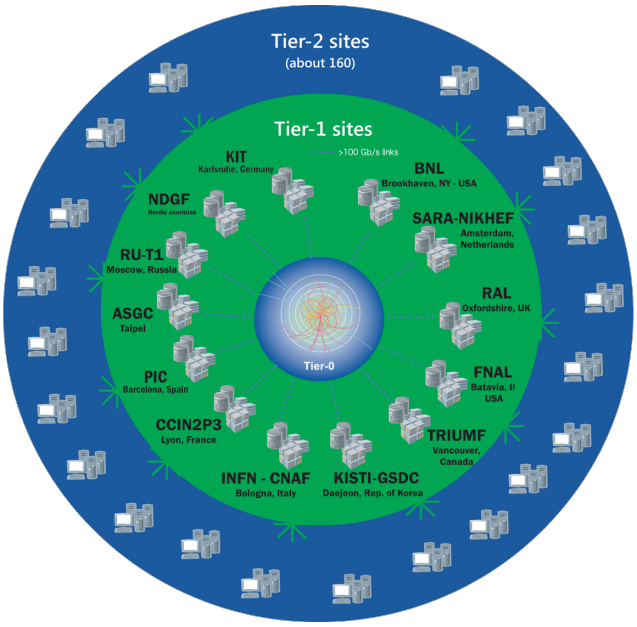

Figure 1.11: Schematic representation of the Tier levels of the WLCG [52].

1.4.2 CMSSW application framework

The collection of software used for the entire CMS analysis pipeline (simulation, alignment and calibration, and reconstruction of collision events) is built around a framework, an Event Data Model (EDM, which will be described in detail in Section 1.4.3), and a suite of services. It is known as CMSSW and has the primary goal of facilitating the development and deployment of reconstruction and analysis software [54].

At the heart of CMSSW is `cmsRun`, a single, lightweight executable that is dynamically configured at runtime via job-specific Python configuration files. These scripts specify the input data sources, define execution paths, configure individual processing steps with their parameters, and determine output destinations. The framework parses this configuration, dynamically loads only the necessary components, and executes them in the defined order. The components are called *modules*, and are self-contained C++ plugins implementing specific functionalities. Each module is compiled as a plug-in and registered with the framework via the plug-in manager, which resolves and loads the required shared libraries at runtime based on the configuration.

The execution model is organized into paths, which are ordered sequences of modules that define logical processing chains, and is centered around the production and analysis of an *event*. The framework ensures that modules are executed only once, even if they appear in multiple paths, thereby optimizing usage of computing resources [54].

Modules are categorized by function:

Sources handle the data reading, either from ROOT files [55], DAQ, or by creating an empty object that generators could later fill;

EDProducers read data from an event, produce something from the data, and output the product back into the event. A succession of modules may produce a series of intermediate products, all of which are stored in the event;

EDFilters read data from the Event and returns a boolean value to decide whether an event should continue down a given processing path;

EDAnalyzers read the data from an event, but it cannot modify its content or the execution path. It is used to analyze the data and produce outputs such as ROOT histograms;

OutputModules read the data from the Event and, once all the paths have been completed, stores the output of the execution.

In addition to the event loop, CMSSW incorporates a service system for handling global tasks such as geometry loading, logging, and memory monitoring. These services are initialized before event processing begins, and remain active throughout the job life cycle.

There is a strict separation between configuration and implementation. This allows the same module code to be reused across multiple workflows by simply changing parameters in the job configuration. Each module's configuration, the version of the code used, and the sequence of operations performed are recorded alongside the output; this allows for the complete reconstruction of the computational history of any dataset, which is necessary for the reproducibility of the reconstruction and analysis.

1.4.3 Event Data Model

The event is the central concept around which the Event Data Model is defined. Physically, an event results from a single readout of the detector electronics and the signals generated by particles in several bunch crossings.

Data are arranged into data tiers, where each tier contains different levels of information about the event:

RAW data are the unprocessed output from the detectors at Tier-0, containing low-level details like detector hits; it is not used directly in analyses, but for reprocessing and reconstruction;

RECO is the first processed version of RAW, with reconstructed physics objects (like tracks and jets); it is detailed and suitable for analysis, but the size is generally too big for widespread use;

AOD (Analysis Object Data) is a more compact version of RECO, optimized for most physics analyses by balancing data size and information complexity.

For the final analysis data format, two additional slimmed formats are defined:

MiniAOD where AOD-level objects are passed through more selection so that information can be discarded. The MiniAOD format is based on CMS's "Physics Analysis Toolkit" (PAT) object classes, which were originally developed for analysis-specific skims of AOD files [56];

NanoAOD is currently the most compact CMS analysis format, containing only high-level object information in a "flat" ROOT tree format. It is intended to hold enough information for most CMS analyses [57].

For Monte Carlo simulation, three additional data tiers are defined [54]:

GEN (Generation) data simulates the physics process at the particle level, generating the particles produced in a collision using theoretical models (no detector effects yet);

SIM (Simulation) data passes the GEN particles through a detailed detector simulation (typically GEANT4) to model how particles interact with the detector materials;

DIGI (Digitization) is the simulated detector responses into a format that mimics what the real detector would output; DIGI files are then mixed with pileup events to produce a simulated version of the RAW data.

Data from the CMS cavern is initially transferred in a temporary binary format and reformatted into RAW files in a process known as repacking. RAW files are then archived on disk and tape.

A subset of the data, typically 40 Hz, is selected as Express data. This is repacked and reconstructed immediately in small jobs, becoming available to users within 48 hours. The Tier-0 Prompt Calibration Loop (PCL) accesses the Express stream for automated calibration. Additional information on the calibration step is presented in Part III.

Prompt Reconstruction jobs on the full RAW dataset are launched 48 hours after data collection, producing RECO and AOD files. Skimming workflows run on RECO and AOD at Tier-1s to create targeted analysis datasets. Skims usually select under 10% of the original sample; larger selections use the full AOD.

Simulation jobs are primarily processed at Tier-2 centers, with Tier-1 centers contributing spare resources. Both tiers handle the reconstruction of simulated events,

adding pileup, and produce AOD and RECO formats. Users run analysis jobs primarily at Tier-2 centers and on available Tier-1 resources.

In all workflow steps, selection and compression are performed to reduce the data size. Additional information on compression is found in Part II.

1.4.4 Physics objects

Reconstructing physics objects in the CMS detector is a multi-step process that transforms raw signals from the subdetectors into meaningful physical quantities. At the core of this process is the identification and measurement of charged particles and the reconstruction of *tracks*. This reconstruction chain is crucial for understanding the events generated in collisions.

The charged particles produced from the proton collisions leave hits in the CMS tracker detectors, which are input into the track reconstruction algorithm to be grouped and their trajectories computed. The algorithm then reconstructs the charged particles responsible for the hits and estimates their momentum and position parameters. After getting the full information of the trajectories with all the hits, a selection is applied on the tracks to reject the misidentified tracks not associated with charged particles and evaluate the compatibility of their positions with the *primary vertices*.

The primary vertices are estimates of the positions of the proton-proton collisions reconstructed from the detector objects. The reconstruction algorithms first select the tracks that are compatible with being produced from the primary interaction region. This is achieved by adjusting the distance between the tracks and the beamline in the transverse plane, as well as the quality of the track fit. These tracks are then clustered into groups and assigned to primary vertices. Further detail on this process is presented in Chapter 5.

In the calorimeters, clustering algorithms group energy deposits from incident particles, such as electromagnetic showers from electrons and photons in the ECAL and hadronic showers in the HCAL. A *cluster* is a data object representing a localized group of detector signals spatially correlated and likely to originate from a single particle [58].

The *particle-flow* (PF) event reconstruction algorithm combines the signals from various subdetectors to determine the particle types, momenta, and energies. It is based on the concept of *global event reconstruction*, which correlates the basic elements to identify all particles in the event and measure their properties [59]. For example, electrons are identified by linking the track and the cluster in the electromagnetic calorimeter, leaving no signal in the hadronic calorimeter, while muons are identified as a track in the tracker detector that is linked to the corresponding track in the muon system.

The particle-flow algorithm is fundamental to reconstruct *jets*. Jets are composite objects formed from the collimated sprays of particles resulting from the hadronization

of quarks and gluons produced in high-energy collisions. Since quarks and gluons are not directly observable due to color confinement, jets serve as their experimental signatures.

The tracking, vertexing, clustering, and particle-flow reconstruction algorithms provide a detailed picture of each collision event. These reconstructed objects then form the basis for the successive offline higher-level analyses.

1.5 Summary

This chapter provided the experimental context of this thesis, introducing the Standard Model of particle physics, the Large Hadron Collider, and the Compact Muon Solenoid experiment. The description of CMS's detector systems highlighted its capabilities and the significant data challenges inherent to its operation, which will grow with the upcoming HL-LHC upgrade.

The presentation of CMS's trigger systems and computing model particularly emphasized the experiment's data processing pipeline, where the physics requirements and technical constraints create opportunities for new advanced computing solutions to be implemented. These data challenges will be partially answered by the work developed in this thesis. Chapter 2 will present the collaboration between the CERN Research and Computing section and the ATLAS and CMS experiments, to extend the performance of the trigger systems beyond the original planned upgrades of Phase-2.

Chapter 2

The Next Generation Triggers project

The Phase-2 upgrade of the CMS experiment is set to handle the higher requirements of the High-Luminosity environment in Run 4 of the LHC and beyond. As described in Chapter 1, the increased PU will result in a tenfold increase in the amount of data read by the detectors. To manage the substantial volume of data effectively, significant upgrades to the trigger systems for both the CMS and ATLAS experiments are planned.

In CMS, upgrades to the detector readout, L1T, and HLT systems, will allow up to 750 kHz compared to the current 100 kHz, up to 12.5 μs L1 Latency compared to the current 4 μs, and up to 7.5 kHz permanent event storage rate compared to the current 1 kHz [26].

In this context, the Next Generation Triggers (NextGen, logo in Figure 2.1) project is a collaboration between CERN (the Experimental Physics, Theoretical Physics, and Information Technology Departments) and the ATLAS and CMS experiments. The purpose of the project, with a five-year expected duration, is to extract more physics information from the HL-LHC data, beyond the original planned upgrades of Phase-2 [5].

The origins of the NextGen project trace back to 2022, when a group of private donors visited CERN to learn more about its scientific mission and future plans. This

Figure 2.1: The Next Generation Triggers logo [60].

initial meeting started discussions on the strategic needs of high-energy physics in the upcoming decades, particularly in the context of computing and trigger technologies.

Following a series of discussions and technical reviews, this initial engagement matured into a formal partnership in 2023. In October of that year, the CERN Council approved a collaboration with the Eric and Wendy Schmidt Fund for Strategic Innovation (under the management of the Hillspire Foundation) to support R&D on future trigger systems [61, 62]. The resulting project was officially launched in January 2024.

By enhancing the efficiency of event selection and background rejection, the project aims to improve sensitivity to rare or unexpected signatures that may indicate physics beyond the Standard Model. To achieve this goal, the project explores methodologies across multiple domains. These include the deployment of neural networks, the optimization of data formats and algorithms for real-time processing, and novel approaches such as quantum-inspired algorithms. In parallel, the project invests in high-performance computing solutions and more efficient hardware architectures, aiming to improve processing throughput and latency in the trigger pipeline.

The supporting strategy of the NextGen project also aims to achieve long-term impact with its developments. A central aspect is establishing partnerships with experts from both academia and industry, leveraging them to complement CERN's in-house capabilities. In line with the CERN Open Science Policy, intellectual property generated by the project will be released under open licenses, promoting transparency, reproducibility, and reusability of its results [63].

2.1 Project's organization

To improve coordination across the project, the effort is divided into several work packages (WPs). The division allows each part of the project to address specific technical or operational challenges independently. This granularity enables precise scheduling and tracking of progress into *milestones*. The following sections describe the project WPs and their goals.

2.1.1 WP1: infrastructure, algorithms, and theory

The first work package covers the development of common applications to be used across experiments and in other work packages [64]. The activities are split in discrete tasks.

Task 1.1 focuses on designing, procuring, deploying and operating the computing infrastructure (hardware and software) and platforms required to support the common tasks in WP1 and the specific activities in WP2 and WP3. This includes a local, dedicated cluster of the latest generation GPUs interconnected with a low-latency, high-bandwidth network in a setup similar to HPC supercomputer centers. Activities

of the WP also include the management of Continuous Integration and Development (CI/CD) services made available to the rest of the collaboration.

Task 1.2 focuses on developing a framework for fast inference of complex network architectures for the LHC online systems. This activity aligns with the existing efforts in the collaboration on common tools for Machine Learning (ML) inference on FPGAs, as part of the hls4ml project [65, 66].

Task 1.3 involves development on the software infrastructure necessary for enabling hardware-aware neural network training workflows. This activity is seamlessly integrated with the previous task and the hls4ml project.

Task 1.4 utilises quantum-inspired methods, such as Tensor Networks, to study complex quantum systems, test quantum hardware on small systems, and develop tools for quantum ML.

Task 1.5 covers new computing strategies for data modeling and interpretation, with the first objective being the development of software and algorithms for efficiently exploiting next-generation computer architectures in Lattice Quantum Field Theory simulations on the clusters provided by the work package. Additionally, the task aims to adapt the algorithms of Monte Carlo event generators to the new hardware infrastructure.

Task 1.6 has the ambitious goal of evaluating the impact of the new triggers on enhancing sensitivity to new physics scenarios. Theorists will develop benchmarks in close collaboration with the experiments and will include the study of concrete models that could extend the Standard Model.

Task 1.7 focuses on improving how HEP software leverages heterogeneous computing architectures such as GPUs and FPGAs. The work covers several core areas such as efficient scheduling of tasks across CPUs and GPUs using C++ coroutines, optimizing memory layouts for better performance on accelerators, developing GPU-ready HEP libraries with multiple backend support, enabling fast and portable ML inference from GPUs, and exploring alternative programming languages like Julia and Mojo to simplify development while maintaining performance.

2.1.2 WP2: enhancing the ATLAS trigger and data acquisition

Work package 2 has the goal of extending the ATLAS trigger capabilities for Phase-2 of the LHC. The focus is on R&D and novel approaches, while maintaining the same line as the existing efforts. The work is organized into seven tasks that cover different aspects of the trigger system [67].

Task 2.1 will focus on the new Level-0 Global Trigger subsystem of the detector, providing a full offline-like reconstruction of the calorimeter data in real-time for the entire 40 MHz rate. The initial studies involved the development of a generic Convolutional

Neural Network (CNNs) kernel to explore several use cases, such as characterizing objects like MET and jets under high PU, improving noise suppression, and estimating the primary vertex position.

Task 2.2 focuses on incorporating additional hit data from other subsystems into the muon reconstruction currently used at the Level-0 Trigger (L0), the first-layer hardware trigger of ATLAS. This will expand the L0 use case to encompass areas not included in the baseline HL-LHC system by developing triggers for particle signatures that are currently undetected. Some approaches include using CNNs for pattern recognition, based on drift distances from hits, to achieve high reconstruction efficiency.

The objective of Task 2.3 is to enhance the performance of the data acquisition system, extending the data available for selection in the trigger. The current system uses custom hardware, and the goal is to evaluate alternative, off-the-shelf technologies or to merge them with existing servers to simplify and reduce costs.

Task 2.4 involves developing upgrades for the Event Filter track trigger. It employs classical numerical and ML techniques to replace the current chain. The aim is to optimize the physics processing performance and utilize heterogeneous hardware architectures. The project is evaluating multiple pipeline candidates, including upgrading the current CPU algorithms and testing alternative classical options, such as graph-based seeders, Kalman Filtering, Graph Neural Networks (GNNs) for tracking and seeding, and clustering on FPGAs.

Task 2.5's goal is to fully exploit the extended coverage of the Level-0 muon trigger (Task 2.2) and the novel tracking infrastructure (ACTS) developed in Task 2.6 to improve the physics performance of the Event Filter muon track reconstruction. By migrating classical reconstruction to ACTS, the aim is to improve segment finding. Additional studies are being performed to test the effectiveness of algorithms based on GNNs.

Task 2.6 works in parallel with tasks 2.4 and 2.5 to develop the infrastructure needed to integrate advanced Event Filter tracking into the ATLAS software. The aim is to offload reconstruction algorithms to accelerators to improve the performance of the common reconstruction tools. A unified geometry is in use for reconstruction across multiple subsystems, with initial GPU support via external libraries. CPU and GPU pipelines have been optimized, demonstrating strong multi-core scaling and significant improvements in GPU seeding, memory utilization, and numerical stability.

Task 2.7 explores advanced reconstruction algorithms to boost ATLAS sensitivity to non-standard signatures, building on developments from Task 2.4 and 2.5. It focuses on novel tracking approaches, integration with other detector data, and ML. The goal is to enable efficient detection of unexpected physics signals, informed by theoretical studies from Task 1.6.

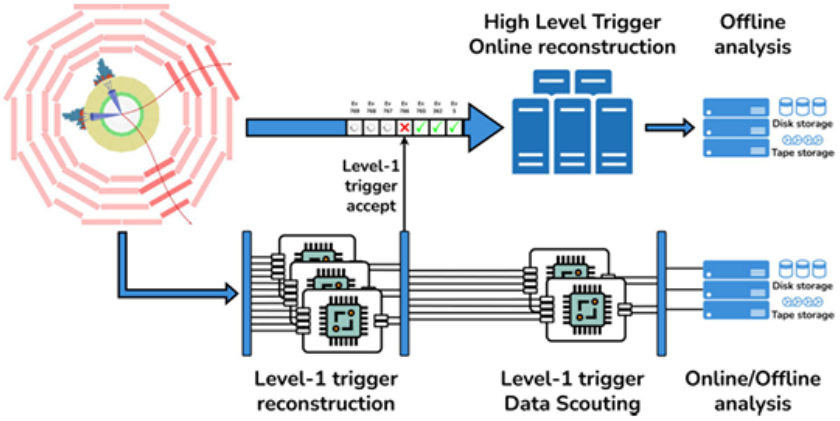

Figure 2.2: Overview of the data flow for the CMS trigger system with the proposed L1T Data Scouting stream.

2.1.3 WP3: Rethinking the CMS real-time data processing

Parallel to WP2, which focuses on ATLAS, work package 3 aims to rethink the CMS data acquisition system [68]. This objective is achieved through developments in both trigger layers (L1T and HLT), exceeding what was currently planned for Phase-2 (See Section 1.3.7). The integration of advanced computational techniques, as well as ML, is central to enhancing the triggers' ability to select and reconstruct events in real-time. This approach enables a more robust and adaptable data acquisition framework, addressing the evolving demands of HL-LHC operations, and reduces the need to reject events in the L1T and HLT. The project aims to complement the current workflow with a set of data streams that enable the analysis of data currently discarded by the trigger selection. Prior proof-of-concept R&D projects have already validated the foundational ideas underpinning these project tasks [69], demonstrating their potential to advance the state of the art in high-energy physics data processing.

WP3 tasks are divided into two main activities. Tasks 3.1 through 3.4 focus on upgrades for the HLT and will be covered in more detail in Section 2.2. Tasks 3.5 through 3.7 cover the development of a novel L1T Data Scouting stream.

Task 3.5 covers the enhancement of the Scouting system, in order to fully explore the potential of the detector for HL-LHC. The goal of Scouting is to capture the objects reconstructed by the L1T at full event rate of 40 MHz to run physics analysis on [70]. The scheme in Figure 2.2 shows the proposed data flow: the RAW data coming from the detector will be reconstructed in the L1T, the high-level data will be used for the selection on the full rate stream leading to the HLT; a dedicated Scouting stream will propagate the reconstructed data together with packed low-level data for additional analysis. This would allow targeting and studying signatures that would normally evade

the standard trigger chain, for example, large irreducible backgrounds or too complex algorithms that cannot fit the regular trigger system. The Scouting stream is decoupled from the regular trigger system, which is constrained by the latency required to perform the selection for the HLT, and grants the time and resources to run arbitrarily complex algorithms. A demonstrator of the Scouting system is already available for Run 3 but is currently limited by the L1T reconstruction [69, 71]. This task aims to leverage the improved L1T object reconstruction quality provided by the Phase-2 upgrade and tasks 3.6 and 3.7 of the project, to harness additional data for the Scouting stream through custom integrated boards. The goal of the task is to go beyond the preliminary studies and test alternative approaches for processing on the hardware and for data acquisition schemes, such as using accelerators with AI engines [72].

Task 3.6's objective is to enhance the L1T reconstruction using real-time AI for trigger decisions. Reconstructing high-level objects from low-level detector data, with physics algorithms and ML [73]. The first goal of this task is to utilize neural networks to collect raw, low-level information and propagate it downstream to the event selection, rather than discarding it with the first-pass reconstruction, as is currently done [74]. The second goal is to create a robust management infrastructure to handle the neural networks developed, to store the artifacts of the inference, and to manage datasets and models [73].

Task 3.7 focuses on developing advanced compression and anomaly detection techniques to make L1T Scouting feasible for HL-LHC conditions. The estimated RAW data volume for L1T is between 100 and 1000 PB per year, making storage impractical without aggressive data reduction strategies. This task explores the use of non-linear, lossy compression based on AI algorithms, such as autoencoders, to significantly reduce dataset size while preserving relevant physics content [75]. In addition to compression, autoencoders will be studied for their potential in real-time anomaly detection, an application already viable for Run 3 data [76].

2.1.4 WP4: education programmes and outreach

In addition to its research and development activities, NextGen incorporates a comprehensive, multidisciplinary training programme for its researchers [77]. The project will also organise events, workshops, and conferences to engage the scientific community and promote the dissemination of the results.

Task 4.2, additionally, focuses on developing the CERN Software Training, Education, and Advanced Modules programme (STEAM), which trains postgraduates and researchers in advanced computing and data science for HEP. Covering topics of interest for the project, such as ML, trigger systems, and quantum computing. It offers lectures, hands-on sessions, and hackathons led by experts from academia and industry.

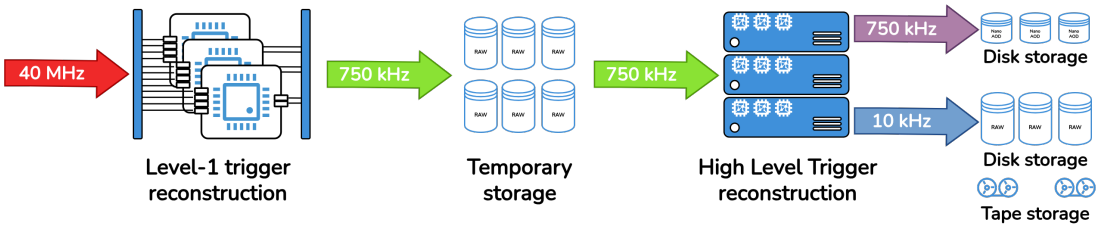

Figure 2.3: Overview of the HLT data flow with the proposed data stream of 750 kHz for Phase-2, and the existing 10 kHz stream of Run 3.

2.2 NextGen for the High Level Trigger of CMS

Tasks 3.1 to 3.4 of WP3 focus on the upgrade planned for Phase-2 to the HLT system of CMS. In particular, they plan to extend the reconstruction capabilities of the HLT beyond what is currently planned for the upgrade. The ambitious goal is to provide offline-like performance to the reconstruction for the full event rate after the L1T selection of 750 kHz and save the reconstructed object in a dedicated stream analogous to the L1T Data Scouting stream for successive analysis. Figure 2.3 shows the proposed data flow for HLT.

2.2.1 Real-Time Reconstruction Revolution

Faster reconstruction

NextGen's Task 3.1 aims to reconstruct all events selected by the L1T at a rate of 750 kHz during the Phase-2 data collection. The goal is to achieve reconstruction quality comparable to that of offline methods at the HLT. This approach will enhance physics analyses by utilizing the rich statistics and high-quality reconstructed objects without the need for the exact precision that offline reprocessing provides. To achieve this goal, the main objective of the task is to increase the reconstruction throughput to keep up with the data rate provided by L1T [68, 78].

The success of previous experiences in CMS, such as Patatrack's pixel track reconstruction [79], has demonstrated that it is possible to enhance the physics quality and reconstruction throughput of selected physics objects by leveraging heterogeneous architectures. Following what has been achieved with the pixel tracks, the Real-Time Reconstruction Revolution (R^3) project aim at tackling the redesign of other significant physics objects reconstruction: muons, electrons and photons, taus, jets, missing transverse energy, and Particle Flow Global Event interpretation [59].

The first milestone of the task reported on the current performance of the online reconstruction and investigated the missing factor in hardware and software performance that would be needed to reach the goal of reconstructing the full 750 kHz stream directly at HLT [5].

Table 2.1: Estimate of the missing performance factor (in bold) for Run 4 and Run 5, based on the estimates of the HLT TDR [6, 78].

Scenario	**PU**	**Throughput**	**Gain/year**	**Missing Factor CPU-Only**	**Fraction On GPU**	**Missing Factor w/ GPUs**
Run 4 (2028)	140	32.2 ev/s	+20%	2.5x	50%	**1.6x**
Run 5 (2032)	200	13.4 ev/s	+20%	5.0x	80%	**2.5x**

CMS has a defined budget for acquiring hardware for Phase-2. This budget has an impact on the amount of computational resources available for HLT and reconstruction. The performance improvements needed in the HLT reconstruction process are then extrapolated from the available budget.

These measures are available as estimates from the DAQ and HLT TDR [6]. Performance metrics such as throughput and timing were evaluated using L1-accept and $t\bar{t}$ events under PU 140 and 200. The measurements were extrapolated to estimate the requirements for HL-LHC, including the necessary corrections. The input rates assumed are 500 kHz for Run 4 (planned to start in 2028 at the time of the TDR, see Figure 1.4) and 750 kHz for Run 5 (in 2032). The estimate assumed that 50% of the HLT code would run on GPUs by Run 4, increasing to 80% by Run 5. Performance improvements are projected for both CPUs and GPUs. CPU performance was benchmarked using HS06 [80], while GPU performance metrics were derived from a benchmark of the Patatrack pixel reconstruction on NVIDIA T4 hardware.

The results are summarized in Table 2.1, with all the described corrections accounted for. The table illustrates a challenging scenario that can, however, be achieved through the increased use of heterogeneous computing resources. Providing a speed-up equal to or higher than the one presented in the last column would enable CMS to meet the requirements of Phase-2 as indicated by the TDR.

To estimate the increased requirement of the task, measurements were repeated under two different scenarios of the NextGen project: what would be required to run the full offline reconstruction at 500 kHz for Run 4 and 750 kHz for Run 5, and what would be necessary to run a scaled HLT reconstruction without filters.

Measurements were performed using simulated $t\bar{t}$ events at 200 PU and L1-accept skims from minimum bias samples at 140 and 200 PU. The benchmarking was conducted on an AMD EPYC “Bergamo” 9754 CPU. All results are reported in HS23 units [81]. Extrapolations were carried out with the same considerations as Table 2.1 and realigned with the latest HL-LHC schedule (see Figure 1.4).

The results for the two scenarios are visible in Tables 2.2 and 2.3. The tables illustrate very challenging scenarios for CPU-only reconstruction. The complete breakdown of the data and additional consideration are available in the annual report, where the milestone was delivered [78, 82].

Table 2.2: Estimate of the missing performance factor (in bold) for Run 4 and Run 5, in a scenario where the full offline reconstruction is run in the HLT [78].

Scenario	PU	Throughput	Gain/year	Missing Factor CPU-Only	Fraction On GPU	Missing Factor w/ GPUs
Run 4 (2030)	140	12.0 ev/s	+20%	23.8x	50%	**14.6x**
Run 5 (2036)	200	6.2 ev/s	+20%	31.4x	80%	**14.4x**

Table 2.3: Estimate of the missing performance factor (in bold) for Run 4 and Run 5, in a scenario where the Phase-2 HLT reconstruction is run without filters.

Scenario	PU	Throughput	Gain/year	Missing Factor CPU-Only	Fraction On GPU	Missing Factor w/ GPUs
Run 4 (2030)	140	47.6 ev/s	+20%	6.0x	50%	**3.7x**
Run 5 (2036)	200	32.5 ev/s	+20%	6.0x	80%	**2.7x**

The following goals of the task aim to meet the speedup requirement presented in the report. This will be achieved through the introduction of additional accelerator-based algorithms, the development of advanced data structures that support GPU hardware (see Section 2.2.1), and the introduction of AI-driven solutions coupled with fast GPU inference.

The second milestone of the task is to define and develop a validation suite for the CMSSW framework, which can accurately measure the performance of the R^3 reconstruction for key physics objects. It should provide a measure of performance for representative physics signals under realistic data-taking conditions. Work on this started in the second year of the project, adding support in the NanoAOD data format for bidirectional links between Monte Carlo truth information and reconstruction data, as well as a dedicated scouting-like HLT path that contains instructions for the full reconstruction without any filters.

In the meantime, the first progress on developing novel reconstruction algorithms began, with improvements to the pixel track Cellular Automaton algorithm to include the first layers of the Outer Tracker. This aims to enhance efficiency and misidentification rate, improve parameter resolution, and increase coverage in all regions of η. This algorithm will be described in more detail in Chapter 5, along with a tool used for automatic parameter tuning.

Together with the improvement to the pixel track reconstruction, a new seeding module for Standalone Muons was implemented, enabling direct matching of muon chamber segments with L1T primitives. This reduces seeding redundancy while maintaining physics performance. In addition, muon reconstruction transitioned to a streamlined approach that simplified the logic and achieved compatible physics performance with a $\sim 10\%$ reduction in total muon reconstruction timing [83].

Optimized data structures for heterogeneous platforms

Task 3.1.2 focuses on developing a flexible and portable Structure of Arrays (SoA) framework to support heterogeneous reconstruction and ML workflows in the CMSSW framework.

A SoA is a memory layout pattern in which data are organized by field rather than by object. Instead of storing an array of objects where each object contains multiple attributes (as in an Array of Structures, AoS), SoA stores each attribute in a contiguous array. This layout improves memory access patterns for parallel and SIMD (Single Instruction, Multiple Data) operations, which is particularly beneficial for heterogeneous computing environments like GPUs [84]. In the context of CMSSW, the objective is to provide an efficient and portable columnar data structure that can operate across different devices and architectures while enabling high performance with minimal data copying.

This involves addressing several challenges, such as the lack of reflection support in the C++ standard used by the current builds of CMSSW (C++20) [85], the need for constructors that allow a dynamic number of elements, and providing a user-friendly interface that enables flexible composition of columns from different SoAs. Another critical requirement is efficient serialization and integration into the existing CMS event data model.

To meet these requirements, the SoA framework is implemented using BOOST::PP macros for generating structures and template metaprogramming to manage different types of columns (for example, scalars, vectors, matrices...) at compile-time [86]. The system supports instantiation on both CPUs and GPUs, and integrates with ROOT via TTree for serialization [55]. Importantly, the backend does not manage memory allocation, giving users complete control and flexibility on where to allocate resources.

Recent developments include the integration of SoA with the PyTorch ML framework, allowing models to run directly on GPU without needing to copy or convert data formats [87]. Additionally, work is ongoing with ROOT developers to enhance the performance of serialization using RNTuple to support the demanding Input/Output (I/O) requirements of HL-LHC. Some performance benchmark of the RNTuple data format in compressing data is shown in Part II.

2.2.2 Evolving CMSSW into a client-service distributed application

Task 3.2's objective is to introduce support for fully distributed applications in the CMSSW framework. This would give higher flexibility in the design of the HLT farm for Phase-2. The current software framework is highly scalable, with over 5000 different modules' instances running on hundreds of threads. The software already features heterogeneous computing support using the Alpaka library, allowing the same codebase

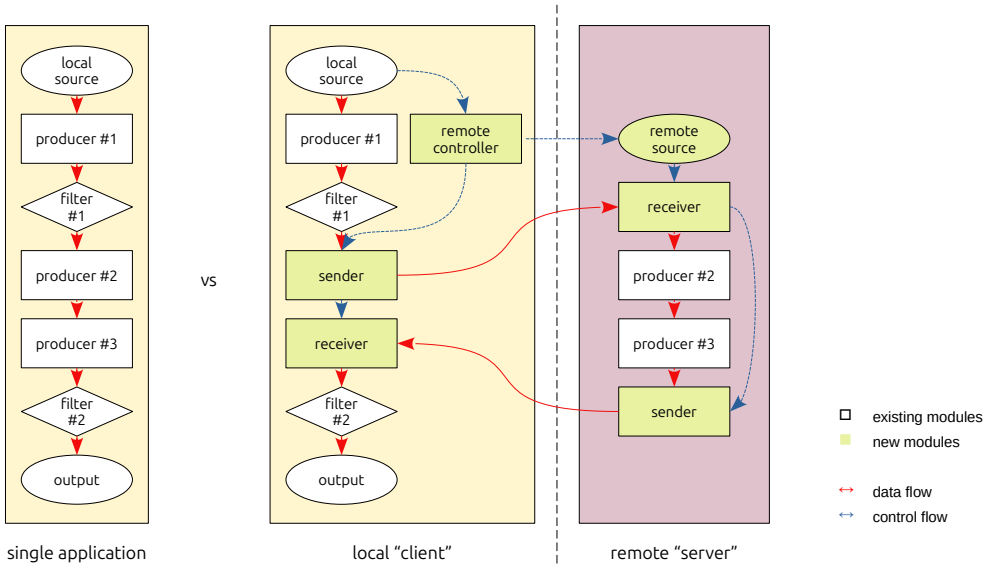

Figure 2.4: Diagram of the current CMSSW single node workflow, and the proposed distribution model that makes use of MPI to deploy modules across a second machine [89].

to compile for different architectures (i.e., CPUs, NVIDIA GPUs, AMD GPUs), and the framework can choose the best option at runtime, transparent to the user, depending on the available machines [88].

The research in the core software framework is now extending these capabilities by developing an extension that allows running multiple processes of the same logical program across multiple interconnected nodes. The original motivation for exploring this direction came during the deployment of a GPU-equipped HLT farm, where the balance between CPU and GPU memory and processing power is fixed at the time of procurement. Over time, as the codebase evolves, the ratio becomes suboptimal, resulting in the underutilization of either one [41]. The proposed distribution model enables the separation of compute responsibilities and the incremental addition of new GPU nodes, which can be utilized remotely over high-speed networks while maintaining low latency.

The first working prototype consists of two cooperating processes. The first reads raw data and sends it over MPI to a second machine, which performs GPU-based reconstruction. All data movement is handled transparently by MPI. This setup demonstrated that it is possible to synchronize two CMSSW applications running as separate processes, exchange data structures efficiently, and continue with standard HLT processing.

A second, more advanced prototype offloads only part of the HLT chain, specifically the hadronic calorimeter local reconstruction and particle flow, from the main process

to a remote GPU-based one. The first process reads the input, sends the raw data over the network, and receives the processed results back to continue the remaining HLT tasks. This demonstrates selective offloading and bidirectional communication, even for structured data in SoA format. The system has been tested, and it reconstructs the data collections on both ends with full consistency [89].

A diagram of the prototype is shown in Figure 2.4.

The final goal of this task is to develop a fully distributed, multi-threaded client-server application within the CMSSW framework that can efficiently run across multiple nodes and heterogeneous architectures with minimal changes to the existing software modules. This system aims to maximize resource utilization by enabling flexible offloading of compute-intensive tasks, supporting complex communication topologies, and ensuring robustness and scalability for future HLT requirements.

2.2.3 Reduction of the RAW data size

As presented in Section 1.2.2, the integrated luminosity collected after the HL-LHC is projected to be approximately ten times larger than the current LHC Run. The expected size of events is estimated to reach approximately 10MB each, which is roughly an order of magnitude larger than the current event size. Given that the hardware trigger can accommodate event rates of up to 750kHz, storing all the data in RAW format (see Section 1.4.3) is not a feasible option, as it would generate approximately 100EB of data over a ten-year period.

In this context, the objective of NextGen Task 3.3 is to reduce the size of RAW data stored by CMS, thereby increasing the maximum trigger rate available for physics analyses. The task investigates several data compression strategies, each offering different trade-offs between compression factor and flexibility for future data reprocessing. Part II provides a detailed overview of this task and the potential compression strategies that CMS may utilize in the coming years.

One direction focuses on lossy compression storing high-level physics objects, such as muons, electrons, or jets, instead of RAW data, similar to the CMS scouting format [71]. This approach can achieve compression factors of around 100, which would be necessary to record the full 750 kHz trigger rate; however, it limits the possibility of re-reconstructing data with updated calibrations or algorithms. Section 8.2 and appendix B present a breakdown of the compression potential of replacing RAW data with low-level reconstructed objects.

Another strategy under study involves lossy compression at the level of low-level physics objects, for instance, by replacing tracker or calorimeter RAW data with reconstructed hits and energy deposits. This technique, already in use for heavy ion collisions through the RAW' format [90], achieves more modest compression ratios, typically below a factor of 10, but preserves the ability to rerun reconstruction with more precise

methods and updated calibrations. Results and extensions to this approach are presented in Section 8.2

A third approach focuses on lossless compression by exploring existing algorithms and the potential to offload compression to hardware accelerators, such as GPUs. Section 8.1 presents benchmarks on lossless compression strategies to evaluate the performance of different algorithms.

The task aims to evaluate these techniques in terms of performance, practicality, and physics impact, to inform future choices for CMS data acquisition and storage.

2.2.4 Optimal calibration

The LHC physics program depends on accurate reconstruction of charged particle trajectories, as well as precise measurements of vertices and impact parameters. During data-taking, factors such as radiation damage from high particle flux can affect detector performance and impact position measurements. To maintain the physics performance of the CMS detector, calibration constants must be updated quickly to reflect changing conditions. This fast and reliable calibration is essential for ensuring efficient event selection by the HLT and for delivering high-quality initial reconstruction of physics objects.

The Alignment Calibration and Database (AlCaDB) group in CMS plays a central role in maintaining the detector's physics performance throughout data-taking [91]. This is achieved by maintaining a database of calibration constants, which are stored and propagated to the object reconstruction. These quantities remain constant from event to event and are referred to as condition data. A quick turnaround for calibration and alignment enables the delivery of datasets ready for analysis within a few hours of acquisition. The workflow designed for continuous monitoring and low-latency computation of calibration conditions is known as the Prompt Calibration Loop [92]. It provides updated conditions for Prompt reconstruction within 48 hours of the data taking.

For workflows that require more than 48 hours to execute or require different granularity, a tool for automating the orchestration of calibration jobs was developed for Run 3. Details of this Automation Framework and my contributions to the tool are outlined in Part III.

The objective of Task 3.4 of the NextGen project is to design a fast calibration workflow that meets or exceeds the resolution provided by Tier-0 in the Prompt reconstruction. With the final goal of bringing Prompt-like calibration to the HLT, which would be required for having HLT reconstruction with the same level of quality as the offline reconstruction [93].

This will be accomplished by implementing a system capable of buffering the full event stream at 750 kHz from L1T for 8 to 12 hours. Shortly after data-taking, a small

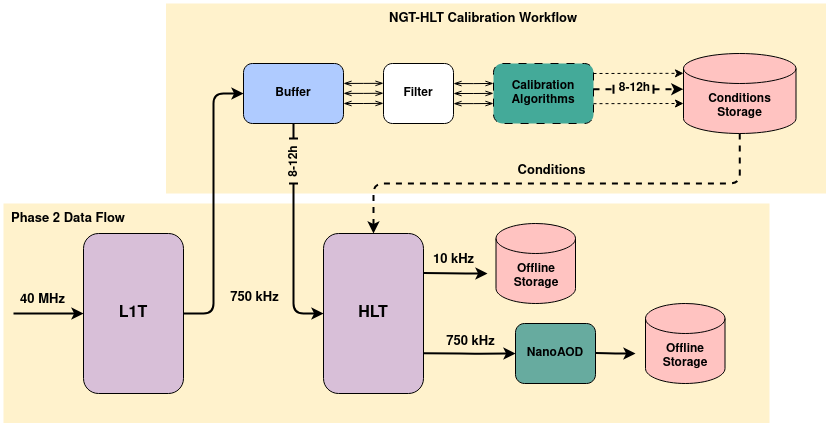

Figure 2.5: Diagram of the proposed calibration workflow for NextGen Task 3.4 [93].

fraction of the buffered events will be processed through an accelerated calibration workflow. The HLT would then use the conditions obtained from this workflow to select events for RAW storage at 10 kHz and perform the HLT reconstruction, which is saved as NanoAOD. Figure 2.5 shows a schematic representation of the proposed pipeline.

The first step of the task is to identify the existing calibration workflows that will be the ideal candidates for the upgrade. This has been achieved in the first milestone of the task [82]. The workflows identified cover several essential calibration tasks. These include the *beamspot* region correction, which calibrates the position of the centroid of the interaction point and the crossing angles of the beams; the bad-component masking for the silicon Pixel and Strip detectors; and corrections for the alignment of the Pixel trackers. The workflows also include the gain factors used in the conversion between the raw signal from the detector and the physical data for the Strip and HCAL detectors, the laser correction to the ECAL crystal that deteriorates due to radiation damage, and the pedestal data of the HCAL detector that measures the base noise level of the energy signals.

The Optimal Calibrations system will be integrated into the CMS DAQ system, serving both as a demonstrator for use with Run 3 data and as a foundation for the final Phase-2 implementation. The proposed Phase-2 architecture is a direct evolution of the current Run 3 DAQ, scaled to accommodate HL-LHC data rates. A key design choice concerns where to buffer data of the NextGen calibration workflow. Based on the LHCb experience of using a buffer stream of $\sim$ 30 PB of data on SSD [94], and using Run 4 parameters, a buffer size of $\sim$ 200 PB is estimated. The work on the Run 3 demonstrator is underway, with detailed benchmarks of SSD performance and integration with the current DAQ system [68].

2.3 Summary

This chapter introduced the Next Generation Triggers project, a collaborative initiative between CERN, ATLAS, and CMS, aimed at enhancing the trigger and data acquisition systems for the HL-LHC era. The key focus of this chapter was on WP3 and its efforts to extend the CMS High-Level Trigger capabilities beyond the baseline Phase-2 upgrades. The proposed enhancements include reconstructing all L1-accepted events at 750 kHz, optimizing data structures for heterogeneous platforms, reducing the size of RAW data through compression, and implementing fast calibration workflows.

The following chapters will delve into my contributions to specific tasks within WP3, detailing the methodologies, implementations, and results of my work. Part I focuses on my activity with a tool for automated parameter tuning for reconstruction algorithms, which is currently being used in task 3.1.1 of the project; Part II details the bulk of the activities related to task 3.3; Part III presents the Automation Framework used in calibration and analysis workflows, presenting an orchestration tool that could be used for the fast calibration pipeline of task 3.4. These contributions align with the broader NextGen objectives presented in this chapter.

Part I

Parameter Tuning for Track Reconstruction

Chapter 3

Multi-Objective optimization

Optimization is a fundamental challenge that arises in virtually every discipline where data plays a central role. From computer science and engineering to statistics, economics, and scientific research, it is the basis for making optimal decisions, improving performance, and efficiently allocating resources.

In the CMS experiment, and specifically in the context of the work presented in this thesis, the interest in optimization is motivated by the large number of parameters required to efficiently reconstruct particle tracks from detector data. The reconstruction algorithms consist of multiple functions, each with its own parameters, which determine how tracks are built and which detector hits are selected for reconstruction. Properly tuning these parameters requires not only a thorough understanding of the underlying physics but also a comprehensive understanding of the reconstruction software itself. Moreover, testing different parameter combinations is often computationally expensive and time-consuming. The high dimensionality of the problem makes it infeasible to explore all possible solutions through brute-force approaches.

To address this challenge, the Patatrack group (Accelerated Pixel Track Reconstruction in CMS) initiated an effort to simplify parameter tuning by developing a framework for automated optimization based on intelligent search algorithms. The first attempt, conducted during the 13th Patatrack Hackathon [95], resulted in the development of a Multi-Objective Particle Swarm Optimization (MOPSO) algorithm for tuning track reconstruction in the CMS pixel detector. The prototype demonstrated promising results, motivating further work to extend and refine the approach.

This part of the thesis presents the activities carried out to develop and extend the prototype further. This chapter introduces the theoretical concepts underlying multi-objective optimization, with a particular focus on the MOPSO algorithm employed in this work. Chapter 4 describes the implementation design of the `patatune` framework and provides benchmarks of its performance. Finally, Chapter 5 presents the results

obtained by applying the algorithm to optimize the reconstruction parameters for both the CMS pixel tracker and the TrackML simulation.

3.1 Heuristic optimization

In its more general definition, optimization is the process of maximizing or minimizing a real function f by choosing the optimal solution from a set X of possible values. The concept of optimality for a minimization problem can be described as follows:

Definition 3.1.1 (Optimal Solution)**.** Given a minimization problem (X, f), $f : X \to \mathbb{R}$, a solution $x_{opt} \in X$ is optimal iff:

$$x_{opt} = \min_{x \in X} f(x)$$

that is:

$$\forall x \in X : f(x_{opt}) \leq f(x)$$

The definition is analogous for maximization problems, and since

$$f(x_{opt}) \geq f(x) \iff -f(x_{opt}) \leq -f(x)$$

all maximization problems can be reduced to a minimization problem (or vice versa).

X is also known as *search space*, therefore optimization techniques are also known as search techniques.

The elements $x \in X$ are called *candidate solutions*, and the function f is commonly known as *objective function*.

Optimization problems can be divided into two categories: those with *continuous variables* and those with *discrete variables*, where the solution forms a countable set of objects [96].

Problems belonging to the second category are varied and well-studied [97]. Typical examples include the Traveling Salesman Problem, which seeks the shortest route visiting a set of cities exactly once and returning to the start; the Minimum Spanning Tree problem, which finds the minimum weight tree connecting all nodes in a weighted graph based; the Quadratic Assignment Problem, which assigns facilities to locations in a way that minimizes total cost; and Scheduling or Timetabling problems, which aim to allocate tasks or events to time slots while satisfying various constraints. Due to their practical importance, many algorithms have been developed to tackle these problems.

These algorithms can be broadly classified as *complete* or *approximate*. Complete algorithms aim to find the optimal solution within a bounded time for any finite solution space; however, for NP-hard problems (non-deterministic polynomial-time), they may

require exponential time to reach a solution. Approximate algorithms, on the other hand, explore only a subset of the solution space, providing a good solution within a short time but without any guarantee of finding the optimal solution. These methods are also known as *heuristics* [96].

The basic optimization methods can be further split into *constructive* methods or *local searches*. Constructive methods build a solution incrementally, adding components to an initially empty solution until it is complete [98]. For example, in a network connectivity problem, a constructive method might iteratively add edges to a graph until all nodes are connected, verifying the relevant criteria at each step of the process. Local search algorithms, instead, begin with a candidate solution and iteratively attempt to replace it with a better solution within its neighborhood in the search space. One example is the Hill Climbing algorithm, which searches for neighbors of the candidate solution and moves only towards the one that improves the current solution.

A specific class of local search algorithms is *metaheuristics* [99]. The term originates from the Greek meaning "to find beyond". This class of algorithms extends the combinatorial algorithms using iterative optimization strategies to guide an underlying, problem-agnostic heuristic. Fundamentally, these methods aim to overcome the limitations of local search methods, which can become trapped in local optima: points that represent a minimum or maximum only within a local region of the search space. By combining various concepts for exploring the search space, a metaheuristic enables the search to escape suboptimal solutions. This is often achieved by allowing the acceptance of non-improving moves and relying less on pure randomness. They generally introduce a well-defined bias into the search, which can take the form of: a *descent bias*, based on the objective function; a *memory bias*, that considers past decisions; or an *experience bias*, derived from prior performance. This intelligent, biased use of randomness is what distinguishes a metaheuristic from a simple random search. Furthermore, metaheuristics are characterized by their generality, serving as a flexible algorithmic framework that can be adapted with relatively few modifications to address different optimization problems [99].

Metaheuristic algorithms try to strike a balance between *exploration* and *exploitation*. Exploration is the process of searching a vast and unexplored area of the solution space to find new, potentially better solutions with a lower risk of falling into local optima. Exploitation, on the other hand, is the process of using the information already obtained to improve the current best-known solution. It focuses on refining and intensifying the search within a promising area of the solution space. The goal is to converge on the best possible solution within a specific region. Excessive exploitation can lead to premature convergence to a local suboptimal solution, while too much exploration can make the search inefficient and prevent convergence to a good solution.

There are different search strategies for metaheuristics. One category consists of extensions of local search algorithms that incorporate intelligent mechanisms to escape local minima. Another category focuses on learning correlations between decision variables to guide the search towards high-quality regions of the solution space.

A further classification is between *population-based* algorithms and *single-point search*: The former work with a set of candidate solutions at each iteration, allowing information to be shared among them and enabling a more diverse exploration of the search space; examples include Genetic Algorithms and Ant Colony Optimization: Ant Colony Optimization simulates the behavior of ants depositing and following pheromone trails, where pheromone levels encode learned information about good solutions and are updated iteratively to strengthen promising paths [100]; Genetic Algorithms model biological evolution, where a population of candidate solutions undergoes variation through mutation and crossover, and selection favors the fittest individuals, gradually improving solution quality over generations.

In contrast, single-point search algorithms operate on a single candidate solution at a time, modifying and improving it step by step until a stopping criterion, such as a maximum number of iterations or convergence, is met; examples include Simulated Annealing and Tabu Search: Tabu Search maintains a "taboo list" of recently visited solutions or moves, preventing the algorithm from cycling back to them and encouraging exploration of new areas in the search space [101]; Simulated Annealing is inspired instead by the annealing process in metallurgy, introducing a temperature parameter that controls the probability of accepting worse solutions, starting high to encourage exploration and gradually cooling to focus on exploitation.

In recent decades, metaheuristic algorithms have been recognized as powerful global optimization techniques, with new and improved versions of existing algorithms regularly presented [102, 103].

3.1.1 Particle Swarm Optimization

Particle Swarm Optimization (PSO) is a population-based metaheuristic optimization method for continuous non-linear functions, with roots in evolutionary computation and artificial life, particularly in the field of bird flocking and swarming theory [104].

PSO demonstrates effectiveness in various applications, excelling particularly in multimodal problems that lack specialized methods. Its versatility spans biological, medical, electrical, and computational intelligence domains, as well as combinatorial problem solving [105]. Its simplicity and adaptability make it easy to integrate into different application domains.

In recent formulations [106], a set of N particles (a *swarm*) is deployed in the D dimensional space X, with velocity v_{id} and position x_{id} for each particle i in dimension d defined as:

```
1: for each particle do
2:     Initialize particle position
3:     Initialize particle velocity
4: end for
5: while max number of iterations not reached do
6:     for each particle do
7:         Evaluate fitness
8:         if fitness is better than personal best then
9:             Update personal best
10:        end if
11:    end for
12:    Select solution with best fitness as global best
13:    for each particle do
14:        Update velocity based on Equation (3.1)
15:        Update position based on Equation (3.2)
16:    end for
17:    Increase iteration counter
18: end while
19: return global best
```

Figure 3.1: Sample implementation of the Particle Swarm Optimization algorithm

$$v_{id}^{t+1} = wv_{id}^{t} + c_1 r_1^t (p_{id}^t - x_{id}^t) + c_2 r_2^t (g_d^t - x_{id}^t) \tag{3.1}$$

$$x_{id}^{t+1} = x_{id}^t + v_{id}^{t+1} \tag{3.2}$$

where: p_{id}^t is the personal best in dimension d at iteration t of particle i; g_d^t is the global best for dimension d at iteration t; w is the inertial weight; c_1 is the cognition coefficient and c_2 is the social coefficient. Together, w, c_1, and c_2 form the parameters of the optimization process. c_1 and c_2 regulate *exploration* and *exploitation* respectively. Finally, r_1^t and r_2^t are two random values uniformly distributed in the range $[0, 1]$.

A possible implementation in pseudo code of the basic PSO is shown in Figure 3.1. At each iteration, the algorithm first evaluates the objective function for each particle's current position. If a particle's new position yields a better value than its personal best, that personal best is updated. Once all particles have been checked, the global best is determined as the best among all updated personal bests. Then, for each dimension, the velocity and position of every particle are updated using Equations (3.1) and (3.2). This process repeats until the maximum number of iterations is reached, at which point the global best is returned as the algorithm's final solution.

3.2 Multi-Objective optimization

When the optimization problem has more than one objective that needs to be optimized simultaneously, it becomes a Multi-Objective optimization problem [107].

Definition 3.2.1 (Multi-Objective Optimization Problem)**.** A pair (X, F) where $F(x) = (f_1(x), f_2(x), \ldots, f_N(x))$ is a vector of functions $f_i : X \to \mathbb{R}$ for $i \in \{1, 2, \ldots, N\}$

The function $F(x)$ maps the *search space* X to the *objective space*, which is the Cartesian product of the images of the functions f_i.

In a multi-objective problem, the objectives can be *non-conflicting* or *conflicting*. Non-conflicting objectives are correlated, so an improvement in one also leads to improvement in the others, effectively reducing the problem to a single-objective optimization. Conflicting objectives, on the other hand, cannot be improved simultaneously. Enhancing performance in one dimension typically comes at the expense of another, making the problem fundamentally different from the single-objective case and requiring specific methods to balance trade-offs. The implication is that, with conflicting objectives, we seek a set of solutions that represent the trade-offs between them, rather than a single optimal solution.

Unlike with single-objective optimization, where a complete order exists, there is not necessarily a trivial ordering of the objective space. We can define the relationship between candidate solutions with the concept of Pareto Dominance.

3.2.1 Pareto dominance

To evaluate and compare solutions in the objective space, we define a partial ordering based on the concept of dominance. A solution is said to dominate another if it is at least as good as, or better than, the other in all objectives and strictly better than the other in at least one.

Definition 3.2.2 (Dominance)**.** Given a multi-objective minimization problem (X, F), a solution $F(x_1)$ dominates $F(x_2)$, denoted by $F(x_1) \preceq F(x_2)$, iff

$$\forall i \in \{1, 2, \ldots, N\} : f_i(x_1) \leq f_i(x_2) \quad \wedge \quad \exists j \in \{1, 2, \ldots, N\} : f_j(x_1) < f_j(x_2)$$

The collection of all solutions that are not dominated by any other solution in the search space forms a set in the objective space known as the Pareto front. This set represents the optimal trade-off between conflicting objectives.

Definition 3.2.3 (Pareto Front)**.** Given a multi-objective problem (X,F), its Pareto front PF is the set of objective vectors of non-dominated solutions such that:

$$\text{PF} = \{F(x^*) \mid \nexists\, x \in X : F(x) \preceq F(x^*)\}$$

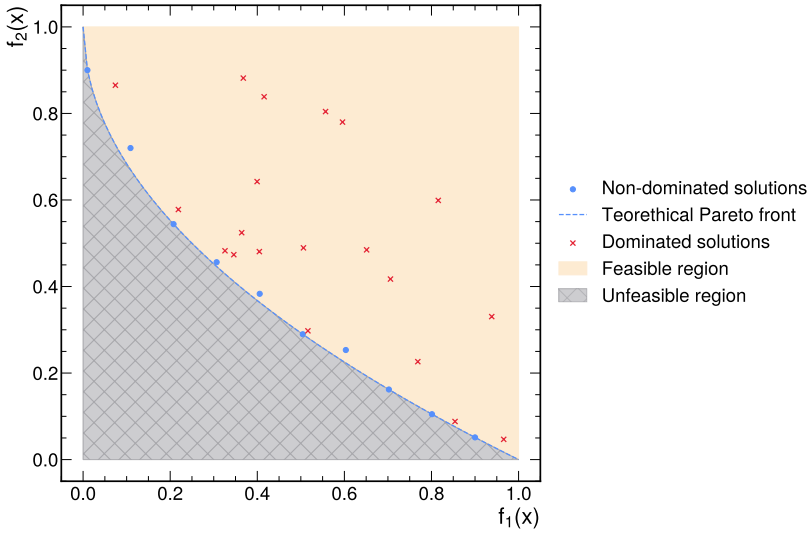

Figure 3.2: Plot showing a 2-dimensional objective space for a multi-objective minimization problem.

Any point in X that maps to the Pareto front is considered a Pareto optimal solution, meaning you cannot improve any objective without making at least one other objective worse.

The plot in Figure 3.2 shows a 2D objective space for a multi-objective optimization problem. The Pareto front, represented with the blue line, divides the space into two regions. The feasible region in the plot is the orange-colored area where solutions to the multi-objective problem can exist. This region contains both dominated (red crosses) and non-dominated (blue dots) solutions. The unfeasible region, shown in gray, is the area of the objective space that no valid solution of the problem can reach. The points in the non-dominated set lie on the edge of the feasible region, approximating the theoretical Pareto front.

The goal of multi-objective optimization is to find a set of solutions that approaches the theoretical Pareto front as much as possible.

3.2.2 Multi-Objective Mixed Integer Programming

Many real-world optimization problems involve both continuous and discrete decision variables and must optimize more than one objective simultaneously. In such cases, the problem is formulated as a Multi-Objective Mixed Integer Programming (MOMIP) problem [108].

MOMIP extends classical multi-objective problems by introducing integer or categorical variables alongside continuous values, allowing the modeling of decisions that represent discrete configurations and logical choices. Formally, a MOMIP problem can be defined as follows:

Definition 3.2.4 (Multi-Objective Mixed Integer Programming Problem)**.** A Multi-objective problem (X, F) with $X \in \mathbb{R}^n$ for which $\exists j \in \{1, \ldots, n\} : \forall x \in X : x_j \in \mathbb{Z}$. Where x_j is the j-th component of x.

Solving a MOMIP problem exactly is computationally challenging. Even for a single-objective, mixed integer programming problems are NP-hard, and adding multiple objectives further increases the complexity [109]. The search space grows exponentially with the number of integer variables, and the objective functions are often non-linear or non-convex, making exact approaches infeasible for large-scale or high-dimensional problems.

Traditional mathematical programming techniques can find Pareto-optimal solutions for small, convex MOMIP instances. However, their applicability is limited when the objective functions are defined by complex simulations or lack analytical gradients, as is the case in many scientific and engineering contexts. In such scenarios, heuristic and metaheuristic optimization algorithms become a practical alternative. They can efficiently explore large, mixed search spaces and approximate the Pareto front without requiring differentiability.

Among these, population-based methods such as Multi-Objective Particle Swarm Optimization (MOPSO) provide a flexible and efficient framework to approximate non-convex Pareto fronts. The next section introduces the MOPSO algorithm in detail and explains how it extends the standard PSO formulation to handle multiple conflicting objectives simultaneously.

3.2.3 Multi-Objective Particle Swarm Optimization

The extension of PSO to multi-objective settings was pioneered in [110]. The work introduced Pareto dominance into PSO, complemented with an archive to store non-dominated solutions and guide the swarm's search. This presented the Multi-Objective Particle Swarm Optimization (MOPSO) as a competitive alternative to evolutionary multi-objective algorithms, combining the fast convergence of the PSO algorithm with strategies to approximate and maintain a well-distributed Pareto front.

MOPSO shares many similarities with the single-objective implementation, both involving a series of iterative adaptations of a set of solutions based on the optimization goal. The main difference is in the solution assessment and the selection of the personal and global best for the particles.

The general implementation of the MOPSO algorithm is represented in pseudo code in Figure 3.3. The code is very similar to the PSO, with the main difference involving the concept of dominance and the introduction of the archive of solutions. At each iteration, the archive is updated by adding all particles from the new iteration to the

```
1: for each particle do
2:     Initialize particle position
3:     Initialize particle velocity
4: end for
5: Initialize empty archive of solutions
6: while max number of iterations not reached do
7:     for each particle do
8:         Evaluate fitness
9:         if fitness dominates personal best then
10:            Update personal best
11:        end if
12:    end for
13:    Update archive with non-dominated particles
14:    for each particle do
15:        Select global best from archive
16:        Update velocity based on Equation (3.1)
17:        Update position based on Equation (3.2)
18:    end for
19:    Increase iteration counter
20: end while
21: return archive of solutions
```

Figure 3.3: Possible algorithmic implementation of MOPSO

archive if they dominate any of the solutions in the archive, and removing from the archive all solutions that are dominated by the newly added particles.

The global best for each particle is then selected from this archive, with different selection methods resulting in different behaviors of the swarm in terms of elitism (the selection of the best solutions) and diversity (the attempt to find a solution set that spans a broader range of non-dominated solutions).

Different methods can be identified when selecting the global and local best solution:

Random the most straightforward method. For each particle, a random solution from the archived solution is chosen as the global best;

Crowding Distance The crowding distance, depicted in Figure 3.4, is an estimate of the density of solutions surrounding a particular point in the population [112]. It is used to sort solutions according to their spread and to favor diversity inside the population. It is calculated as the size of the largest cuboid that surrounds a point and does not include any other point;

Round Robin inspired by process scheduling in computer systems. Each particle is assigned a global best solution from the archive in sequential order by crowding distance. If the number of particles exceeds the number of archived solutions,

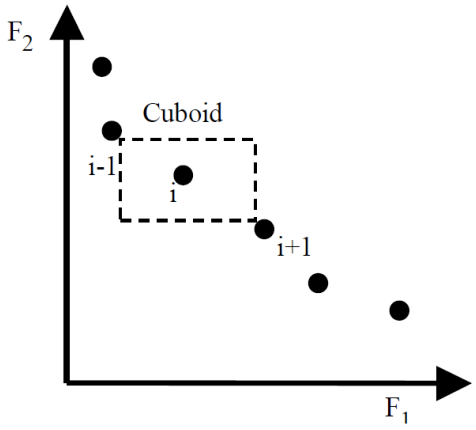

Figure 3.4: Crowding distance computation in a 2D space [111].

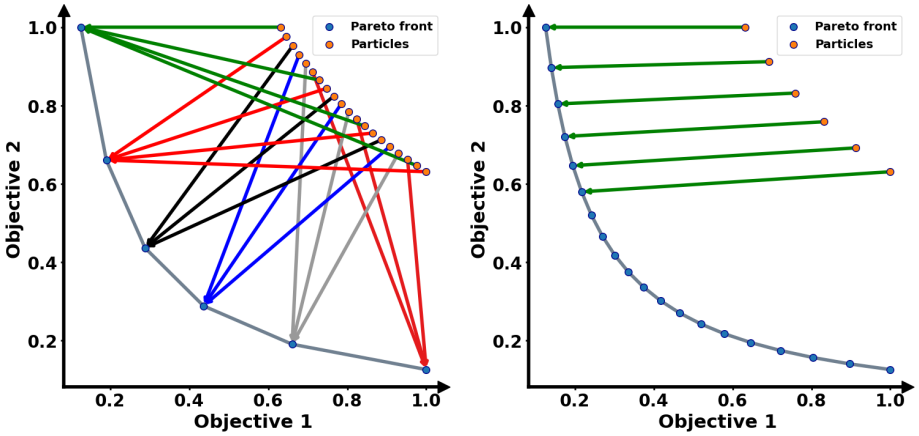

Figure 3.5: Two different cases for the Round-Robin topology: left, when $n > |PF|$; right, when $n < |PF|$.

the assignment continues cyclically from the beginning of the archive. Figure 3.5 illustrates this assignment scheme.

3.3 Summary

This chapter provides a theoretical foundation for the multi-objective optimization techniques that are applied in this thesis to tune the parameters of particle track reconstruction algorithms in the CMS experiment. The challenge stems from the large number of parameters, making brute-force approaches computationally infeasible. To overcome this, the work employs an automated framework based on the MOPSO algorithm. The core of this approach is a population-based metaheuristic inspired by the collective behavior of swarms. The goal is not to optimize a single solution but to find a set of optimal trade-offs, known as the Pareto front. MOPSO approximates the Pareto front by maintaining an external archive of non-dominated solutions. This method ensures both convergence toward the optimal front and diversity among the solutions, providing a comprehensive view of the trade-offs. The implementation of this algorithm will be discussed in the following chapter on the `patatune` framework, and its specific applications to CMS pixel tracking will be discussed in Chapter 5.

Chapter 4

Patatune: a framework for multi-objective optimization

This chapter provides a comprehensive analysis of `patatune`, a software package written in Python designed to solve multi-objective optimization problems. The tool has been developed following the activity started at the 13th Patatrack Hackathon and has been extended to meet the requirements of the CMS experiments, while also maintaining generalization and extensibility in mind [95]. The discussion in this chapter explores its foundational architectural principles, the theoretical and practical implementation of its core algorithms and the utility components that ensure its robustness in a scientific context. It follows an overview of the methodologies for evaluating the performance of these algorithms, and the additional tools developed for visualizing and analyzing the results. The final section focuses on benchmarking the MOPSO algorithm implemented in the tool using standard test problems. This analysis is framed within the context of its intended application.

The development of `patatune` is guided by the goal of providing a flexible framework for multi-objective optimization in HEP. The aim was to design a tool with a simple, adaptable interface that allows users to extend it across different use cases easily. The framework was conceived to be modular, allowing new algorithms and features to be incorporated without modifying the user interface. Python was chosen as the implementation language because it provides flexibility, a low barrier to entry, and multiple scientific libraries to facilitate development and integration. A central design principle was the separation of the optimization logic from the definition of objectives, to have the optimization workflow operate the same way regardless of how the objective functions work. Support for heterogeneous parameter types, including continuous, discrete, and boolean variables, was a requirement in the CMS use cases. The intent has also been to provide flexible mechanisms for saving and checkpointing, allowing results and state to be exported in multiple formats to fit diverse computational environments,

and optimization to be paused and continued at a different moment. Finally, validation and evaluation are meant to remain independent from the optimization algorithms to promote reusability and to allow analysis to be run separately from the optimization itself.

4.1 Design and implementation

The architecture of `patatune` reflects this intent, prioritizing modularity, flexibility, and reproducibility. This design is evidenced by its file structure, which includes distinct directories for source code (`src/patatune`), illustrative examples (`example`), and test benchmarks (`tests`).

The source code is again split into functional components. With a script dedicated to the definition of the objective functions (`objective.py`), one for the common code for the optimization algorithms (`optimizer.py`), one for the utilities and common functions (`util.py`) and one for the performance metrics (`metrics.py`). The design and implementation of each will be described in the following sections.

To ensure the modularity of the project, the code related to the algorithms themselves is defined in dedicated subfolders. The project currently implements the MOPSO algorithm as presented in the previous chapter, and its implementation is written in the `mopso/mopso.py` and `mopso/particle.py` scripts.

The project utilizes a `pyproject.toml` file to define the setup instructions, adhering to contemporary Python packaging standards. The software is easily installable with standard tools like `pip` and is available on the Python Package Index [113, 114]. This packaging is crucial for the project, enabling its integration in the CMS software framework.

4.1.1 Objectives definition

`patatune` package provides a simple interface for the user to define multi-objective problems, offering two distinct classes: `Objective` and `ElementWiseObjective`.

`Objective` is the base class, which allows the definition of one or more objective functions that can run simultaneously on all candidate solutions. The `evaluate` method takes a list of arrays, one for each candidate solution, runs the objective functions by passing the entire list as an argument, and returns the output of each objective function. The intention is to use it to define a function that handles a list of candidate solutions simultaneously. It is used, for example, with CMSSW configurations that utilize accelerators to run multiple producers simultaneously. An example will be provided in Chapter 5. The class also allows defining the names

of the objective functions for visualization and exporting the results in a human-readable format.

`ElementWiseObjective` inherits from the `Objective` class and is tailored for problems where the objective function has to be evaluated independently on each candidate solution. The `evaluate` method receives a single position in the search space as input and returns the corresponding fitness values. This design choice enables the parallelization of the optimization process by leveraging Python's vectorization capabilities. It is used, for example, in the test functions described in Section 4.2 that are defined as standard Python functions.

This dual approach to objective function definition allows users to select the most suitable method for their specific application. The objective definition supports a wide range of objective functions, from functions defined in the same Python script to anything that can be run as a separate process.

Both classes have also been extended to support batched processing through the `asyncio` Python library. The library allows running concurrent code using a simple `async/await` syntax. The `AsyncBatchObjective` and `AsyncElementWiseObjective` classes allow the use of objective functions defined by the user as coroutines, and divide the input solution into batches to be run simultaneously on the machine.

Thanks to the modular approach, it should be simple to define other evaluation procedures to support, for example, different batch processes or run evaluations on heterogeneous architectures.

4.1.2 Optimization algorithms definition

`patatune` is built to support different optimization algorithms. The `Optimizer` base class is called when running the optimization, requiring only the definition of a `step` and `optimize` function. The first is used to run a single iteration of the optimization process, while the second is used to run multiple iterations until a stop condition is met. The specific algorithm implementation then specializes the base class.

Currently, `patatune` implements the MOPSO algorithm defined in Section 3.2.3. The implementation of MOPSO in `patatune` is highly configurable through several key parameters, providing the user with granular control over the optimization process. These parameters are necessary for tuning the balance between global exploration of the search space and local exploitation of promising regions. This tuning is fundamental to the efficiency of metaheuristic algorithms.

`num_particles` defines the number of particles in the swarm. A larger swarm increases the likelihood of exploring diverse regions of the search space, but also raises the computational cost.

`inertia_weight`,`cognitive_coefficient` and `social_coefficient` control the w, c_1 and c_2 parameters of Equations (3.1) and (3.2).

`lower_bounds` and `upper_bounds` are arrays that define the lower and upper limits of the search space X for each of its dimensions. The lower and upper bounds are also used to determine the data type of each dimension of the search space, supporting continuous values with real numbers, discrete values with integers, and binary options with booleans.

`initial_particles_position` is used to define how the population of particles is initialized. Either with a uniform *random* spread in the search space, or in a *gaussian* distribution around a `default_point`, or with all positions matching the *lower_bounds* or *upper_bounds* arrays.

`topology` allows to specify the method by which the global best is picked, as described in Section 3.2.3. The selection can be done in a random order, using the Round Robin method, or by selecting particles based on the crowding distance: in the densest areas of the Pareto front to promote exploitation, or in the least crowded regions to promote exploration.

`max_pareto_length` is used to limit the number of the archived solution, by discarding those with the lower crowding distance, to have high diversity in the Pareto front.

The `optimize` method is used to run the optimization process for a specified number of iterations, which the user can set. An additional method to improve diversification is built into the algorithm, which allows the particles to be spread in the search space when, for a specified number of iterations, their fitness has not improved. This behavior is also configurable by the user, with options to enable (`exploring_particle`) and to specify the number of stale iterations after which the process should occur (`max_iterations_without_improvement`). Aside from the previous parameters that allow for changing the optimization behavior, the algorithm also implements a way to name the dimension in the search space for visualization and storing the results.

To populate the archive of solutions, the algorithm implements a `get_dominated` function. If available in the environment, the function leverages the *Numba JIT* (Just-In-Time) compiler to accelerate its core loop [115]. Numba compiles the Python code into highly optimized machine code at runtime to speed up numerical operations. This is particularly effective for the nested loops required to compare every solution against every other solution in the multi-objective space, and it directly impacts the overall runtime of the MOPSO algorithm.

4.1.3 Utilities

Beyond its core optimization routines, `patatune` integrates several utility components that are essential for supporting the user.

The `Randomizer` class allows for the setting of a reproducible seed for random number generation and ensures consistency with all the libraries used by the tool. In scientific research, reproducibility is a requirement. This feature ensures that any optimization run can be replicated precisely, allowing consistent results and confirming that any observed performance improvement is due to a change in the optimization parameters or problem formulation rather than a statistical fluke.

The `FileManager` class provides a robust checkpoint system. This functionality is necessary for optimization tasks that run for a long time, typical in large-scale computations, such as those used by the CMS experiments. The ability to save the current state of `patatune` and resume the run from the last saved point optimizes the use of computational resources and time. The `FileManager` is also used to store the results of the optimization in various formats, ranging from human-readable `csv` files to serialized binary for processing by other scripts.

Additionaly, a logging component is provided to offer a mechanism for monitoring the optimization progress in real-time and for recording the status of the process.

4.1.4 Performance metrics

Evaluating the performance of a multi-objective optimization algorithm is more complex than that of a single-objective one, as it requires assessing the quality of an entire set of solutions rather than a single value. Moreover, sometimes the optimum set of solutions is not known, so other techniques must be used to evaluate performance without knowing the theoretical Pareto front. Standard metrics exist to quantify the quality of a solution set. `patatune` implements three different metrics: the Generational Distance ($\mathcal{GD}$), the Inverted Generational Distance ($\mathcal{IGD}$), and the Hypervolume ($\mathcal{HV}$) [116].

Definition 4.1.1 (Generational Distance)**.** Given a multi-objective problem (X, F), a set of reference points $Z \subseteq \mathrm{PF}, |Z| < \infty$ in its Pareto front, and a set of points $Y \subseteq X, |Y| < \infty$ that defines a set of solution, $A = F(Y)$, the Generational Distance of A, is:

$$\mathcal{GD}(A) = \frac{1}{|Y|}\left(\sum_{i=1}^{|Y|} d_i^2\right)^{1/2}$$

where d_i is the Euclidean distance from the i-th point in A to its nearest reference point in Z, on the left in Figure 4.1.

The $\mathcal{GD}$ performance indicator computes the average distance from the archive of solutions A to the theoretical Pareto front.

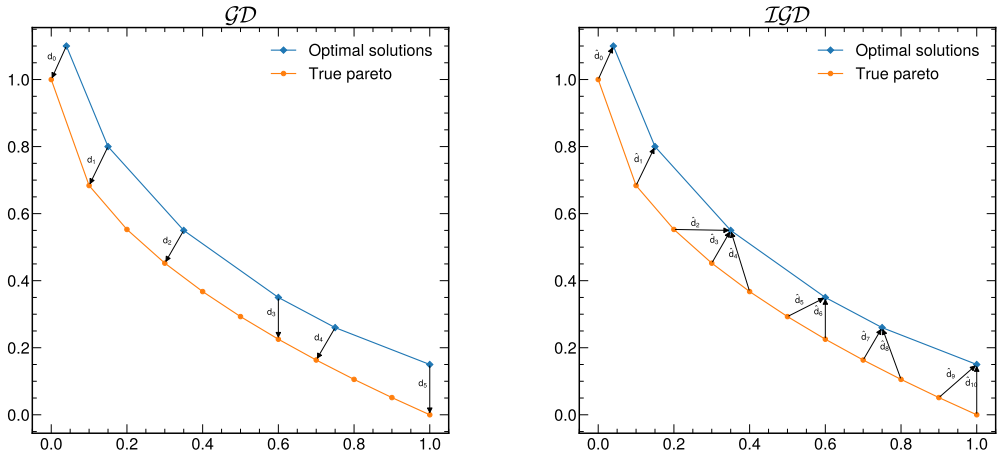

Figure 4.1: On the left, a representation of the measure of the distances d_i for the $\mathcal{GD}$ indicator. On the right, the measure of the distances $\hat{d}_i$ for the $\mathcal{IGD}$ indicator.

Definition 4.1.2 (Inverted Generational Distance)**.** Given a multi-objective problem (X, F), a set of reference points $Z \subseteq \text{PF}, |Z| < \infty$ in its Pareto front, and a set of points $Y \subseteq X, |Y| < \infty$ that defines a set of solution $A = F(Y)$, the Inverted Generational Distance of A is:

$$\mathcal{IGD}(A) = \frac{1}{|Z|} \left(\sum_{i=1}^{|Z|} \hat{d}_i^2 \right)^{1/2}$$

where $\hat{d}_i$ is the Euclidean distance from the i-th point in Z to its nearest reference point in A, on the right in Figure 4.1.

The $\mathcal{IGD}$ performance indicator inverts the $\mathcal{GD}$ and measures the distance from any point in Z to the closest point in A.

When the theoretical Pareto front is not available, the performance indicator that can be used in `patatune` is the Hypervolume.

Definition 4.1.3 (Hypervolume)**.** Given a set of points $S \subset \mathbb{R}^N, |S| < \infty$ and a reference point $r \in \mathbb{R}^N | \forall p \in S : r \preceq p$, the Hypervolume $\mathcal{HV}$ is the measure of the region dominated by S and bounded above by r:

$$\mathcal{HV}(S, r) = \lambda_N \left(\bigcup_{p \in S} [p, r] \right)$$

Where $\lambda_N(\cdot)$ is the Lebesgue measure and $[p, r] = \{q \in \mathbb{R}^N | p \preceq q \preceq r\}$ is the N-dimensional box delimited below by p and above by r.

Figure 4.2 shows a two-dimensional example of the Hypervolume.

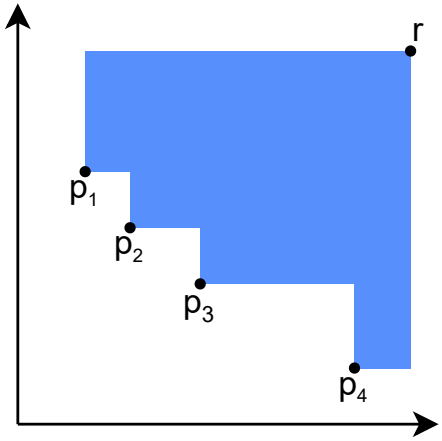

Figure 4.2: Example in two dimensions of the Hypervolume indicator for a points set $\{p_1, p_2, p_3, p_4\}$ and reference point r [117]

In a multi-objective problem, the Hypervolume is the measure of the region dominated by the archived set with respect to a reference point. The goal of an algorithm that solves a multi-objective minimization problem is to maximize this indicator.

4.1.5 Visualization

To facilitate the visualization of the optimization results, we developed a prototype web dashboard capable of reading the output of a `patatune` run and plotting the resulting solutions with interactive controls to study their behavior [118]. The visualization module is a browser-based dashboard built with Python Dash and Plotly [119], providing an interactive environment for exploring MOPSO outputs uploaded as `csv` files. Figure 4.3 shows the dashboard as currently implemented.

The upper panel titled "Upload CSV File" allows loading data from the local disk. On file upload, the application parses the file, detects the parameters and objective columns (with an option for manual override), and constructs an internal data model that preserves both the original dataset and any user-driven modifications.

The primary view, located in the center, is a dynamic matrix that plots every pairwise relationship between the objectives. Each cell of the matrix is an independent, linked scatter plot, so that selecting points in one cell highlights the corresponding points across all others.

Interactive filtering is provided through range controls tied to each objective and to any parameter columns. Adjusting a filter in the "Filters" panel immediately updates the plots, allowing for rapid exploration of subsets of solutions and the trade-off between objectives.

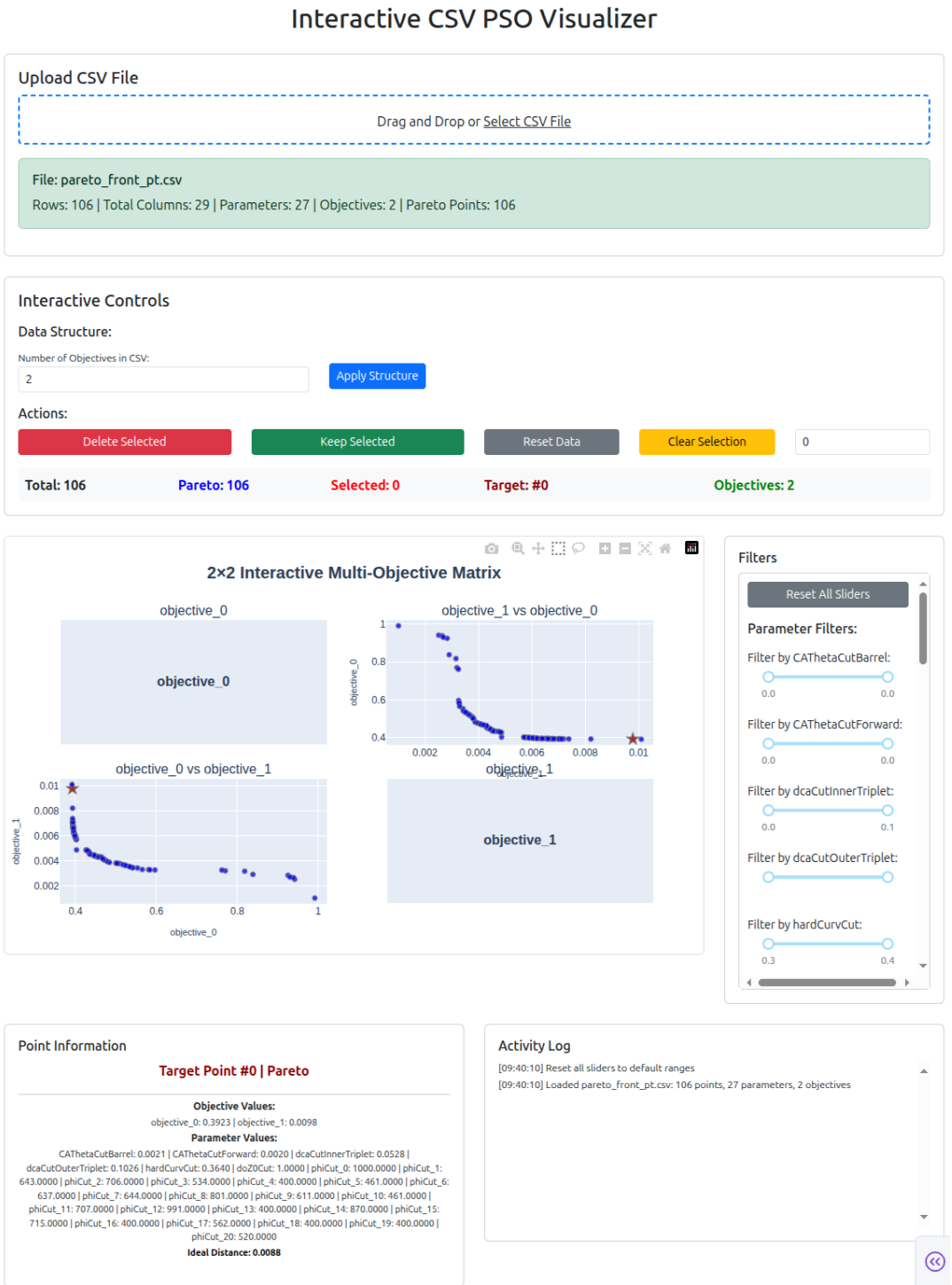

Figure 4.3: The visualization dashboard for `patatune`.

Point selection supports single-click selection and rectangular (drag) selection for groups. Selections are persistent across views: when a subset is selected in one matrix cell, it is highlighted everywhere and can be acted upon with different operations. The "Interactive Controls" panel offers various actions: Delete Selected (removes the selected items from the working dataset), Keep Selected (prunes all other items), Reset Data (restores the original uploaded file), and Clear Selection (deselects without altering the data). All these operations update the in-memory state model and are reflected instantly in the plots.

The "Point Information" panel shows summary statistics for the current working set (average for each objective and selected parameters). "Activity Log" captures user actions (uploads, filtering steps, selections, and exports) in the session state to review the analysis sequences.

The interface is deployed on CERN's general-purpose Platform-as-a-Service (PaaS) [120, 121]. The service is based on the community version of OpenShift: OKD4 [122]. The service is used to deploy and host the applications as containers. The OKD4 infrastructure provides a high level of redundancy and automatically redistributes the workload in the event of incidents. It monitors and reports on the application's health, and maintains backups of resources and volumes. In this deployment, the application runs statelessly (session state held in-memory) and the access is protected behind CERN Single Sign On (SSO) authentication platform.

This visualization layout supports an exploratory workflow in which users can filter, inspect trade-offs across objectives, evaluate the effects of different parameter values on objectives, and identify candidate configurations in the optimal solutions archive.

4.2 Benchmarks with test problems

To assess the performance of the multi-objective optimization algorithm, we rely on standard benchmark problems. These benchmarks provide controlled environments where specific difficulties can be systematically studied. These difficulties can be: multimodality, where different solutions share similar objectives; deceptive optima, where the solution seems to follow a Pareto front while actually getting stuck in local minima; lack of diversity, where the solution clusters in a single part of the Pareto front leaving other parts uncovered; complex Pareto front geometries, where characteristics such as convexity, non-convexity, discontinuities, or non-uniform spacing make maintaining a well-distributed approximation challenging.

The design of these benchmarks allows researchers to test whether an algorithm can both converge to the true Pareto-optimal front and maintain a well-distributed set of solutions across it. The use of standard benchmarks also ensures that improvements on an algorithm are not tied to a particular problem instance but hold more generally.

The origins of many widely adopted functions can be traced back to the work of Deb and collaborators in the late 1990s and early 2000s, who formalized sets of problems with varying difficulty features to expose the strengths and weaknesses of evolutionary approaches [123].

4.2.1 ZDT

The ZDT family of functions [124], named after the initials of the authors, is designed to test the various challenges encountered in searching for a Pareto front, including convergence difficulties and different front geometries such as convexity, non-convexity, discreteness, and non-uniformity.

Each of the functions is structured in the same manner:

Definition 4.2.1 (ZDT Family of Functions)**.** Given the functions f_1, g, and h, a $\mathcal{ZDT}$ test function is a multi-objective minimization problem $(X, \mathcal{ZDT})$ where:

$$x = (x_1, x_2, \ldots, x_n) \in X$$

$$\mathcal{ZDT}(x) = (f_1(x_1), f_2(x))$$

$$f_2(x) = g(x_2, \ldots, x_n)h\left(f_1(x_1), g(x_2, \ldots, x_n)\right)$$

The $\mathcal{ZDT}$ functions differ in f_1, g, and h, as well as the number of variables n and the search space X.

Definition 4.2.2 ($\mathcal{ZDT}_1$)**.** $\mathcal{ZDT}_1 = (f_1(x_1), f_2(x))$ is a test function belonging to the $\mathcal{ZDT}$ family of functions, where $n = 30, X = [0, 1]$:

$$f_1(x_1) = x_1$$

$$g(x_2, \ldots, x_n) = 1 + \frac{9}{n-1}\sum_{i=2}^{n} x_i$$

$$h(f_1, g) = 1 - \sqrt{\frac{f_1}{g}}$$

The Pareto front of $\mathcal{ZDT}_1$ is obtained with $g = 1$.

The $\mathcal{ZDT}_1$ function is used to test the capability of the optimization algorithm to converge to a convex Pareto front.

Definition 4.2.3 ($\mathcal{ZDT}_2$)**.** $\mathcal{ZDT}_2 = (f_1(x_1), f_2(x))$ is a test function belonging to the $\mathcal{ZDT}$ family of functions, where $n = 30, X = [0, 1]$:

$$f_1(x_1) = x_1$$

$$g(x_2, \ldots, x_n) = 1 + \frac{9}{n-1} \sum_{i=2}^{n} x_i$$

$$h(f_1, g) = 1 - \left(\frac{f_1}{g}\right)^2$$

The Pareto front of $\mathcal{ZDT}_2$ is obtained with $g = 1$.

The $\mathcal{ZDT}_2$ function is the counterpart of $\mathcal{ZDT}_1$ and is used to test the capability of the optimization algorithm to converge to a non-convex Pareto front.

Definition 4.2.4 ($\mathcal{ZDT}_3$)**.** $\mathcal{ZDT}_3 = (f_1(x_1), f_2(x))$ is a test function belonging to the $\mathcal{ZDT}$ family of functions, where $n = 30, X = [0, 1]$:

$$f_1(x_1) = x_1$$

$$g(x_2, \ldots, x_n) = 1 + \frac{9}{n-1} \sum_{i=2}^{n} x_i$$

$$h(f_1, g) = 1 - \sqrt{\frac{f_1}{g}} - \left(\frac{f_1}{g}\right) \sin(10\pi f_1)$$

The Pareto front of $\mathcal{ZDT}_3$ is obtained with $g = 1$.

The $\mathcal{ZDT}_3$ test function is discrete, with discontinuity in the Pareto front but a continuous search space.

Definition 4.2.5 ($\mathcal{ZDT}_4$)**.** $\mathcal{ZDT}_4 = (f_1(x_1), f_2(x))$ is a test function belonging to the $\mathcal{ZDT}$ family of functions, where $n = 10, x_1 \in [0, 1], x_2, \ldots, x_n \in [-5, 5]$:

$$f_1(x_1) = x_1$$

$$g(x_2, \ldots, x_n) = 1 + 10(n-1) + \sum_{i=2}^{n} \left(x_i^2 - 10\cos(4\pi x_i)\right)$$

$$h(f_1, g) = 1 - \sqrt{\frac{f_1}{g}}$$

The Pareto front of $\mathcal{ZDT}_4$ is obtained with $g = 1$.

The $\mathcal{ZDT}_4$ test function contains 21^9 local Pareto fronts, testing the ability of the optimization algorithm to deal with multimodal solutions. The best local Pareto that is not the optimal Pareto front is at $g = 1.25$.

Definition 4.2.6 ($\mathcal{ZDT}_5$)**.** $\mathcal{ZDT}_5 = (f_1(x_1), f_2(x))$ is a test function belonging to the $\mathcal{ZDT}$ family of functions, where $n = 11, x_1 \in \{0, 1\}^{30}, x_2, \ldots, x_n \in \{0, 1\}^5$ (x_1 is represented with 30 bits and each of x_2 to x_n are represented by 5 bits):

$$f_1(x_1) = 1 + u(x_1)$$

$$g(x_2, \ldots, x_n) = \sum_{i=2}^{n} v(u(x_i))$$

$$h(f_1, g) = \frac{1}{f_1}$$

$u(x_i)$ is the *unitation* function that gives the number of ones in the bit vector x_i, and:

$$v(u(x_i)) = \begin{cases} 2 + u(x_i) & \text{if } u(x_i) < 5 \\ 1 & \text{if } u(x_i) = 5 \end{cases}$$

The Pareto front of $\mathcal{ZDT}_5$ is obtained with $g = 10$.

The $\mathcal{ZDT}_5$ test function has candidate solutions x_i that are binary strings. The best local Pareto that is not the optimal Pareto front is at $g = 11$.

Definition 4.2.7 ($\mathcal{ZDT}_6$)**.** $\mathcal{ZDT}_6 = (f_1(x_1), f_2(x))$ is a test function belonging to the $\mathcal{ZDT}$ family of functions, where $n = 10, X = [0, 1]$:

$$f_1(x_1) = 1 - e^{-4x_1} \sin^6 6\pi x_1$$

$$g(x_2, \ldots, x_n) = 1 + 9 \left(\frac{\sum_{i=2}^{n} x_i}{n - 1} \right)^{1/4}$$

$$h(f_1, g) = 1 - \left(\frac{f_1}{g} \right)^2$$

The Pareto front of $\mathcal{ZDT}_6$ is obtained with $g = 1$.

The $\mathcal{ZDT}_6$ test function has a solution not uniformly distributed along the Pareto front with a bias for solutions f_1 near 1. Moreover, the density of the solutions is lower near the Pareto front and highest away from it.

4.2.2 Results

The performance of the algorithm has been tested on the $\mathcal{ZDT}$ functions. Since the desired front is known in the test problems, the convergence metrics $\mathcal{GD}$ and $\mathcal{IGD}$ have been used as performance indicators. The performance has been measured by running the algorithm 10 times with different random seeds and averaging the results. For each test function, different values of w, c_1, c_2 have been compared.

The plots in Figure 4.4 show the values of the performance indicators as a function of the inertial weight w, for the different combinations of cognition and social coefficients, c_1 and c_2, for the $\mathcal{ZDT}_1$ function. The optimization has been run for 150 iterations

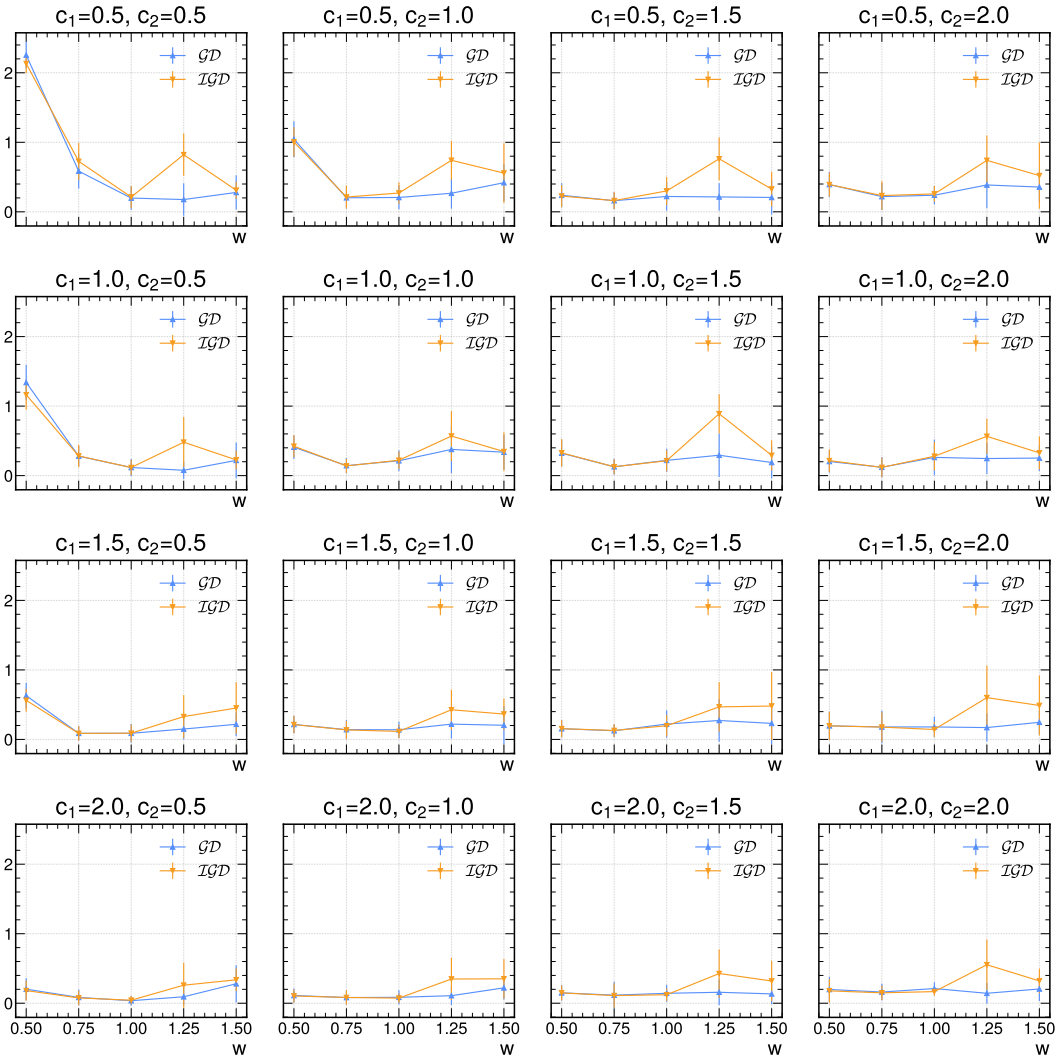

Figure 4.4: The value of the $\mathcal{IGD}$ and $\mathcal{GD}$ performance indicators as a function of w, c_1 and c_2 for the $\mathcal{ZDT}_1$ function. The error bars represent the standard deviation of the averages across 10 runs.

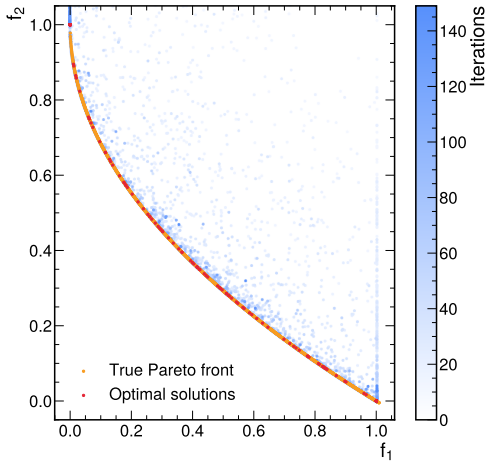

Figure 4.5: Optimal solution found by running `patatune` on the $\mathcal{ZDT}_1$ test function. The blue points show the solution found in each iteration. The red points show the archived solution on the last iteration. The orange points correspond to the true Pareto front.

with 100 particles, limiting the size of the archive of solutions to the 200 solutions with the highest crowding distance. The benchmark shows that the best combination of w, c_1, c_2, with the lowest average and standard deviation for the two distances, is for $w = 1, c_1 = 2, c_2 = 1$. The plot in Figure 4.5 shows the solutions obtained by running the algorithm with that configuration.

Appendix A shows the same analysis for the other $\mathcal{ZDT}$ functions.

4.3 Future developments: Reinforcement Learning

To reduce the computational time of the optimization process, a Reinforcement Learning (RL) extension to `patatune` has been studied to skip unnecessary objective function evaluations. The intention is to train a model to avoid evaluations that are not expected to improve the final solution and, consequently, to accelerate convergence. The outcome of this study, originally presented in the MSc thesis reported in [125], is briefly summarized in this section.

Reinforcement learning is a ML paradigm in which an agent learns to perform a task through interaction with an environment [126]. Training consists of learning a mapping from *states* to *actions* that maximize a numerical *reward*. The learner is not explicitly instructed which actions to take, but instead discovers the ones that yield the greatest long-term reward.

In this framework, two main components can be identified. The *agent* is the learner and decision maker, while the *environment* represents everything the agent interacts with. The interaction between agent and environment is continuous: after an action is selected, the environment responds by providing a new state and a reward associated with the action. Solving a reinforcement learning task corresponds to finding an optimal *policy* that allows the agent to maximize the cumulative reward in the long run.

Multi-Agent RL (MARL) extends this paradigm to multiple agents interacting within a shared environment. Each agent seeks to maximize its own reward, which leads to complex coordination or competition dynamics. Depending on the task, agents can be trained to cooperate toward a common goal or to compete under different policies.

To accelerate the optimization, each particle in MOPSO is replaced by an intelligent agent that makes decisions according to a policy. A MARL learning algorithm was trained to learn such a policy. The reinforcement learning algorithm selected for this purpose was the Proximal Policy Optimization (PPO) algorithm [127]. To describe the environment, each run of the MOPSO algorithm was considered an *episode* divided into steps, one for each iteration. At every step, each agent decides whether to evaluate its position or not, and receives a reward based on the outcome. The environment then provides each agent with the new state resulting from the chosen action. Each agent's state is defined by information extracted from an N-dimensional sphere, where N is the dimension of the search space, defined around each agent's position at every iteration. The state includes the number of non-dominated points within the sphere, obtained from the current archive of non-dominated solutions, and the number of dominated points in the same region. Figure 4.6 shows a representation of the agent state in a two-dimensional parameter space.

The reward provided to each agent is composed of three terms: The *hypervolume term* rewards the final approximation of the Pareto front and is given only at the end of each episode, computed as per Definition 4.1.3, rewarding solutions close to the true Pareto front; The *evaluation term* that gives a negative reward for each function evaluation performed; The *non-dominated term*, which rewards evaluations that yield a non-dominated point. The final reward function is a linear combination of the three components.

The training was performed on test functions closely related to the $\mathcal{ZDT}$ family of functions described in Section 4.2.1, however with some alteration to use a smaller parameter space. The trained policy leverages each agent's local surroundings to decide whether to evaluate or skip an objective function. This behavior not only resulted in faster convergence, but also produced a modest improvement in the final Hypervolume, at the expense of a reduced number of Pareto points.

The trained policy was applied to a specific use case, namely the tuning of the hyperparameters of the clustering algorithm used in the Phase-2 HGCAL reconstruc-

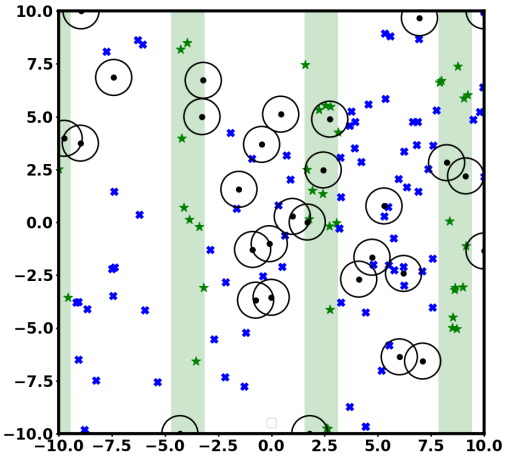

Figure 4.6: MOPSO environment with several particles (black dots) moving in a 2D parameter space. The state of each agent is characterized by the number of non-dominated points (green stars) and dominated points (blue crosses) in a sphere (black circle) around the particle. Evaluations within the green area results in a non-dominated point [125].

tion [58]. In this context, the agents achieved an improvement in the corresponding Hypervolume of approximately 18% while skipping about 42% of the evaluations, resulting in a reduction of roughly 40% of computation time. These results were obtained with a relatively simple multi-agent reinforcement learning approach, leaving considerable room for further exploration and refinement. Future developments could focus on redefining state and action representations, improving reward design, and exploring alternative learning architectures to enhance the performance and generalization of the approach.

In summary, this study represents an initial step toward integrating reinforcement learning techniques into `patatune`. While the proposed approach remains exploratory, it demonstrates the potential of learning-based methods to improve efficiency in complex optimization problems and provides a foundation for future developments.

4.4 Summary

This chapter provides a comprehensive analysis of `patatune`, a Python software package designed to solve multi-objective optimization problems. The tool was built with a modular and flexible architecture to meet the specific requirements of the CMS experiment. The discussion covers its foundational architectural principles, including its design to decouple optimization logic from the problem domain. `patatune`'s core is

its highly configurable MOPSO algorithm, which allows users to fine-tune the balance between exploration and exploitation by controlling its parameters. To evaluate its performance, `patatune` implements standard metrics like Generational Distance ($\mathcal{GD}$), Inverted Generational Distance ($\mathcal{IGD}$), and Hypervolume ($\mathcal{HV}$). The package also includes a custom web-based dashboard for interactive visualization and analysis of results. The chapter concludes with an overview of benchmarks conducted on standard $\mathcal{ZDT}$ test functions to validate the effectiveness of the algorithm in handling various challenges, such as different Pareto front geometries and multimodality. These tests demonstrate the ability of the algorithm to successfully converge to the true Pareto front while maintaining a diverse set of solutions, confirming its suitability for the complex optimization problems it is intended to solve.

Chapter 5

Optimizing track reconstruction

The measurement of charged particle parameters and trajectories is essential for any HEP experiment. It essentially involves building a list of particle track candidates from a collection of 3D positions of hits in a tracker detector.

The reconstructed tracks of charged particles are among the most fundamental objects in the reconstruction of particle collisions. Tracks are used for the reconstruction of electrons, muons, hadrons, taus, and jets, as well as in the determination of the primary interaction vertices.

These tracks are then used to study the physical properties of the triggering particle and analyze the collision event in detail.

Collisions in the HL-LHC produce a pileup effect, where multiple proton-proton collisions occur per bunch crossing. The highly dense environment expected at HL-LHC will result in thousands of charged particles with tens of thousands of hits per event, with only a fraction of those linked to particles of interest. In these conditions, particle tracking will be the most time-consuming component of the event reconstruction chain, and a considerable effort is being devoted to studying these algorithms to improve their performance. As presented in Section 2.2.1, Task 3.1 of the NextGen project involves the extension of these algorithms to improve efficiency beyond the baseline objectives of HL-LHC. This work will require additional effort in selecting the parameters involved in the reconstruction.

This chapter presents the current implementation of the CMS pixel track reconstruction algorithm, along with the results of optimizing its parameters using the `patatune` algorithm. The final section presents the results of using the optimization framework to tune the parameters of the reconstruction algorithm, adapting it to a different detector without requiring direct modifications to the reconstruction code.

5.1 Patatrack pixel track reconstruction

The Patatrack pixel track and vertex reconstruction algorithm is a software developed within the official CMS reconstruction software to utilize heterogeneous computing and improve the performance of the reconstruction chain for Run 3 and Phase-2 of the experiment.

In the current setup, HLT pixel track reconstruction is implemented using a hit-chain maker based on Cellular Automata (CA) [79, 128, 129], which enables a level of parallelism suitable for GPU architectures while improving physics performance compared to previous propagation-based methods.

In this process, the event data are first reconstructed to extract information about individual hits in the detector. Raw data are unpacked to create *digis* that represent single pixels in the detector with charge exceeding the noise threshold. Neighboring digis are grouped into clusters through an iterative process, and the cluster shape and total charge are used to determine the hit position in the detector's local coordinates. The track reconstruction proceeds in several stages. First, a graph structure is built from the *seeding layers* to form track seeds. The CA-based algorithm then constructs track candidates through the following steps: By iterating over all pairs of consecutive seeding layers, the algorithm collects all compatible hit pairs (*doublets*) that could belong to the same track in the *doublet building* step. For each hit on an outer layer, the algorithm searches for compatible hits on the inner layer within geometrical windows. Each doublet forms a *cell* in the CA containing information about its two hits. In the *doublet connection* step, cells sharing a common hit are tested for compatibility. Two cells form valid neighbors if they satisfy multiple geometrical criteria that ensure three-hit alignment in both the longitudinal $(r - z)$ and transverse $(x - y)$ planes, and compatibility with tracks originating from the beamspot region. The cellular automaton then evolves iteratively, propagating chain-length information from outer to inner cells and building *n-tuplets*. After the evolution completes, cells whose innermost hit lies on a seed layer and have the appropriate state are identified as valid seed origins. A Depth-First search from each root cell extracts all connected n-tuplets by traversing the neighbor connections established during the connection step. The Fishbone algorithm [79] then resolves ambiguities when multiple n-tuplets correspond to the same particle due to overlapping sensitive layers or shared hits between candidates. The resulting n-tuplets are fitted using the Broken Line fit algorithm [130] to compute final track parameters and their uncertainties. A χ^2 quality criterion determines which candidates proceed to primary vertex reconstruction. Only tracks that fulfill quality requirements are stored for downstream processing and physics analysis. The algorithm involves over 40 parameters that control the doublet building and connection steps, as well as the track-quality selections. These parameters define geometrical acceptance windows

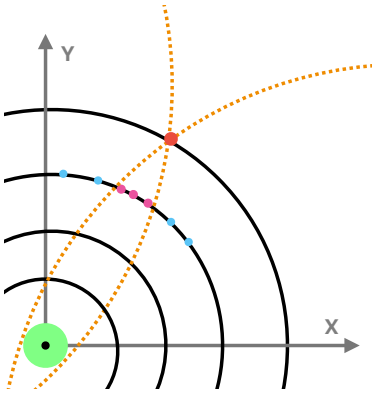

Figure 5.1: An example of use of the *phiCut* parameter for the doublet building step of the Pixel Track reconstruction algorithm [131].

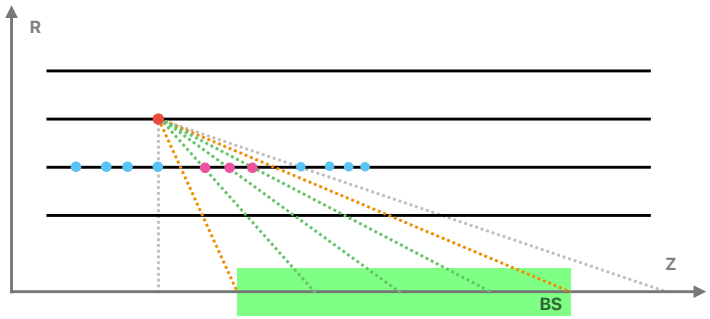

Figure 5.2: An example of the use of the *z0Cut* parameter for the doublet building step of the pixel track reconstruction algorithm [131].

(*cuts*), compatibility criteria, and quality thresholds that directly impact both the reconstruction efficiency and misidentification rate. The specific parameters optimized in this work are detailed in Section 5.1.1, while the performance metrics used to evaluate different parameter configurations are defined in Section 5.1.2.

5.1.1 Geometrical cuts

Of the parameters identified, 20 are used for the doublet building step.

- the angular cut-offs for the difference between each doublet inner and outer hit on the azimuthal angle around the beam axis on the different modules of the detector. Each of the possible layer pairs has its own cut, resulting in 19 independent *phiCuts*; Figure 5.1 shows an example of one of the *phiCuts*, represented by the orange dashed lines. It is a window in ϕ (azimuthal angle) opened in the transverse plane $(x - y)$. For each hit on the outer layer (red) the criteria selects the compatible hits (magenta) between all hits on the inner layer (light blue).

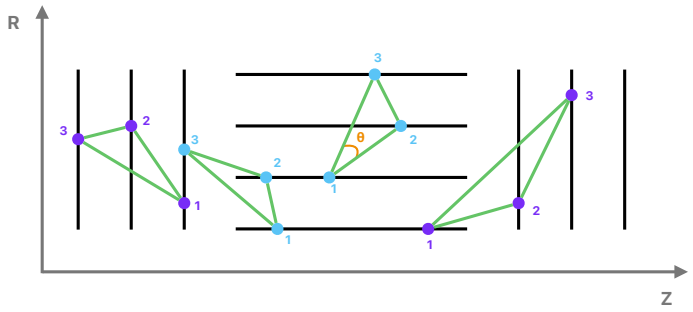

Figure 5.3: Illustration of the *caThetaCuts* for the doublet connection step of the pixel track reconstruction algorithm [131].

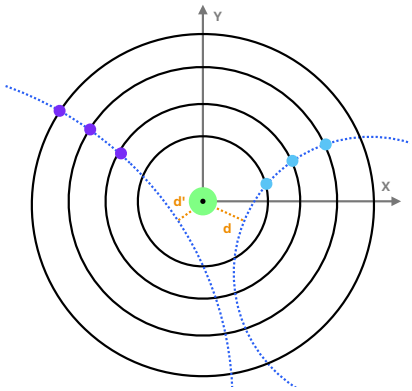

Figure 5.4: Picture of the *dcaCut* for the doublet connection step of the pixel track reconstruction algorithm [131].

- the cut-off on the displacement with respect to the beamspot (*z0Cut*). Figure 5.2 shows an illustration of the *z0Cut* represented by the orange dashed lines. This criterion selects a window in z in the longitudinal plane ($r - z$). The segments connecting the outer hit to the inner hits are prolonged to the beamspot. Only the inner hits for which the prolonged segment lies within the beamspot region (green) are considered compatible with the outer hit.

For the doublet connection step, five parameters are defined, as represented in Figures 5.3 to 5.5.

- The *CAThetaCut* (in Figure 5.3) is a selection on the maximum of θ angle, which acts as a proxy of the three hits alignment in the longitudinal plane ($r - z$). Two independent values are defined for this criterion: If the two innermost hits (1;2) lie on the barrel layers, the selection applied is *CAThetaCutBarrel* (light blue hits); in all the other cases, the *CAThetaForward* (purple hits).

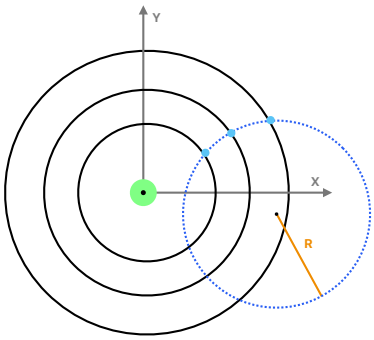

Figure 5.5: Diagram of the *hardCurvCut* for the doublet connection step of the pixel track reconstruction algorithm [131].

- Figure 5.4 shows the maximum on the distance of closest approach (DCA) to the beamspot region. The distance d is measured between the fit circle (blue) and the beamspot (green) in the transverse plane ($x - y$). Two independent values are defined for this criterion: The hits selected lying on the first barrel layer (light blue) use the *dcaCutInnerTriplet*; all the other cases (purple hits) use the *dcaCutOuterTriplet*.

- Figure 5.5 shows the selection of the minimum on the curvature of the tracks in the transverse plane (*hardCurvCut*). It is a selection of the maximum inverse of the radius R of the fit (in orange). The circle is fit to three hits forming a triplet (light blue) in the transverse plane ($x - y$).

5.1.2 Efficiency and fake rate

To ensure unbiased physics results, the algorithm needs high efficiency in the reconstruction. This means a high percentage of charged particles within a specific region must be identified as track candidates. The *efficiency* is defined as the percentage of simulated tracks (N_{sim}) that have been associated with at least one reconstructed track ($N_{ass(reco \to sim)}$) (Equation (5.1)). A reconstructed track is associated with a simulated track if more than 75% of the hits come from the same simulated tracks.

$$efficiency = \frac{N_{ass(reco \to sim)}}{N_{sim}} \tag{5.1}$$

In track candidate processing, additional effort is devoted to ambiguity resolution, which aims to eliminate track candidates from random hit combinations (known as fakes) and to identify track duplicates resulting from shared hits between track candidates. The rate at which the algorithm incorrectly identifies noise or unrelated hits as valid track candidates is know as *fake rate*. It is defined as the fraction of all

the reconstructed tracks (N_{rec}) that are not uniquely associated to a simulated track ($N_{ass(sim \to reco)}$) (Equation (5.2)).

$$fake\ rate = \frac{N_{rec} - N_{ass(sim \to reco)}}{N_{rec}} \tag{5.2}$$

Keeping the fake rate low (or the fake and duplicate rate when considering duplicates) is essential for managing the number of track candidates, computational costs, and physics performance.

Relationship between efficiency and fake rate

Optimizing for high efficiency often involves setting more permissive criteria for track identification, allowing the algorithm to capture as many true tracks as possible. However, this increased leniency can also lead to a higher number of fake tracks, as the algorithm may incorrectly identify noise or artifacts as valid tracks. It may similarly increase the number of duplicates, as slightly varying paths of the same particle might be reconstructed multiple times.

Conversely, tightening the criteria to reduce fake and duplicate rates means that the algorithm becomes more strict in what it considers a valid track. This is the likelihood of false positives and duplicates, but it can result in missing true tracks, reducing efficiency.

For these reasons, these two objectives are considered conflicting and, as is often the case, improvements in one can lead to trade-offs in the other, requiring a balance to achieve an optimal set of solutions rather than a single optimal point.

Optimizing for these two parameters is therefore a multi-objective optimization problem, as defined in Chapter 3, and is a suitable use case for `patatune`.

5.1.3 Parameter tuning

The MOPSO algorithm is used to optimize the parameters of the Patatrack pixel track reconstruction algorithm. The overall pixel tracks efficiency (Equation (5.1)) and fake rate (Equation (5.2)) are used as the objective functions. The optimization involves the 25 parameters described in Section 5.1.1 and is run on 100 events. The algorithm is executed over 150 iterations using 200 particles, with $w = c_1 = c_2 = 1$. The whole run took approximately 5 hours on an HLT node with a single job, 8 threads, and 1 GPU assigned. The performance has been measured in a simulated $t\bar{t}$ sample with superimposed pileup events (with $< PU > \simeq 50$). The detector conditions are simulated to include an issue affecting the Barrel Pixel Layers 3 and 4 (BPix3 and BPix4) as of the TS1 in 2023: 27 modules in BPix3 and BPix4 became inoperable due to a problem with the LHC clock distribution [132]. These modules have been turned off since this incident. To manage this issue, an additional pixel doublet recovery iteration has been

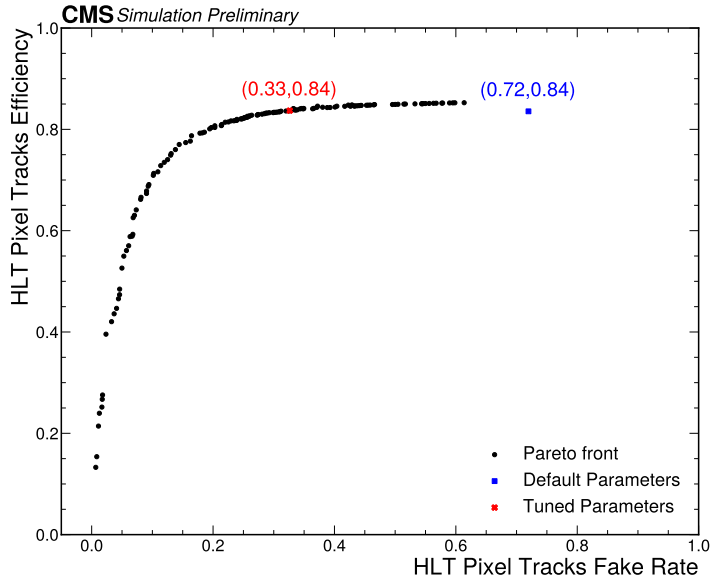

Figure 5.6: The Pareto front (in black) obtained by running the optimization. The blue square dot is the result obtained with the default parameters. The red cross is the chosen working point [131].

added to the HLT tracking sequence [131]. The resulting Pareto front is shown in Figure 5.6. The new working point has been chosen because it maintains the same efficiency as the default while achieving the lowest fake rate.

The comparison of the default and tuned parameters in Table 5.1 reveals notable adjustments. These configurations are the ones used to produce the validation plots in the following section.

The *CAThetaCutBarrel* and *CAThetaCutForward* parameters have been reduced by 41% and 33%, respectively. The *hardCurvCut* has also been slightly decreased by 8%. Meanwhile, the *dcaCutInnerTriplet* has seen a substantial increase of 300%, while the *dcaCutOuterTriplet* has increased by 50%. Additionally, the *cellZ0Cut* has been reduced by 26%. The *phiCuts* parameters have undergone significant modifications, with individual values either increasing or decreasing. Some notable changes include increases of 38% and 59% in some entries, while others have been reduced by as much as 34%. These adjustments aim to optimize the overall performance while maintaining the desired efficiency. These configurations have been used to generate the validation plots first presented in [133] and are reported in the next section.

Table 5.1: Value of the default and tuned set of parameters and the relative percentage change.

	Default	Tuned	Relative change
CAThetaCutBarrel	0.002	0.001180	-41%
CAThetaCutForward	0.003	0.001997	-33%
hardCurvCut	0.0328407225	0.0302418577	-8%
dcaCutInnerTriplet	0.15	0.5994313104	+300%
dcaCutOuterTriplet	0.25	0.3738067689	+50%
cellZ0Cut	12.0	8.846	-26%
phiCuts	[522, 730, 730, 522, 626, 626, 522, 522, 626, 626, 626, 522, 522, 522, 522, 522, 522, 522]	[661, 598, 595, 718, 707, 858, 656, 832, 677, 411, 650, 767, 703, 682, 637, 601, 675, 745, 797]	[+27%, -18%, -18%, +38%, +13%, +37%, +26%, +59%, +8%, -34%, +4%, +47%, +35%, +31%, +22%, +15%, +29%, +43%, +53%]

5.1.4 Results

The HLT pixel tracks form the basis of the complete HLT tracking chain, providing fast track and vertex reconstruction for event selection in the trigger system [134]. Pixel tracks are used both as standalone objects and as seeds for the subsequent full tracking iterations that incorporate strip detector information. The quality of pixel tracks directly impacts the trigger efficiency, as fake tracks can lead to incorrect classification of primary vertices and inefficient use of computational resources. When used as seeds for full tracking, pixel tracks provide initial track parameter estimates that guide the pattern recognition in the strip detector. High-quality pixel track seeds reduce the combinatorial complexity of full tracking by constraining the search space for compatible strip hits.

The HLT pixel tracks with the tuned setup exhibit compatible efficiency compared to the default setup across the full phase space in p_{T} (Figure 5.7), η (Figure 5.8), ϕ (Figure 5.9), and pileup conditions (Figure 5.10). Critically, the fake rate is reduced by approximately 50% almost uniformly across all these variables. This substantial reduction in fake rate directly benefits the HLT timing budget by reducing the number of seed vertices that must be explored in subsequent tracking iterations, while maintaining the same level of efficiency in reconstructing tracks [131].

The impact of improved pixel track quality propagates through the full tracking chain. The HLT full tracks, seeded by the optimized HLT pixel tracks, demonstrate compatible efficiency and fake rate compared to the default configuration (Figures 5.11 to 5.14). This result validates that the 50% reduction in pixel track fake rate successfully translates into full tracking without compromising the overall efficiency.

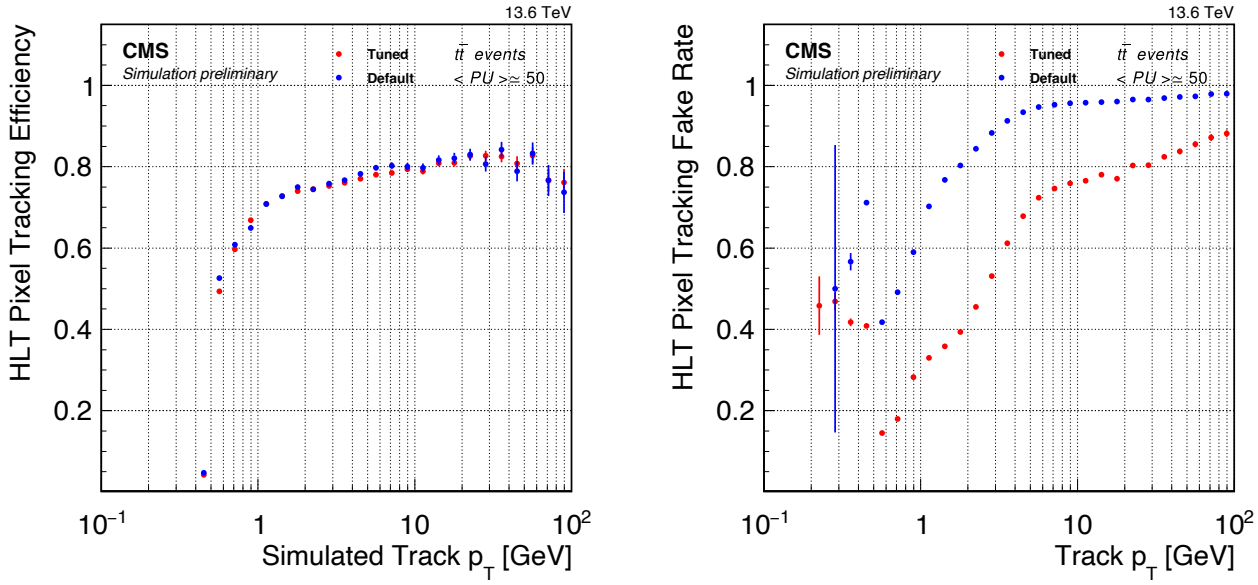

Figure 5.7: On the left, the HLT pixel tracking efficiency as a function of the simulated track p_{T}. On the right, the HLT pixel tracking fake rate as a function of the reconstructed track p_{T}. In blue, the results for default parameters, in red for the tuned ones [131].

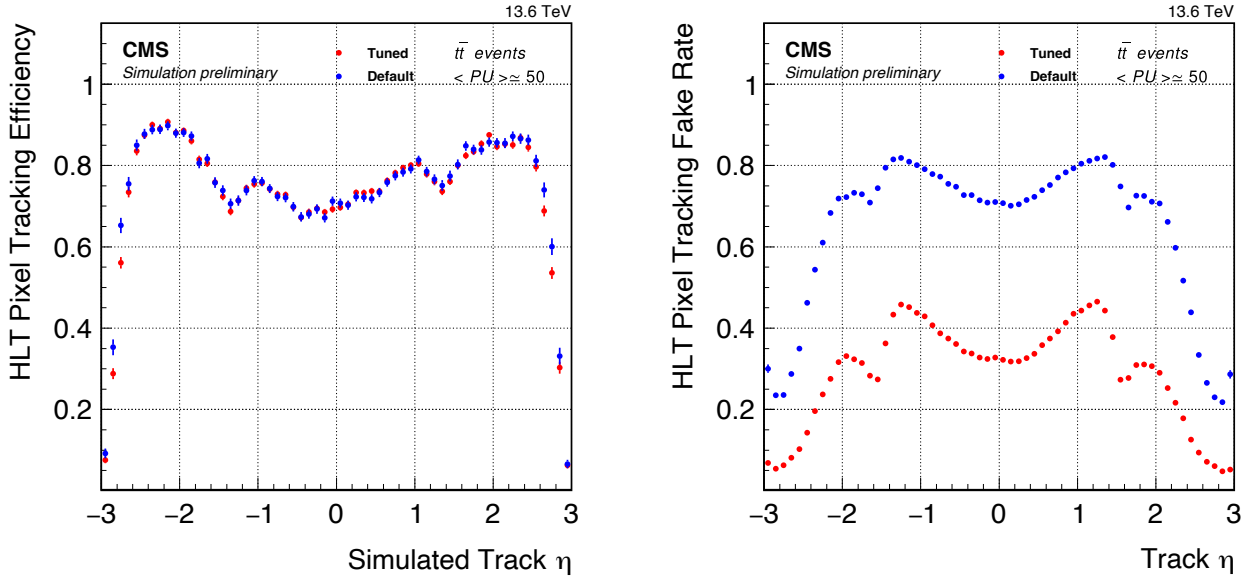

Figure 5.8: On the left, the HLT pixel tracking efficiency as a function of the simulated track pseudorapidity η. On the right, the HLT pixel tracking fake rate as a function of the reconstructed track pseudorapidity η [131].

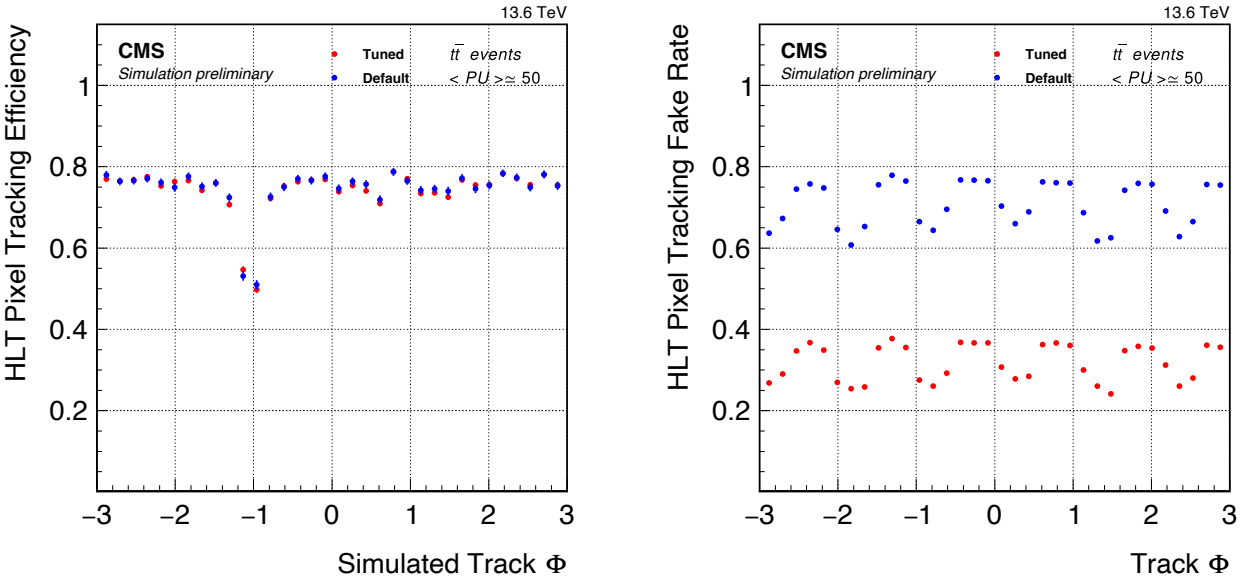

Figure 5.9: On the left, the HLT pixel tracking efficiency as a function of the simulated track azimuthal angle ϕ. On the right, the HLT tracking fake rate as a function of the reconstructed track azimuthal angle ϕ. In blue, the results for default parameters, in red for the tuned ones [131].

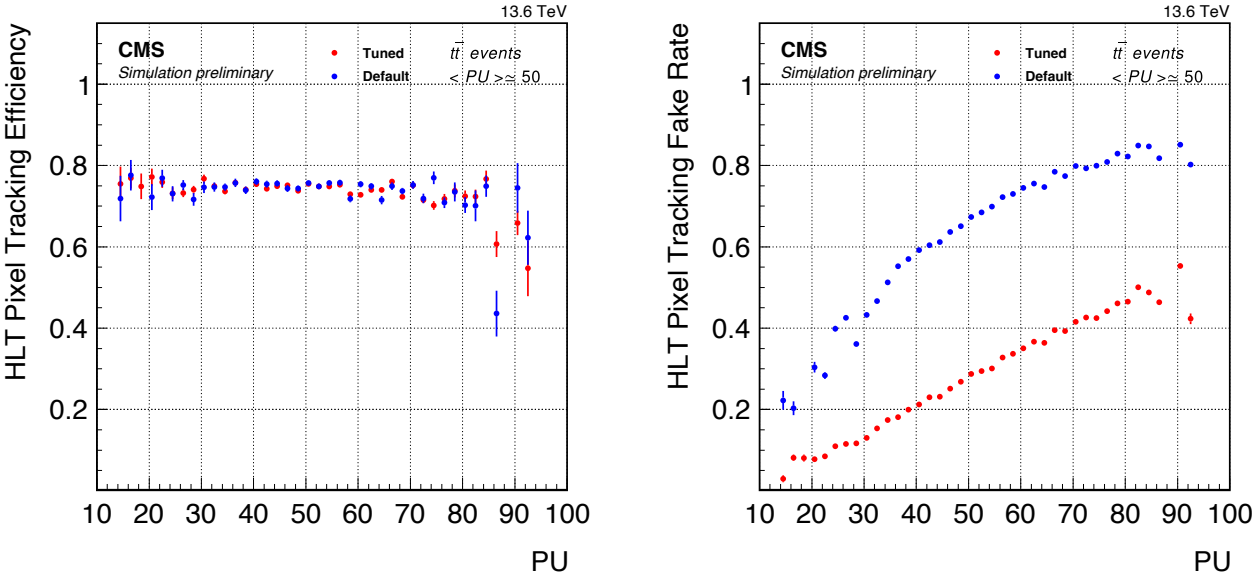

Figure 5.10: On the left, the HLT pixel tracking efficiency as a function of the event pileup (PU). On the right, the HLT pixel tracking fake rate as a function of the event pileup (PU). In blue, the results for default parameters, in red for the tuned ones [131].

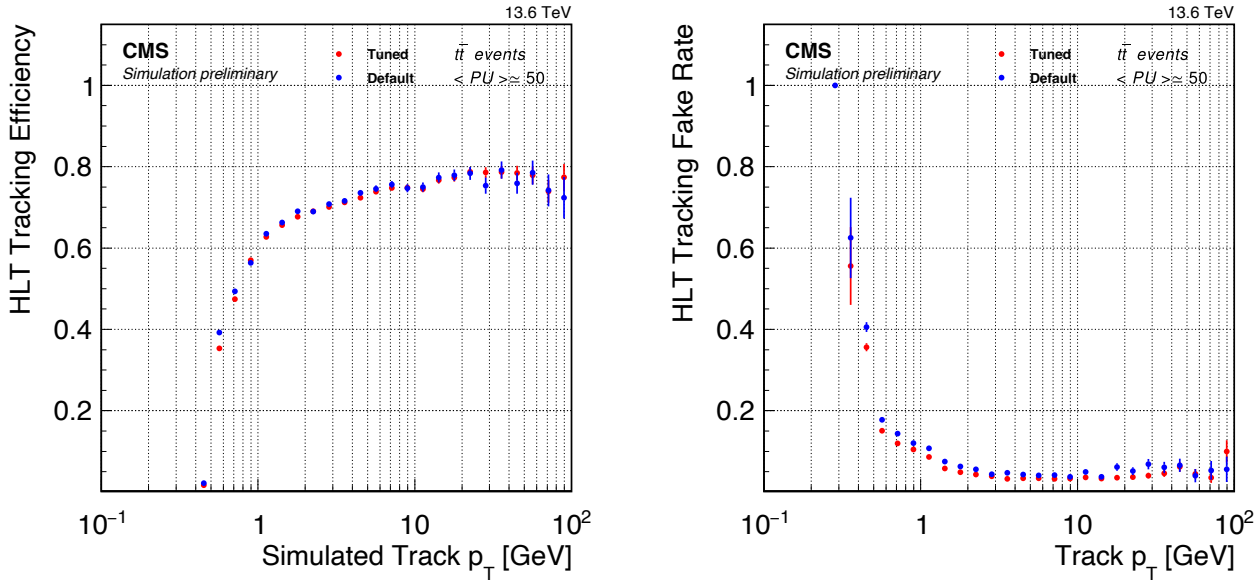

Figure 5.11: On the left, the HLT tracking efficiency as a function of the simulated track p_{T}. On the right, the HLT tracking fake rate as a function of the reconstructed track p_{T}. In blue, the results for default parameters, in red, for the tuned ones [131].

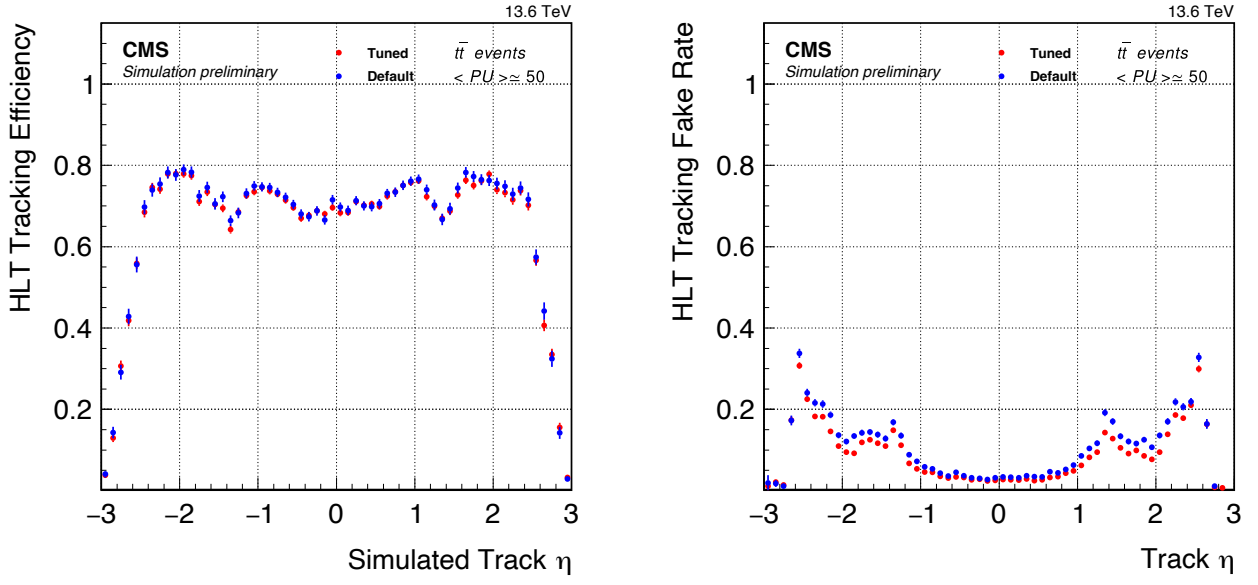

Figure 5.12: On the left, the HLT tracking efficiency as a function of the simulated track pseudorapidity η. On the right, the HLT tracking fake rate as a function of the reconstructed track pseudorapidity η. In blue, the results for default parameters, in red for the tuned ones [131].

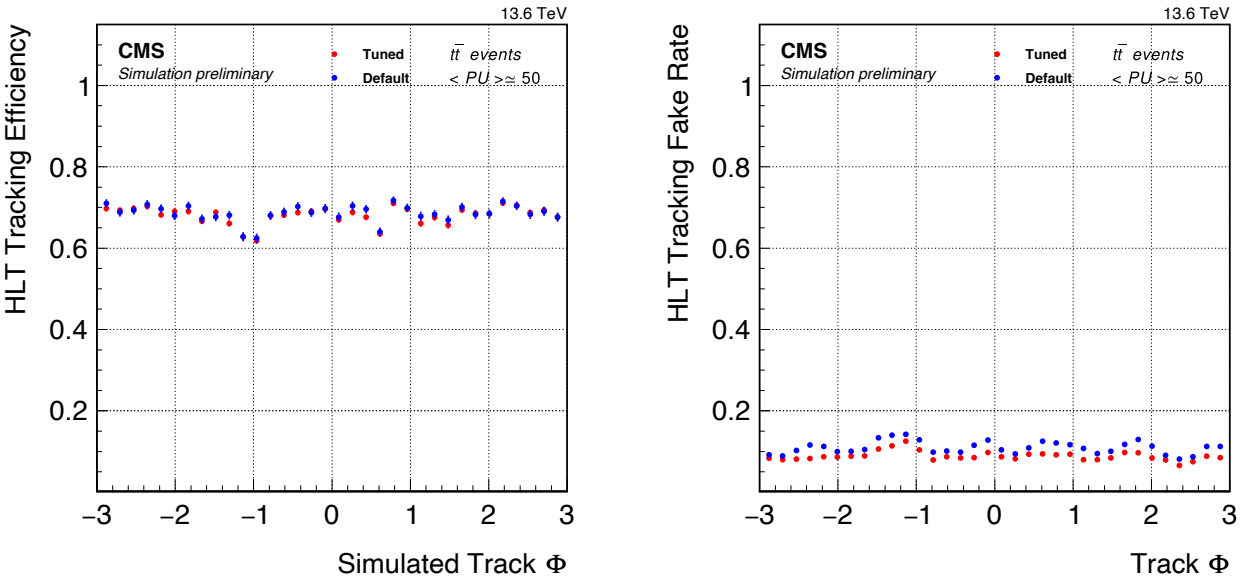

Figure 5.13: On the left, the HLT tracking efficiency as a function of the simulated track azimuthal angle ϕ. On the right, the HLT tracking fake rate as a function of the reconstructed track azimuthal angle ϕ. In blue, the results for default parameters, in red for the tuned ones [131].

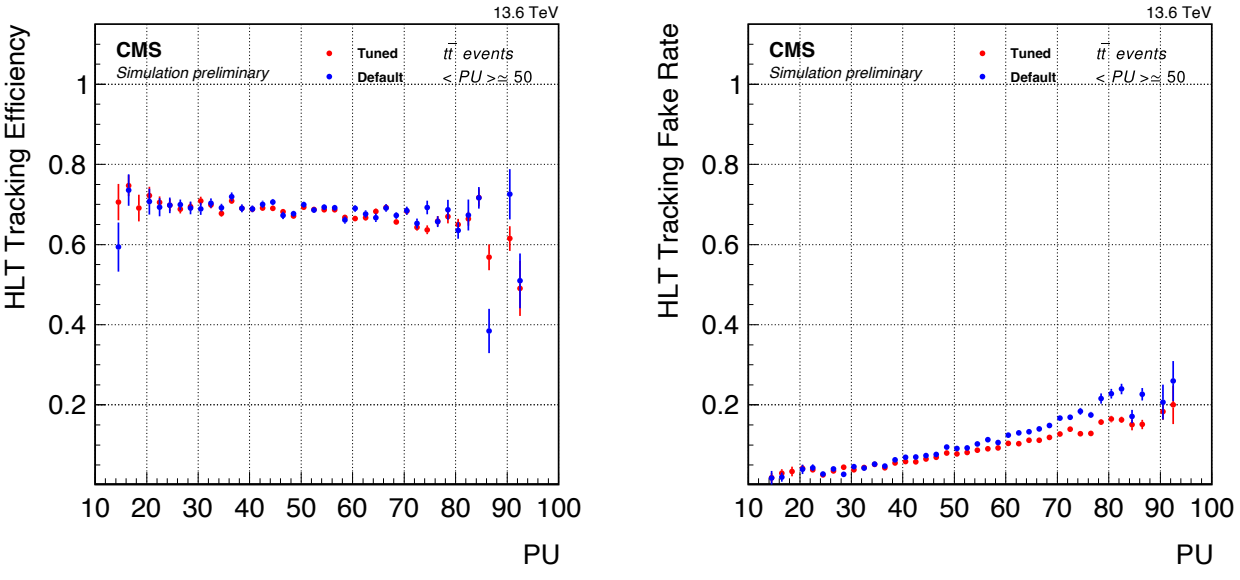

Figure 5.14: On the left, the HLT tracking efficiency as a function of the event pileup (PU). On the right, the HLT tracking fake rate as a function of the event pileup (PU). In blue, the results for default parameters, in red for the tuned ones [131].

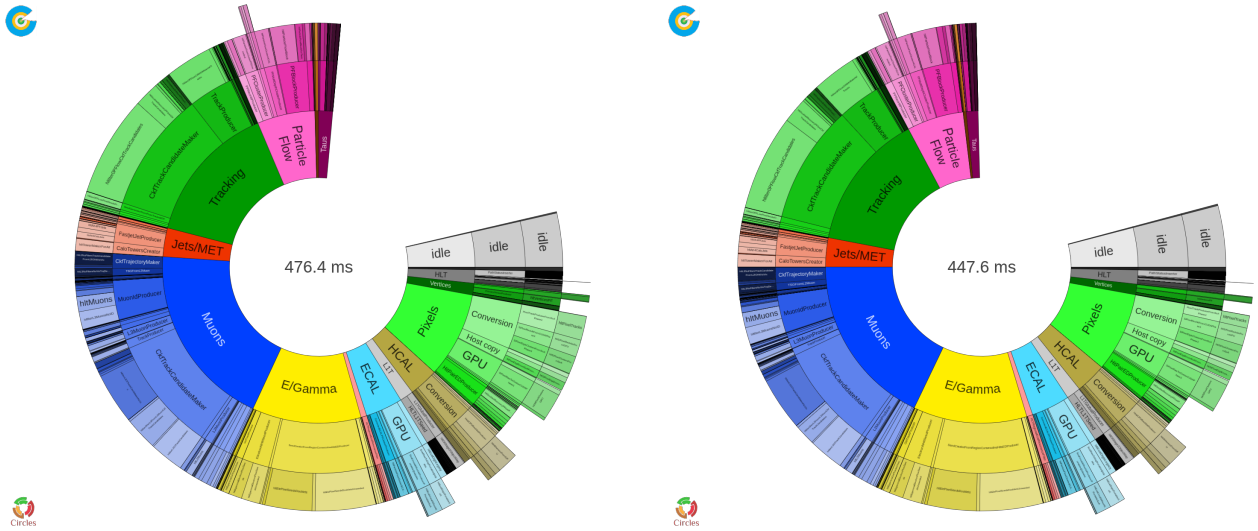

Figure 5.15: Time spent in one job of the HLT reconstruction run using the default setup (on the left) and the new tuned parameters on the right. Each slice represents the fraction of time spent by each part of the reconstruction. The empty slice indicates time spent outside of event reconstruction algorithms.

Timing performance

A preliminary investigation was conducted to assess the impact of parameter tuning on the timing of the HLT reconstruction. This study involved processing a sample of 11,639 physics events collected on July 7, 2023, which had passed the Level-1 (L1) selection. The event sample represents the typical data stream processed by the HLT during data taking, characterized by an average pileup ($< PU >$) of approximately 61. The tests were run on a machine identical to those in the HLT farm [41], which is equipped with two AMD EPYC 7763 "Milan" CPUs, 256 GB of system memory, and two NVIDIA T4 GPUs. The measurements were performed using eight concurrent jobs, each with 32 threads and 24 streams, and were repeated three times to ensure consistency.

Figure 5.15 shows the timing performance. The pie charts represent the time spent per event on HLT reconstruction for one of the eight jobs run when using the default setup for HLT pixel tracks, on the left, and the new tuned parameters, on the right. The average time spent per event is 476.4 ms with the default set of parameters, and 447.6 ms for the tuned parameters. For the whole machine running eight simultaneous jobs, the measured average throughput is 539.8 ± 1.1 events/s in the first case and 569.5 ± 2.5 events/s in the [131].

These preliminary results indicate a speedup of approximately 5.5% for the overall HLT reconstruction process when using the new tuned parameters compared to the default values.

5.1.5 NextGen CA extension to pixel tracking

Within the context of task 3.1 for the Real-Time Reconstruction Revolution project (see Section 2.2.1), the NextGen collaboration is extending the Cellular Automata algorithm of the pixel track reconstruction to include more regions of the detector.

With the upgraded tracker for Phase-2 (see Section 1.3.7 the reconstruction algorithm must be optimized for the higher pileup and the new geometry. In particular, the new extension adds information from the macro-pixels of the first three layers in the barrel of the Outer Tracker, tightening the requirement on the fit of short tracks to improve fake rejection [42].

Given the high combinatorial complexity of parameter selection and the interdependencies between parameters as described in Section 5.1.1, `patatune` was applied to optimize the reconstruction algorithm. Unlike the previous optimization study where efficiency and fake rate were computed as single global values, this optimization employed a more granular approach where the metrics defined in Equations (5.1) and (5.2) were binned in η and p_{T}, with efficiency and fake rate in each bin treated as a separate objective. This resulted in a 12-objective optimization problem that captures the algorithm's performance across different regions of phase space. The problem studies the trade-offs between performance in the barrel and the two endcap regions and between low, medium, and high momentum tracks.

Eight parameters were selected for optimization, including those described in Section 5.1.1 but excluding the layer-pair-specific cuts. The Pareto front obtained from this optimization is shown in Figure 5.16. For comparison, two reference configurations are displayed: the *default* point corresponds to the initial parameter set for the pixel track reconstruction algorithm, while the point labeled "Single Muon" represents a manually tuned configuration optimized specifically for single muon events without pileup. This simplified scenario is used for algorithm development but does not represent realistic conditions. The archive of non-dominated solutions identified by `patatune` enables the selection of working points tailored to specific physics requirements. For instance, "Point 16" (Section 5.1.5) achieves higher efficiency than the default configuration across most η while reducing the fake rate in some phase space regions.

These results further validate the potential of `patatune` for tuning complex reconstruction algorithms. The test was done to determine whether binning the objective functions would enable the optimization to balance performance across different detector regions without finding solutions that excel in one area at the expense of others.

This study further highlighted a limitation of the automatic optimization process: the computational cost of the optimization scales directly with the time required to reconstruct and validate each candidate parameter set. The total running time can become prohibitive as the number of particles or iterations increases. This constraint

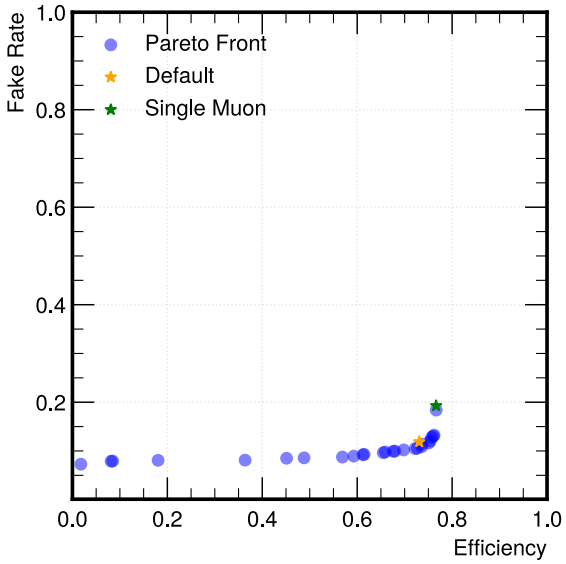

Figure 5.16: Pareto front (blue points) obtained from the optimization of the extended Cellular Automaton reconstruction. The yellow star indicates performance with default parameters, while the green star shows results obtained with parameters manually optimized for a Single Muon sample without pileup.

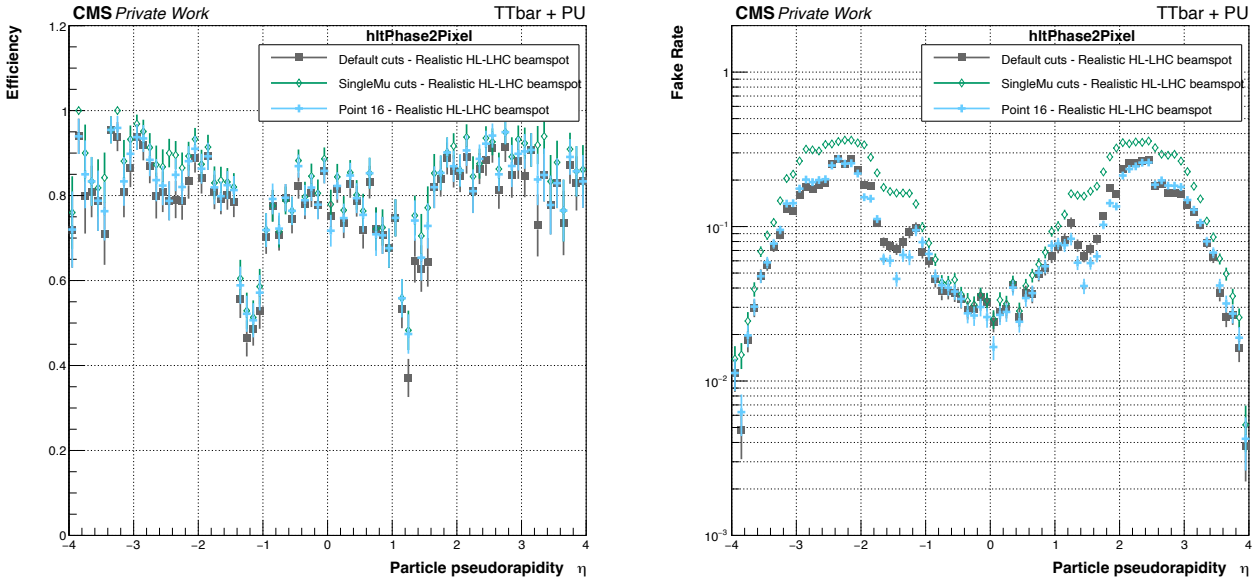

Figure 5.17: Efficiency (on the left) and fake rate (on the right) as a function of the pseudorapidity η for the Phase 2 HLT pixel tracks, reconstructed with the default set of parameters (in gray), the Single Muon optimized set (in green), and the tuned working point (in light blue).

underscores the need to develop strategies to reduce the number of objective function evaluations required to converge to high-quality solutions, such as the reinforcement learning approach described in Section 4.3, and to improve reconstruction throughput.

Additional work in the NextGen project will focus on efficiency improvements to make automated optimization practical in operational workflows.

5.1.6 Discussion

The optimization of particle track reconstruction currently focuses exclusively on efficiency and fake rate as objective functions, even if split in regions as is the case for the NextGen extension. This is mainly due to the fact that the two figures are fast to compute during the reconstruction process and are a robust metric for of the performance of the reconstruction as is shown in validation (See Section 5.1.4). The `patatune` framework proves effective in optimizing the tracking efficiency and fake rate, offering a continuous spectrum of solutions from which an optimal working point can be selected.

However these are not the only measurements of track quality that could potentially be used for optimization. In CMS, the trajectory of a particle at the vertex is described by five parameters: d_0, the transverse distance from the beamspot; z_0 the longitudinal distance from the beamspot; ϕ the angle of the track on the transverse plane; $\cot\theta$ the cotangent of the polar angle; p_T the transverse momentum [34]. The performance of the fit can be measured by evaluating the resolution of these parameters with respect to the simulated tracks. This is done by computing the distribution of the track residuals and measuring the half-width of the interval that contains 68% or 90% of all entries, centered on the most probable value of the residuals [135]. For optimization with `patatune`, an additional objective based on the residual widths of the track parameters could be considered. As a first approximation, however, this metric was not considered since, for track reconstruction, it is more important to efficiently reconstruct particles while maintaining a low fake rate. Moreover, the track resolution can only be computed after the fitting step, which is not required when evaluating efficiency and fake rate alone, and would therefore introduce additional computational cost during optimization.

Timing is another metric that could be an interesting target for optimization. For track reconstruction, timing is measured over the full detector reconstruction. In this work, it was not considered as an optimization metric, since performing the complete reconstruction would lead to significantly longer computing times. Moreover, it has been shown that the reconstruction time is strongly correlated with the fake rate, as a lower number of fake tracks results in reduced computing time for the full track reconstruction. To include timing as an objective function, careful consideration should be taken to ensure that only the pixel track reconstruction timing is considered in the execution, and there is no strong correlation with other optimization metrics.

The core pixel track reconstruction algorithm was developed specifically for the CMS detector. However, the software was designed with minimal external dependencies. To evaluate its efficacy and performance across a range of heterogeneous platforms, a dedicated project fork was created to operate independently of the standard CMSSW software framework and to support Open Data formats, as described in [136].

To explore the potential of the `patatune` framework in adapting existing algorithms for different environments, we used it to optimize the track reconstruction parameters in the standalone version of the Patatrack pixel track reconstruction software for an alternative detector. Specifically, we performed a test using the simulated detector from the *TrackML challenge* that will be presented in the next section.

5.2 TrackML

TrackML is a ML challenge organized by a collaborative team from various LHC experiments aimed to find novel algorithms with high accuracy and throughput for particle track reconstruction [137]. The competition was hosted on *Kaggle*, an online platform that organizes data science challenges and provides a shared environment where participants can submit their models and compare results on public leaderboards [138].

TrackML provides a Monte Carlo simulated dataset of top quark pair production from proton-proton collisions in an environment that replicates the challenges encountered in reconstructing particle tracks during the HL-LHC phase. The TrackML detector simulates a realistic detector model based on the CMS and ATLAS trackers, featuring three concentric cylindrical layers of all-silicon pixel detectors (barrel) and six circular disks (endcaps) that cover a 4π solid angle of hits (Fig. 5.18). The innermost detector is a pixel detector with a spatial resolution of 50 μm $\times$ 50 μm with modules placed to overlap and cover up to $\eta = 3$. To simulate high pileup conditions, the data are enriched with noise from various low-momentum particle interactions.

The data are structured in independent event files with a specific event identifier. For each event, four files are provided in `csv` format, providing information on hits (x,y,z coordinates in the detector geometry and a unique hit identifier), directional information of the tracking particle, and the unique identifier of the particle that created the hit (with id 0 for noise).

5.2.1 Parameter tuning

The MOPSO algorithm is applied to tune the Patatrack pixel track reconstruction algorithm's parameters, using efficiency, and fake and duplicate rates as objective functions. The input data for the reconstruction are obtained from the TrackML dataset, utilizing 100 events for parameter tuning and 5000 events for performance evaluation.

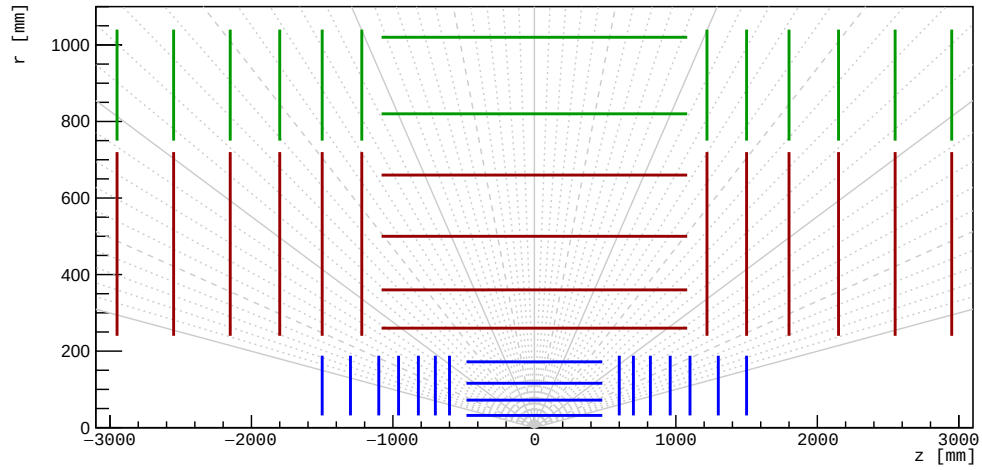

Figure 5.18: TrackML detector schematic showing the top half of the detector projected on the $r - z$ plane.

Since the TrackML dataset already provides hit coordinates data in the detector geometry, the clustering process used to find the hit position from the sensor data is not required. The data are directly loaded as hits and processed to find n-tuples and fit tracks. Together with the hit data, it is populated with the truth values of the originating particle (particle id, p_{T}, dR, v_z, and number of hits), as well as the layer index of the detector and the module index that maps the hit to a volume layer in the detector.

The track reconstruction is performed only on particles with at least four hits in the barrel and endcap layers of the detector, and p_{T} greater than 0.9. The parameters tuned in the optimization are 26 in total and consists of: the cut on the theta angle in the barrel (*CAThetaCutBarrel*) and forward region of the detector (*CAThetaCutForward*), the cut on the Distance of Closest Approach for particles in the inner (*dcaCutInnerTriplet*) and outer (*dcaCutOuterTriplet*) layers of the detector, the cut-off on the curvature of the tracks (*hardCurvCut*), and the angular cut offs for the azimuthal angle around the beam axis on the different modules of the detector (21 different *phiCuts*) as described in Section 5.1.1. Table 5.2 shows the lower and upper bounds of the parameters that the swarm can explore. All parameters are treated as continuous, while *phiCuts* are discrete.

The efficiency is calculated as the total number of associated tracks across all events over the total number of simulated tracks (Equation (5.1)). The number of simulated tracks is determined by the number of unique particle ids, which must have at least four hits and a p_T value greater than 0.9, in the event. The number of associated tracks is calculated as the number of simulated tracks which has been associated with at least

Table 5.2: Parameters' ranges used for the optimization

Parameter	Lower Bound	Upper Bound
CAThetaCutBarrel	0.0	0.003
CAThetaCutForward	0.0	0.005
dcaCutInnerTriplet	0.0	0.2
dcaCutOuterTriplet	0.0	1.0
hardCurvCut	0.03	0.09
21 values of phiCuts	400	1000

one reconstructed track (*sim to reco*). For a particle track to be associated, more than 75% of the hits in the track must have the same particle id.

The fake and duplicate rate is computed as the sum of fakes and duplicates across all events divided by the total number of reconstructed tracks. The number of fakes is the number of reconstructed tracks that are not associated with any simulated track (Equation (5.2)). The number of duplicates is the number of reconstructed tracks that are associated with the same simulated tracks ($N_{ass(reco \to sim)} - N_{ass(sim \to reco)}$).

The choice of the personal best is made in a deterministic manner, selecting the first non-dominated solution found by the single particle and replacing it only when a new solution dominates the previously found one. This promotes elitism in local search. The global best is chosen randomly for each particle across all non-dominated solutions in the archive, to promote exploration.

The resulting Pareto front is shown in Figure 5.19.

5.2.2 Results

From the archive of optimal solutions, two points in the search space were selected for comparison with the default parameter set of the Patatrack pixel track reconstruction algorithm. The first was chosen because it achieved an efficiency comparable to the default configuration but with a significantly lower fake rate. The second corresponds to the point at which the optimization algorithm achieves its highest efficiency.

The tuned parameters were validated by measuring efficiency, fake rate, and duplicate rate as functions of the reconstructed tracks' p_{T}, η, and Φ. Validation was performed on 5000 events from the TrackML dataset that were not used during optimization. The results are presented in Figures 5.20 to 5.22.

The plots exhibit behavior similar to that observed for pixel track reconstruction on CMS data, confirming the Patatrack algorithm's capability to operate effectively on a different detector geometry.

On this geometry, the overall efficiency of the algorithm is lower, while the fake rate is considerably reduced. The tuned parameters display the expected behavior.

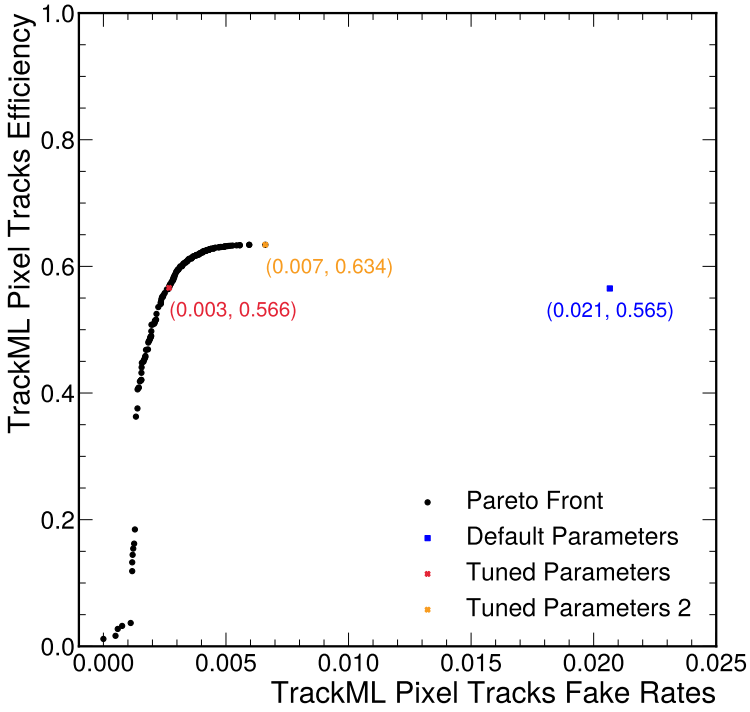

Figure 5.19: The Pareto front (in black) obtained by running the optimization. The blue square dot is the result obtained with the default Patatrack reconstruction parameters. The red cross marks the tuned parameters at the same level of efficiency. The yellow cross shows the parameters for the highest level of efficiency reached by the algorithm.

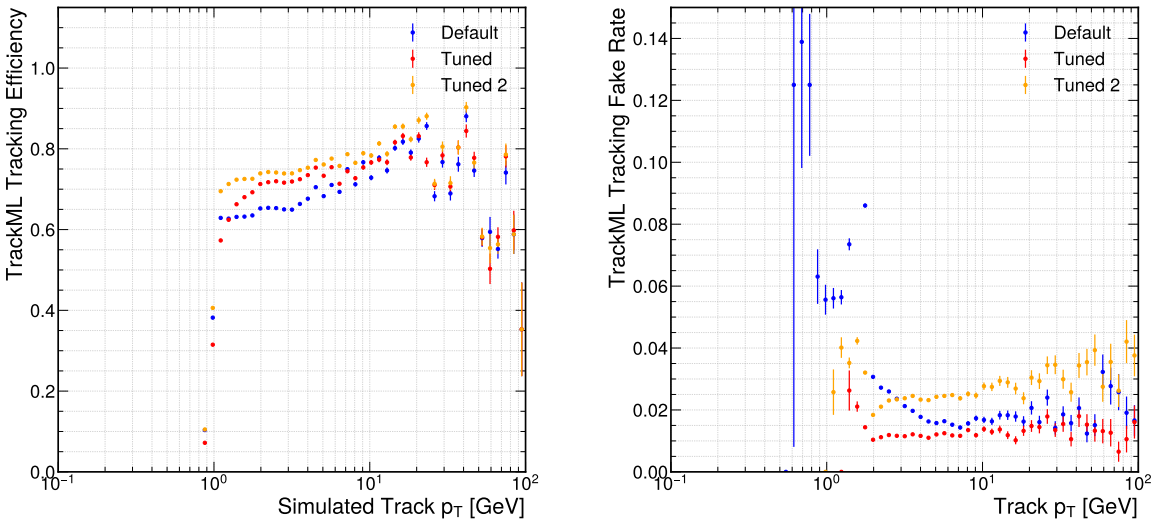

Figure 5.20: On the left, the TrackML pixel tracking efficiency as a function of the simulated track p_T. On the right, the TrackML pixel tracking fake rate as a function of the reconstructed track p_T. In blue, the results for default parameters, in red for the tuned one with the same level of efficiency, and in yellow, the tuned one with the highest efficiency.

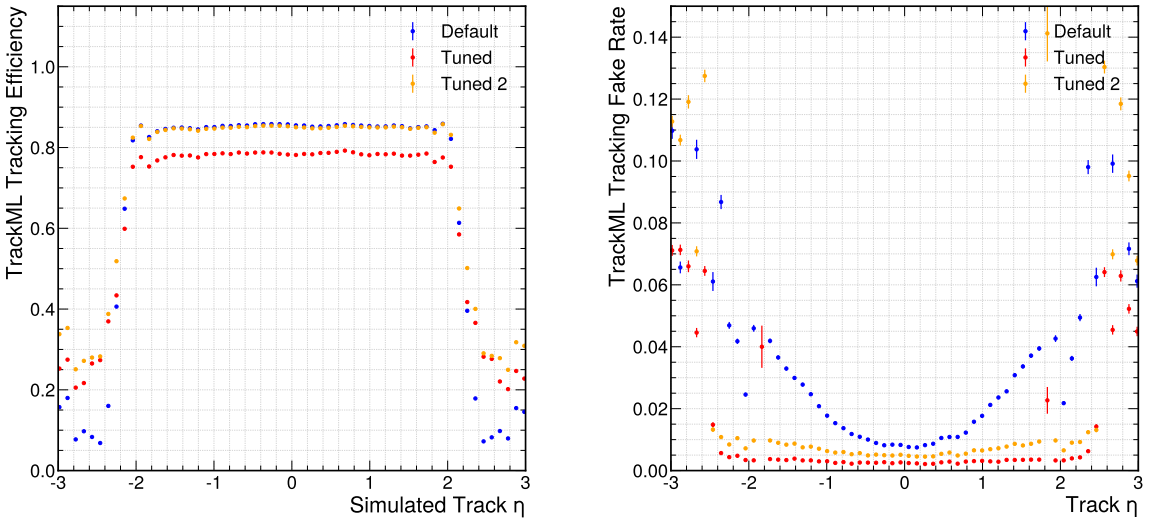

Figure 5.21: On the left, the TrackML pixel tracking efficiency as a function of the simulated track pseudorapidity η. On the right, the TrackML pixel tracking fake rate as a function of the reconstructed track pseudorapidity η. In blue, the results for default parameters, in red for the tuned one with the same level of efficiency, and in yellow, the tuned one with the highest efficiency.

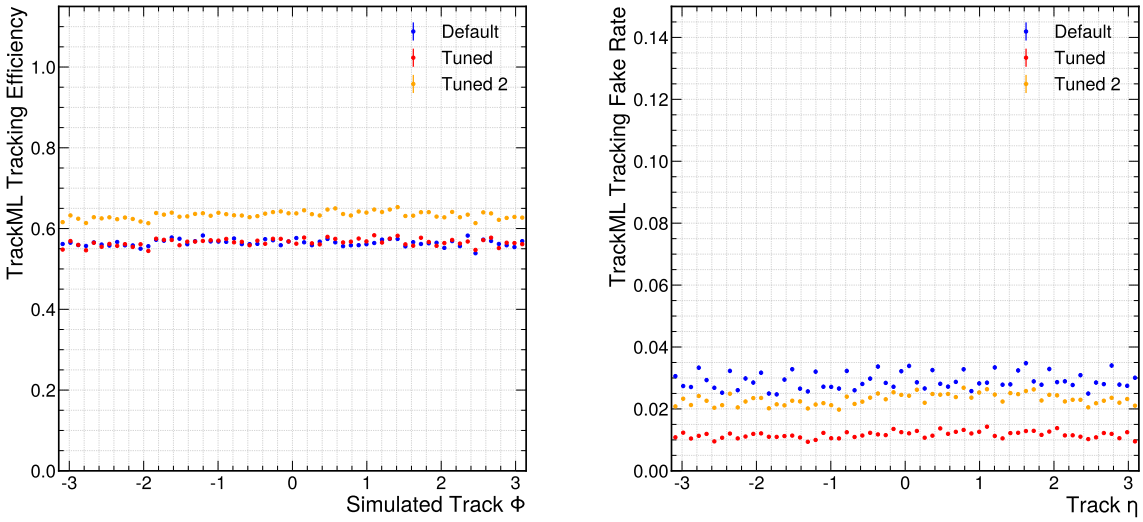

Figure 5.22: On the left, the TrackML pixel tracking efficiency as a function of the simulated track p_{T}. On the right, the TrackML pixel tracking fake rate as a function of the reconstructed track azimuthal angle ϕ. In blue, the results for default parameters, in red for the tuned one with the same level of efficiency, and in yellow, the tuned one with the highest efficiency.

The first selected point, corresponding to a configuration with efficiency comparable to the default, achieves a substantially lower fake rate. In Figure 5.21, the efficiency plot shows reduced performance in the central pseudorapidity region, compensated by higher efficiency for $|\eta| > 2$. Meanwhile, the performance is comparable across ϕ and p_{T} (Figures 5.20 and 5.22).

The second point achieves higher efficiency across all observables. In Figure 5.21, it performs similarly in the central pseudorapidity region but shows a clear improvement elsewhere. It also provides enhanced efficiency as functions of both p_{T} and ϕ. In both tuned configurations, the fake rate is reduced, with the first point achieving the lowest values, as expected.

These results demonstrate the value of multi-objective optimization in identifying trade-offs and guiding the selection of parameters best suited to a specific use case.

The TrackML exercise also demonstrated the portability of `patatune`. Without requiring modifications to the algorithm itself, the framework successfully optimized reconstruction on a new detector description and event topology. This flexibility highlights the potential of the approach as a versatile tool for algorithm development and validation across various detector designs.

5.3 Summary

This chapter addressed the optimization of track reconstruction, focusing on the CMS pixel track reconstruction algorithm and its adaptation to different detector geometries. Track reconstruction is one of the most crucial and computationally demanding components of event processing at the HL-LHC, where extreme pileup conditions result in large numbers of hits per event.

The chapter began with a detailed overview of the Patatrack pixel track reconstruction algorithm, developed within CMS to exploit GPU acceleration. A large number of tunable parameters control each step of the algorithm. Using the `patatune` framework with the MOPSO algorithm, 25 parameters were optimized with respect to efficiency and fake rate in simulated CMS events. The resulting Pareto front provided a set of trade-offs, from which a new working point was selected. Compared to the default setup, the tuned parameters yielded a reduction in the fake rate of approximately 50%, while preserving efficiency.

To show the generality of the method, the optimization was repeated on the TrackML simulated detector. The resulting Pareto front demonstrated that the algorithm could be effectively tuned for a significantly different detector geometry, maintaining high efficiency while substantially reducing fake and duplicate rates.

Overall, this part demonstrated the capability of `patatune` to optimize complex reconstruction algorithms in a multi-objective context. This establishes the framework as a valuable tool for both current and future particle tracking challenges at the HL-LHC and in the context of the NextGen project.

Part II

Reduction of the RAW Data Size

Chapter 6

Data compression

As presented in the previous parts of this thesis, the challenge for the HLT of the CMS experiment in the HL-LHC upgrade will derive primarily from the increase in data volume resulting from higher pileup. The objective of the NextGen project is not only to manage this increase while extending the acceptance rate to maximize the data available for analysis, but also to handle the increased volume of data to be safely stored for further processing in the Tier-0 system.

With the increase in luminosity, event sizes are estimated to reach the order of 10 MB per event [6], which is roughly one order of magnitude larger than the current Run 3 data. Given the increased event rates discussed in Chapter 2, it is not realistic to store all the data in its RAW format, as a rate of 750 kHz would result in approximately 100 EB per year.

In this context, the objective of Task 3.3 of the NextGen project is to reduce the size of the RAW data stored by the experiment to handle the increase in the maximum trigger rate. The project proposes the exploration of three distinct data compression solutions.

One solution is to replace raw data with high-level physics objects (i.e., muons, electrons, jets), similar to the CMS scouting data format [139]. This type of strategy can achieve a compression ratio on the order of 1/100, which would be necessary to store all 750 kHz of data, but with limited possibility of re-performing the reconstruction when newer algorithms or calibrations become available.

Another solution is to replace RAW data with low-level physics objects (i.e., replacing tracker or calorimeter sensor data with the reconstructed particle hit positions and energies), similar to the RAW' method [140] currently used in lead-lead collisions in CMS. This technique achieves a more limited compression factor, typically less than 10, but it preserves the possibility of re-reconstructing high-level objects with newer algorithms or calibrations.

A third approach would focus on lossless compression. It would require testing new compression algorithms and potentially offloading them to accelerators to increase throughput. This approach may also involve the use of reconstructed physics objects (e.g., tracks) to enhance the information available for lossless compression algorithms. A similar approach has been explored for compressing the Time Projection Chamber data used by the ALICE experiment [141].

This part of the thesis presents the preliminary work undertaken during the first 18 months of the project to address the challenge. This first chapter provides an introduction to data compression from the perspectives of information theory. Chapter 7 presents a characterization of the RAW data from Run 3 and the simulations for Phase-2, including estimations of event data size. Chapter 8 focuses on the compression benchmarks used to test the current solutions, covering both algorithms and data formats, as well as tests with lossy solutions.

6.1 Fundamentals of data compression

Data compression is a fundamental concept in computer science and information theory, referring to the process of encoding information using fewer bits than the original representation [142]. The goal of compression is to reduce the size of data for more efficient storage, faster transmission, and reduced resource consumption in computing systems. With the growing volume of digital data across domains, compression is fundamental to ensure scalability and efficiency across multiple research fields, from scientific computing and multimedia to cloud storage and ML.

In scientific computing, compression plays a crucial role when managing the vast volumes of data generated by large-scale experiments, such as CMS at CERN. In this context, it introduces a stringent requirement: the reconstructed data must still allow for the same level of accuracy in subsequent physics analyses as would be obtained from the original uncompressed data.

Compression can be broadly divided into two different categories: *lossless* compression and *lossy* compression.

Definition 6.1.1 (Compression Algorithm)**.** A *compression algorithm* is a function

$$C : X \to X_c$$

that maps an input dataset X to its compressed representation X_c.

A corresponding *reconstruction* (or *decompression*) algorithm is defined as:

$$R : X_c \to Y$$

which reconstructs Y from the compressed data X_c.

If $Y = X$, the compression is said to be *lossless*; otherwise, it is *lossy*.

In general, the representation X_c requires fewer bits to store the information contained in X, and lossy compression algorithms generally provide higher compression than their lossless counterparts.

A compression algorithm can be evaluated in several different ways. One could assess the relative complexity of the algorithm, the memory requirements for its execution, the throughput of the algorithm on a specified machine, the compression achieved, and the fidelity of the reconstruction in relation to the original data. To assess the amount of compression for a set of data, we can look at the ratio of the number of bits in X_c compared to the number of bits in X. In lossless compression, the data can be perfectly reconstructed after decompression, whereas in lossy compression, the efficiency of a compression algorithm can be determined by quantifying the difference between Y and X. This difference is often referred to as *distortion*. When differences in the reconstruction are difficult to model mathematically (as is the case for image or video compression), the terms *fidelity* and *quality* are also used to describe the differences between the reconstruction and the original [142]. These concepts, however, often rely on subjective metrics and are not formally defined.

At the core of any compression method lies the observation that most real-world data contains a significant amount of redundancy. Redundancy arises from patterns, correlations, or repeated structures within the data that do not contribute new information. For example, texts can contain frequent letters or predictable sequences of words, while sensor data and scientific measurements can exhibit correlations between consecutive values or long sequences of repeated digits. Compression algorithms can exploit such regularities by representing those patterns in a more compact form. The more redundancy present in the data, the greater the potential for compression.

From a computational perspective, compression is the process of encoding information into binary representations that capture only the essential content of the data. Efficient compression is achieved when the representation minimizes the total number of bits used while preserving the information necessary for reconstruction.

Information Theory provides a precise mathematical definition of information and a limit on how well data can be compressed. The following section provides a brief overview of the topic; more detailed descriptions are available in the references.

6.1.1 Shannon's entropy

Introduced by Claude Shannon in 1948 [143], Information Theory establishes the mathematical foundations that describe how information can be quantified, transmitted, and compressed. At its core lies the concept of *information content*, which measures the

reduction in uncertainty that occurs when a particular event happens. Intuitively, a highly probable event conveys little new information, whereas a rare or unexpected event carries more information.

Definition 6.1.2 (Information Content)**.** Let a discrete random variable X represent a random event with possible values x occurring with probability $p_X(x) = P(X = x)$. The *information content* (also known as *self-information*) of x is:

$$I_X(x) = -\log_2 [p_X(x)] = log_2 \left[\frac{1}{p_X(x)}\right]$$

where the logarithm is in base 2, so that information is measured in bits.

The use of the logarithm yields an intuitive result: it associates a low probability with high information (since $-log(x)$ increases as x decreases from 1 to 0) and a high probability with low information (since $log(1) = 0$).

Shannon also introduces the concept of *entropy* that measures the expected amount of information needed to describe the state of the discrete random variable, considering the distribution of probabilities across all potential outcomes:

Definition 6.1.3 (Shannon Entropy)**.** The *entropy* of a discrete random variable X is the expected information content of its outcomes:

$$H(X) = -\sum_{x \in X} p_X(x) \log p_X(x).$$

The *source coding theorem*, demonstrated in [143], establishes the statistical limits to possible data compression for data where the information can be described as independent identically distributed (i.i.d.) random variables, and gives operational meaning to the Shannon entropy.

Theorem 6.1.1 (Source Coding Theorem)**.** Let $X_1, X_2, \ldots, X_n$ be n i.i.d. random variables X with entropy $H(X)$.

- (**Direct part**) For any $\varepsilon > 0$, there exists an n_0 such that for all $n \geq n_0$, it is possible to represent the sequence $(X_1, \ldots, X_n)$ using at most $n\big(H(X) + \varepsilon\big)$ bits on average.
- (**Converse part**) Conversely, if the sequence $(X_1, \ldots, X_n)$ is represented using fewer than $nH(X)$ bits, then as $n \to \infty$, the probability of perfectly reconstructing the sequence approaches zero.

This establishes the theoretical limit for any lossless compression algorithm in the case where no correlation exists between the samples. In real-world scenarios, however,

data rarely satisfies this strict assumption. Most datasets exhibit redundancy, patterns, or predictable structures, which can be exploited to compress the data beyond the limits suggested by Shannon's entropy.

A complementary perspective is offered by Kolmogorov complexity, which examines compression from an algorithmic perspective [144]. Kolmogorov complexity defines the complexity of a dataset as the length of the shortest computer program that can reproduce it exactly. Unlike entropy, which considers average symbol probabilities, Kolmogorov complexity considers the intrinsic algorithmic structure of the data: sequences with patterns or regularities can be generated by shorter programs, whereas truly random sequences require a program essentially as long as the sequence itself. While Kolmogorov complexity is uncomputable in general, it provides a theoretical benchmark for the best possible compression of any dataset, independent of the probabilistic assumptions that Shannon entropy relies on.

While these concepts describe the theoretical limits of compression, they do not by themselves provide a practical method for encoding the data in a compressed format.

6.2 Simple entropy coding

In Information Theory, *coding* is the mapping of a source dataset to sequences of bits that can be stored or transmitted. The set of binary sequences is called a *code*, and the individual members of the set are called *codewords*.

Defining a code allows us to realize compression in practice by representing data according to its statistical or algorithmic structure. The insights from Shannon and Kolmogorov guide how we design codes: Shannon entropy tells us how short the average codewords can be, while Kolmogorov complexity reminds us that any underlying regularities can potentially be exploited to reduce the total representation length.

When the number of bits in the codeword for a letter is fixed, we talk of *fixed-length code*. An example is the ASCII code, which encodes 128 characters (95 printable numbers, punctuation marks, and letters of the English language, and 33 non-printable control characters) as a value from 0 to 127, stored as a seven-bit integer [145].

To reduce the number of bits required to represent different messages, one can assign a different number of bits to different symbols. By using fewer bits for symbols that occur more frequently, the average number of bits per symbol, called the *rate* of the code, can be minimized. A similar code is called *variable-length code* This principle is used, for example, in Morse code, where letters that appear more often in text are assigned shorter codewords. For example, the letter E is represented by a single dot ($\cdot$), while the letter Z is represented by the longer sequence - - $\cdot\cdot$ [146].

When designing variable-length codes, an important challenge is ensuring that the resulting code can be uniquely decoded. If some codewords share *prefixes*, decoding

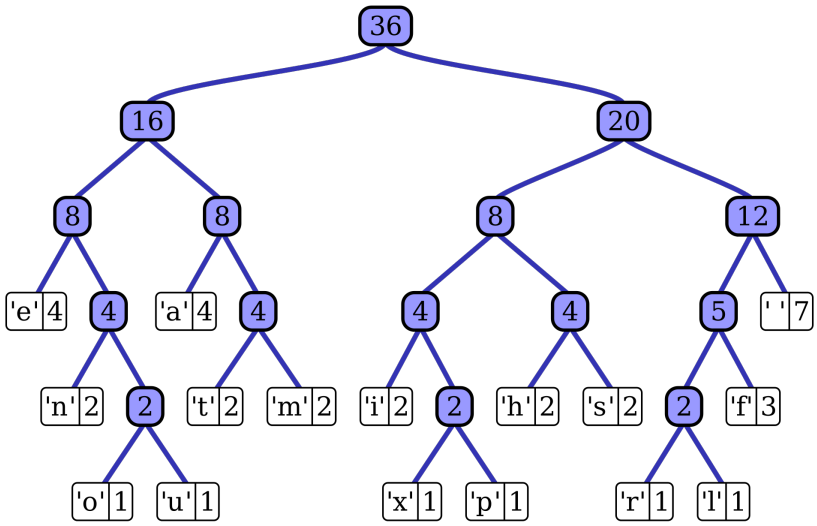

Figure 6.1: Huffman tree generated from the exact frequencies of the text "this is an example of a huffman tree" [147].

can become ambiguous because a sequence of bits might correspond to more than one possible combination of symbols. To verify unique decodability, one examines the so-called *dangling suffixes* that arise when one codeword is a prefix of another. If any of these suffixes is itself a valid codeword, the code is not uniquely decodable. A straightforward way to avoid this problem is to construct a code in which no codeword is the prefix of another. Such codes are known as *prefix codes*.

6.2.1 Huffman coding

The Huffman code is a particular type of optimal prefix code commonly used for lossless data compression [148]. The algorithm generates a code from the estimated probability or frequency of occurrence for each possible value of the source data. As per Shannon's entropy, more common symbols are generally represented using fewer bits than less common symbols.

The technique works by creating a binary tree of nodes, each containing the symbol itself, its frequency of appearance (weight), and a link to a parent node. The process begins by selecting the two nodes with the smallest probability and creating a new node with them as children. The weight of the new node is set to the sum of the weights of the children. The process is then applied again to the new node and the remaining nodes, until only one node remains, which is the root of the Huffman tree. Figure 6.1 shows the tree generated from the text "this is an example of a huffman tree", where the symbols are the characters and the occurrences of each character in the sentence give the weights. Once the Huffman tree has been generated, it is traversed to generate a dictionary that maps symbols to binary codes, where the left child of a

node is typically labeled as 0 and the right as 1. The final encoding of any symbol is then read by concatenating the labels along the path from the root node to the symbol. In the example of Figure 6.1, ‘ ’(the space character) would be encoded with 111, ‘a’ with 010, ’x’ with 10010, etc...

For a set of symbols with uniform probability and a size that is a power of two, Huffman coding reduces to simple fixed-length coding, such as ASCII. When naively applying Huffman coding to binary strings, no compression is achieved. Since there are only two symbols, Huffman encoding assigns 1 bit to each, resulting in a code of the same length as the input. A straightforward way to address this limitation is to group symbols to form a new *alphabet*, where each new symbol represents a block of consecutive original symbols.

Huffman's original algorithm is optimal when the probability distribution of the input symbols is known. Compression algorithms can approximate this optimality for complex alphabets by learning a *model* of the data: a prediction of which pattern of symbols will occur in the data. The more accurate this prediction is, the closer the output will be to optimal.

When the probability distribution is not known in advance, a variation called adaptive Huffman coding can be used [149–151]. The algorithm involves calculating probabilities dynamically based on recent actual frequencies in the sequence of source symbols and adjusting the coding tree structure to match the updated probability estimates. In this implementation, the weight of a node is given by the number of times a symbol is encountered. Neither compression nor reconstruction algorithms know the statistics of the alphabet at the start of operation. They start with the same tree structure, and by following the same updating procedure, the encoding and decoding processes remain synchronized.

6.2.2 Arithmetic coding

An alternative widely used coding scheme is *arithmetic coding* [153]. Arithmetic coding is particularly effective for sources with small or highly skewed alphabets. Unlike Huffman coding, which assigns each symbol its own distinct codeword, arithmetic coding represents the entire message as a single real number in the range $0 \leq q < 1$. The core idea is to progressively narrow an interval based on the probability of each symbol in the message. The encoding process begins with the full interval $[0, 1)$, which is partitioned into sub-intervals whose widths are proportional to the probabilities of the symbols in the current context. Recursively, each sub-interval is itself split into fractions proportional to the probability of each symbol in that interval. Depending on the model, these probabilities are not necessarily the same in each step. Once all symbols have been encoded, any number within the resulting final interval can uniquely represent the entire message.

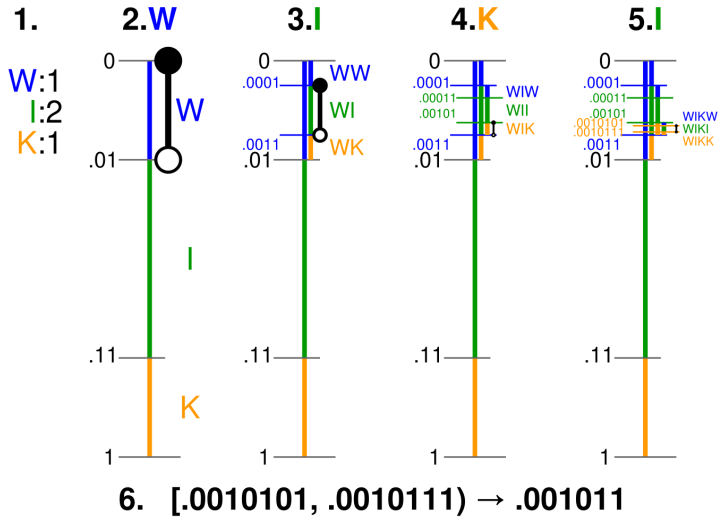

Figure 6.2: Visualization of the arithmetic coding to encode the message "WIKI". The interval $[0, 1)$ is iteratively partitioned based on the frequencies. A value in the final range is chosen to represent the message [152].

Figure 6.2 shows the encoding of the message "WIKI" in arithmetic coding. The initial range is split into three intervals based on the frequencies of the letters W, I and K, arranged from top to bottom. Subsequent sub-intervals correspond to the next letter in the message. The final encoded message can be chosen from any value in the final range.

In *adaptive arithmetic coding*, the probability model can be updated dynamically as the message is processed, rather than remaining fixed throughout encoding [154]. The encoder and decoder both start with the same initial probability distribution, using a uniform distribution when no prior knowledge is available. As each symbol is encoded, both the encoder and decoder update their probability estimates based on the symbols seen so far. The decoded data matches the original data as long as the frequency table used for decoding is replaced in the same way and at the same step as the one used for encoding. This adaptability comes at the cost of additional computational complexity, however there is no need to preserve a tree, as with adaptive Huffman codes, making it a simpler implementation. Moreover, arithmetic coding can achieve compression rates closer to the theoretical entropy limit compared to Huffman encoding.

Entropy coding methods treat the data primarily as a sequence of independent symbols drawn from a probability distribution. While grouping symbols into blocks can capture some local structure, this approach does not exploit long-range redundancies that frequently occur in real-world data.

6.3 Dictionary-based methods

In many practical scenarios, such as with text and structured data, specific sequences of symbols recur frequently throughout the message, while some patterns may appear only rarely. Identifying and exploiting these recurrences can be an effective compression method. A common approach is *dictionary*-based compression, where a list of frequently occurring patterns is maintained alongside the encoded data. The encoded data consists of references to the dictionary when a pattern is present, or is compressed less efficiently when the pattern does not appear in the dictionary. For this technique to be effective, the set of frequently occurring patterns and, consequently, the size of the dictionary must be much smaller than the total number of possible patterns [142].

A *static* dictionary can achieve high compression when sufficient information about the data is known. A static dictionary is a dictionary that remains unchanged throughout the encoding process and whose complete code is determined before coding begins. A static dictionary can be reused across data sources that share similar structures, eliminating the need to transmit it along with the encoded data.

Adaptive dictionaries, instead, allow the same method to compress different types of data without prior knowledge of their specific structure. Most adaptive dictionary-based techniques are based on the Lempel-Ziv family of algorithms [155, 156]. The two papers present two different approaches to the problem, known commonly as LZ77 and LZ78. These algorithms serve as the foundation for many variations, such as LZW [157], LZSS[158], and LZMA[159]. Adaptive dictionaries require the dictionary to be encoded alongside the data, so it can be read by the decoding process to reconstruct the original information.

6.3.1 LZ77 approach

The LZ77 algorithm works by maintaining a dictionary of previously seen data patterns, using a *sliding window* approach to identify repeated sequences. This sliding window is divided into two parts: the *search buffer*, which contains the previously seen data, and the *look-ahead buffer*, which contains the upcoming data to be compressed. The encoder searches for matches between the look-ahead buffer and the search buffer to identify repetitions within the data.

When a match is found, the algorithm outputs a token that specifies the distance to the previous occurrence (the *offset*), the *match-length*, and the next symbol in the look-ahead buffer. To detect these matches, the encoder must keep track of a certain amount of recently processed data. Typically, the last 2 KB, 4 KB, or 32 KB are stored within the sliding window. This structure gives LZ77 its alternative name of *sliding-window compression*.

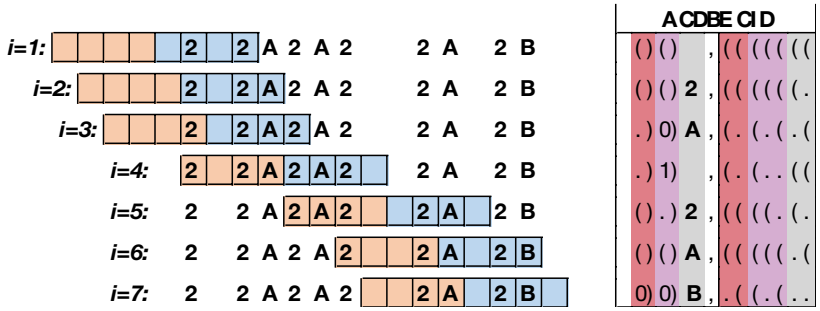

Figure 6.3: Diagram showing an example of the LZ77 encoding process. Image based on [160].

Figure 6.3 shows an example of encoding the word "ABABCBCBAABCABe" with LZ77. For every index, the table shows the position, the match length, and the next symbol in the look-ahead buffer. The second column shows the corresponding binary codeword. The next symbol in the buffer is stored to address the situation where no match is found in the search buffer. In this case, the offset and length values are 0 and the third element of the triple is the code for the symbol itself.

The downside of the LZ77 approach is that it assumes recurrence occurs close to the look-ahead buffer. As such, any pattern that recurs over a period longer than the sliding window will not be captured. Moreover, if the search buffer is even one symbol shorter than the sequence to match, none of the new symbols will be matched and will have to be represented by separate codewords.

6.3.2 LZ78 approach

The LZ78 approach solves the issues of LZ77 by building an explicit dictionary, instead of relying on the search buffer. The dictionary is constructed by coding the entire input into a tuple, containing the index of the previous longest match and the code for the symbol following the matched pattern. If a match to a character is not found in the dictionary, a new dictionary entry is created. The character is then stored in the created entry together with the index 0. After the process is completed, each tuple is converted into a binary bitstream.

Figure 6.4 shows an example of the procedure for the word "AABABCAABBAC".

While the LZ78 algorithm solves the shortcomings of the LZ77 approach, it presents a serious drawback. By capturing patterns dynamically and holding them, it creates a situation where the dictionary can grow without bounds. In a practical situation, the dictionary cannot grow indefinitely, and the rest of the data must be encoded without searching for new patterns, treating the dictionary as a static dictionary.

Many popular compression methods emply the LZ family of encoders. The DEFLATE compression algorithm combines the LZSS variation with Huffman coding [162]:

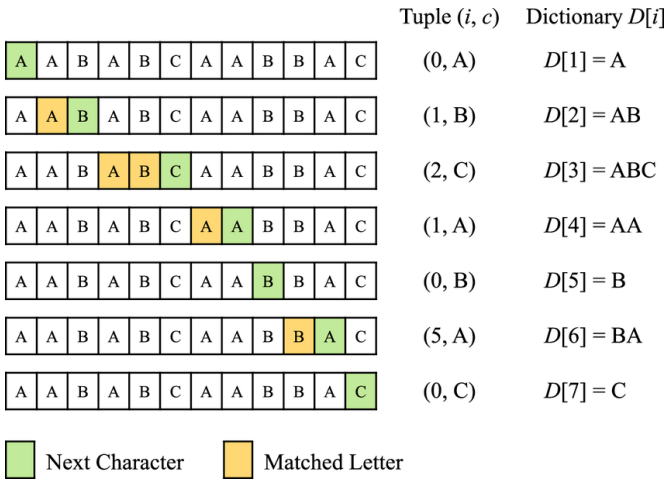

Figure 6.4: Diagram showing an example of the LZ78 encoding process [161].

The algorithm divides the input data into a series of blocks; The LZ algorithm identifies duplicate strings and replaces them with a pointer to their first occurrence, followed by the length of the string; Huffman coding then performs bit reduction by identifying commonly used symbols and replacing them with symbols that have shorter bit sequences within each LZ77 compressed block. The algorithm is used, for example, in the ZLIB compression library [163]. The LZ4 compression algorithm implements a fast LZ77 scheme, focusing on high-speed compression and decompression without additional entropy coding [164]. ZSTD extends LZ77 with longer match distances, fast dictionary matching, and entropy coding based on Finite State Entropy [165]. LZMA uses a very large sliding window LZ77 matcher combined with range coding and probabilistic modeling to achieve very high compression ratios [159]. The compression libraries will be discussed and benchmarked in more detail in Chapter 8.

6.4 Block-sorting

In a different approach to dictionary encoding, techniques like run-length encoding (RLE), move-to-front (MTF), and Quantized Local Frequency Coding (QLFC) represent consecutive occurrences of the same symbol as a single symbol–count pair [166–168]. These techniques are particularly effective when similar symbols are grouped, as this reduces redundancy in long runs. These methods are often used in combination with *block-sorting* algorithms that rearrange the input data, making it more amenable to subsequent transformations [142].

In contrast to dictionary-based methods such as LZ77 or LZ78, which operate in a streaming manner and build their models incrementally as data are read, block-sorting techniques require that the entire sequence be available to the encoder before compression begins. A prominent example of this strategy is the Burrows–Wheeler

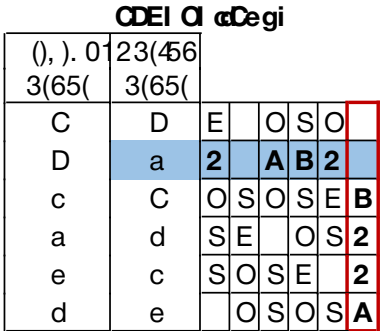

Figure 6.5: Diagram showing the block-sorting process.

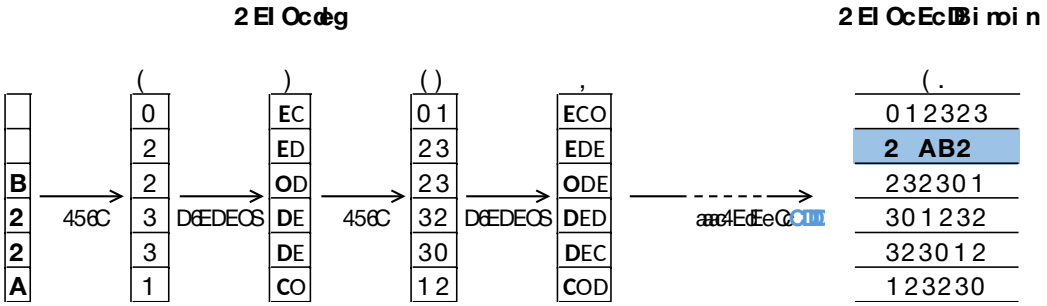

Figure 6.6: Diagram showing the block-sorting reconstruction process.

Transform (BWT), which performs a reversible block-based sorting of the input to expose a pattern in the symbols, which can then be encoded in the following steps to compress the data [169].

The BWT operates on a block of data by generating all its cyclic rotations and sorting these rotations lexicographically. The output is a tuple containing the last column of the resulting matrix along with the index of the original string in the sorted order. This process results in clusters of symbols that appear in similar contexts in the input, leading to long runs that later stages can easily compress. The transformation is fully reversible, and the original sequence can be reconstructed using the index value and the relationship between the first and last columns of the sorted rotations.

Figure 6.5 illustrates the block-sorting process using the input word “DECODE”. Each row represents a cyclic rotation of the original word sorted lexicographically. The column labeled “Original order” gives the index of each rotation before sorting, and the “Sorted order” column shows the order of these rotations after. Once all rotations are sorted, the last column (highlighted in red) forms the transformed output, which in this example is “EEDDOC”. The number 2 indicates the row corresponding to the original string in the sorted list; this index is passed as output and is needed for reconstruction.

Figure 6.6 illustrates the inverse of the block-sorting process. Starting from the transformed sequence, the decoder begins with these characters as the first column. It

then repeatedly sorts the column and adds the transformed sequence as a new leftmost column. This iterative process gradually reconstructs all cyclic rotations of the original word. Once the table is complete, the rotations are fully sorted again. The row indicated by the stored index (row 2, in the picture) corresponds to the original word.

A different transformation, known as the *Sort Transform* (ST) or *Schindler Transform*, was later introduced to address some of the computational drawbacks of the BWT, particularly its uncertain runtime and poor scalability with large blocks [170, 171]. The ST achieves this by sorting the data based on a limited context (typically a few bytes of the input) instead of considering the entire data.

In a typical implementation, an ST of order k, given an N-character source text, orders all the rotated texts derived from it by sorting only by comparing the first k characters, where $k \in \{1, \ldots, N\}$. The result is a permutation of the input symbols that preserves part of the local clustering achieved by the BWT, with significantly faster forward transformation. This makes it suitable for real-time or high-throughput applications such as data streaming or hardware-based compression. The throughput of the encoding, however, highly depends on the block size, and the inverse transform is somewhat slower, making it less viable for scenarios where the dataset is compressed only once and decompressed multiple times.

Both the BWT and ST exemplify the principle of transforming data to expose redundancy rather than encoding it directly. By reordering symbols so that those appearing in similar contexts are grouped, these methods produce sequences that are highly compressible by subsequent encoding stages, forming the foundation of modern compressors such as *bzip2* [172] or *bsc* [173] that will be discussed in Section 8.1.3. The downside of these approaches compared to the dictionary-based method is that they require loading the entire block into memory for compression, which can be computationally intensive for large datasets.

6.5 Summary

This chapter laid the theoretical foundations necessary for understanding data compression in the context of high-energy physics experiments. The challenge faced by CMS during the HL-LHC era requires evaluating different compression strategies and the trade-offs they present.

The compression methods discussed fall into different categories, each with distinct characteristics relevant to our application. Entropy coding methods like Huffman and arithmetic coding are fast and effective when symbol frequencies are skewed. Combined with dictionary-based methods, which excel at finding repeated sequences, they form the backbone of modern fast compressors like LZ4, ZSTD, and LZMA. Block-sorting techniques like the Burrows-Wheeler Transform can instead achieve superior

compression by reorganizing data to expose context-based redundancies. However, they typically require more computational resources.

The choice of compression algorithm for CMS will ultimately depend on several competing factors: the compression ratio achievable, the throughput required to keep pace with the trigger rate, the computational resources available, and the need to preserve data fidelity for physics analysis. Understanding these trade-offs requires not only theoretical knowledge, but also empirical testing on real detector data. The next chapter will characterize the CMS RAW data from both Run 3 and Phase-2 simulations. Chapter 8 will then benchmark the algorithms discussed here on actual CMS data, measuring their compression ratios and throughput to determine which approaches are most promising for development within the NextGen project.

Chapter 7

Characterizing Run 3 and Phase-2 RAW data

As part of the first-year milestones of the NextGen project, Task 3.3 was responsible for delivering a comprehensive report on the contributions of various subdetectors to the overall RAW data size [5]. This report serves as a foundational step for all subsequent work in data compression.

This chapter highlights the most significant contributions to the data size, providing a detailed breakdown by subdetector. By comparing Run 3 data with Phase-2 simulations, we can assess the anticipated increase in data volume and identify key areas for potential intervention. This characterization is crucial for developing effective compression strategies, ensuring that our efforts are focused on the most significant data contributors. The findings from this analysis will guide the design and implementation of the compression algorithms that will be the focus of the project activities in the following years.

Additionally, the report characterizes the potential for data reduction by storing high-level reconstructed objects instead of low-level RAW data. This is compared with estimates from the CMS Phase-2 HLT TDR [6].

The results of this activity are presented in the following sections.

7.1 Methodology

To characterize the data, different datasets have been utilized for Run 3 and Phase-2. To measure the event size, we utilized the CMSSW_14_2_0_pre1 software release [174]. The configurations were executed on the Linux Public Login User Service (LXPLUS)[175], as performance was not a critical metric for this research. The machines on this service are part of the CERN OpenStack Private Cloud, with each node featuring 30 GB of RAM, 10 GB of swap, 8 CPUs, and solid-state disks on the hypervisors.

The measurements were performed using configuration scripts available in the "NGT RAW Size Impact" GitHub repository [176].

For Run 3 data, a custom Front-End Driver (FED) selector, `EvFEDSelector`, was employed to measure the RAW event size contribution from each detector. The FED is a specialized electronic board that collects and formats RAW data from a detector subsystem before sending it to the central data acquisition system. The specific FED numbers for each detector are obtained by assigning a unique identification number to each FED board, allowing for the precise tracking of data origin within the experiment's complex architecture. The data sample used for this analysis consisted of 100 events from proton-proton collisions, specifically from the muon dataset of run 382216 and fill 9804, recorded in June 2024. These events were chosen from luminosity sections with an average pileup of approximately 60. To study overall Run 3 event sizes, we used a CMSSW configuration to produce an output file with RAW data, digitized data (*digis*), reconstructed hits (*rechits*), and *clusters*.

For Phase-2 studies, we run the same configuration on a 14 TeV $t\bar{t}$ sample from the Phase-2 Spring 2024 campaign, using files with PU $\sim$ 200 as the primary input. In addition to RAW data, digis, rechits, and clusters, the *mixTracker* object, which simulates the Outer Tracker data, is included in the output file.

The average size of these objects can be determined with the `edmEventSize` command.

7.2 Data size visualization

`edmEventSize` is a command-line program used to measure the compressed and uncompressed size of each *branch* in a ROOT *tree*.

In ROOT, a tree is a hierarchical data structure designed to store large datasets of events efficiently. Each tree is composed of branches, which group related data fields, and each branch contains *leaves* representing the individual variables or arrays of primitive types actually written to disk. Within CMSSW, a typical ROOT file contains an *Events* tree where each branch corresponds to a specific data product. For example, the branch `recoCaloClusters_hgcalMergeLayerClusters__RECO` stores reconstructed HGCAL layer clusters collections. Inside this branch, the leaves hold the actual quantities, such as the position components, the hits, or the energy for each cluster. In summary, the branch defines the product as a whole, while the leaves represent the individual numerical entries that characterize each event's objects.

Since the script is distributed with every CMSSW release, it is available inside the CMSSW environment. It can be executed as `edmEventSize -v file.root` to print the branch sizes to the terminal or as `edmEventSize -o file.txt file.root` to write the output to a text file. The tool provides options to sort the results alphabetically, adjust

the formatting of product names, and export the measurements as ROOT histograms or as PNG plots. Although developed for CMS event data, it can analyze any ROOT tree by specifying the tree name with the `--tree-name` option, with the understanding that some features are specific to CMS data structures [177].

One limitation of the original tool was the absence of any detailed report on the size and type of individual leaves within a ROOT tree. This level of granularity is important when identifying which variables contribute most to the overall data volume of a stored object. To address this, the script was extended to include an additional option that exposes this information [178].

When executed with the `-L` option, it now reports to the terminal both the compressed and uncompressed size of every leaf in the selected tree, in addition to the usual per-branch summary. This enhancement allows for evaluating directly the relative contribution of each stored variable to the overall event size, making it easier to identify where to intervene in data reduction or restructuring.

The extension also introduces a structured JSON output whose schema is shown in Figure 7.1. This format records, for each branch and its leaves, the measured sizes, along with auxiliary metadata describing the metrics. It provides an interface that can be consumed by downstream tools, while still being human-readable. The JSON schema is compatible with a visualization framework based on the Carrotsearch Circle library [179], which is already employed within CMS to display the resource usage of CMSSW modules in hierarchical pie charts. We adapted that interface to accept the new leaf size metrics, enabling interactive visualizations in which users can explore the contribution of individual leaves or entire branches to a ROOT file. This integration enables the display of how different parts of the event data impact the storage budget.

An example of its use is shown in Figure 7.2, where the highlighted slice in the pie chart corresponds to the size of the `recoCaloCluster` branch. This collection stores information on clusters reconstructed from hits in the HGCAL detector layers, including the x, y, and z coordinates of the cluster barycenter and the cluster energy, as leaves. Traditionally, these quantities are stored in double precision, even though the physical precision of the values would allow the use of smaller data types.

The chart on the left shows the branch with its original double-precision values. The chart on the right illustrates the same branch with position and energy stored as floats, which results in a significantly smaller footprint.

Table 7.1 presents a breakdown of the file size for HGCAL objects, the `recoCaloCluster` object, and its floating-point variant. The compression has been performed with the LZMA algorithm at level 4. More details on the compression algorithm will be discussed in Section 8.1. While the absolute file size is notably smaller for the float version, the compression ratio, defined as $\frac{\text{compressed size}}{\text{uncompressed size}}$ (lower is better), is lower for double-precision data. This occurs because double-precision arrays typically con-

```
{
  "modules": [
     ...
    {
      "events": 10,
      "type": "recoCaloClusters_hgcalMergeLayerClusters__RECO",
      "label": "obj.energy_|Double_t",
      "size_compressed": 93707,
      "size_uncompressed": 233943,
      "ratio": 0.400555
    },
    {
      "events": 10,
      "type": "recoCaloClusters_hgcalMergeLayerClusters__RECO",
      "label": "obj.position_.fCoordinates.fX|Double_t",
      "size_compressed": 92185,
      "size_uncompressed": 234105,
      "ratio": 0.393776
    },
     ...
      ],
  "resources": [
    {
      "name": "size_uncompressed",
      "description": "uncompressed size",
      "unit": "B",
      "title": "Data Size"
    },
    {
      "name": "size_compressed",
      "description": "compressed size",
      "unit": "B",
      "title": "Data Size"
    }
  ],
  "total": {
    "events": 10,
    "size_uncompressed": 281359338,
    "size_compressed": 47726784,
    "ratio": 0.169629
  }
}
```


Figure 7.1: Snippet of the `edmEventSize` script's JSON output, showing the size of the energy and x coordinate position of the `hgcalMergeLayerClusters` object in a ROOT tree.

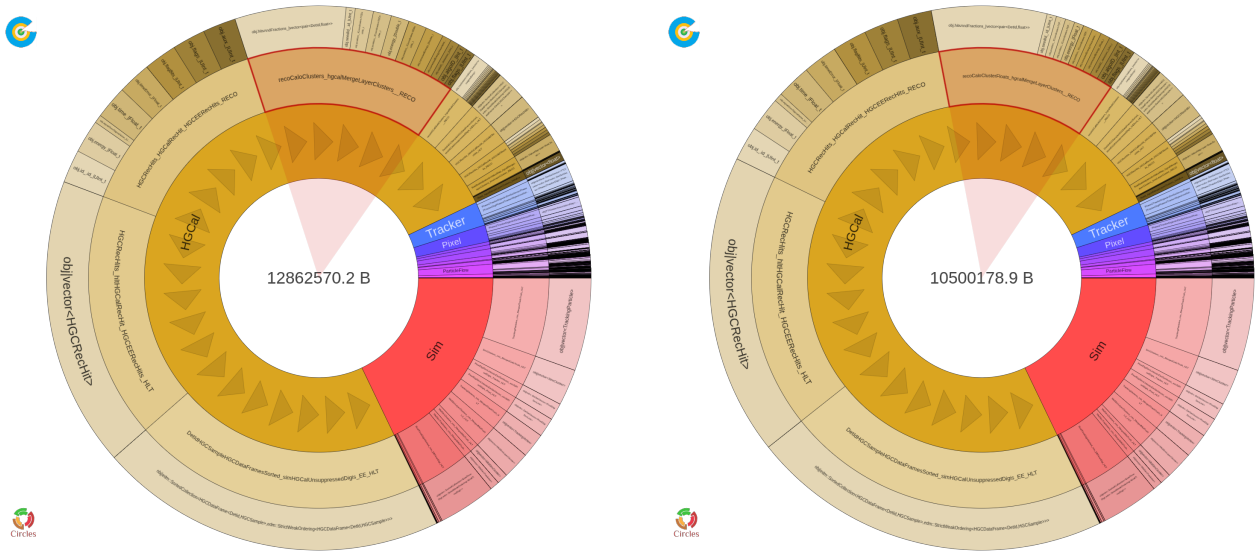

Figure 7.2: Pie charts showing the uncompressed size of each branch and leaf in a ROOT tree for the Monte Carlo Phase-2 simulation of the HLT reconstruction. The expanded portion of the pie corresponds to the HGCAL detector, and the highlighted slice is the `recoCaloCluster` branch. The left chart displays the original format, where position and energy are stored as `double`, while the right chart shows the same object with these quantities stored as `float`.

tain more repeated patterns and higher redundancy in the least significant bits, which compression algorithms exploit more effectively than the smaller, less redundant float representation. Consequently, although storing data as floats reduces the uncompressed size, the relative gain after compression is smaller compared to storing data as doubles.

7.3 RAW event size in Run 3

The measured RAW event sizes for each detector in Run 3 are listed in Tables 7.2 and 7.3. Because the standard module for splitting RAW data by FED could not isolate the HCAL contribution, its size was estimated by subtracting the contributions of the other detectors from the total event size.

These measurements, of the data described in Section 7.1, were compared with results from Run 3 $t\bar{t}$ simulations, which had an average pileup of about 65 (Table 7.4). The compression has been performed with the LZMA algorithm at level 4.

The corresponding fractions are shown as pie charts in Figure 7.3 for both the data and the $t\bar{t}$ sample. The overall agreement between the data and the simulation is good. The main discrepancies appear in the ECAL preshower and HCAL, likely reflecting the higher electron and jet multiplicities in the $t\bar{t}$ simulation relative to the Muon data.

Table 7.1: Breakdown of the file size per event for HGCAL subdetector objects in a Phase-2 Monte Carlo simulation. The values of the `recoCaloCluster` branch in the original double-precision representation are compared with a reduced-size float version, showing both uncompressed and compressed sizes in kilobytes, as well as the compression ratio.

		Average Uncompressed Size (kB/Event)	**Average Compressed Size (kB/Event)**	**Compression Ratio**
Double	HGCAL	642.2	114.7	18%
	recoCaloCluster	125.6	24.0	19%
Float	HGCAL	618.0	113.9	18%
	recoCaloClusterFloats	102.5	23.4	23%

Table 7.2: Average RAW uncompressed and compressed event size per subdetector in Run 3 data [180].

Module Name	**Average Uncompressed Size (kB/Event)**	**Average Compressed Size (kB/Event)**	**Compression Ratio**
RAWDataCollector	1477.180	1054.290	28.6%
partialRAWDataRepackerStrips	759.204	603.656	20.5%
partialRAWDataRepackerPixel	325.380	283.859	12.8%
partialRAWDataRepackerECAL	96.492	29.032	69.9%
partialRAWDataRepackerMuons	78.407	24.289	69.0%
partialRAWDataRepackerES	57.161	19.220	66.4%
partialRAWDataRepackerOther	59.144	5.306	91.0%

Table 7.3: Average RAW uncompressed and compressed event size per subdetector in Run 3 Monte Carlo simulation [180].

Module Name	**Average Uncompressed Size (kB/Event)**	**Average Compressed Size (kB/Event)**	**Compression Ratio**
RAWDataCollector	1997.460	1339.050	33.0%
partialRAWDataRepackerStrips	1006.810	730.825	27.4%
partialRAWDataRepackerPixel	318.142	285.794	10.2%
partialRAWDataRepackerECAL	121.908	36.110	70.4%
partialRAWDataRepackerMuons	121.258	26.406	78.2%
partialRAWDataRepackerES	183.134	104.145	43.1%
partialRAWDataRepackerOther	31.240	2.714	91.3%

Table 7.4: Comparison of the average compressed RAW size per event, per subdetector, for Run 3 data and Monte Carlo simulation [180].

Module	**Run 3 Data Average Compressed Size (kB/Event)**	**Run 3 Monte Carlo Average Compressed Size (kB/Event)**
Strips	603.7	730.8
Pixel	283.9	285.8
ECAL	29.0	36.1
ECAL pre-shower	19.2	104.1
Muons	24.3	26.4
Other	5.3	2.7
HCAL (estimate)	88.9	153.1
RAWDataCollector	**1054.3**	**1339.1**

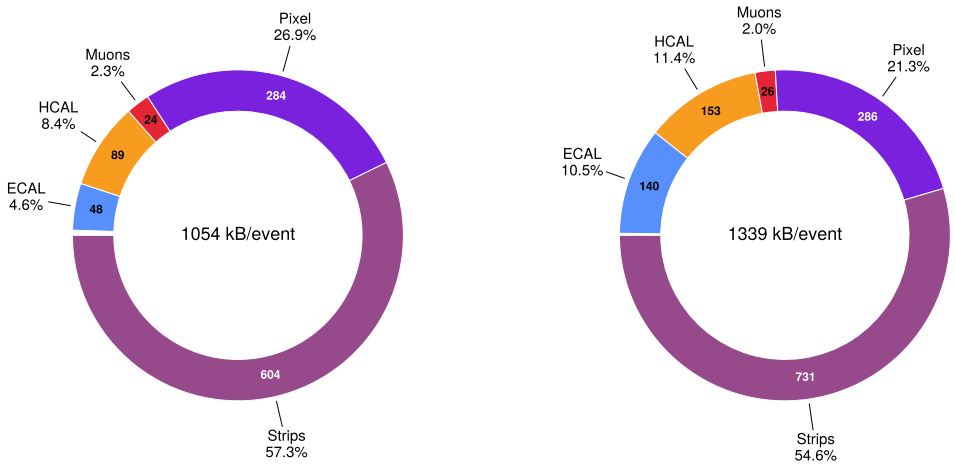

Figure 7.3: RAW percentage per detector in Run 3 data (left) and $t\bar{t}$ simulation (right).

Table 7.5: Comparison of simDigi size per event for Run 3 (PU $\sim$ 60) and Phase-2 (PU $\sim$ 2000), per subdetector [180].

Subdetector	**Run 3 simDigi Size (kB/Event)**	**Phase-2 simDigi Size (kB/Event)**
Strips/Outer Tracker	731	705
Pixel/Inner Tracker	284	1375
ECal (no preshower)	36	66
HCal	153	80
HGCal	–	2705
MTD	–	617
Muon	26	129
Total	**1282**	**5671**

In summary, strips and pixels dominate the RAW data, contributing 57.3% and 26.9% of the event size, respectively. Together they account for over four-fifths of the total, while all other detectors combined contribute less than one-fifth.

7.4 Event size in Phase-2 simulation

In the present Phase-2 samples with PU $\sim$ 200, the RAW data are only about 300 kB per event, far below the expected 100 MB. The full RAW event content for Phase-2 has not yet been defined in simulation. Most Phase-2 workflows currently create only a limited subset of RAW collections, because many detector subsystems have not finalized or enabled their full RAW data formats in the production release. As a result, we use the simulated digitized detector data (*simDigis*), which include all subdetector contributions even when the formal RAW packing is absent, to estimate the eventual RAW size of Phase-2 detectors.

7.4.1 Comparison with Run 3

Table 7.5 compares the average compressed simDigis size per event in Phase-2 with the corresponding Run 3 values. The compression has been performed with the LZMA algorithm at level 4.

For the Phase-2 strip detector, the numbers are taken from the `mix` tracker object (see Appendix B for more details). The HCAL and ECAL endcaps will be replaced by the HGCal detector (see Section 1.3.7), which accounts for the significant size difference between the detectors.

The table shows that HGCal is expected to be the largest single contributor to the Phase-2 RAW size, followed by the Inner Tracker. Interestingly, the Phase-2 Outer

Tracker is projected to produce a smaller event size at pileup 200 than the Run 3 strip detector at pileup 60. This reduction occurs because the Outer Tracker stores only fully digitized hits, without recording the charge deposited in each strip [33].

Appendix B provides a detailed breakdown of the event content contributed by each subdetector in the Phase-2 upgrade. Where relevant, tables highlight key differences between Run 3 and Phase-2 to place these contributions in context.

To assess the potential for lossy compression, low-level reconstructed objects from the detector's local reconstruction have been included in the measurements.

7.4.2 Low-level reconstructed objects

Low-level reconstructed objects provide a practical proxy for estimating how much the RAW event size could be reduced through a lossy compression strategy. Appendix B breaks down the average compressed size per event of the principal low-level objects for each Phase-2 subdetector.

Across the upgraded detectors, these objects are usually smaller than the corresponding simDigis or the expected RAW data.

For the tracking detectors, storing rechits rather than clusters or RAW data reduces the event size by factors ranging from approximately 50% for the strip detector to nearly 90% for the pixel detector.

For the calorimeters, the situation is more varied. The ECAL low-level reconstructed objects (rechits and clusters) are in some cases larger than the simulated RAW data (Tables B.3 and B.4). This suggests that simple replacement of RAW with rechits would not yield compression for these subsystem. The HGCAL presents a different profile. With approximately 6 million channels distributed across 50 sampling layers, the detector generates roughly 2.7 MB of simulated digitization data per event at 200 pileup (Table B.5). Storing uncalibrated rechits reduces this to about 1.1 MB, a 58% reduction, while calibrated rechits occupy 1.6 MB. However, storing only clusters eliminates the possibility of re-reconstructing with improved algorithms or updated calibrations, as the algorithm has already made decisions about which hits to associate and how to compute cluster properties.

The MIP Timing Detector (MTD) shows perhaps the most dramatic potential for size reduction through lossy compression. The raw digitization data total approximately 617 kB per event, while storing tracking rechits that contain only the subset of timing measurements associated with reconstructed tracks occupies just 21 kB, a 96% reduction (Table B.6

Overall, storing such low-level reconstructed objects instead of full RAW content could reduce the Phase-2 average event size by more than half, while preserving the essential physics information needed for most offline reconstruction [180].

Table 7.6: Comparison of Phase-2 event size estimate in HLT TDR and simulations [180].

Subsystem	**HLT TDR Est. Size (MB/Event)**		**Phase-2 Simulation Size (MB/Event)**
Inner Tracker	1.44	1.44	1.38
Outer Tracker - PS	0.72	1.15	0.70
Outer Tracker - 2S	0.43		
MIP Timing Det. - BTL	0.24	0.68	0.62
MIP Timing Det. - ETL	0.44		
ECAL Barrel	0.60	0.60	0.66
HCAL Barrel	0.24	0.33	0.80
HCAL HO	0.03		
HCAL HF	0.06		
HGCAL	3.00	3.00	2.71
Muon DT	0.15	0.76	0.13
Muon CSC	0.47		
Muon GEM - GE1/1	0.00		
Muon GEM - GE2/1	0.00		
Muon GEM - ME0	0.12		
Muon RPC	0.01		
Total	**8.42**	**7.96**	**5.65**

7.4.3 Comparison with HLT-TDR estimates

Table 7.6 shows the estimated RAW event size for each subsystem according to the HLT TDR [6] from 2021, compared with the Phase-2 simulation data. The total Phase-2 event size from the simulation is 5.65MB per event, whereas the HLT TDR estimated 7.96MB.

Part of the larger estimate comes from the Level-1 (L1) trigger contribution, which accounts for approximately 0.47MB per event. This includes the Track Finder Trigger Primitive (about 0.01MB), the HGCAL Trigger Primitive Stage 1 (0.15MB), Stage 2 (0.05MB), and the Level-1 Trigger itself (0.26MB).

These contributions are not present in the Phase-2 simulation data. Excluding Trigger Primitive information would therefore yield a significant reduction in event size, particularly for the HGCAL detector [180].

7.5 Summary

This chapter provides a detailed characterization of Run 3 RAW event content, as well as a study of the Phase-2 data that can serve as a reliable proxy for the full RAW collections that are not yet available in the current simulations. By comparing subdetector contributions between Run 3 and Phase-2, and analyzing simulated digi-

tization (*simDigis*) and low-level reconstructed objects, we identified the objects that most closely approximate the eventual RAW data.

The study of low-level reconstructed objects shows that significant data volume reductions could be achieved by storing, for example, *rechits* or *clusters* instead of the full RAW content. These results highlight both the potential and limitations of using reconstructed data as a proxy for RAW data, emphasizing that effectiveness depends on the detector and the level of granularity.

Finally, the overall event sizes obtained in the Phase-2 simulations are in good agreement with the HLT TDR estimates. Differences, such as the absence of Level-1 trigger contributions in the simulations, are understood and do not affect the validity of using these data for subsequent studies. Altogether, this analysis confirms that the available Phase-2 datasets provide a realistic approximation of the RAW content, forming a solid foundation for the compression benchmarks discussed in the next chapter.

Chapter 8

Compression benchmarks on CMS data

Each compression approach investigated by task 3.3 of the NextGen project presents different trade-offs between compression ratio and the information available for reprocessing. The overall goal of the task is to achieve a significant reduction in RAW data size by considering both lossy and lossless algorithms, as well as by exploring physics-driven compression strategies that replace low-level detector information with higher-level quantities derived from it.

In this chapter, we present the activities carried out during the first year of the project to achieve this objective. We first discuss benchmarks of lossless data compression techniques already available in the ROOT software framework. These benchmarks serve a dual purpose: on the one hand, to evaluate the performance of existing compression models on Run 3 data and on Phase-2 simulations, which provide a reliable approximation of the missing RAW samples as presented in the previous chapter. On the other hand, they allow us to assess the effectiveness of compression on the new experimental RNTuple data format. RNTuple is expected to enter production towards the end of 2025, representing a key opportunity to improve storage efficiency while providing compatibility with modern data access patterns [181].

Section 8.1.3 presents results on using GPU-accelerated compression algorithms on the same data. Some promising results have already been achieved, which will need to be further explored in the continuation of this work.

The chapter will end with preliminary work on lossy compression strategies. In particular, Section 8.2 presents studies based on applying low-level reconstruction techniques to strip detector data, inspired by similar efforts already carried out by the Heavy Ion group in the context of the RAW' project. This line of research will be further expanded in the coming years.

8.1 Performance comparison of lossless compression algorithms

Efficient lossless compression is fundamental for storing RAW data from the CMS experiment while preserving accurate physics performance. The following sections report on the performance of various lossless compression algorithms for CMS RAW data in Run 3 and Phase-2 simulations, comparing the traditional ROOT *TTree* data format with the newer *RNTuple* format [181]. Benchmarks include measurements of read time, compression ratio, and throughput.

All benchmarks were executed on a cluster of nodes made available for the NextGen project, and managed through a Kubernetes orchestrator [182]. The nodes used for benchmarking compression within CMSSW provided, for each job, an AMD EPYC 9534 CPU (64 cores, 2.45 GHz) with 180 GB RAM. Each compression job was executed via a GitLab CI/CD pipeline, which orchestrated the deployment of different combinations of input files, algorithms, and compression levels. The repository containing the CMSSW configuration and the code required to generate the plots of the results is publicly available on the CERN GitLab instance [183].

8.1.1 Benchmarks with Run 3 RAW data

Input files were generated from a proton-proton collision dataset consisting of 6951 events ($\sim$ 140 s of data taking) from Run 382229 of Fill 9806 in June 2024. The dataset corresponds to fully hadronic triggers at 13.6 TeV with an average pileup of ~ 64.5.

ROOT files with different numbers of events were produced in both the TTree and RNTuple formats. The output was generated using the `PoolOutputModule` and the `RNTupleOutputModule` from the experimental `CMSSW_15_1_RNTUPLE_X` integration build [184], based on ROOT version 6.35.99, saving only the `FEDRawDataCollection` object: the collection of all RAW data corresponding to each subsystem's FED (see Section 7.1).

To ensure an accurate measurement of the compression ratio, both output modules were modified to output the original file without compression. This was necessary because, by default, an *uncompressed* option is not available in CMSSW, since a compression algorithm must always be explicitly specified in the output module's configuration.

Read timing

Figure 8.1 shows the time required to read uncompressed files. TTree format files (blue) and RNTuple format files (orange) were generated for various event counts (1, 10, 100,

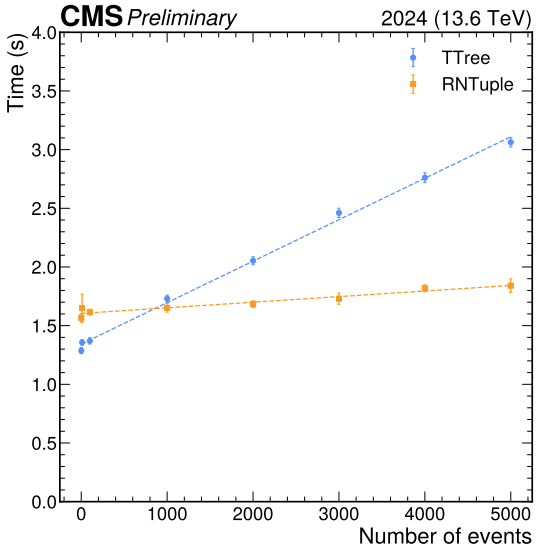

Figure 8.1: Read time of Run 3 RAW data as a function of the number of events for uncompressed TTree and RNTuple formats [185].

1000, 2000, 3000, 4000, and 5000). Reading was performed using the `PoolSource` and `RNTupleSource` modules. A `GenericConsumer` analyzer was employed to ensure that all collections were read.

This step is necessary because CMSSW is optimized to avoid unnecessary processing of data that is not being modified. For example, reading a file and attempting to write it again through an output module without explicitly processing the contained collections would result in no action, even if the output file were to be written in a different format. The `GenericConsumer` analyzer therefore "touches" every collection in the file with negligible overhead, ensuring that the entire processing pipeline is executed under realistic timing conditions.

Processing times were measured on a single thread, with 10 executions averaged after discarding the first five warm-up runs. This was done to cache the data and minimize the overhead caused by file lookups.

The plot shows that the reading time scales linearly with the number of events. RNTuple exhibits a slower increase compared to TTree, leading to significantly faster reading performance for large datasets [186].

This improvement is primarily due to the internal structure of the RNTuple file. In the new experimental format, data are stored in a columnar layout where all collections across events are concatenated, and an auxiliary array containing the starting indices of each event is used for lookup. As a result, the metadata required to describe the file contents does not scale significantly with the number of events, and reading operations require less computation.

Compression comparison

Throughput and compression ratios were evaluated for the four lossless compression algorithms implemented in ROOT: LZMA [159], ZSTD [165], ZLIB [163], and LZ4 [164].

LZMA is a high-compression algorithm based on the LZ77 algorithm, with a huge dictionary size followed by a variant of arithmetic coding to achieve very strong compression. This typically comes at the cost of having a very low throughput.

ZLIB is a widely used implementation of the DEFLATE algorithm, which relies on LZ77 compression combined with Huffman coding for entropy reduction. It provides moderate compression ratios and relatively fast performance, making it a standard reference.

ZSTD is a modern compression algorithm developed by Meta, designed to provide fast compression and decompression with tunable compression levels. It combines dictionary-based techniques with advanced entropy coding, offering an excellent balance between speed and compression efficiency.

LZ4 is a fast compression algorithm optimized for speed that uses a lightweight dictionary approach. While it achieves worse compression ratios than the other algorithms, it provides high-speed compression and decompression, making it suitable for high-throughput workflows.

Data containing 5000 events ($\sim$ 7.65 GB) were compressed at different levels for each algorithm. ROOT provides ten compression levels for each algorithm. Where 0 corresponds to no compression and 9 corresponds to the highest level of compression. These ten levels map to different configurations of each algorithm.

The compression ratio is defined as $\frac{\text{compressed size}}{\text{uncompressed size}}$, with lower values being preferable.

Figure 8.2 shows the throughput (events/s) versus compression ratio for TTree files and RNTuple files, respectively.

Similar benchmarks are shown in Figure 8.3, where the throughput is plotted as a function of the compressed size. The vertical dashed line shows the original uncompressed file size.

Overall trends are consistent across formats: LZ4 provides the highest throughput at lower compression, LZMA achieves the best compression at lower throughput, and ZSTD outperforms ZLIB in all cases.

Both TTree and RNTuple behave similarly, with RNTuple showing slightly lower throughput for LZ4, but with somewhat improved compression across all algorithms, primarily due to the larger original uncompressed size [186].

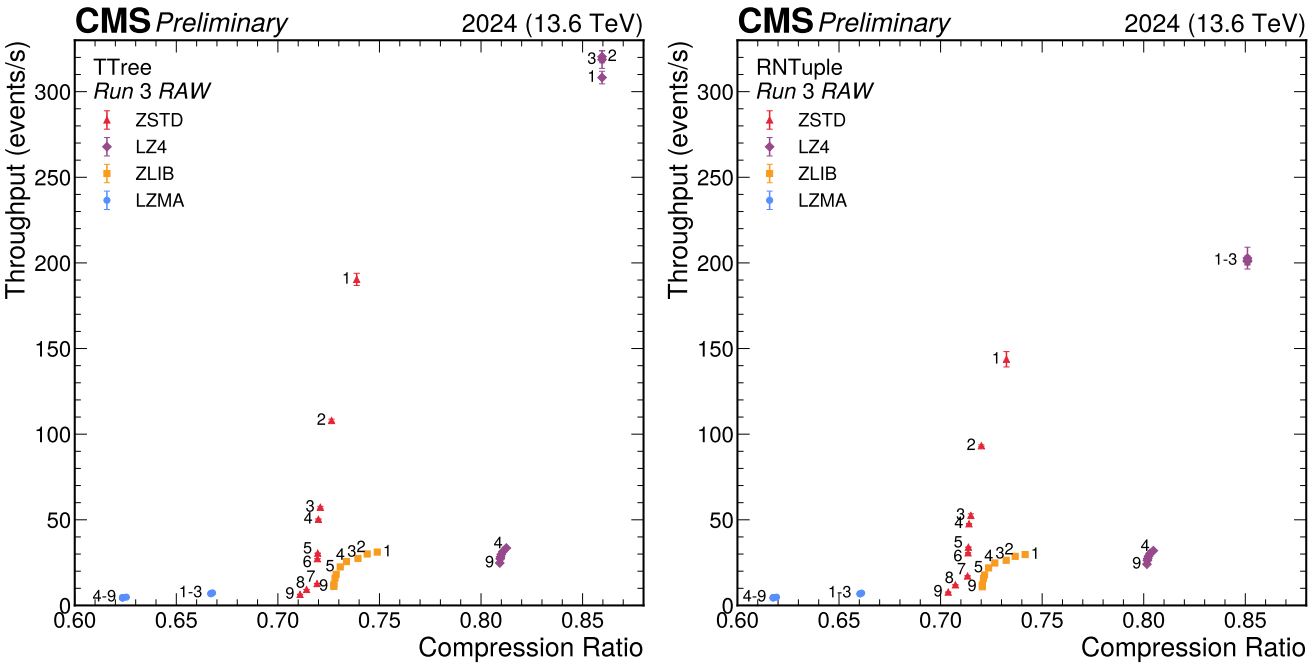

Figure 8.2: Throughput versus compression ratio of Run 3 RAW data in TTree files, on the left, and RNTuple, on the right. Labels on the points indicate the compression level [185].

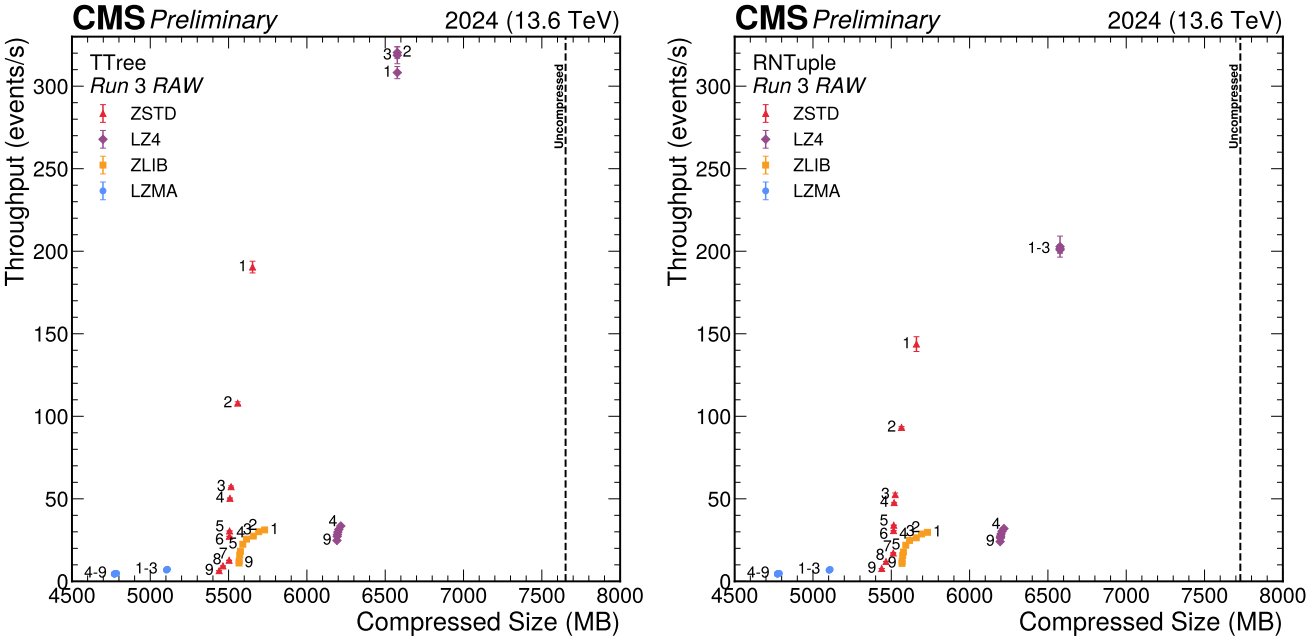

Figure 8.3: Throughput versus compressed size of Run 3 RAW data in TTree files, on the left, and RNTuple, on the right. Labels on the points indicate the compression level. The vertical dashed line indicates the uncompressed size [185].

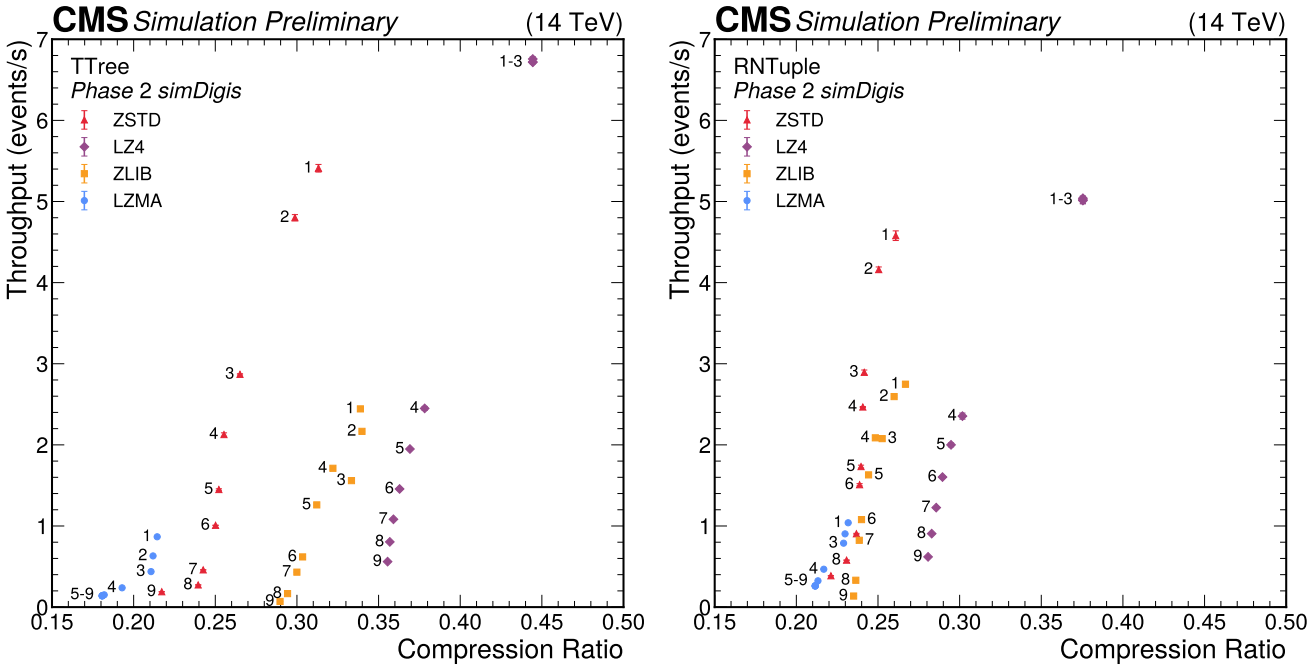

Figure 8.4: Throughput versus compression ratio of Phase-2 Monte Carlo simulation in TTree files, on the left, and RNTuple, on the right. Labels on the points indicate the compression level [185].

8.1.2 Benchmarks with Phase-2 Monte Carlo simulation

Phase-2 simulations used $t\bar{t}$ samples with an average pileup of 200 at a center-of-mass energy of 14 TeV. ROOT files containing 500 events were produced in both the TTree and RNTuple formats. Only the simDigis objects were stored, serving as an approximation of RAW detector data, as presented in Chapter 7. These samples are used for benchmarking compression performance in conditions that closely resemble the high-luminosity environment expected in Phase-2.

Figures 8.4 and 8.5 show throughput versus compression ratio and compressed size for TTree and RNTuple files.

Phase-2 simDigi files generally compress more efficiently than Run 3 RAW files because the structured arrays of detector information in simDigis allow compressors to exploit recurring data patterns. In contrast, RAW data are essentially unstructured bitstreams, which limits the effectiveness of dictionary-based compression. Throughput for TTree is slightly higher at low compression levels, whereas RNTuple achieves better compression on average, except for LZMA.

In summary, among the algorithms, LZ4 provides the highest throughput, LZMA achieves the best compression, and ZSTD delivers a balanced performance between speed and compression ratio. The benchmarks indicate that RNTuple offers faster

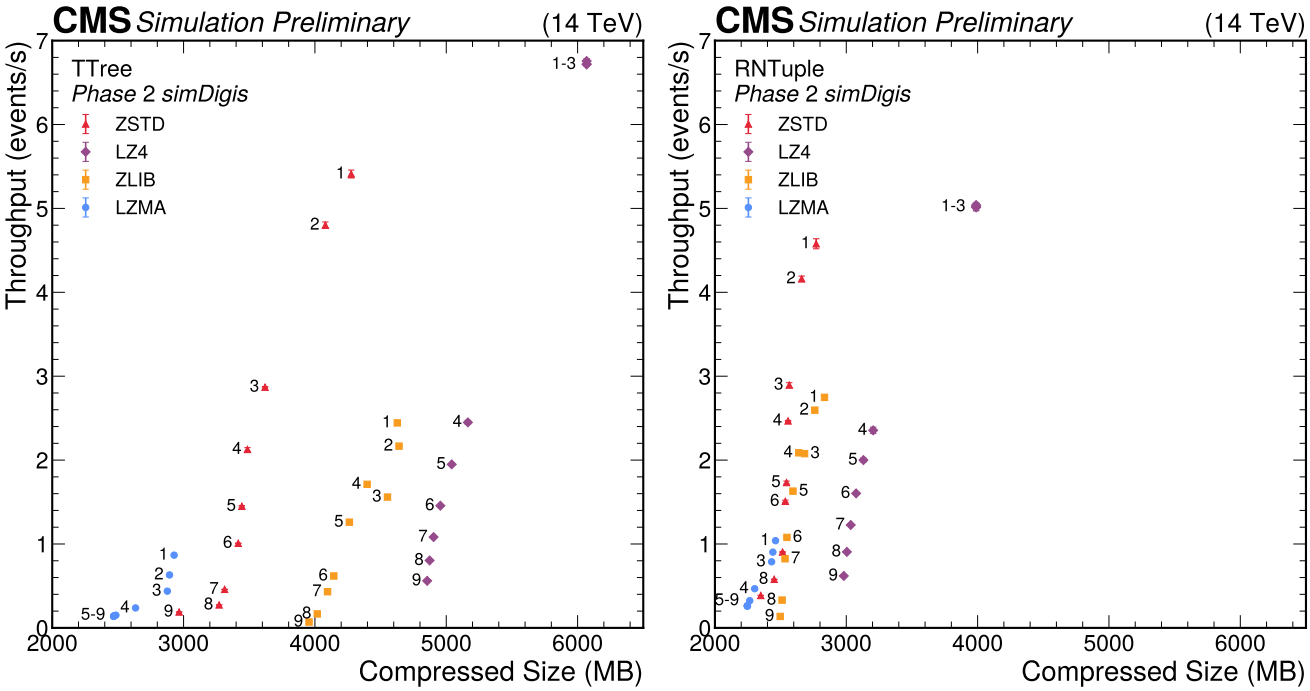

Figure 8.5: Throughput versus compressed size of Phase-2 Monte Carlo simulation in TTree files, on the left, and RNTuple, on the right. Labels on the points indicate the compression level [185].

reading performance compared to TTree, with improved compression results, particularly in cases involving structured data.

Although the improvement for RAW data is modest, RNTuple will likely be the default format for Phase-2 data going forward. This is especially relevant given that RNTuple is still experimental, and further optimizations are expected in the coming years as the developers gather additional requirements from the experiments. Its columnar layout and ongoing development make it well-suited to meet the performance and scalability goals of the NextGen project.

8.1.3 Lzbench

The ROOT framework implements the four algorithms described in the previous section. CMSSW exposes the same algorithms through its output modules by directly relying on ROOT compression. To benchmark alternative algorithms, it is therefore necessary to use external tools. For this thesis, the tool selected is `lzbench` [187]. `lzbench` is a lightweight library designed for in-memory compression benchmarking of open-source encoders and decoders (*codecs*).

The benchmark integrates more than 40 different compression codecs implemented in C or C++. These algorithms can be roughly divided into three groups, depending on their performance characteristics and intended use [142].

The first group consists of slow but strong algorithms. These methods typically achieve the highest compression ratios, but at the cost of very high CPU and memory usage. They are most suitable for scenarios where data are compressed once and decompressed many times. LZMA falls into this category, and many algorithms in this class are variations of it.

The second group consists of medium-speed algorithms, with throughput ranging from 10 to 500 MB/s. These are often based on DEFLATE, such as ZLIB, or on more modern designs like ZSTD. They offer a good balance between compression ratio and speed, with higher throughput than slow algorithms but with generally weaker compression. In most cases, as shown in the previous section, ZSTD-based algorithms outperform their DEFLATE counterparts in both speed and compression ratio.

Finally, there are the fast algorithms, which can reach throughput on the order of gigabytes per second. These codecs are designed to maximize speed, with compression and decompression rates that can rival or even surpass disk I/O performance. They are particularly suited for real-time data processing or memory-efficient storage pipelines. Traditional examples include LZO, while more recent approaches such as LZ4 have become widely adopted in high-performance applications [164].

This classification highlights the trade-offs between compression ratio and throughput, reflecting the same balance discussed in the context of Shannon entropy and dictionary-based methods in the previous chapter.

Additionally, the tool integrates some GPU-based compression algorithms, allowing tests of accelerator use for compression. Most compression algorithms do not benefit from multithreading, as splitting the data into blocks often results in a lower overall compression ratio.

`lzbench` includes in its library an LZ4 implementation on CUDA, exposed via the NVIDIA *nvCOMP* library, as well as *libbsc*, a CUDA-based implementation of the BSC algorithm.

The nvCOMP library is NVIDIA's GPU-accelerated compression and decompression framework. It supports standard formats such as LZ4, ZSTD, and Deflate, as well as GPU-optimized codecs like GDeflate and Cascaded compression [188]. `lzbench` utilizes the LZ4 implementation from nvCOMP, which is the latest open-source version, as NVIDIA closed the source code for the GPU implementation starting from version 4.2.3 [189].

The nvCOMP LZ4 codec divides the input into blocks and compresses them in parallel on the GPU. The block size is a tunable parameter in its API, affecting both compression ratio and throughput. Due to this parallel block-based design, the GPU

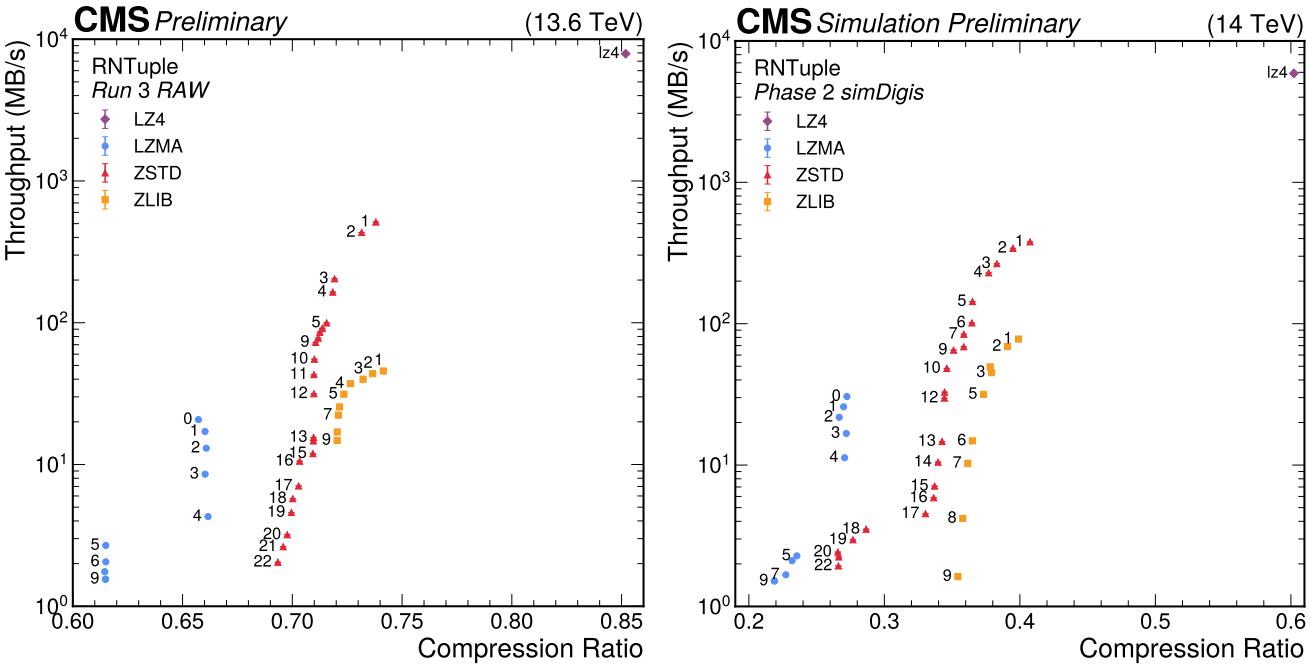

Figure 8.6: Throughput in MB/s versus compression ratio of Run 3 RAW data size on the left, and Phase-2 Monte Carlo simulation, on the right, in RNTuple file format. Labels on the points indicate the compression level for LZMA, ZSTD and ZLIB. LZ4 provides a single compression setting.

implementation can achieve high throughput rates, particularly when processing large structured data [190].

The libbsc library implements a high-performance compression algorithm, based on the Burrows–Wheeler Transform (see Section 6.4). It compresses data in blocks and supports parallel block processing on multicore CPUs and GPUs [173]. GPU support is provided through NVIDIA CUDA. Memory requirements for compression and decompression depend on both block size and the number of blocks processed in parallel.

By using `lzbench`, we extended the study beyond the four ROOT-provided algorithms to explore whether alternative codecs may offer better performance for CMS RAW and simulation data.

For the benchmarks, the same files identified in Sections 8.1.1 and 8.1.2 have been used. In particular, 5000 events in RNTuple data format were used for Run 3 RAW data, and 500 events in RNTuple format for Phase-2 simulation. The data was compressed by running the `lzbench` benchmark for 5 iterations and averaging the results. The node used for compression has an NVIDIA H100 GPU, 180 GB of RAM, and a 96-core CPU. To utilize multi-threading capabilities, compression was performed on 500 MB blocks, using 8 threads for CPU compression.

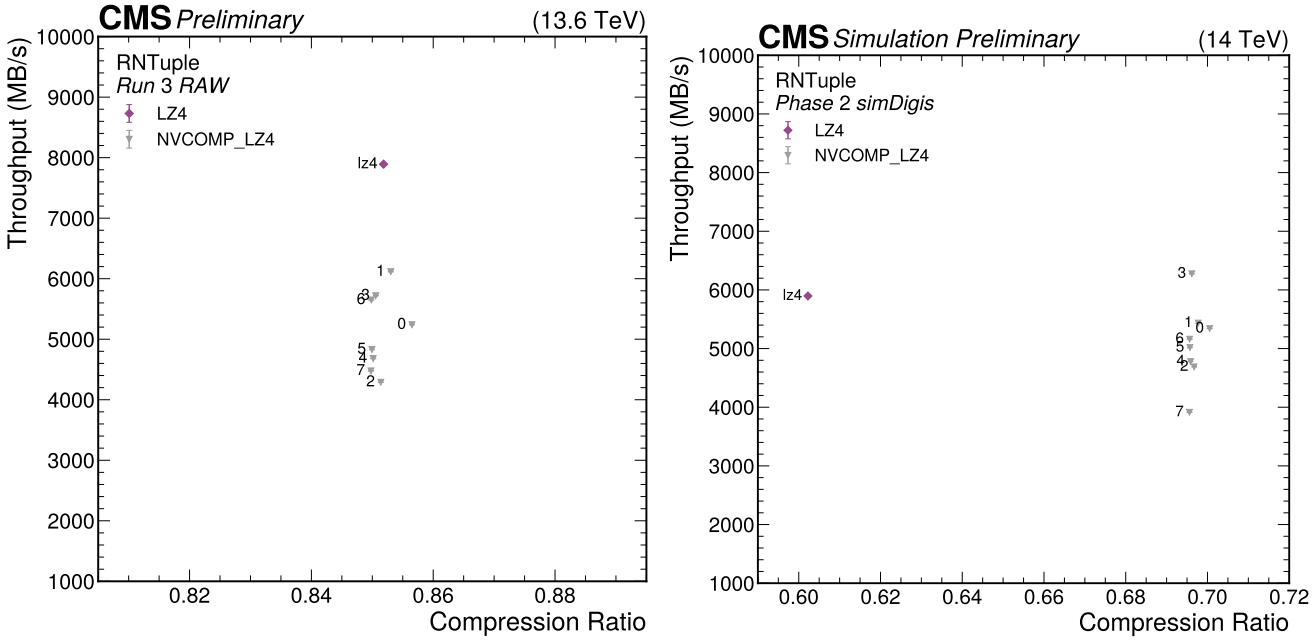

Figure 8.7: Throughput in MB/s versus compression ratio of Run 3 RAW data size on the left, and Phase-2 Monte Carlo simulation, on the right, in RNTuple file format. Labels on the points indicate the compression level. Labels on the points indicate the compression level for the nvComp LZ4 implementation. The CPU implementation provides a single compression setting.

Figure 8.6 shows the behavior of the four algorithms, also available in CMSSW, when used from the `lzbench` library. The behavior matches the results found in the previous section, with slightly weaker compression, as the compression performed in CMSSW is optimized for the ROOT file structure, while in this benchmark the file is compressed as a single bitstream.

Figure 8.7 shows the comparison between LZ4 and its implementation via nvCOMP. While the compression ratio is comparable for Run 3 RAW data, the compressed size is significantly higher for Phase-2 data, as the division into blocks reduces the effectiveness of the RNTuples columnar storage. Moreover, the performance does not improve when using a GPU, resulting in worse performance than the CPU equivalent.

Figure 8.8 shows the comparison between LZMA, ZSTD, and the CUDA implementation of BSC. In both cases, BSC proves effective at reducing the data size while maintaining very high throughput. In most configurations, the speed is higher than that of ZSTD, with compression ratios comparable to its highest levels. While not achieving compression as strong as LZMA, the throughput is orders of magnitude higher and, especially for RAW data, BSC proves to be a very valid alternative.

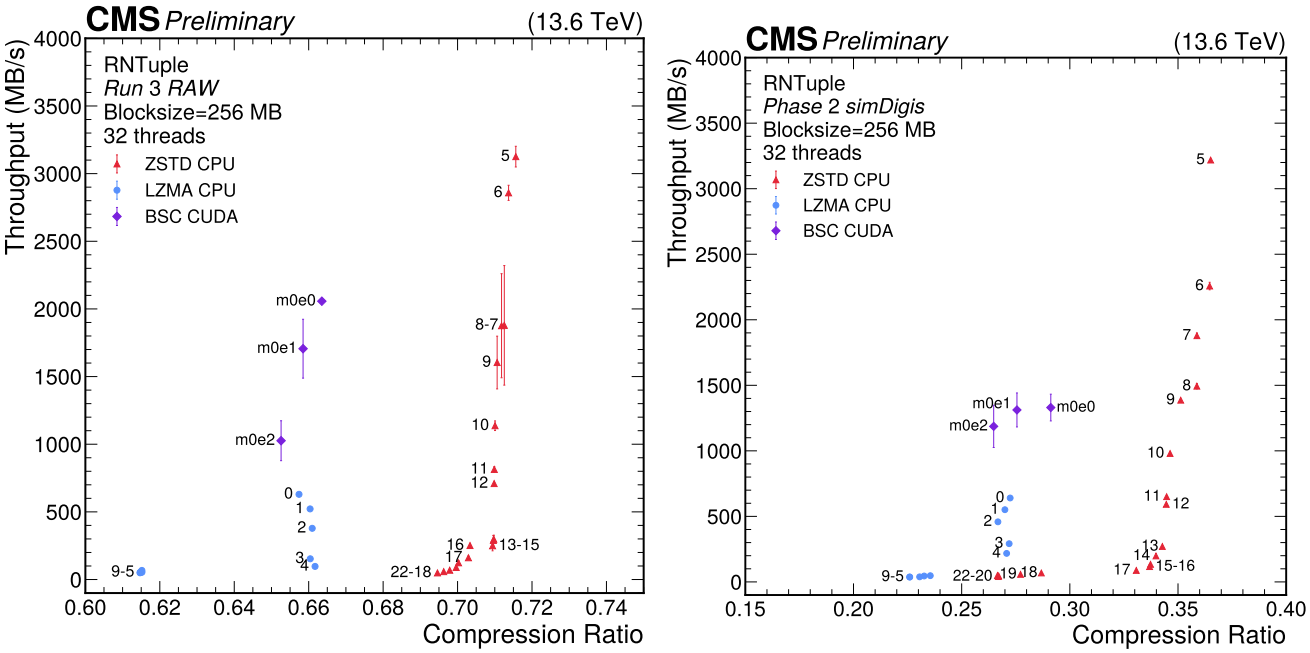

Figure 8.8: Throughput in MB/s versus compression ratio of Run 3 RAW data size on the left, and Phase-2 Monte Carlo simulation, on the right, in RNTuple file format. Labels on the points indicate the compression levels for LZMA and ZSTD, as well as the configuration for BSC.

Additional tests need to be performed to compare different block sizes, as well as to investigate a deeper integration with ROOT to improve performance on structured data that leverages the columnar layout of RNTuple more effectively.

8.2 Strips RAW' compression

Lossless compression reduces the size of RAW data by encoding it in a format that can be fully recovered, at the cost of additional CPU processing. An alternative approach is to store low-level reconstructed objects rather than RAW data. This preserves most reconstruction possibilities and maintains physics performance while significantly reducing file size.

For Heavy Ion collisions, this lossy approach has been implemented in the RAW' format [140]. During PbPb collisions, the maximum trigger rate is limited by the DAQ bandwidth and the event size. To reduce event size and increase the maximum trigger rate, reconstructed strip clusters replace the strip detector RAW data. Instead of storing ADC counts for all strips (see Section 1.3.3), only the charge barycenter and the total number of strips are saved. The charge barycenter is calculated as the weighted average of the number of strips using their charges as weights. Figure 8.9

SST cluster format	RAW		RAW'	
Stored content	**Strip index**	**ADC counts (8-bit int)**	**Approximated cluster properties**	

Example stored tracker data	**First strip** 123 **(16-bit int)**	75	• **Barycenter** = 125.5 **(10-bit int)** (We store 10x barycenter as int)
			• **Size** = 6 **(6-bit int)**
			• **Average charge** = 100 **(8-bit int)**
			• **Cluster shape: (1-bit Boolean)**
			  • **Saturated strip**
			  • **Peak filter**
				[Event-basis] FED modules & readout error info

	(derived by first strip & ADC list) 124	103		
	125	127		
	126	94		
	127	161		
	128	42		
Example total bits per cluster	16 + 8*6 = 64 bits		10 + 6 + 8 + 1*2 = 26 bits + smaller FED error contribution	

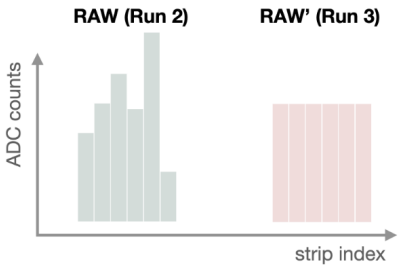

Figure 8.9: An example of RAW' compression for the strip detector data. On the left, the count of bits used to save a cluster of strips using RAW and RAW'. On the right, the signal (ADC) saved per each strip using RAW and RAW' [140].

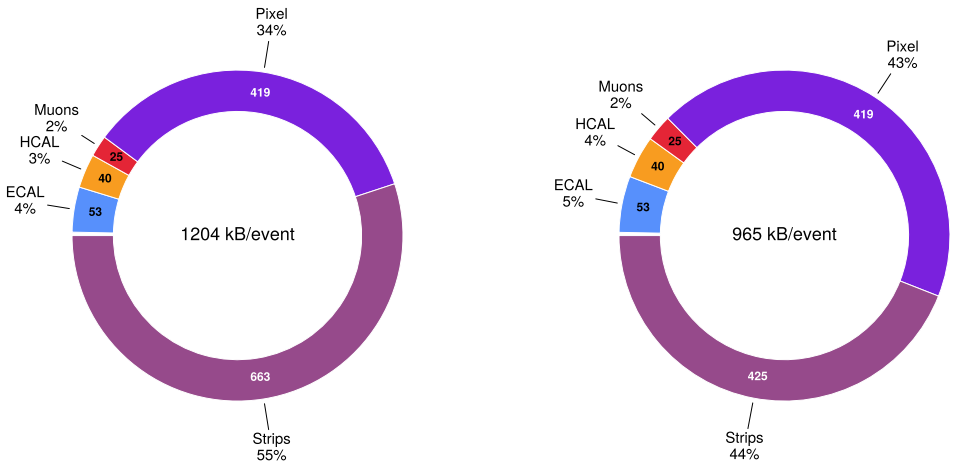

Figure 8.10: Data fraction per detector in Run 3 proton-proton collisions using RAW (left) and RAW' (right). The current RAW' version compresses only the strip detector.

illustrates the number of bits saved per strip cluster, compared to the ADC counts, stored in RAW and RAW'.

8.2.1 RAW' in proton-proton collisions

The RAW' format has been tested in proton-proton collisions with an average pileup of ~ 60. For the strip detector, the compression achieved is -36%. The size is reduced from 663 kB/event to 425 kB/event. This corresponds to an overall event size reduction of -20%.

The results are shown in the pie charts of Figure 8.10.

A significant fraction of reconstructed strip clusters originates from out-of-time pileup collisions. The primary feature of these clusters is the small amount of collected

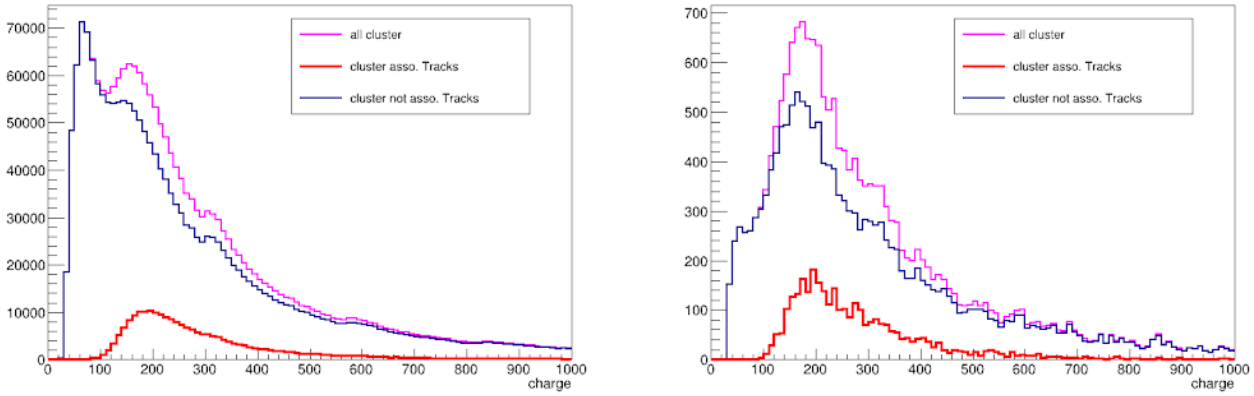

Figure 8.11: Distribution of strip cluster total amplitudes in $t\bar{t}$ simulation with pileup 60 (left) and no pileup (right).

charge. Such clusters are typically excluded from tracking by applying a minimum charge threshold.

To further reduce the event size, only strip clusters passing a tight charge requirement are stored. In this selection, the sum of ADC values of all strips in a cluster is considered. This sum is then normalized by the module thickness, as defined in the detector geometry [191]. The resulting value must exceed 1945 ADC/cm. This cut is applied at the `SiStripClusterizer` EDProducer step [192]. The cut has been chosen to optimize the tracking efficiency and to reduce the fake rate. A tight cluster charge can reduce the size of the strip detector event by approximately 13%.

In Figure 8.11, the total cluster amplitude distribution is shown. The figure also displays two sub-distributions: clusters associated with reconstructed tracks and clusters not associated with any track. Track hits are considered to match a strip cluster if they belong to the same strip module and if their positions overlap. Almost all track hits are matched to clusters.

We find that about 90% of clusters are not associated with any track. This indicates that most clusters currently stored in RAW' are unused in tracking. Many originate from low-charge out-of-time pileup. Others come from soft tracks produced in nuclear interactions with the tracker material. These tracks are not reconstructed and therefore their strip clusters appear as “not associated” in the plot. Removing such clusters from RAW' could significantly reduce the event size. However, the potential loss of useful information must be carefully assessed.

8.2.2 Discussion

The compression strategies investigated in this chapter span a wide range of approaches, each presenting trade-offs between data volume reduction and reconstruction perfor-

mance. Understanding these trade-offs is necessary for making informed decisions about which strategies to deploy for Phase-2 of the experiment.

The removal of strip cluster measurements below a charge threshold, as described in Section 8.2, demonstrates an event size reduction of approximately 20% in proton-proton collisions without compromising the quality of prompt track reconstruction used in most physics analyses. Tracks from the primary vertex with sufficient transverse momentum and moderate impact parameters are reconstructed with essentially unchanged efficiency and resolution when using the RAW' format, as these tracks produce clusters well above the applied charge cut. However, this approach has significant implications for physics signatures that rely on non-standard track topologies. Low-p_T tracks, which deposit less charge in the silicon sensors and are more likely to produce clusters near the threshold, leads to reduced reconstruction efficiency if stringent charge requirements are applied. The impact is particularly pronounced for tracks below approximately 0.5 GeV, where a substantial fraction of clusters may fall below the selection threshold. These tracks can be relevant for certain analyses, such as studies of minimum bias events, searches for soft displaced vertices or tracks from long-lived particles.

The implications extend beyond simple efficiency losses. Many searches for new physics rely on these characteristic signatures. If the lossy compression systematically removes clusters from displaced tracks, it degrades not only the reconstruction efficiency but also the ability to distinguish signal hinting at anomalous events. Analyses targeting these particles may therefore require access to the full RAW data or a modified RAW' format with relaxed charge requirements in regions of phase space relevant to their signatures.

The low-level reconstructed objects analyzed in Section 7.4.2 (and the additional data in Appendix B) leads to different degrees of information loss compared to the original RAW detector data. For the tracking detectors, storing reconstructed hits rather than clusters or RAW data reduces the event size by factors ranging from approximately 50% for the strip detector to nearly 90% for the pixel detector. These rechits preserve the essential geometric information but discard details about cluster shape, charge distribution, and individual strip or pixel ADC values.

For most tracking applications, this level of compression is entirely acceptable. Standard track reconstruction algorithms use hit positions as input and do not require the full cluster information. However, cluster shape can provide discrimination between particle types and inform corrections for local inefficiencies or calibration issues and may benefit from or require access to RAW data.

The implementation of any lossy compression strategy for production data storage requires rigorous validation to ensure that physics performance is not degraded in unacceptable ways. A robust validation framework should include several components. First, a direct comparisons of reconstruction performance metrics, such as tracking ef-

ficiency, fake rate and resolution, vertex position resolution (See Section 5.1.2). This must be performed on identical event samples processed with both the full RAW data and the compressed format and binned in relevant kinematic variables (p_T, eta, ϕ) and event properties to identify any regions of phase space where performance degrades.

Second, the impact on physics object reconstruction must be assessed. Even if tracking efficiency remains acceptable, changes in fake rate or track parameter resolution can propagate through the particle flow algorithm and affect higher-level objects.

Third, the effect on actual physics measurements must be evaluated. This includes both precision Standard Model measurements and searches for new physics. For the latter, particular attention must be paid to signatures that might be disproportionately affected by the compression choices, as discussed before.

Finally, this validation cannot be a one-time effort performed before deployment. As detector conditions evolve and algorithms improve the validation must be repeated to ensure that the compressed format continues to meet physics requirements. This suggests the need for an automated validation framework that continuously monitors key physics performance metrics and alerts experts if degradation is detected.

The Next Generation Triggers project, in continuing the work on Task 3.3, should prioritize the development of such a validation framework as an essential complement to the compression studies. The work that will be presented in Part III could present a baseline for this future development.

8.3 Summary

In this chapter, we presented the results of the first year of work on Task 3.3 of the NextGen project, focused on exploring compression strategies for CMS RAW data. Lossless compression benchmarks with both Run 3 RAW and Phase-2 simulated data showed that the new RNTuple format outperforms the legacy TTree in terms of read performance and generally provides better compression, particularly for structured data. Among the tested algorithms, LZ4 proved to be the fastest, LZMA achieved the strongest compression, and ZSTD offered a balanced compromise between speed and efficiency.

We extended these studies using the `lzbench` library, which allowed us to evaluate a broader set of codecs, including GPU-based implementations. While the nvCOMP LZ4 codec did not provide significant improvements, the CUDA implementation of BSC showed competitive compression ratios and very high throughput, making it a promising candidate for future studies.

Finally, we investigated lossy compression strategies, with a focus on the RAW' format for strip detector data. Preliminary results demonstrated that a 20% event size reduction can be achieved in proton–proton collisions, without degrading track

reconstruction performance. Further size reductions may be possible by discarding low-charge clusters associated with out-of-time pileup, although the impact on physics performance will require careful evaluation.

Together with the characterization of the RAW data and the Phase-2 simDigis presented in the previous chapter, the results presented in this part provide a solid foundation for continued exploration of compression strategies in the NextGen project.

Part III

Automating Detector Calibrations

Chapter 9

Automation framework for calibrations

Within the Physics Performance and Dataset (PPD) area, the AlCaDB group is responsible for developing and monitoring the alignment and calibration workflows, ensuring that calibration constants are stored and efficiently propagated to the physics object reconstruction for all CMS subsystems [91].

These constants are essential for ensuring that physics objects are reconstructed with high precision and stability, and are organized into Global Tags (GTs) [193]. A Global Tag represents a consistent set of all necessary conditions, ranging from detector geometry and subdetector calibration constants to higher-level calibrations for analysis. This setup decouples the conditions data from the CMSSW. A single CMSSW release can be used with different Global Tags by specifying the desired tag in the configuration file. Multiple GTs are maintained for different purposes. For data-taking, the Global Tags are *Prompt*, *Express* and *HLT*

Constants that can be determined within 24 hours of data taking and require regular updates are fed back into the conditions database to be used in the Prompt reconstruction. This pipeline is the Prompt Calibration Loop (PCL) [92].

The PCL includes several workflows that run automatically on the Tier-0 processing farm for each collected run. These workflows are designed around the 48-hour window between data acquisition and the Prompt reconstructions. The PCL is embedded in the overall CMS data-taking workflow and uses the Event Data model of the CMSSW framework. While the PCL is running, the full physics dataset is stored on a disk buffer at Tier-0 for up to 48 hours.

The Express stream is a subset of approximately 10% of the total event data, intended for prompt physics analysis, alignment and calibration, detector and trigger commissioning, and fast physics validation. This stream, along with any additional required data, undergoes express reconstruction to generate the AlCaReco skims within

1-2 hours of data taking. AlCaReco datasets are skims of reconstructed data, containing the minimal information required as input to a given calibration or alignment algorithm. Calibration algorithms then run in parallel on these datasets to generate intermediate products, such as histograms for DQM and preliminary calibration constants for the Express stream. These outputs are then aggregated in the final step, known as AlCaHarvesting, which combines them into the set of conditions used for Prompt reconstruction. Once prompt alignment and calibration are complete, the full data sample is passed through prompt reconstruction using the updated conditions and is subsequently made available for analysis. The PCL ensures that calibrations needed for Prompt reconstruction are computed and validated within a 48-hour window.

This workflow is the target of the NextGen project's task 3.4, as presented in Section 2.2.4. The objective is to improve this calibration loop to achieve prompt-like calibration within a few hours, making it readily available for the HLT reconstruction.

Not all workflows are suited for the PCL. The PCL offers automatic data processing to provide per-run calibration constants. It operates explicitly on the Express stream and is bound to CMSSW's EDM, based on C++. For calibration tasks that require data from multiple runs and utilize different data streams, a new automation framework has been developed with flexibility in mind. This framework, referred to as the Automation Framework (AF), can execute tasks written in any code by utilizing containers and allows defining any output format. It supports workflows of arbitrary complexity in terms of the number of tasks and grouping of runs to be processed together.

9.1 ECAL Automation Framework

The particular case of the CMS electromagnetic calorimeter, which is highly sensitive to radiation damage (see Section 1.3.4), requires regular and frequent calibrations, necessitating the development of a system to automate the procedure as much as possible. Up to Run 1 and Run 2, delivering optimal resolution for electron and photon energy in the calorimeter required continued work to derive new calibration data. The repetitive nature of these tasks made them a perfect candidate for an automated system, which would provide accurate and reliable calibration data while freeing human resources. During the LHC's second long shutdown, this automation system was developed, and it was deployed in 2022 for the start of Run 3 [194].

Although intended as a commissioning year, the system provided conditions that were actually used to update the calibrations in the ongoing data-taking. The system operated without interruption throughout the six-month data-taking period, with an average of 500 jobs per day.

The good results obtained in the first year of the project led to its extension for the use by other groups and subsystems. By the end of 2023, the Automation Framework

had been included in the responsibilities of the AlCaDB group, providing a tool for the entire CMS experiment to automate calibration beyond what is possible with the PCL. As of the start of data-taking in 2025, the AF is used by multiple subsystems [195]. ECAL remains the primary user, with over 30 distinct workflows spanning various stages of the calibration pipeline.

In the automated system ECAL implemented workflows to compute the alignment with the tracking system, monitor the energy scale of π_0 and $Z \to e^+e^-$, synchronize the timing of the ECAL channels, and track the evolving pulse shape of each channel since the start of the AF operations. Additionally, it integrates multiple workflows for data reconstruction and calibration streams that are not currently processed at T0, intended for offline analysis and calibration. More recently, new workflows were integrated to derive updated pulse shapes from different datasets that describe the time-profile signals in the ECAL channels [196]. Additionally, improved timing calibration was implemented to align timing across the channels, ensuring signals are synchronized [197]. The AF is also used to validate workflows and calibration data, enabling the faster deployment of new conditions.

HCAL has also adopted the Automation Framework to improve the calibration of its subsystems. Pedestal conditions, including their production, validation, and uploading, have been fully automated. The framework operates independently, requiring experts only to review the outputs, such as plots, logs, and condition updates. Pedestal tables generated by the automated workflows have already been employed in the 2024 re-reconstruction campaign. Progress has also been made in automating the gains calibration, which is essential for converting energy to photocurrent and monitoring radiation damage, particularly in the HE and HF regions. A pipeline for gains calibration is now operational for HE and HF, though additional laser runs are still required [198].

The Precision Proton Spectrometer (PPS) is a subsystem of the CMS designed to detect and reconstruct protons that are scattered at very low angles from collisions [199]. The PPS pixel sensors are housed in vessels called Roman Pots, which are inserted and retracted from the LHC beam pipes at distances of hundreds of meters from the interaction vertex. The regular repositioning necessitates continuous recalibration of the alignment with respect to the beam axis. In the past two years, workflows for the AF have been developed for PPS [200]. The automation uses a dedicated AlCa dataset containing only PPS products and trigger information, and runs multiple workflows to measure tracking efficiency, correct timing alignment, and evaluate timing resolution. Time-walk and alignment corrections, previously performed in the PCL, have been reimplemented within the automation framework, allowing for easier customization, updates, and reprocessing when needed. A validation workflow has also been introduced to ensure the quality of results [201].

Additional work is underway for utilizing the AF in Multi-Run Harvester workflows, Tracker Alignment, and reconstruction and calibration of the jets and missing transverse energy [195], demonstrating the need and interest in using the tool.

9.1.1 Pipeline

The automation frameworks handle workflows as a collection of tasks. As discussed in the introduction, these tasks can be any executable script. The execution of tasks along the workflow chain is subject to a locking mechanism, where a task starts only after a previous one is completed. The execution of a task can also be locked while waiting for payloads to be delivered by other data processing. Multiple locking conditions can be combined to create complex interlocking workflows. Workflows can have forks in their flow, utilizing the locking mechanism, as well as a merging workflow that waits for the completion of different workflows to proceed.

The source data for the task can vary. A standard interface is provided to retrieve run files from the centralized CMS data system on the GRID (See Section 1.4.1), as well as files available locally and on the EOS storage system [202, 203]. The output of the previous task can also be transferred between tasks and the workflow to be recalled as input.

Each task can be divided in stages: the *submit* stages checks for new tasks and run jobs on the source data provided that the locking conditions are satisfied; the *resubmit* stage reprocess failed jobs; the *check* stage controls if every job in a task is completed to mark the task as done for the run.

The job submission, execution and update of the completion state is done by *task handlers*. To support iterative workflows, a dedicated task handler was implemented [204]. This component monitors the status of tasks, checks for iteration conditions, and manages a counter field in the database to coordinate successive cycles. The overall iteration task remains in the running state until all cycles are completed, at which point it is marked as completed.

Figure 9.1 shows a flowchart of the execution of a single task (above), and of tasks within a workflow (below). A run controller task (RunCtrl) in the automation runs regularly to check when a new run is acquired during data taking. As soon as the CMS DAQ system marks the run as complete, its processing is triggered by setting the status of the workflows assigned to that run to *new*. Run metadata, such as the amount of integrated luminosity, the status of the subdetectors or the configuration of the DAQ, determines the workflow assigned to each run. Manual injection can also be done to override the automatic mechanism and allow, for example, the reprocessing of data.

Each workflow is executed at a regular interval, starting tasks based on the locking condition. Tasks that require data from an entire LHC fill or a larger amount of data will wait until the set of runs matches the task's requirements to start.

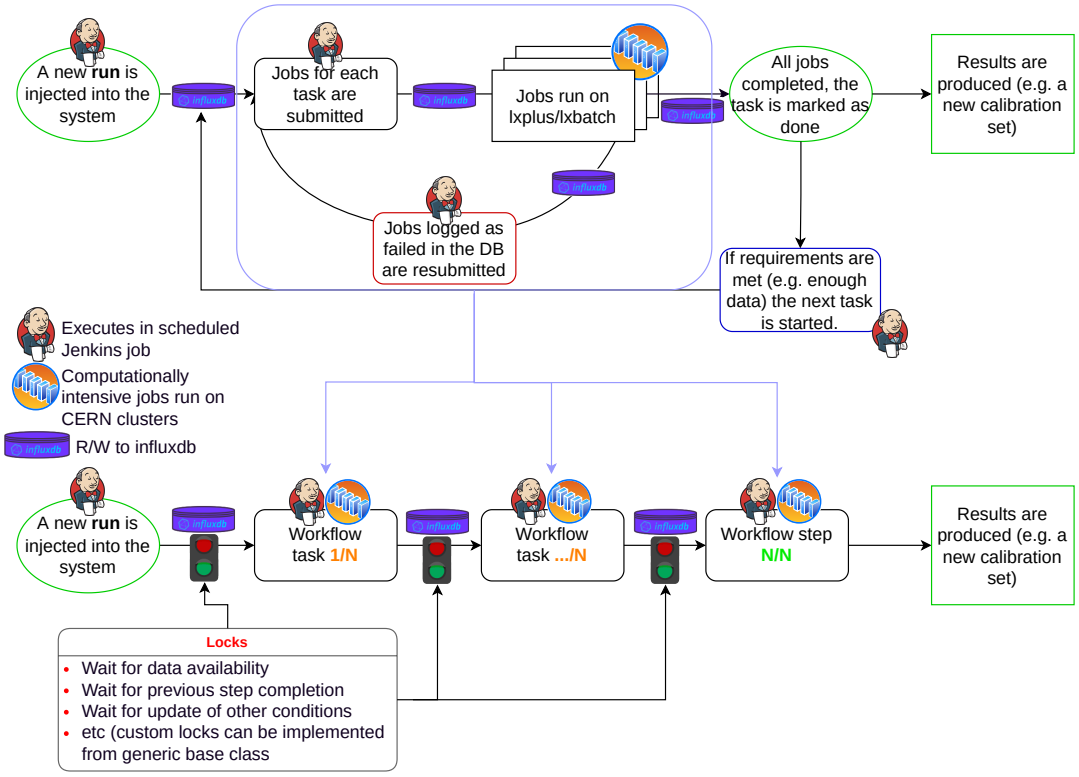

Figure 9.1: Task and workflow execution diagram for the Automation Framework [194].

The task handlers will submit jobs and update the status of the run to 'processing'. Jobs that require large computation can be run on the HTCondor batch system [205]. At completion, a job is marked as *completed*, or as *failed* in case of failure. After all jobs are completed and the *check* stage is run, the task is marked as *done* for that specific run and the resulting data are linked in the database.

9.1.2 Tools

The AF utilizes various industry-standard tools and manages communication between them using a Python package. To deploy the framework applications, CERN's PaaS has been used (see Section 4.1.5).

The run number, along with their corresponding metadata, task status, and workflows for each run, as well as the job status, including input and output, are all stored in an InfluxDB database hosted on the PaaS. InfluxDB is a database designed to support, store, and process time series data [206]. It supports fast queries and provides a Python interface that is easy to integrate into packages. The connection to the database ensures the persistent storage of the automation's status and the fast retrieval of data. It also enables the communication of output from tasks, allowing for interdependencies between workflows without requiring data movement from the shared EOS storage.

The workflows are handled through an instance of Jenkins deployed on PaaS. Jenkins is an open-source orchestration software [207]. It provides an interface for scheduling builds and executing workflows, facilitating continuous integration and development. The workflows are run through Jenkins on a remote agent, specifically instances of the LXPLUS interactive logon service of CERN. This cluster of machines provides access to a common file system, such as EOS, and enables the deployment of jobs on HTCondor, as well as the execution of custom containers. From the Jenkins interface, users can check the status of workflows with strict role permissions, allowing different groups to work on separate automations without interfering with each other.

The Jenkins workflows are described in a git repository stored on the CERN Gitlab instance [208]. The repository contains the configuration for the workflows, the script to be run locally or on HTCondor, the definition of the Jenkinsfile used by Jenkins to schedule and stage the workflow build, and all the necessary files for task execution. Jenkinsfiles use Groovy, a scripting language with a Java-like syntax. A snippet of the configuration for the Zero Bias workflow in Jenkins is shown in Figure 9.2 (more details on this specific workflow will be presented in the next chapter). The language allows defining shared libraries for common functions, as well as environment variables. The pipeline definition follows, and is split into stages, each defining the script to run. Post and pre keys are used to define the action to take before and after the workflow. One example is the `sendMattermostNotification` that sends a message via web API to a dedicated chat, to warn of failed workflows.

The repository also contains the description and the build instructions of a Docker image [209]. The Docker image, specific to each group, defines the software required to run each job and the correct release to use. The image is mounted on the LXPLUS agent via Apptainer [210], and the workflows are run through it. The workflows use the `automationApptainerStep` to load the Docker image and launch the script passed as an argument in a shell inside the container, as shown in Figure 9.2. The build of the image is handled through the Gitlab CI/CD process. New builds of the image are then unpacked into the CVMFS shared filesystem and made available to Apptainer for faster deployment [211, 212].

Finally, the last tool hosted on CERN PaaS is Grafana. Grafana provides a user interface to monitor the status of the AF. It's a web application that provides analytics and visualisation for different data, and connects directly to InfluxDB [213].

9.2 Investigating the transition to Gitlab CI/CD

In June 2025, CERN IT started decommissioning the centrally managed Jenkins application on the PaaS. Following this update, we began investigating the potential switch to GitLab CI/CD as a more modern and maintainable solution.

GitLab CI/CD is already in use for deploying Docker containers with Apptainer, and a pipeline is operational in each repository of the Detector's Performance Groups (DPGs) that utilize the automation. Additionally, the pipeline is used to validate the Jenkinsfile by accessing the Jenkins API with credentials defined in the Gitlab CI/CD variables, which provide a layer of masking integral to secure requests to the APIs.

GitLab CI/CD provides a solution that covers source code management, issue tracking, continuous integration, and continuous deployment within a single interface. In contrast, Jenkins functions as a dedicated automation server, primarily focused on orchestrating build, test, and deployment pipelines. While highly extensible through its vast plugin ecosystem, Jenkins typically requires integration with external tools for source control, artifact management, and other DevOps functionalities. The switch could streamline overall operations by reducing the number of required applications and simplifying credential handling, which is already robust in the repository's group access management.

Gitlab CI/CD uses a static, YAML-based configuration file usually named `.gitlab-ci.yml`. A snippet of the configuration file for the Zero Bias workflow for the GitLab CI/CD platform is shown in Figure 9.3. This presents a very similar structure to the Jenkinsfile. The key difference lies in the presence of repeated instructions for each stage, as they are more independently defined, unlike the polymorphic structure that is possible with the Java-based language of the Jenkinsfile.

A significant advantage of the GitLab CI/CD implementation is that it allows launching the workflow on dedicated runners within a Kubernetes cluster. In the Jenkins implementation, the Apptainer step is required to run the custom image on the LXPLUS node and subsequently submit the individual jobs to the batch system. With Gitlab CI/CD, the runner can directly deploy a custom image and connect to the HTCondor interface from there. The runners are shared in the collaboration; however, there is the option to connect a dedicated VM as a runner in case additional resources are required.

Jenkins offers greater flexibility in scheduling builds. Each workflow can have a different schedule, and the application automatically staggers the builds to avoid having too many builds run simultaneously. In GitLab, a single schedule is shared across the entire pipeline, so workarounds are necessary to support multiple schedules. Moreover, there is a limit on the number of jobs that can be defined in a pipeline, which could prevent the pipeline from starting if too many workflows are defined in the repository.

Moving away from a separate Jenkins instance could lead to potentially simplified infrastructure and reduced maintenance overhead. However, transitioning to a more extensive GitLab CI/CD role also presents challenges and potential downsides. A significant hurdle is the migration effort for existing Jenkins pipelines. Re-implementing complex Jenkinsfiles into `.gitlab-ci.yml` configurations would require substantial de-

```
@Library("dt-shared-libs")
import static dtctrl.vars.GlobalVars.*
setenv(this, env.JOB_BASE_NAME)

pipeline {
  agent {
    label 'lxplus-dt'
  }
  options {
    timeout(time: 30, unit: 'MINUTES')
  }
  stages {
    stage('zerobias-ntuple-production-resubmit') {
      steps {
        dir('workflows/zerobias'){
          automationApptainerStep """
          python3 ntuple_production.py --campaign $campaign --db ↩
              $dbinstance --logurl $logurl --wdir \$CMSSW_BASE --eosdir \↩
              $EOSDIR resubmit --template template_ntuple_production.sub
          """
        }
      }
    }
...
    stage('zerobias-ntuple-harvest-check') {
      steps {
        dir('workflows/zerobias'){
          automationApptainerStep """
          python3 harvest_by_fill.py --campaign $campaign --db ↩
              $dbinstance --logurl $logurl --wdir \$CMSSW_BASE --eosdir \↩
              $EOSDIR check --max-retries 10
          """
        }
      }
    }
  }
  post {
    unsuccessful {
      sendMattermostNotification "ERROR", "$JOB_BASE_NAME: failed job ↩
          execution.", env.ett_notify_url
    }
  }
}
```

Figure 9.2: The `Jenkinsfile` file with the configuration of the Zero Bias workflow for Jenkins.

```
default:
  tags:
    - k8s-eos

include:
  - local: shared-lib/ci-templates/.shared_templates.yml

stages:
  - production-resubmit
  - production-submit
  - production-check
  - harvest-resubmit
  - harvest-submit
  - harvest-check

zerobias-ntuple-production-resubmit:
  extends:
    - .shared_library_dev
  stage: production-resubmit
  script:
     - cd workflows/zerobias
     - python3 ntuple_production.py --campaign $CAMPAIGN --db $DBINSTANCE↩
          --logurl $LOGURL --wdir $CMSSW_BASE --eosdir $EOSDIR resubmit ↩
         --template template_ntuple_production.sub
  timeout: 30m
...
zerobias-ntuple-harvest-check:
  extends:
    - .shared_library_dev
  stage: harvest-check
  script:
    - cd workflows/zerobias
    - python3 harvest_by_fill.py --campaign $CAMPAIGN --db $DBINSTANCE --↩
       logurl $LOGURL --wdir $CMSSW_BASE --eosdir $EOSDIR check --max-↩
       retries 10
  timeout: 30m
```

Figure 9.3: The `gitlab-ci.yml` file with the configuration of the Zero Bias workflow for GitLab CI/CD.

velopment time and expertise, especially for highly intricate workflows. There would also be a learning curve for new features and paradigms. Furthermore, while GitLab's integrated features are robust, there might be limitations for highly specialized or custom integrations that were easily achieved with Jenkins. The path forward may involve an initial hybrid approach, where GitLab CI/CD gradually assumes more workflow orchestration responsibilities while critical or complex Jenkins pipelines are migrated incrementally.

This transition aligns with the long-term objectives of the NextGen project, where GitLab CI/CD has already been explored in the context of work package 1 (see Section 2.1.1), and it provides a potential solution for orchestrating the fast calibration pipeline for task 3.4.

9.3 Summary

The chapter presents the Automation Framework developed within the CMS experiment to automate and orchestrate the execution of calibration workflows across multiple subsystems, complementing the existing Prompt Calibration Loop. It allows tasks written in any language to be executed in containers, supports complex locking and interdependent workflows, and handles large datasets across multiple runs.

ECAL was the primary driver for the AF but has since been extended to HCAL, PPS, DT, and other subsystems, streamlining multiple workflows.

The framework relies on modern tools deployed on CERN's PaaS, ensuring reproducible execution environments and keeping the cost of maintaining and supporting the tool low. With the decommissioning of Jenkins, the AF is investigating migration to GitLab CI/CD. This approach offers a unified interface for source control, continuous integration, and deployment, allowing workflow execution on Kubernetes runners with direct container support. The AF has proven robust and efficient, processing hundreds of jobs daily, and is now an essential tool for automating CMS calibration, validation, and monitoring workflows in Run 3 and will be a key component for the NextGen Fast Calibration task.

In the next chapter, we present a use case of the AF involving the production of an unbiased dataset for the analysis of the muon background.

Chapter 10

Muon use-case

The CMS muon detector is a crucial component for reconstructing and identifying muons produced in proton-proton collisions. The three main subsystems – the DT chambers in the barrel region, the CSCs in the endcaps, and the RPCs covering both the barrel and endcap regions – are described in detail in Section 1.3.5. Each subdetector provides redundant and precise measurements of muon trajectories and timing information.

The detector faces increased challenges during operation, especially with the ramp-up in luminosity of each run. These conditions have led to significantly higher background rates, which complicate the detection and reconstruction of muons. This chapter presents an effort to use the AF to automate the production of custom NanoAOD files, which include the necessary datasets for characterizing the background in the muon detector. This is done to simplify this type of analysis and provide a reliable source for future background studies.

10.1 Background characterization

While sources like natural radioactivity, cosmic rays, and interactions of circulating protons with residual gas in the beam pipe contribute to the background, the dominant source of background hits in the muon system originates directly from proton-proton interactions themselves. Specifically, charged hadrons that exit the calorimeters can interact within the muon chambers, depositing energy and generating spurious hits [214].

Accurately characterizing and understanding these backgrounds ensures the high precision and stability of physics object reconstruction. Still, it also aids in the design and planning of detector upgrades for the HL-LHC, and it provides essential input for detector monitoring and calibration workflows. This input is fundamental because the

background is a reliable measure of the continuous flux of radiation that damages the detector's electronics [215].

The analysis of backgrounds in the CMS muon detector primarily relies on data collected through *Zero Bias* triggers. Unlike typical physics triggers that select events based on specific high-energy signatures, Zero Bias triggers capture an unbiased sample of events whenever proton bunches cross within the detector [216]. This is achieved by triggering specifically on Beam Pick-up Timing eXperiment (BPTX) signals [217]. The experiment features two beam position monitors (BPMs) located approximately 175 m from the CMS interaction point on each end. These detectors measure the signal left by the proton bunches traveling in the vacuum chamber in the form of an image current of free-moving electrons. This approach provides a unique dataset that is sensitive to detector noise, beam-gas interactions, and collision-induced backgrounds without selection biases towards specific physics processes.

The Zero Bias dataset allows for a detailed study of the various background components. By analyzing data from bunch crossings with and without collisions, it's possible to distinguish between *fast* background components and *slow* ones. The fast background typically includes prompt particles, such as those that might traverse the gaps between the barrel and endcap regions. In contrast, the slow background often consists of neutron gas that permeates the experimental cavern during a run. Separating these two components is crucial for understanding how to design and optimize shielding to protect the detector's sensitive electronics as well as their potential interference with muon reconstruction in the DT and CSC chambers.

Background conditions evolve with luminosity, beam conditions, and detector aging. Automating the processing of the Zero Bias dataset ensures that analysis-ready data are available as quickly as possible after acquisition. Moreover, manually managing the processing of large datasets is a time-consuming task. For this reason, the production of NanoAOD data for Zero Bias dataset has been chosen as an use case for the AF by the muon DPG.

The automated framework streamlines job submission, monitoring, error handling, and output collection in the Zero Bias NanoAOD production and harvesting. This not only frees up human resources for more complex analytical tasks but also ensures that the same processing chain and configurations are consistently applied across all data runs, ensuring that the produced NanoAOD are uniform and that analyses performed on them are reproducible over time.

With an automated pipeline, it becomes feasible to continuously produce and update Zero Bias datasets ready for analysis, facilitating future studies of background as the LHC progresses towards HL-LHC.

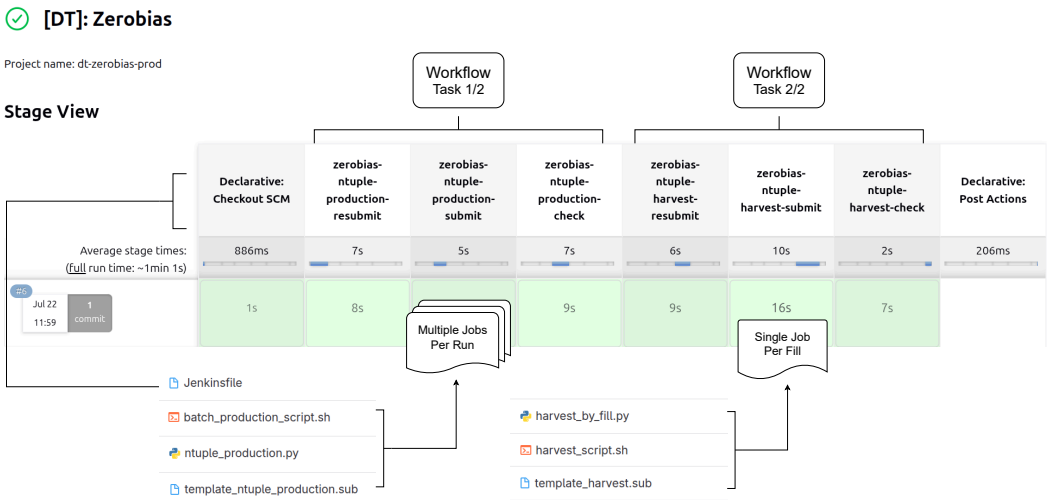

Figure 10.1: Diagram showing the relation between the GitLab repository containing the workflows configuration and the Jenkins dashboard of the Zero Bias workflow with its two tasks.

10.2 Zero Bias Ntuple production and harvest

Figure 10.1 shows the Jenkins dashboard for the Zero Bias workflow. The Jenkinsfile defines the workflow pipeline and the stages of each task. The repository contains the configuration for deploying the job in the batch system and the scripts to be executed.

The workflow consists of two tasks, "Zero Bias Ntuple Production" and "Zero Bias Ntuple Harvest". Both tasks present the default three stages: submit to launch a processing job on the input data, check to verify the job status and mark the run as completed, and resubmit to reattempt failed jobs.

The "Zero Bias Ntuple Production" task implements a TaskHandler that reads the data from the CMS Database Bookkeeping Service (DBS), the catalogue for Monte Carlo and recorded data of the experiment [218]. The data are then filtered to select the runs with files in the `/Zero Bias/*/RAW` database, containing the data that passed the Zero Bias trigger. When no file is present, the run is marked as *skipped*. The task then collects the files for each run and deploys jobs on the batch system to process the files in batches of ten. The output files in NanoAOD format are then saved in a dedicated folder in EOS, and the path to the files is stored in the InfluxDB to be read by the subsequent steps.

The "Zero Bias Ntuple Harvesting" task implements a TaskHandler to read the data output of the Ntuple production task and groups them by fill. Only when a fill is complete, with all its runs marked as either *done* or *skipped*, the task is started. All input files are collected in a single job on the batch system to merge the Ntuples into a single file. The output file is then stored in EOS, ready for analysis.

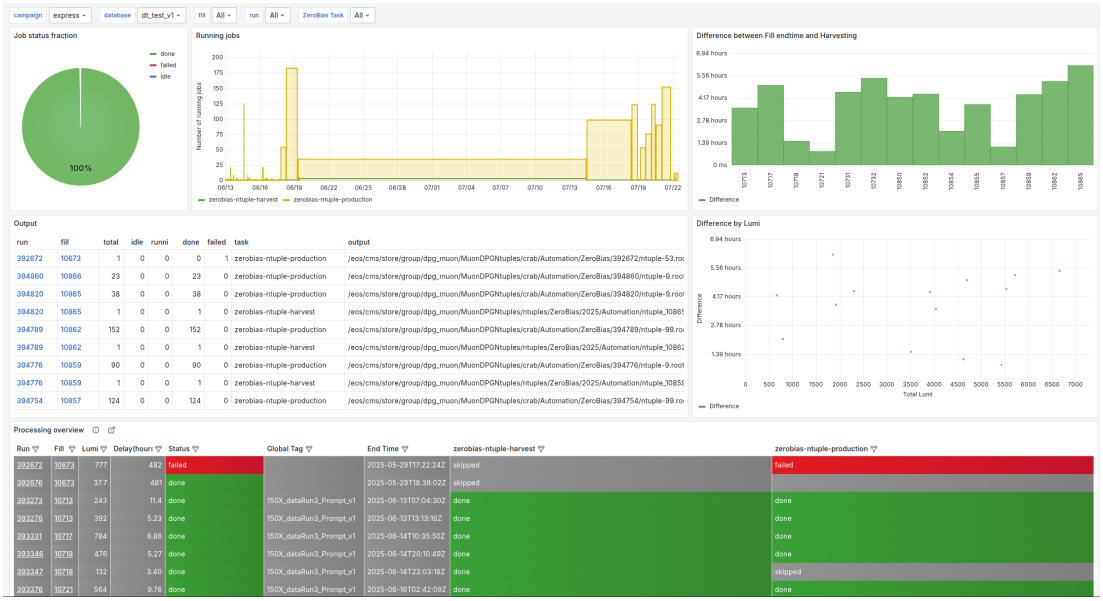

Figure 10.2: Grafana Dashboard that shows the status of the Zero Bias workflow.

The produced Zero Bias Ntuples are specifically tailored to contain all the information required to perform detailed studies of detector backgrounds. They represent a custom NanoAOD "flavor" designed for the muon system, including information from the DT chambers such as digis, rechits, clusters, and segments that are essential for characterizing both fast and slow background components. Beyond the DTs, the Ntuples also include relevant information from the CSCs, GEMs, and RPCs, ensuring that the dataset provides a complete view of the muon system response during Zero Bias triggers.

To evaluate the performance of the automated workflow in terms of timing, we measured the time it takes for the run to be marked as *done* in the AF for the Harvesting task. This time is compared to the moment the run is marked as complete in the CMS Online Monitoring System (OMS), which exposes an API to access information on run and fill status [219]. This difference represents how quickly the workflow finishes from when the data are available on DBS.

The data are aggregated by fill and visualized through the Grafana interface, where InfluxDB is queried to obtain the data, and some transformations are applied to extract the required information. The data are then plotted and presented in a custom dashboard, allowing the user to see the status of the workflows. Figure 10.2 shows the current Grafana interface for the Zero Bias workflow. The panels on the left display the status of the job in the batch system, while the panel at the center-left also lists the path to the output files. The panels on the right display timing information, including the delay in hours from the completion of the run in the database and the completion

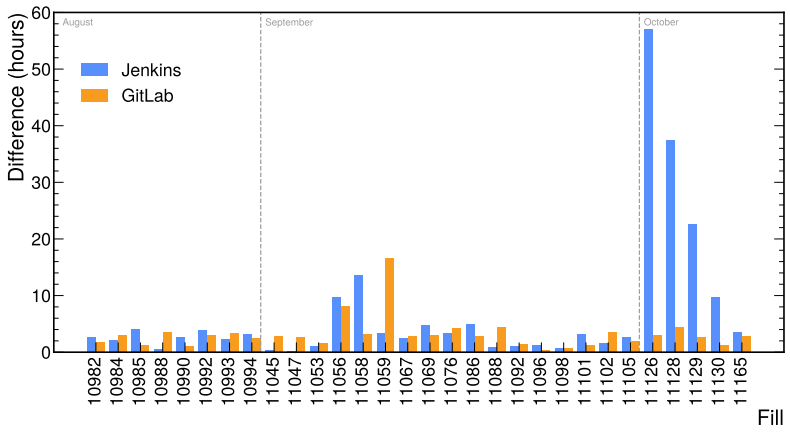

Figure 10.3: Plot of the delay in hours from the completion of the run in the database and the completion of harvesting via the Automation Framework per fill. In blue the results using Jenkins, in orange using Gitlab CI/CD.

of harvesting. The panel at the bottom displays the status of the runs in the dataset, both for the single task and for the entire workflow.

Figure 10.3 shows the time required to complete the automated workflow, defined as the difference between when the last run's Ntuples are harvested and when the corresponding fill is marked as done in OMS. The blue bars correspond to the workflows items orchestrated by Jenkins, while the orange bars correspond to the workflow running via GitLab CI/CD, as described in Section 9.2. The plot covers several fills during the data-taking period from late August to early October 2025. The trend is generally stable, with a peak at the beginning of September caused by a DDoS attack that temporarily took the CERN GitLab instance offline [220]. A larger peak at the beginning of October is present in the Jenkins values, due to an HTCondor update that was not correctly reflected in the Docker image used by the LXPLUS agent. The peak on the Gitlab values is absent, since the Docker image is not used in that version of the AF as each stage on GitLab CI/CD is deployed live with the most up-to-date software versions.

Figure 10.4 shows the delay of the automated framework as a function of the total lumi collected in the fill for the Gitlab CI/CD orchestration. Taking into account the delay due to the DDoS attack, the plot shows that there is no strict correlation between the delay and the amount of data taken. The Automation Framework proves to be effective at handling varying data volumes without significant degradation in performance, maintaining a consistent turnaround time for the Zero Bias workflow. This demonstrates its robustness and scalability, ensuring timely availability of processed Ntuples for background studies.

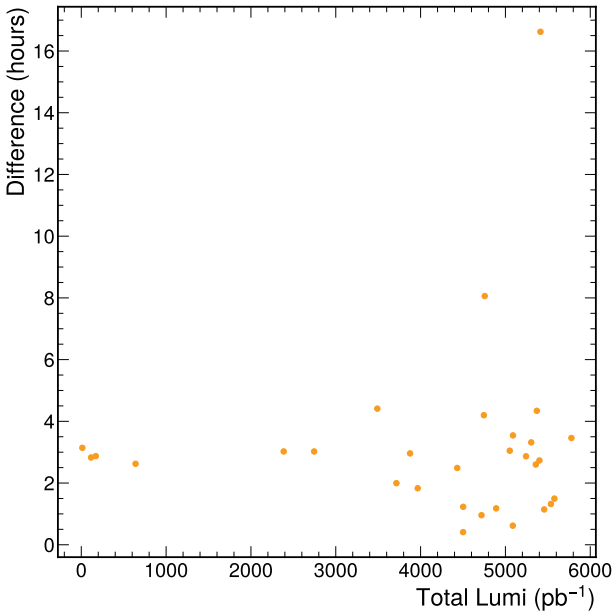

Figure 10.4: Plot of the delay in hours from the completion of the run in the database and the completion of harvesting via the Automation Framework per total luminosity of the fills.

10.3 Summary

This chapter presents the use case of the CMS muon detector for the automated production and harvesting of Zero Bias datasets using the Automation Framework. The muon system faces high background rates from proton-proton interactions, beam-gas interactions, and radiation, which must be carefully characterized for precise physics reconstruction and detector monitoring.

Zero Bias triggers provide an unbiased dataset for background studies, capturing events independently of physics selection criteria. The AF automates the processing of this dataset, producing NanoAOD Ntuples tailored to the muon system, including DT digis, hits, clusters, and segments, as well as relevant CSC, GEM, and RPC information. The automation ensures consistent processing and rapid availability of this data.

The Zero Bias workflow reads run files from the CMS DBS, submits batch jobs to process them in NanoAOD format, and stores outputs in EOS. The output is then merged by fill once all runs are complete, producing a single file ready for analysis. Workflow timing and status are monitored via InfluxDB and visualized through Grafana dashboards, allowing users to track job completion, output locations, and latency from data acquisition.

Overall, the AF streamlines the Zero Bias dataset processing, freeing human resources and providing a reliable foundation for future background studies in preparation for HL-LHC operations.

Conclusions

This thesis has presented advanced computing techniques developed for the CMS experiment to address the computational challenges of the High-Luminosity Large Hadron Collider era.

Part I introduced `patatune`, a flexible Python framework for multi-objective optimization based on the PSO algorithm. The framework successfully tunes complex reconstruction algorithms. It handles numerous mixed-type parameters by automatically exploring high-dimensional parameter spaces and identifies Pareto-optimal solutions to balance competing objectives.

When applied to CMS Patatrack pixel track reconstruction, `patatune` achieved a 50% reduction in fake rate while maintaining reconstruction efficiency, demonstrating clear physics performance improvements. Beyond the performance gains, `patatune` provides an approach to parameter tuning that can prove helpful with the Phase-2 upgrade as detector complexity grows and reconstruction algorithms evolve. The successful application of the framework to the TrackML detector simulation validated its potential as a general-purpose tool for algorithm optimization, proving effective across various detector geometries and problems.

The exploratory work on reinforcement learning looked promising for reducing the computational cost of the optimization. This represents an area of interest for continued development, as the NextGen project increasingly focuses on ML approaches.

Part II provided a comprehensive characterization of CMS RAW data and a systematic evaluation of compression strategies for managing HL-LHC data volumes. The detailed breakdown of Run 3 data and Phase-2 simulations was necessary to characterize the event size, with the Inner and Outer Trackers, and HGCAL as the most significant contributors. The study formed the quantitative baselines for testing compression performance. Benchmarks of lossless algorithms on both TTree and RNTuple formats demonstrated that the new RNTuple data structure offers both improved compression ratios and faster read performance, particularly for structured data, thanks to its columnar design. These results support the ongoing transition to RNTuple as CMS's primary data format for Phase-2.

Exploring GPU-accelerated compression algorithms identified promising implementations. Among them, the BSC algorithm demonstrates its potential for achieving strong compression ratios and throughput that exceed those of ZSTD on heterogeneous hardware. The investigation of lossy compression through the RAW' format achieved approximately 20% event size reduction in proton-proton collisions, suggesting that similar improvements are possible in other subdetectors.

The work presented here establishes the foundation for continued compression research within NextGen Task 3.3. Future directions include deeper integration of GPU compression within ROOT and investigation of compression models that exploit high-level physics objects to model the data.

Part III documented the Automation Framework that has become an essential operational tool for multiple CMS subsystems. It allows processing of hundreds of jobs daily and delivers calibration constants and analysis-ready datasets. The flexibility of the framework, which supports arbitrary executables in containers, complex workflow dependencies, and multiple data sources, has pushed its adoption beyond the original ECAL use case.

Using the Automation Framework for the production of Zero Bias dataset for muon background studies proved effective. The framework is capable of handling analysis workloads with quick turnaround.

The migration from Jenkins to GitLab CI/CD represents a successful initial step in evolving the infrastructure. The effort aligns with CERN IT's strategy and emphasizes the use of more modern and maintainable solutions. While challenges remain in replicating Jenkins's scheduling capabilities, the unified GitLab environment results in safer credential management, reduced maintenance, and better integration with existing development workflows. The Automation Framework can serve as a stepping stone for NextGen Task 3.4's goal of achieving prompt-quality calibration, within hours of data taking, for use directly in HLT reconstruction.

Both `patatune` and the Automation Framework are practical tools now in use in production, directly contributing to the preparation of CMS for HL-LHC.

Future directions

The three technical contributions presented in this thesis address computational challenges through more efficient methods. The first replaces manual parameter tuning with automatic intelligent search, the second reduces data volume through compression rather than increased storage, and the last streamlines repetitive workflows through automation instead of allocating additional human effort. Looking ahead to the start of HL-LHC operations in 2030, several directions for extending this work can be explored.

The `patatune` framework could be integrated more tightly with CMSSW to enable automatic optimization of reconstruction parameters as detector conditions evolve or as new algorithms are developed. The framework's architecture is sufficiently general to optimize parameters for other experiments or tune objectives in any domain with large parameter spaces. The results obtained with the NextGen extension highlight that a large number of parameters makes the reconstruction harder to tune and complicates the identification of a suitable working point within the solution set. Additional effort is therefore needed to define a method to quantify parameter importance and to identify which variables of the tuned algorithm drive the optimization most strongly. To achieve this, one could draw inspiration from machine learning and define a metric analogous to feature importance in weight-based model training.

The compression studies could be extended to include additional algorithms that utilize heterogeneous resources, potentially integrating them into the ROOT framework and training compression specifically on CMS data characteristics. Finally, the Automation Framework could become the orchestration layer for NextGen's fast calibration pipeline.

The techniques presented in this thesis, developed in the context of the NextGen project, contribute to the broader computational efforts of the Phase-2 upgrade of the CMS experiment, aiming at preparing the experiment for future efforts in investigating what lies beyond the Standard Model.

Appendix A

ZDT test functions benchmarks

ZDT2

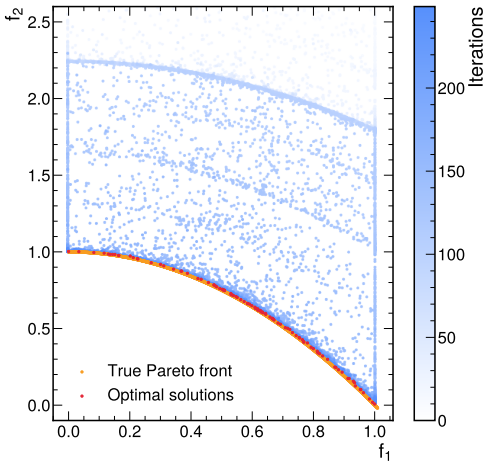

Figure A.1: Optimal solution found by running `patatune` on the $\mathcal{ZDT}_2$ test function. The blue points show the solution found in each iteration. The red points show the archived solution on the last iteration. The orange points correspond to the true Pareto front.

The plots in Figure A.6, at the end of this appendix section, show the values of the performance indicators as a function of the inertial weight w, for the different combinations of cognition and social coefficients, c_1 and c_2, for the $\mathcal{ZDT}_1$ function. The optimization has been run for 150 iterations with 100 agents, limiting the size of the archive of solutions to the 200 solutions with the highest crowding distance. The benchmark shows that the best combination of w, c_1, c_2, with the lowest average and standard deviation for the two distances, is for $w = 1.25$, $c_1 = 2$, $c_2 = 0.5$.

The plot in Figure A.1 shows the solutions obtained by running the algorithm with that configuration.

ZDT3

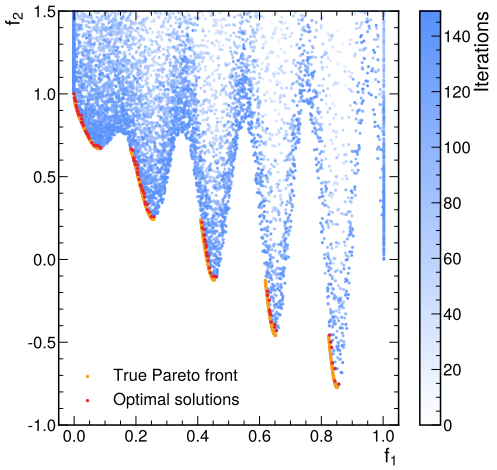

Figure A.2: Optimal solution found by running `patatune` on the $\mathcal{ZDT}_3$ test function. The blue points show the solution found in each iteration. The red points show the archived solution on the last iteration. The orange points correspond to the true Pareto front.

The plots in Figure A.7 show the values of the performance indicators as a function of the inertial weight w, for the different combinations of cognition and social coefficients, c_1 and c_2, for the $\mathcal{ZDT}_3$ function. The optimization has been run for 150 iterations with 100 agents, limiting the size of the archive of solutions to the 200 solutions with the highest crowding distance. The benchmark shows that the best combination of w, c_1, c_2, with the lowest average and standard deviation for the two distances, is for $w = 0.75$, $c_1 = 2$, $c_2 = 0.5$.

The plot in Figure A.2 shows the solutions obtained by running the algorithm with that configuration.

ZDT4

The plots in Figure A.8 show the values of the performance indicators as a function of the inertial weight w, for the different combinations of cognition and social coefficients, c_1 and c_2, for the $\mathcal{ZDT}_4$ function. The optimization has been run for 600 iterations with 100 agents, limiting the size of the archive of solutions to the 200 solutions with

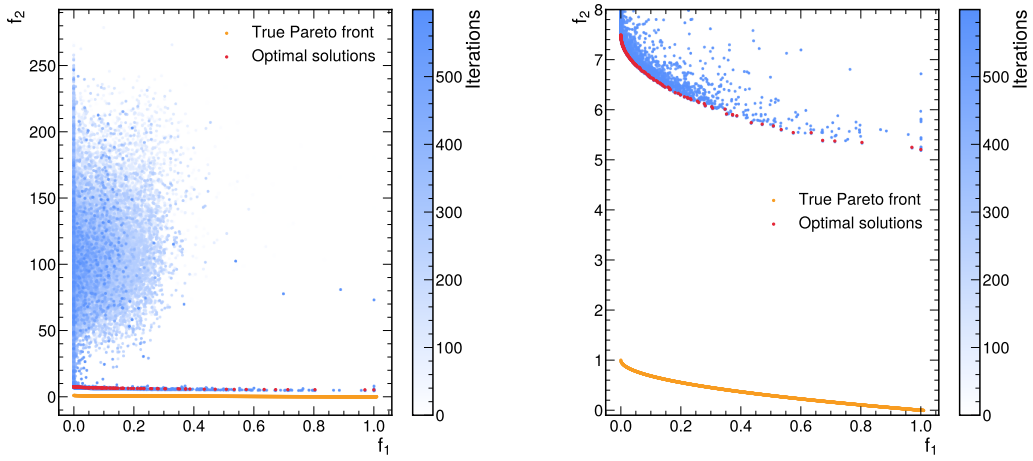

Figure A.3: Optimal solution found by running `patatune` on the $\mathcal{ZDT}_4$ test function. The blue points show the solution found in each iteration. The red points show the archived solution on the last iteration. The orange points correspond to the true Pareto front.

the highest crowding distance. The benchmark shows that the best combination of w, c_1, c_2, with the lowest average and standard deviation for the two distances, is for $w = 0.5$, $c_1 = 2$, $c_2 = 1$.

The plot in Figure A.3 shows the solutions obtained by running the algorithm with that configuration.

ZDT5

The plots in Figure A.9 show the values of the performance indicators as a function of the inertial weight w, for the different combinations of cognition and social coefficients, c_1 and c_2, for the $\mathcal{ZDT}_5$ function. The optimization has been run for 300 iterations with 100 agents, limiting the size of the archive of solutions to the 200 solutions with the highest crowding distance. The benchmark shows that the best combination of w, c_1, c_2, with the lowest average and standard deviation for the two distances, is for $w = 1.5$, $c_1 = 2$, $c_2 = 0.5$.

The plot in Figure A.4 shows the solutions obtained by running the algorithm with that configuration.

ZDT6

The plots in Figure A.10 show the values of the performance indicators as a function of the inertial weight w, for the different combinations of cognition and social coefficients,

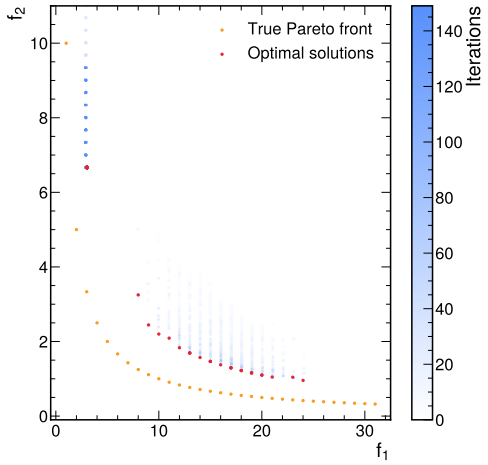

Figure A.4: Optimal solution found by running `patatune` on the $\mathcal{ZDT}_5$ test function. The blue points show the solution found in each iteration. The red points show the archived solution on the last iteration. The orange points correspond to the true Pareto front.

c_1 and c_2, for the $\mathcal{ZDT}_6$ function. The optimization has been run for 300 iterations with 100 agents, limiting the size of the archive of solutions to the 200 solutions with the highest crowding distance. The benchmark shows that the best combination of w, c_1, c_2, with the lowest average and standard deviation for the two distances, is for $w = 0.75$, $c_1 = 2$, $c_2 = 0.5$.

The plot in Figure A.5 shows the solutions obtained by running the algorithm with that configuration.

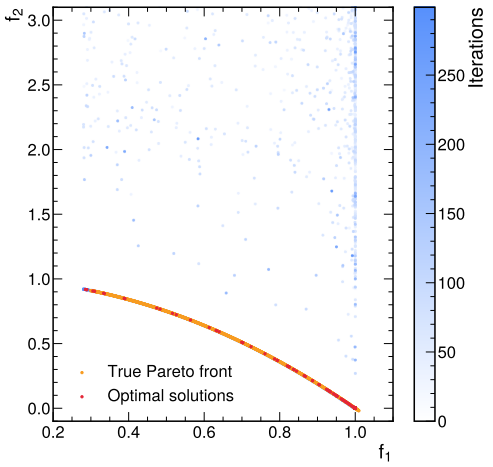

Figure A.5: Optimal solution found by running `patatune` on the $\mathcal{ZDT}_6$ test function. The blue points show the solution found in each iteration. The red points show the archived solution on the last iteration. The orange points correspond to the true Pareto front.

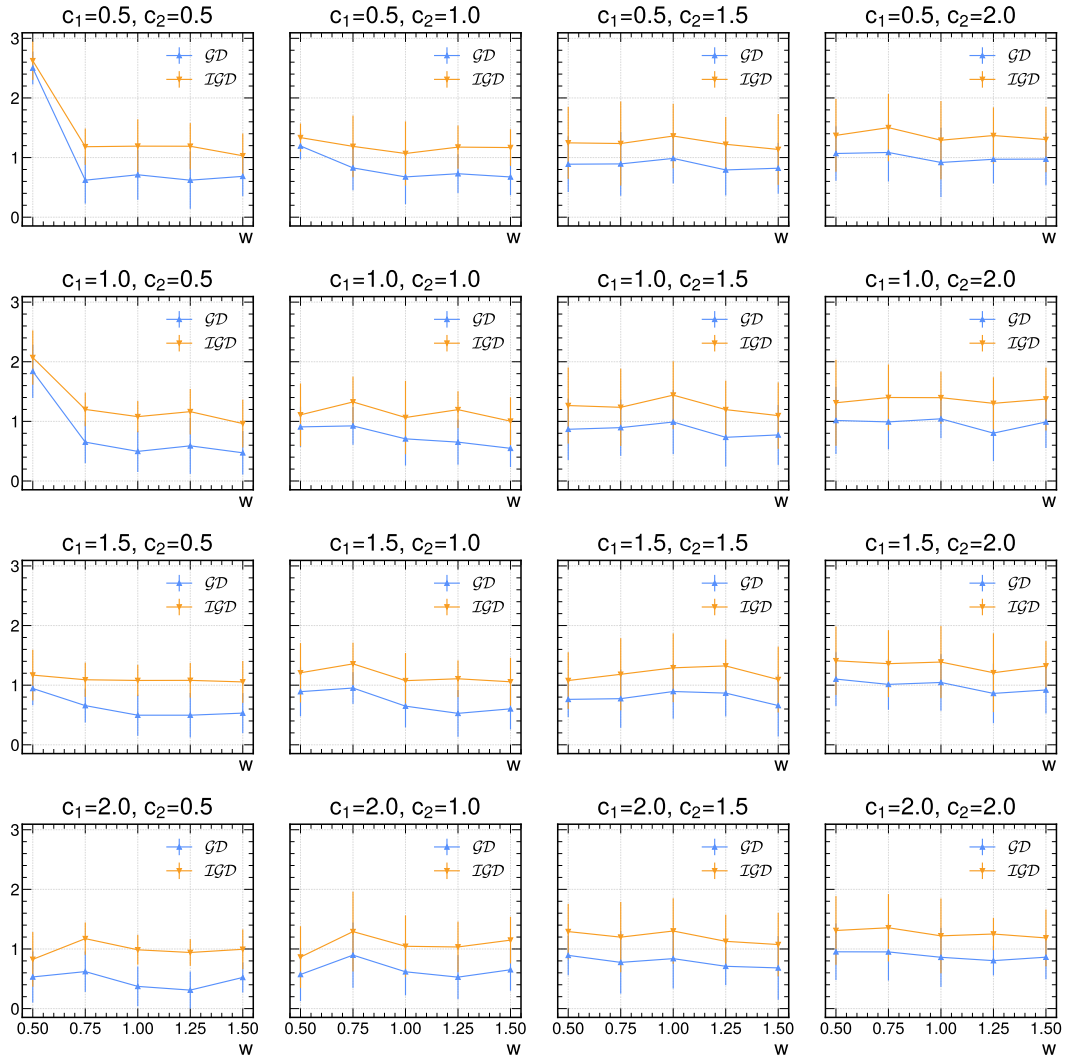

Figure A.6: The value of the $\mathcal{IGD}$ and $\mathcal{GD}$ performance indicators as a function of w, c_1 and c_2 for the $\mathcal{ZDT}_2$ function. The error bars represent the standard deviation of the averages across 10 runs.

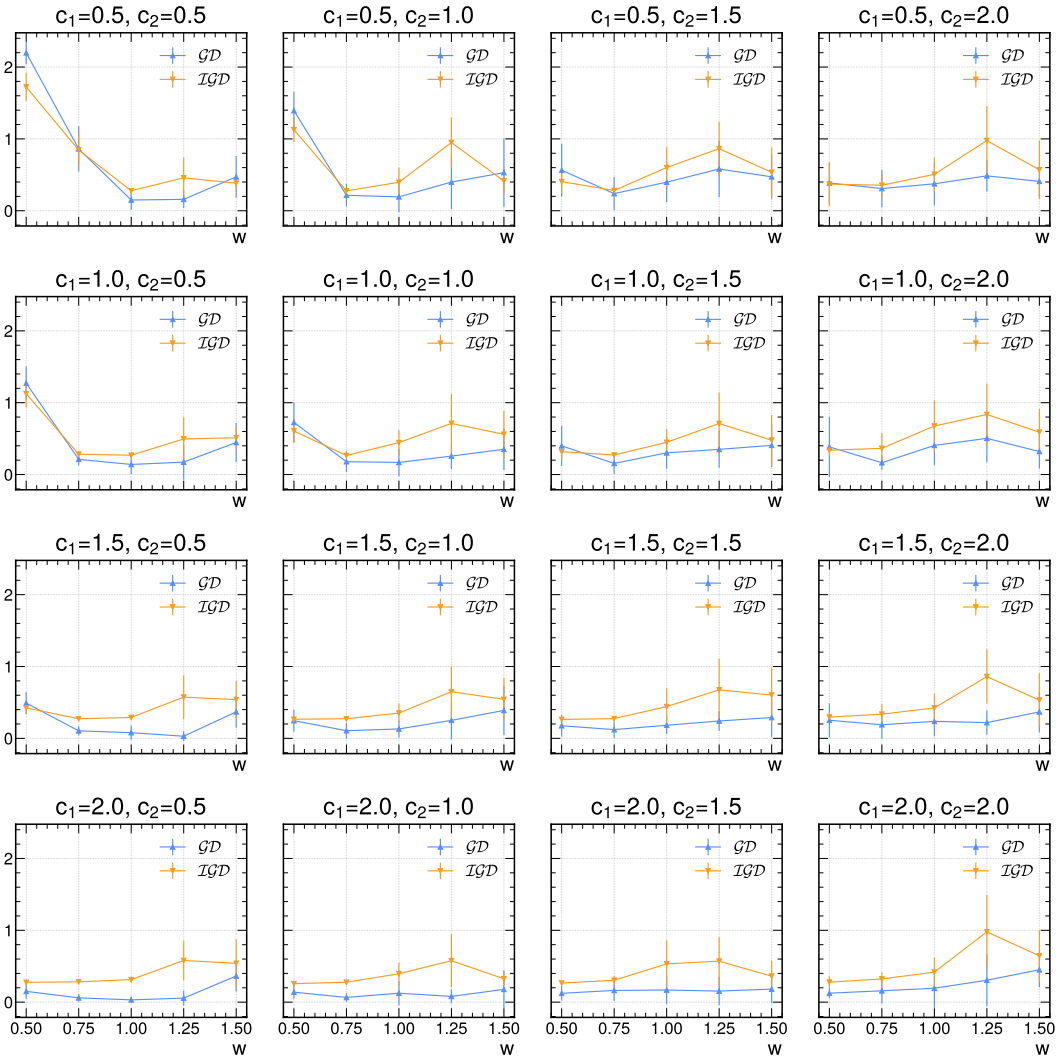

Figure A.7: The value of the $\mathcal{IGD}$ and $\mathcal{GD}$ performance indicators as a function of w, c_1 and c_2 for the $\mathcal{ZDT}_3$ function. The error bars represent the standard deviation of the averages across 10 runs.

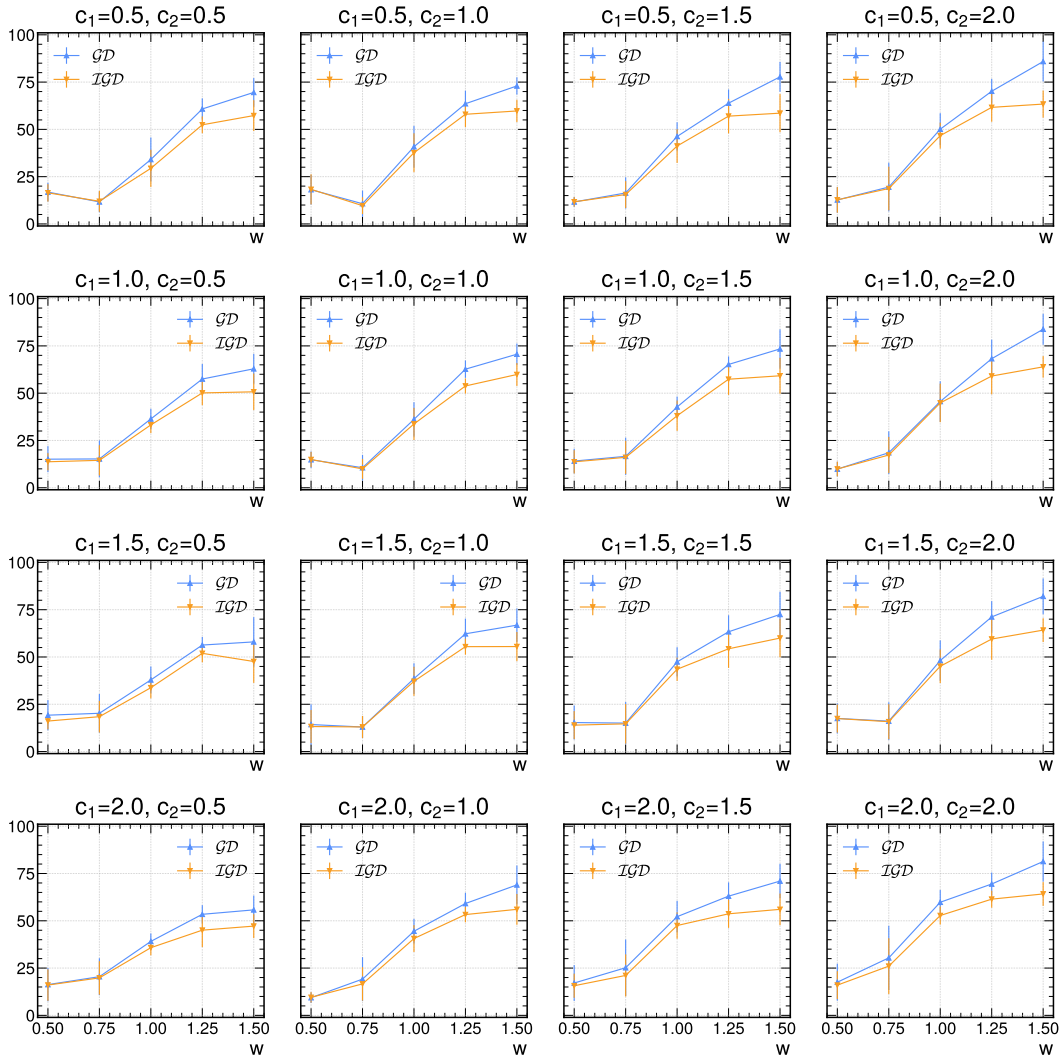

Figure A.8: The value of the $\mathcal{IGD}$ and $\mathcal{GD}$ performance indicators as a function of w, c_1 and c_2 for the $\mathcal{ZDT}_4$ function. The error bars represent the standard deviation of the averages across 10 runs.

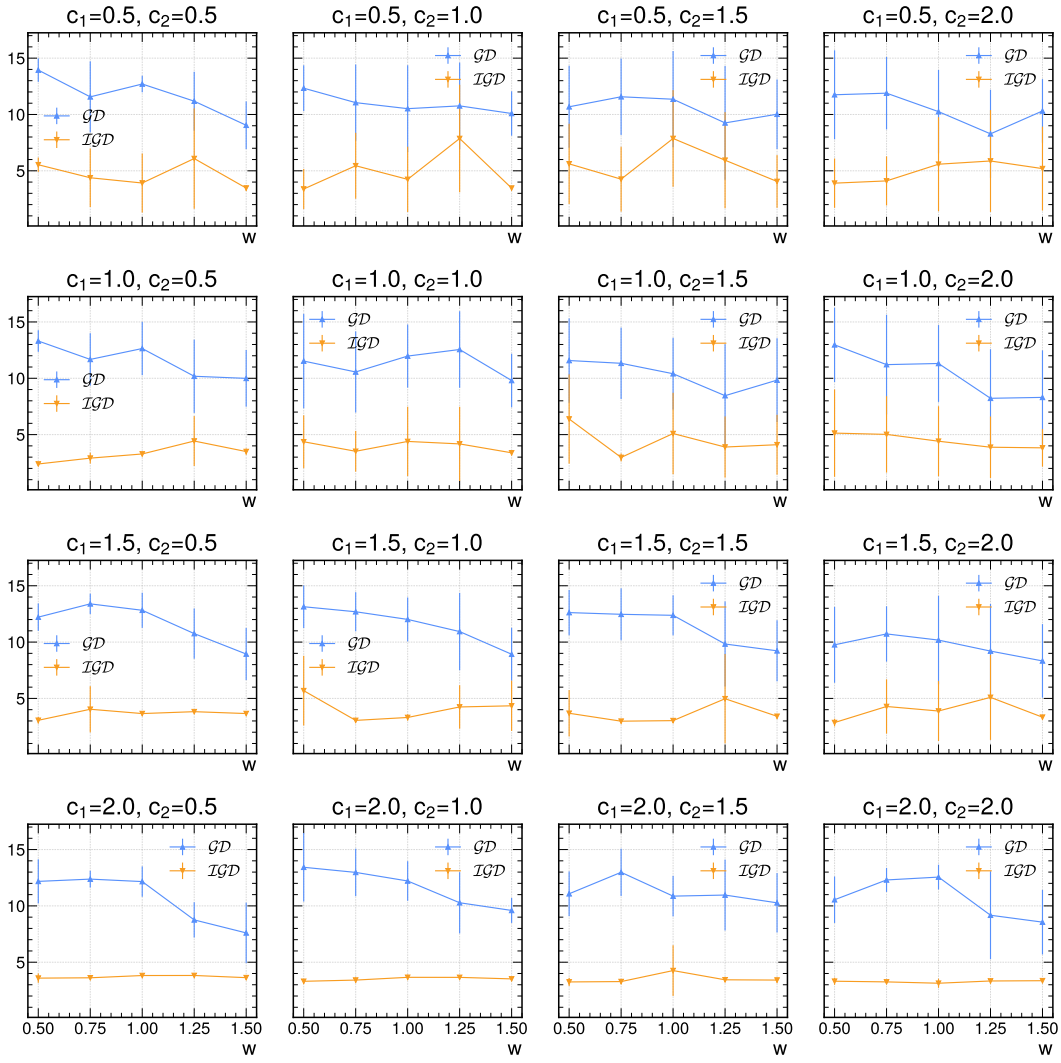

Figure A.9: The value of the $\mathcal{IGD}$ and $\mathcal{GD}$ performance indicators as a function of w, c_1 and c_2 for the $\mathcal{ZDT}_5$ function. The error bars represent the standard deviation of the averages across 10 runs.

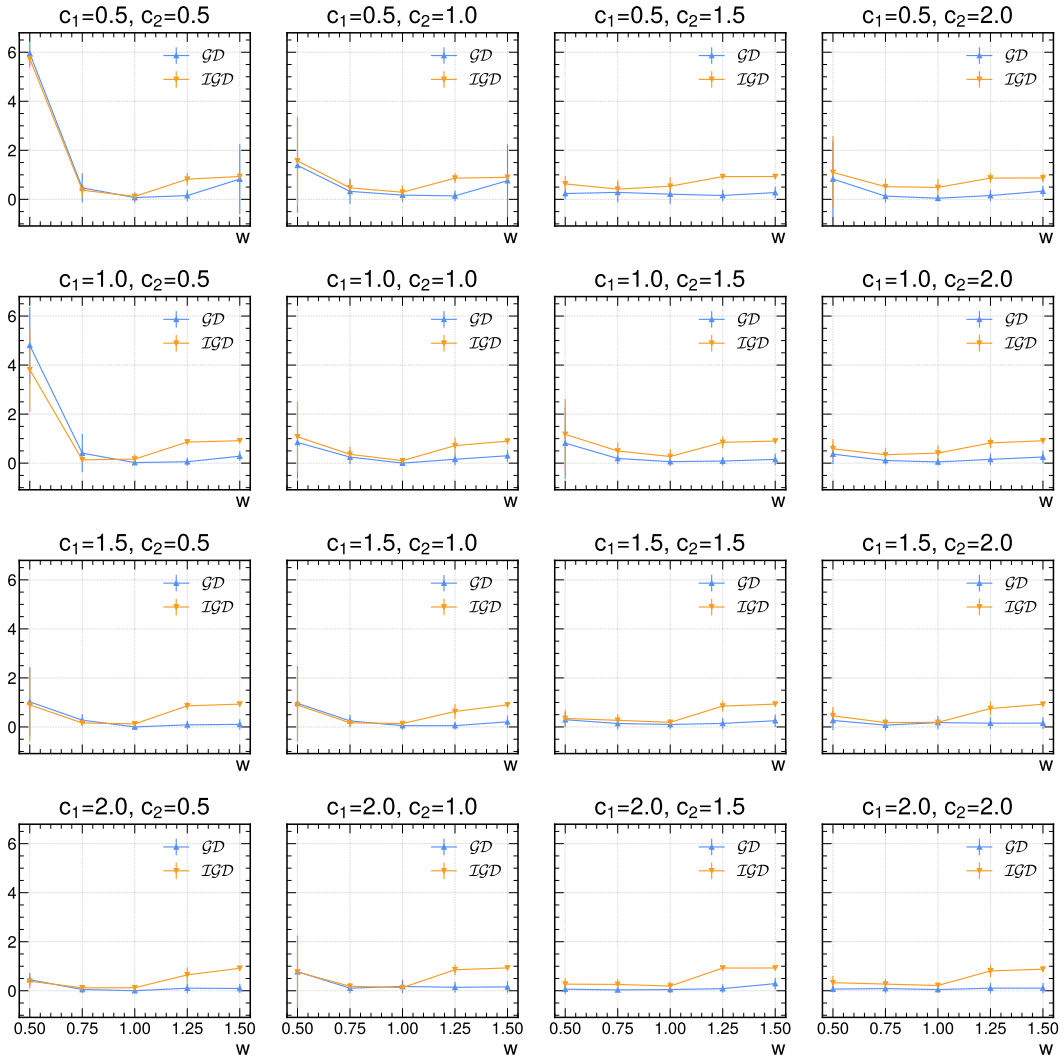

Figure A.10: The value of the $\mathcal{IGD}$ and $\mathcal{GD}$ performance indicators as a function of w, c_1 and c_2 for the $\mathcal{ZDT}_6$ function. The error bars represent the standard deviation of the averages across 10 runs.

Appendix B

Breakdown of subdetectors contribution to RAW Data

Inner Tracker

The Phase-2 Inner Tracker represents a major upgrade over the Run 3 Pixel detector, offering finer granularity and wider coverage to cope with the higher event rates of the HL-LHC. The number of readout channels will rise from about 124 million to roughly 2 billion, and the pseudorapidity acceptance will extend from $|\eta| < 2.4$ to $|\eta| < 4$. The average event size is expected to be about 1.4MB (see Section 7.4.3).

Table B.1 summarizes the average compressed event size for the Pixel detector and the Inner Tracker in Run 3 and Phase-2 simulations. Because no RAW data exist yet for Phase-2, the closest proxy is the size of `simSiPixelDigis`, which should approximate the RAW format.

Table B.1: Comparison of the average compressed size per event, per module, between Pixel detector in Run 3 (PU $\sim$ 60) and Inner Tracker in Phase-2 simulation (PU $\sim$ 200) [180]

Module	**Run 3 Average Compressed Size (kB/Event)**	**Phase-2 Average Compressed Size (kB/Event)**
RAW	283.859	–
siPixelDigis	279.828	–
simSiPixelDigis	1014.335	7010.950
excluding PixelDigiSimLinkedmDetSetVector	278.229	1374.570
siPixelClusters	283.307	–
siPixelRecHits	44.633	147.077

Table B.2: Comparison of the average compressed size per event, per module, between the Strip detector in Run 3 and the Outer Tracker in the Phase-2 simulation [180].

Module	**Run 3 Average Compressed Size (kB/Event)**	**Phase-2 Average Compressed Size (kB/Event)**
RAW	730.8	–
siStripDigis	849.9	–
simSiStripDigis	4582.8	–
excluding StripDigiSimLinkedmDetSetVector	848.9	–
mixTracker	–	704.9
siStripClusters	703.6	–
siPhase2Clusters	–	593.1
siStripMatchedRecHits	333.7	–

For Run 3, the simulated digitization data of the Pixel detector amount to roughly 278kB per event once the `PixelDigiSimLinkedmDetSetVector` objects, used only for Monte Carlo truth, are excluded. This is essentially identical to the RAW event size and to the `siPixelDigis` measurement.

In the Phase-2 simulation, the `simSiPixelDigis` for the Inner Tracker, excluding the same Monte Carlo truth objects, reach about 1375kB per event. This larger size reflects both the higher pileup and the increased detector granularity. No `siPixelDigis` are available yet.

Low-level reconstructed objects

Low-level reconstructed data provide a way to estimate potential savings if reconstructed formats replace RAW data. In Run 3, `siPixelClusters` measure 283kB per event, close to the RAW size, because they retain all pixel ADC counts that record the collected charge. By contrast, `siPixelRecHits` requires only 45kB per event, implying a possible 84% reduction in storage.

For Phase-2, the `siPixelRecHits` occupy about 142kB per event, suggesting that substituting these for the full RAW content could reduce data volume by up to 89%.

Outer Tracker

The Phase-2 Outer Tracker will replace the Run 3 Strip Detector, introducing major improvements to meet the demanding HL-LHC conditions. Its finer granularity allows operation under higher luminosity and pileup. The total number of readout channels will grow from about 10 million to 211 million: the innermost modules will host both pixel sensors (around 170 million channels) and strip sensors (about 11 million chan-

nels), while the outer modules will consist of paired strip sensors (around 31 million channels). Measurements from these module pairs will be combined to provide a 40 MHz track trigger. The expected average event size for the Phase-2 Outer Tracker is about 1.15MB (see Section 7.4.3).

Table B.2 summarizes the average compressed event size in the Strip Detector and Outer Tracker for Run 3 and Phase-2 simulations.

For Run 3, the RAW event size of the Strip Detector is about 731kB per event. `siStripDigis` modules, containing the digitized strip data, measure roughly 850kB per event. Simulated digitization data, `simSiStripDigis`, reach about 849kB per event after removing the MC-truth objects (`StripDigiSimLinkedmDetSetVector`), which is essentially the same as the RAW and `siStripDigis` sizes.

In Phase-2, the `mixTracker` module, which simulates Outer Tracker data, amounts to about 705kB per event. This format temporarily replaces the `simSiStripDigis` data used in Run 3 simulations.

Low-level reconstructed objects

Low-level reconstructed data provides an estimate of the potential savings that could be achieved by replacing RAW data with reconstructed formats. For Run 3, `siStrip Clusters` measure about 704kB per event, as they include all strip ADC counts. `siStripMatchedRecHits`, representing hits reconstructed from clusters, measure about 334kB per event, suggesting a possible 53% reduction compared to RAW data, consistent with the current RAW' shown in Figure 8.10.

In Phase-2, the reconstructed object `siPhase2Clusters` measures about 593kB per event, close to the `mixTracker` size. The Outer Tracker will be fully digitized, recording only binary information per strip rather than deposited charge, which explains why the `siPhase2Clusters` size remains similar to `mixTracker`, unlike the larger compression achieved with `siStripMatchedRecHits` in Run 3.

ECAL

With the upgrade, major changes to the ECAL barrel electronics are planned to improve radiation tolerance and enhance the precision of timing measurements. In addition, the full endcaps will be replaced by the new HGCAL detector covering $1.5 < |\eta| < 3.0$. These modifications lead to a substantial increase in the event data size for Phase-2 simulations compared with Run 3.

Table B.3 reports the average compressed event size in the ECAL detector for both Run 3 and Phase-2 simulations.

Table B.3: Comparison of the average compressed size per event, per module, between Run 3 (PU $\sim$ 60) and Phase-2 simulation (PU $\sim$ 200) in ECAL [180].

Module	**Run 3 Average Compressed Size (kB/Event)**	**Phase-2 Average Compressed Size (kB/Event)**
RAW	36.110	–
simEcalDigis	34.142	66.342
ecalDigis	49.640	71.772
ecalRecHit	38.250	82.409
reducedEcalRecHitsES	22.713	33.310
ecalDetailedTimeRecHit	–	90.808
ecalMultiFitUncalibRecHit	109.796	282.103
particleFlowClusterECAL	15.822	22.691

The expected average event size for the ECAL in Phase-2 is about 0.60MB (see Section 7.4.3). This exceeds the current data volume, despite the removal of the endcap, because of the additional timing information included in Phase-2.

For `simEcalDigis`, the size rises from 34kB per event in Run 3 to 66kB in Phase-2. The `ecalDigis` data, which omit trigger primitives, show a similar growth from 50kB in Run 3 to 71kB in Phase-2.

Low-level reconstructed objects

Low-level reconstructed objects exhibit the same upward trend for both *rechits* and *clusters*. It remains unclear whether storing only reconstructed particle hits could effectively reduce the ECAL event size.

HCAL

In Run 3, the RAW data size for the HCAL is about 142kB per event. This estimate comes from subtracting the combined RAW sizes of the other detectors from the total RAW event size.

Table B.4 presents the average compressed event size in the HCAL detector for Run 3 and Phase-2 simulations.

For Phase-2, it is again helpful to compare simulated digitization data. The `simHcal Digis` size is roughly 125kB per event in Run 3 and decreases to 80kB in Phase-2. Likewise, the `hcalDigis` data, which excludes trigger primitives, drops from 125kB in

Table B.4: Comparison of the average compressed size per event, per module, between Run 3 (PU $\sim$ 60) and Phase-2 simulation (PU $\sim$ 200) in HCAL [180].

Module	**Run 3 Average Compressed Size (kB/Event)**	**Phase-2 Average Compressed Size (kB/Event)**
RAW (Est.)	88.928	–
simHcalDigis	125.167	80.100
hcalDigis	156.758	94.252
slimmedHcalRecHits	–	0.096
reducedHcalRecHits	7.558	4.349
particleFlowClusterHCAL	187.237	3.458

Table B.5: Average compressed size per event, per module, in HGCAL Phase-2 simulation [180].

Module	**Phase-2 Average Compressed Size (kB/Event)**
simHGCalUnsuppressedDigis	2704.907
hgcalDigis	2704.870
HGCalUncalibRecHit	1124.202
HGCalRecHit	1641.559
particleFlowClusterHGCALFromSimCl	3234.980
particleFlowClusterHGCAL	80.972

Run 3 to 94kB in Phase-2. This reduction is partly due to the removal of the endcap detectors covering $|\eta| > 1.5$.

Low-level reconstructed objects

For low-level reconstructed objects, the `reducedHcalRecHits` size decreases from 7.6kB in Run 3 to 4.4kB in Phase-2. The `particleFlowClusterHCAL` object shows an even sharper drop, from 187kB in Run 3 to 3.5kB in Phase-2, a change that warrants further investigation.

HGCAL

HGCAL is the new Calorimeter Endcap, replacing both the ECAL and HCAL endcaps in the region $1.5 < |\eta| < 3.0$ for the Phase-2 upgrade (see Section 1.3.7. The data rate

Table B.6: Average compressed size per event, per module, in MTD Phase-2 simulation [180].

Module	**Phase-2 Average Compressed Size (kB/Event)**
mix_FTLBarrel	501.545
mix_FTLEndcap	115.857
mtdUncalibratedRecHits	859.546
mtdRecHits	165.355
mtdClusters	187.665
mtdTrackingRecHits	21.149

from the HGCAL front ends is still under evaluation, as is the specification of the final data formats.

HGCAL is the largest contributing detector, with an estimated event size of about 3MB (see Section 7.4.3). Table B.5 provides the breakdown of event sizes for each object related to HGCAL.

For simulated digitization, the `simHGCalUnsuppressedDigis` objects reach roughly 2705kB per event. A comparable size is observed for `hgcalDigis` in Phase-2 simulations, as no notable differences appear in the current setup.

Low-level reconstructed objects

For low-level reconstructed objects, the `HGCalUncalibRecHit` objects measure 1124kB per event, whereas the calibrated `HGCalRecHit` objects reach 1642kB. Considerably larger are the `particleFlowClusters` objects, totaling 3316kB per event.

Storing *uncalibrated rechits* instead of RAW data could cut the size by about 58%.

MTD

The MTD is another new component of the Phase-2 upgrade. It adds timing layers to both the barrel and endcap regions of the experiment. Table B.6 lists the average compressed size per event for MTD-related modules in Phase-2 simulations.

In simulated data, the `mix_FTLBarrel` module contributes about 502kB per event, and the `mix_FTLEndcap` module adds roughly 116kB. Together, these digitization layers provide an estimate of the detector's RAW data size.

Low-level reconstructed objects

The MTD's low-level reconstructed objects include uncalibrated and calibrated hits as well as clusters. The `mtdUncalibratedRecHits` objects occupy the largest size at about 860kB per event. By comparison, the calibrated `mtdRecHits` and `mtdClusters` are markedly smaller.

The `mtdTrackingRecHits` module represents the most compact reconstructed objects, and with an average size of 21kB per event, it suggests a potential 96% reduction relative to RAW data.

Bibliography

[1] S. Chatrchyan et al. "Observation of a new boson at a mass of 125 GeV with the CMS experiment at the LHC". In: *Physics Letters B* 716.1 (2012), pp. 30–61. ISSN: 0370-2693. DOI: https://doi.org/10.1016/j.physletb.2012.08.021 (cit. on pp. 1, 9).

[2] G. Aad et al. "Observation of a new particle in the search for the Standard Model Higgs boson with the ATLAS detector at the LHC". In: *Physics Letters B* 716.1 (2012), pp. 1–29. ISSN: 0370-2693. DOI: https://doi.org/10.1016/j.physletb.2012.08.020 (cit. on pp. 1, 9).

[3] *The High Luminosity LHC Project*. Accessed: October 14, 2025. URL: https://hilumilhc.web.cern.ch/content/hl-lhc-project (cit. on pp. 1, 16).

[4] C. O. Software and Computing. *CMS Phase-2 Computing Model: Update Document*. Tech. rep. Geneva: CERN, 2022. URL: https://cds.cern.ch/record/2815292 (cit. on pp. 1, 28).

[5] N. G. Triggers. *Next Generation Triggers - Public proposal*. Dec. 2023. DOI: 10.17181/9jcgn-r3690 (cit. on pp. 1, 7, 35, 41, 123).

[6] C. Collaboration. *The Phase-2 Upgrade of the CMS Data Acquisition and High Level Trigger*. Tech. rep. Geneva: CERN, 2021. URL: https://cds.cern.ch/record/2759072 (cit. on pp. 7, 25, 27, 42, 109, 123, 132).

[7] D. Griffiths. *Introduction to Elementary Particles*. Wiley, 1987. DOI: 10.1002/9783527618460 (cit. on p. 8).

[8] S. Navas et al. "Review of particle physics". In: *Phys. Rev. D* 110.3 (2024), p. 030001. DOI: 10.1103/PhysRevD.110.030001 (cit. on p. 8).

[9] P. W. Higgs. "Broken Symmetries and the Masses of Gauge Bosons". In: *Phys. Rev. Lett.* 13 (16 Oct. 1964), pp. 508–509. DOI: 10.1103/PhysRevLett.13.508 (cit. on p. 9).

[10] F. Englert and R. Brout. "Broken Symmetry and the Mass of Gauge Vector Mesons". In: *Phys. Rev. Lett.* 13 (9 Aug. 1964), pp. 321–323. DOI: 10.1103/PhysRevLett.13.321 (cit. on p. 9).

[11] D. R. Curtin. *Exotic Signatures Theory Overview*. July 2025. DOI: 10.17181/958p4-6f848 (cit. on p. 10).

[12] *Highlights of the HL-LHC physics projections by ATLAS and CMS*. Apr. 2025. arXiv: 2504.00672 [hep-ex] (cit. on p. 10).

[13] C. Botta et al. *BSM benchmarks for Next generation Triggers - CMS*. July 2025. DOI: 10.17181/61mvq-xrg37 (cit. on p. 10).

[14] N. Calace. *BSM benchmarks for Next Generation Triggers - ATLAS*. July 2025. DOI: 10.17181/m75cw-s2y66 (cit. on p. 10).

[15] L. Evans and L. R. Evans. *The Large Hadron Collider: a marvel of technology*. eng. Second edition. Fundamental sciences. Lausanne: EPFL Press, 2018. ISBN: 9782889152827 (cit. on pp. 10, 14, 18).

[16] D. Dominguez. *3D cut of the LHC dipole. Coupe 3D du dipôle du LHC*. 2014. URL: https://cds.cern.ch/record/1741036 (cit. on p. 11).

[17] F. Landua. *The CERN accelerator complex layout in January 2022.* 2022. URL: https://cds.cern.ch/record/2813716 (cit. on p. 12).

[18] D. Froidevaux and P. Sphicas. "General-Purpose Detectors for the Large Hadron Collider". In: *Annual Review of Nuclear and Particle Science* 56.1 (Nov. 2006), pp. 375–440. ISSN: 1545-4134. DOI: 10.1146/annurev.nucl.54.070103.181209 (cit. on p. 12).

[19] A. A. Alves et al. "The LHCb Detector at the LHC". In: *JINST* 3 (2008), S08005. DOI: 10.1088/1748-0221/3/08/S08005 (cit. on p. 13).

[20] ALICE Collaboration. "The ALICE experiment at the CERN LHC". In: *Journal of Instrumentation* 3.08 (Aug. 2008), S08002. DOI: 10.1088/1748-0221/3/08/S08002 (cit. on p. 14).

[21] G. Eulisse and D. Rohr. "The O2 software framework and GPU usage in ALICE online and offline reconstruction in Run 3". In: *EPJ Web Conf.* 295 (2024), p. 05022. DOI: 10.1051/epjconf/202429505022. arXiv: 2402.01205 (cit. on p. 14).

[22] *Minutes of the 250th meeting of the Research Board, held on 18th September 2024*. Tech. rep. Geneva: CERN, 2024. URL: https://cds.cern.ch/record/2910550 (cit. on pp. 15, 16).

[23] M. Lamont and R. Jones. *Chamonix Workshop 2024 summary*. 2024. URL: https://cds.cern.ch/record/2898107 (cit. on p. 16).

[24] O. Aberle et al. *High-Luminosity Large Hadron Collider (HL-LHC): Technical design report*. CERN Yellow Reports: Monographs. Geneva: CERN, 2020. DOI: 10.23731/CYRM-2020-0010 (cit. on pp. 16, 17).

[25] O. Brüning and L. Rossi. *The High Luminosity Large Hadron Collider: new machine for illuminating the mysteries of the universe*. Ed. by O. Brüning. Advanced series on directions in high energy physics. World Scientific, 2024. DOI: 10.1142/13487 (cit. on p. 17).

[26] D. Contardo et al. *Technical Proposal for the Phase-II Upgrade of the CMS Detector*. Tech. rep. Geneva: CERN, 2015. DOI: 10.17181/CERN.VU8I.D59J (cit. on pp. 17, 25, 35).

[27] The HEP Software Foundation et al. "A Roadmap for HEP Software and Computing R&D for the 2020s". In: *Computing and Software for Big Science* 3.1 (Mar. 2019), p. 7. ISSN: 2510-2044. DOI: 10.1007/s41781-018-0018-8 (cit. on p. 17).

[28] M. Brice. *Lowering the YE+1 end-cap for CMS. Expérience CMS: Descente du disque bouchon YE+1 le 9 janvier.* 2007. URL: https://cds.cern.ch/record/1010257 (cit. on p. 18).

[29] *The CMS magnet project*. Technical design report. CMS. Geneva: CERN, 1997. DOI: 10.17181/CERN.6ZU0.V4T9 (cit. on pp. 18, 20).

[30] T. Sakuma and T. McCauley. "Detector and Event Visualization with SketchUp at the CMS Experiment". In: *Journal of Physics: Conference Series* 513.2 (June 2014), p. 022032. DOI: 10.1088/1742-6596/513/2/022032 (cit. on p. 19).

[31] Izaak Neutelings. *CMS coordinate system*. Accessed: October 16, 2025. 2025. URL: https://tikz.net/axis3d_cms/ (cit. on p. 20).

[32] S. Chatrchyan et al. "The CMS experiment at the CERN LHC. The Compact Muon Solenoid experiment". In: *JINST* 3 (2008), S08004. DOI: 10.1088/1748-0221/3/08/S08004 (cit. on p. 19).

[33] *The Phase-2 Upgrade of the CMS Tracker*. Tech. rep. Geneva: CERN, 2017. DOI: 10.17181/CERN.QZ28.FLHW (cit. on pp. 21, 25, 26, 131).

[34] V. Karimaki et al. *The CMS tracker system project*. Technical design report. CMS. Geneva: CERN, 1997. URL: https://cds.cern.ch/record/368412 (cit. on pp. 21, 98).

[35] *The CMS electromagnetic calorimeter project*. Technical design report. CMS. Geneva: CERN, 1997. URL: https://cds.cern.ch/record/349375 (cit. on p. 22).

[36] *The CMS hadron calorimeter project*. Technical design report. CMS. Geneva: CERN, 1997. URL: https://cds.cern.ch/record/357153 (cit. on p. 22).

[37] CMS Collaboration. "Construction and test of the final CMS Barrel Drift Tube Muon Chamber prototype". In: *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 480.2 (2002), pp. 658–669. ISSN: 0168-9002. DOI: https://doi.org/10.1016/S0168-9002(01)01227-X (cit. on p. 23).

[38] G. L. Bayatyan et al. *CMS TriDAS project*. Technical design report. CMS. URL: https://cds.cern.ch/record/706847 (cit. on p. 24).

[39] S. Cittolin, A. Rácz, and P. Sphicas. *CMS The TriDAS Project: Technical Design Report, Volume 2: Data Acquisition and High-Level Trigger. CMS trigger and data-acquisition project*. Technical design report. CMS. Geneva: CERN, 2002. URL: https://cds.cern.ch/record/578006 (cit. on p. 24).

[40] A. Hayrapetyan et al. "Development of the CMS detector for the CERN LHC Run 3. Development of the CMS detector for the CERN LHC Run 3". In: *JINST* 19.05 (2024), P05064. DOI: 10.1088/1748-0221/19/05/P05064. arXiv: 2309.05466 (cit. on p. 24).

[41] CMS Collaboration. *Commissioning CMS online reconstruction with GPUs*. 2022. URL: https://cds.cern.ch/record/2851656 (cit. on pp. 25, 45, 95).

[42] *Performance of CMS pixel tracking at the High-Level Trigger for Phase-2*. 2025 (cit. on pp. 25, 96).

[43] CMS Collaboration. *A MIP Timing Detector for the CMS Phase-2 Upgrade*. Tech. rep. Geneva: CERN, 2019. URL: https://cds.cern.ch/record/2667167 (cit. on p. 26).

[44] *The Phase-2 Upgrade of the CMS Barrel Calorimeters*. Tech. rep. Geneva: CERN, 2017. URL: https://cds.cern.ch/record/2283187 (cit. on p. 26).

[45] *The Phase-2 Upgrade of the CMS Endcap Calorimeter*. Tech. rep. Geneva: CERN, 2017. DOI: 10.17181/CERN.IV8M.1JY2 (cit. on p. 26).

[46] *The Phase-2 Upgrade of the CMS Muon Detectors*. Tech. rep. Geneva: CERN, 2017. DOI: 10.17181/CERN.5T9S.VPMI (cit. on p. 27).

[47] *The Phase-2 Upgrade of the CMS Level-1 Trigger*. Tech. rep. Geneva: CERN, 2020. URL: https://cds.cern.ch/record/2714892 (cit. on p. 27).

[48] E. Meschi. "The CMS Level-1 Trigger Data Scouting system for the HL-LHC upgrade". In: *Proceedings of 42nd International Conference on High Energy Physics*. Vol. 476. 2024, p. 864. DOI: 10.22323/1.476.0864 (cit. on p. 27).

[49] K. Bos et al. *LHC computing Grid: Technical Design Report. Version 1.06 (20 Jun 2005)*. Technical design report. LCG. Geneva: CERN, 2005. URL: https://cds.cern.ch/record/840543 (cit. on p. 28).

[50] M. Aderholz et al. *Models of Networked Analysis at Regional Centres for LHC Experiments (MONARC), Phase 2 Report, 24th March 2000*. Tech. rep. Geneva: CERN, 2000. URL: https://cds.cern.ch/record/510694 (cit. on p. 28).

[51] G. L. Bayatyan et al. *CMS computing: Technical Design Report*. Technical design report. CMS. Geneva: CERN, 2005. URL: https://cds.cern.ch/record/838359 (cit. on p. 29).

[52] *Directory listing of WLCG Outreach*. Accessed: May 22, 2025. URL: https://wlcg-docs.web.cern.ch/?dir=outreach/images (cit. on p. 30).

[53] I. Bird et al. *Update of the Computing Models of the WLCG and the LHC Experiments*. Tech. rep. CERN, 2014. URL: https://cds.cern.ch/record/1695401 (cit. on p. 29).

[54] *CMSSW Application Framework*. Accessed: October 23, 2025. URL: https://twiki.cern.ch/twiki/bin/view/CMSPublic/WorkBookCMSSWFramework (cit. on pp. 30, 32).

[55] R. Brun, F. Rademakers, and S. Panacek. *ROOT, an object oriented data analysis framework*. 2000. URL: https://cds.cern.ch/record/491486 (cit. on pp. 31, 44).

[56] G. Petrucciani, A. Rizzi, and C. Vuosalo. *Mini-AOD: A New Analysis Data Format for CMS. A new analysis data format for CMS*. Tech. rep. 7. Geneva: CERN, 2015. DOI: 10.1088/1742-6596/664/7/072052. arXiv: 1702.04685 (cit. on p. 32).

[57] M. Peruzzi, G. Petrucciani, and A. Rizzi. "The NanoAOD event data format in CMS". In: *J. Phys. : Conf. Ser.* 1525.1 (2020), p. 012038. DOI: 10.1088/1742-6596/1525/1/012038 (cit. on p. 32).

[58] M. Rovere et al. "CLUE: A Fast Parallel Clustering Algorithm for High Granularity Calorimeters in High-Energy Physics". In: *Frontiers in Big Data* Volume 3 - 2020 (2020). ISSN: 2624-909X. DOI: 10.3389/fdata.2020.591315 (cit. on pp. 33, 80).

[59] F. Beaudette. *The CMS Particle Flow Algorithm*. 2013. arXiv: 1401.8155. URL: https://cds.cern.ch/record/1645993 (cit. on pp. 33, 41).

[60] CERN Next Generation Triggers Project. *Next Generation Triggers*. Accessed: July 22, 2025. URL: https://nextgentriggers.web.cern.ch/ (cit. on p. 35).

[61] *NextGen Triggers project. Scientific Policy Committee - 335th Meeting*. Tech. rep. 2023. URL: https://cds.cern.ch/record/2886799 (cit. on p. 36).

[62] A. Del Rosso. *The next-generation triggers for CERN detectors*. Accessed: June 6, 2025. Apr. 2024. URL: https://home.cern/news/news/computing/next-generation-triggers-cern-detectors (cit. on p. 36).

[63] *CERN Open Science Policy*. Tech. rep. Geneva: CERN, 2022. DOI: 10.17181/CERN.RTIF.Z8BO (cit. on p. 36).

[64] CERN, ed. *Work Package 1 Sessions*. Nov. 2024. URL: https://indico.cern.ch/event/1421629/sessions/565680/#20241125 (cit. on p. 36).

[65] FastML Team. *fastmachinelearning/hls4ml*. Version v1.0.0. 2024. DOI: 10.5281/zenodo.1201549 (cit. on p. 37).

[66] J. Duarte et al. "Fast inference of deep neural networks in FPGAs for particle physics". In: *JINST* 13.07 (2018), P07027. DOI: 10.1088/1748-0221/13/07/P07027. arXiv: 1804.06913 [physics.ins-det] (cit. on p. 37).

[67] CERN, ed. *Work Package 2 Sessions*. Nov. 2024. URL: https://indico.cern.ch/event/1421629/sessions/565681/#20241126 (cit. on p. 37).

[68] CERN, ed. *Work Package 3 Sessions*. Nov. 2024. URL: https://indico.cern.ch/event/1421629/sessions/565682/#20241126 (cit. on pp. 39, 41, 48).

[69] M. Migliorini. *40MHz scouting at the CMS experiment*. CERN-THESIS-2024-288. May 2024. URL: https://repository.cern/records/3rrqb-tha31 (cit. on pp. 39, 40).

[70] E. Yigitbasi. *CMS L1 Data Scouting for HL-LHC*. Sept. 2025. DOI: 10.17181/a2h9x-1m527 (cit. on p. 39).

[71] R. Ardino et al. "Design and perspectives of the CMS Level-1 trigger Data Scouting system". In: *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 1067 (2024), p. 169719. ISSN: 0168-9002. DOI: https://doi.org/10.1016/j.nima.2024.169719 (cit. on pp. 40, 46).

[72] L. Sieder. *CMS L1 Data Scouting for HL-LHC*. Oct. 2024. DOI: 10.17181/7954q-kk780 (cit. on p. 40).

[73] M. M. Glowacki. *MLOps Pipeline for Continuous Deployment of Machine Learning Algorithms in the CMS Level-1 Trigger*. Sept. 2025. DOI: 10.17181/ms1gz-pes27 (cit. on p. 40).

[74] S. Summers, I. Bestintzanos, and G. Petrucciani. *Reconstructing jets in the Phase-2 upgrade of the CMS Level-1 Trigger with a seeded cone algorithm*. Tech. rep. Geneva: CERN, 2024. DOI: 10.1051/epjconf/202429502024. arXiv: 2310.08062 (cit. on p. 40).

[75] J. A. Pearkes. *CMS Experience with Anomaly Detection in the L1 Trigger*. Mar. 2025. DOI: 10.17181/m0ydv-dfq02 (cit. on p. 40).

[76] *2024 Data Collected with AXOL1TL Anomaly Detection at the CMS Level-1 Trigger*. 2024. URL: https://cds.cern.ch/record/2904695 (cit. on p. 40).

[77] CERN, ed. *Work Package 4 Sessions*. Nov. 2024. URL: https://indico.cern.ch/event/1421629/sessions/565683/#20241127 (cit. on p. 40).

[78] L. Beltrame et al. *Task 3.1 2024 Report - Current performance and future lines of developments*. Annual Report. NextGen, Oct. 2024 (cit. on pp. 41–43).

[79] A. Bocci et al. "Heterogeneous Reconstruction of Tracks and Primary Vertices With the CMS Pixel Tracker". In: *Frontiers in Big Data* 3 (2020). ISSN: 2624-909X. DOI: 10.3389/fdata.2020.601728 (cit. on pp. 41, 84).

[80] M. Michelotto et al. "A comparison of HEP code with SPEC1 benchmarks on multi-core worker nodes". In: *Journal of Physics: Conference Series* 219.5 (Apr. 2010). DOI: 10.1088/1742-6596/219/5/052009 (cit. on p. 42).

[81] D. Giordano et al. "HEPScore: A new CPU benchmark for the WLCG". In: *EPJ Web of Conf.* 295 (2024). DOI: 10.1051/epjconf/202429507024 (cit. on p. 42).

[82] Next Generation Triggers. *NGT Evaluation Report - 2024*. Jan. 2025. DOI: 10.17181/x37nz-pva81 (cit. on pp. 42, 48).

[83] *Muon Tracking at the CMS High-Level Trigger for the HL-LHC Upgrade*. 2025. URL: https://cds.cern.ch/record/2950077 (cit. on p. 43).

[84] L. Beltrame. *Evolution of data structures for heterogeneous reconstruction in CMSSW*. Sept. 2025. URL: https://indico.cern.ch/event/1488410/contributions/6562855/ (cit. on p. 44).

[85] W. Childers et al. *Reflection for C++26*. Working Paper P2996R13. The C++ Standards Committee, June 2025. URL: https://www.open-std.org/jtc1/sc22/wg21/docs/papers/2025/p2996r13.html (cit. on p. 44).

[86] J. Chen, L. Beltrame, and O. G. Rietmann. *A Unified Interface for Different Memory Layouts*. Sept. 2025. DOI: 10.17181/7m7sv-t2j82 (cit. on p. 44).

[87] C. Zeh. *Efficient Data Movement for Machine Learning Inference in Heterogeneous CMS Software*. Sept. 2025. DOI: 10.17181/2h2ge-mtv31 (cit. on p. 44).

[88] J. Alawieh et al. "Experience with the alpaka performance portability library in the CMS software". In: *EPJ Web Conf.* 337 (2025), p. 01141. DOI: 10.1051/epjconf/202533701141 (cit. on p. 45).

[89] A. Bocci, F. Alazemi, and A. Valenzuela. *Evolving the CMS experiment software into a client-service distributed application for HLT*. Annual Report. NextGen, Oct. 2024 (cit. on pp. 45, 46).

[90] CMS Collaboration. *Data compression via partial online processing in CMS: experience in heavy ions and prospects*. 2023. URL: https://cds.cern.ch/record/2891493 (cit. on p. 46).

[91] CMS Collaboration. *AlCaDB Mandate*. Accessed: July 22, 2025. URL: https://twiki.cern.ch/twiki/pub/CMS/AlCaDB/AlcaDBMandate.pdf (cit. on pp. 47, 153).

[92] G. Cerminara and B. T. L. Van Besien. *Automated workflows for critical time-dependent calibrations at the CMS experiment*. Tech. rep. 7. Geneva: CERN, 2015. DOI: 10.1088/1742-6596/664/7/072009 (cit. on pp. 47, 153).

[93] M. Musich et al. *Task 3.4 2024 Report - Optimal HLT Calibrations*. Annual Report. NextGen, Oct. 2024 (cit. on pp. 47, 48).

[94] P. Cifra et al. "The LHCb HLT2 Storage System: A 40-GB/s System Made of Commercial Off-the-Shelf Components and Open-Source Software". In: *IEEE Trans. Nucl. Sci.* 70.6 (2023), pp. 979–984. DOI: 10.1109/TNS.2023.3240882 (cit. on p. 48).

[95] CERN, ed. *13th Patatrack Hackathon*. June 2023. URL: https://indico.cern.ch/event/1246472/ (cit. on pp. 53, 65).

[96] C. Blum and A. Roli. "Metaheuristics in Combinatorial Optimization: Overview and Conceptual Comparison". In: *ACM Computing Surveys* 35.3 (2003), pp. 268–308. DOI: 10.1145/937503.937505 (cit. on pp. 54, 55).

[97] B. Korte and J. Vygen. *Combinatorial Optimization: Theory and Algorithms*. Springer, 2018. DOI: 10.1007/978-3-662-56039-6 (cit. on p. 54).

[98] É. D. Taillard. "Constructive Methods". In: *Design of Heuristic Algorithms for Hard Optimization*. Springer, 2022, pp. 85–101. DOI: 10.1007/978-3-031-13714-3_4 (cit. on p. 55).

[99] F. Glover and G. A. Kochenberger, eds. *Handbook of Metaheuristics*. 1st ed. International Series in Operations Research & Management Science. Springer, 2003. ISBN: 978-0-306-48056-0. DOI: 10.1007/b101874 (cit. on p. 55).

[100] M. Dorigo and T. Stutzle. *Ant colony optimization*. en. A Bradford Book. Cambridge, MA: Bradford Books, June 2004 (cit. on p. 56).

[101] F. Glover. "Future paths for integer programming and links to artificial intelligence". In: *Computers & Operations Research* 13.5 (1986), pp. 533–549. ISSN: 0305-0548. DOI: https://doi.org/10.1016/0305-0548(86)90048-1 (cit. on p. 56).

[102] H. T. Sadeeq and A. M. Abdulazeez. "Metaheuristics: A Review of Algorithms". In: *International Journal of Online and Biomedical Engineering (iJOE)* 19.09 (July 2023), pp. 142–164. DOI: 10.3991/ijoe.v19i09.39683 (cit. on p. 56).

[103] L. Velasco, H. Guerrero, and A. Hospitaler. "A Literature Review and Critical Analysis of Metaheuristics Recently Developed". In: *Archives of Computational Methods in Engineering* 31.1 (Jan. 2024), pp. 125–146. ISSN: 1886-1784. DOI: 10.1007/s11831-023-09975-0 (cit. on p. 56).

[104] J. Kennedy and R. Eberhart. "Particle swarm optimization". In: *Proceedings of ICNN95 - International Conference on Neural Networks*. Vol. 4. 1995, 1942–1948 vol.4. DOI: 10.1109/ICNN.1995.488968 (cit. on p. 56).

[105] R. Poli. "Analysis of the Publications on the Applications of Particle Swarm Optimisation". In: *Journal of Artificial Evolution and Applications* 2008 (Feb. 2008), pp. 1–10. ISSN: 1687-6229. DOI: 10.1155/2008/685175 (cit. on p. 56).

[106] Y. Shi and R. Eberhart. "A modified particle swarm optimizer". In: *1998 IEEE International Conference on Evolutionary Computation Proceedings. IEEE World Congress on Computational Intelligence*. 1998, pp. 69–73. DOI: 10.1109/ICEC.1998.699146 (cit. on p. 56).

[107] S. Bechikh, R. Datta, and A. Gupta, eds. *Recent Advances in Evolutionary Multi-objective Optimization*. Vol. 20. Adaptation, Learning, and Optimization. Cham: Springer International Publishing, 2017. ISBN: 978-3-319-42977-9. DOI: 10.1007/978-3-319-42978-6 (cit. on p. 58).

[108] M. J. Alves and J. Clímaco. "Multi-objective Mixed Integer Programming". In: *Encyclopedia of Optimization*. Ed. by C. A. Floudas and P. M. Pardalos. Boston, MA: Springer US, 2009, pp. 2454–2460. ISBN: 978-0-387-74759-0. DOI: 10.1007/978-0-387-74759-0_421 (cit. on p. 59).

[109] A. Jaber et al. "A Review on Multi-Objective Mixed-Integer Non-Linear Optimization Programming Methods". In: *Eng* 5.3 (2024), pp. 1961–1979. ISSN: 2673-4117. DOI: 10.3390/eng5030104 (cit. on p. 60).

[110] C. Coello, G. Pulido, and M. Lechuga. "Handling multiple objectives with particle swarm optimization". In: *IEEE Transactions on Evolutionary Computation* 8.3 (2004), pp. 256–279 (cit. on p. 60).

[111] C. R. Raquel and P. C. Naval. "An effective use of crowding distance in multi-objective particle swarm optimization". In: *Annual Conference on Genetic and Evolutionary Computation*. 2005. URL: https://api.semanticscholar.org/CorpusID:504289 (cit. on p. 62).

[112] K. Deb et al. “A Fast Elitist Non-dominated Sorting Genetic Algorithm for Multi-objective Optimization: NSGA-II”. In: *Parallel Problem Solving from Nature PPSN VI*. Ed. by M. Schoenauer et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2000, pp. 849–858 (cit. on p. 61).

[113] *Python Package Index - PyPI*. URL: https://pypi.org/ (visited on 03/28/2021) (cit. on p. 66).

[114] S. Rossi Tisbeni and A. Di Florio. *patatune – A Python library for multi-objective optimization*. Version 1.0.1. Aug. 2025. URL: https://pypi.org/project/patatune/ (cit. on p. 66).

[115] S. K. Lam, A. Pitrou, and S. Seibert. “Numba: A llvm-based python jit compiler”. In: *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC*. 2015, pp. 1–6 (cit. on p. 68).

[116] C. Audet et al. “Performance indicators in multiobjective optimization”. In: *European Journal of Operational Research* 292.2 (Nov. 2020), pp. 397–422. DOI: 10.1016/j.ejor.2020.11.016 (cit. on p. 69).

[117] A. P. Guerreiro, C. M. Fonseca, and L. Paquete. “The Hypervolume Indicator: Computational Problems and Algorithms”. In: *ACM Comput. Surv.* 54.6 (July 2021). ISSN: 0360-0300. DOI: 10.1145/3453474 (cit. on p. 71).

[118] G. Oguare. *Multi-Objective Particle Swarm Optimization*. Aug. 2025. DOI: 10.17181/metgp-ybp72 (cit. on p. 71).

[119] Plotly Technologies Inc. *Dash*. Accessed: August 27, 2025. 2024. URL: https://dash.plotly.com/ (cit. on p. 71).

[120] J. Henschel. *Scalable and multi-tenant Kubernetes ingress infrastructure*. 2024. URL: https://cds.cern.ch/record/2900715 (cit. on p. 73).

[121] A. Pimpo. *Modern web hosting where the web was invented*. Talk at Voxxed Days CERN 2024, CERN. Accessed: July 23, 2025. Jan. 2024. URL: https://www.youtube.com/watch?v=E9lzurdOYdE&t (cit. on p. 73).

[122] OKD Project Community. *OKD 4 Documentation*. Accessed: July 23, 2025. URL: https://docs.okd.io/4.18/ (cit. on p. 73).

[123] K. Deb. “Multi-objective Genetic Algorithms: Problem Difficulties and Construction of Test Problems”. In: *Evolutionary Computation* 7.3 (1999). DOI: 10.1162/evco.1999.7.3.205 (cit. on p. 74).

[124] E. Zitzler, K. Deb, and L. Thiele. “Comparison of Multiobjective Evolutionary Algorithms: Empirical Results”. In: *Evolutionary Computation* 8.2 (2000), pp. 173–195. DOI: 10.1162/106365600568202 (cit. on p. 74).

[125] M. Picione. "Reinforcement learning application in a multi-objective particle swarm optimizer". Accessed: October 15, 2025. MA thesis. University of Milano - Bicocca, 2024. URL: https://drive.google.com/file/d/1QraX1bVZDaPQdIkbbZye7SHYqmRy4_67/view?usp=sharing (cit. on pp. 78, 80).

[126] R. S. Sutton and A. G. Barto. *Reinforcement Learning*. 2nd ed. Adaptive Computation and Machine Learning series. Cambridge, USA: Bradford Books, Nov. 2018 (cit. on p. 78).

[127] J. Schulman et al. *Proximal Policy Optimization Algorithms*. 2017. arXiv: 1707.06347 [cs.LG]. URL: https://arxiv.org/abs/1707.06347 (cit. on p. 79).

[128] F. Pantaleo. *New Track Seeding Techniques for the CMS Experiment*. Mar. 2018 (cit. on p. 84).

[129] D. Funke et al. "Parallel track reconstruction in CMS using the cellular automaton approach". In: *Journal of Physics: Conference Series* 513.5 (June 2014), p. 052010. DOI: 10.1088/1742-6596/513/5/052010 (cit. on p. 84).

[130] V. Blobel. "A new fast track-fit algorithm based on broken lines". In: *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 566.1 (2006), pp. 14–17. ISSN: 0168-9002. DOI: https://doi.org/10.1016/j.nima.2006.05.156 (cit. on p. 84).

[131] *A Multi-Objective Optimization Tool for HLT Pixel Track Reconstruction in CMS*. 2024. URL: https://cds.cern.ch/record/2914423 (cit. on pp. 85–87, 89–95).

[132] CMS Collaboration. *Performance of Track Reconstruction at the CMS High-Level Trigger in 2023 data*. 2024. URL: https://cds.cern.ch/record/2890676 (cit. on p. 88).

[133] S. Rossi Tisbeni et al. "A Multi-Objective Optimization Tool for Track Reconstruction in CMS". In Press. 2025 (cit. on p. 89).

[134] *Performance of Run-3 HLT Track Reconstruction*. 2022. URL: https://cds.cern.ch/record/2814111 (cit. on p. 90).

[135] R. Fruhwirth et al. *Data analysis techniques for high-energy physics*. 2nd ed. Cambridge, England: Cambridge University Press, Mar. 2012 (cit. on p. 98).

[136] A. Bocci et al. "Performance portability for the CMS Reconstruction with Alpaka". In: *Journal of Physics: Conference Series* 2438.1 (Feb. 2023), p. 012058. DOI: 10.1088/1742-6596/2438/1/012058 (cit. on p. 99).

[137] S. Amrouche et al. “The Tracking Machine Learning Challenge: Accuracy Phase”. In: *The NeurIPS ’18 Competition*. Ed. by S. Escalera and R. Herbrich. Cham: Springer International Publishing, 2020, pp. 231–264. ISBN: 978-3-030-29135-8 (cit. on p. 99).

[138] *Kaggle*. Accessed: October 15, 2025. URL: https://www.kaggle.com/ (cit. on p. 99).

[139] A. Hayrapetyan et al. “Enriching the physics program of the CMS experiment via data scouting and data parking”. In: *Phys. Rep.* 1115 (2024), pp. 678–772. DOI: 10.1016/j.physrep.2024.09.006. arXiv: 2403.16134 (cit. on p. 109).

[140] *Data compression via partial online processing in CMS: experience in heavy ions and prospects*. 2023. URL: https://cds.cern.ch/record/2891493 (cit. on pp. 109, 145, 146).

[141] M. Richter. “Data compression in ALICE by on-line track reconstruction and space point analysis”. In: *Journal of Physics: Conference Series* 396.1 (Dec. 2012), p. 012043. DOI: 10.1088/1742-6596/396/1/012043 (cit. on p. 110).

[142] K. Sayood. *Introduction to Data Compression*. en. 5th ed. Oxford, England: Morgan Kaufmann, 2017 (cit. on pp. 110, 111, 117, 119, 142).

[143] C. E. Shannon. “A mathematical theory of communication”. In: *The Bell System Technical Journal* 27.3 (1948), pp. 379–423. DOI: 10.1002/j.1538-7305.1948.tb01338.x (cit. on pp. 111, 112).

[144] A. Kolmogorov. “On tables of random numbers”. In: *Theoretical Computer Science* 207.2 (1998), pp. 387–395. ISSN: 0304-3975. DOI: https://doi.org/10.1016/S0304-3975(98)00075-9 (cit. on p. 113).

[145] *American Standard Code for Information Interchange (ASCII)*. Revised as ANSI X3.4-1986. New York, NY, 1963 (cit. on p. 113).

[146] *International Morse Code*. Geneva, Switzerland, 2009 (cit. on p. 113).

[147] Meteficha. *Huffman tree 2*. Wikimedia Commons. Public Domain. 2007. URL: https://commons.wikimedia.org/wiki/File:Huffman_tree_2.svg (cit. on p. 114).

[148] D. A. Huffman. “A Method for the Construction of Minimum-Redundancy Codes”. In: *Proceedings of the IRE* 40.9 (1952), pp. 1098–1101. DOI: 10.1109/JRPROC.1952.273898 (cit. on p. 114).

[149] N. Faller. “An adaptive system for data compression”. In: *Record of the 7th Asilomar Conference on Circuits, Systems, and Computers*. IEEE, 1973, pp. 593–597 (cit. on p. 115).

[150] D. E. Knuth. “Dynamic Huffman coding”. In: *Journal of Algorithms* 6 (1985), pp. 163–180 (cit. on p. 115).

[151] J. S. Vitter. “Design and analysis of dynamic Huffman codes”. In: *Journal of the ACM* 34.4 (Oct. 1987), pp. 825–845 (cit. on p. 115).

[152] Cmglee. *Arithmetic coding visualisation*. Wikimedia Commons. CC BY-SA 4.0. 2017. URL: https://commons.wikimedia.org/wiki/File:Arithmetic_coding_visualisation.svg (cit. on p. 116).

[153] J. Rissanen and G. G. Langdon. “Arithmetic coding”. In: *IBM J. Res. Dev.* 23.2 (Mar. 1979), pp. 149–162. ISSN: 0018-8646. DOI: 10.1147/rd.232.0149 (cit. on p. 115).

[154] D. Marpe, H. Schwarz, and T. Wiegand. “Context-based adaptive binary arithmetic coding in the H.264/AVC video compression standard”. In: *IEEE Transactions on Circuits and Systems for Video Technology* 13.7 (2003), pp. 620–636. DOI: 10.1109/TCSVT.2003.815173 (cit. on p. 116).

[155] J. Ziv and A. Lempel. “Compression of individual sequences via variable-rate coding”. In: *IEEE Transactions on Information Theory* 24.5 (1978), pp. 530–536. DOI: 10.1109/TIT.1978.1055934 (cit. on p. 117).

[156] J. Ziv and A. Lempel. “A universal algorithm for sequential data compression”. In: *IEEE Transactions on Information Theory* 23.3 (1977), pp. 337–343. DOI: 10.1109/TIT.1977.1055714 (cit. on p. 117).

[157] Welch. “A Technique for High-Performance Data Compression”. In: *Computer* 17.6 (1984), pp. 8–19. DOI: 10.1109/MC.1984.1659158 (cit. on p. 117).

[158] J. A. Storer and T. G. Szymanski. “Data compression via textual substitution”. In: *J. ACM* 29.4 (Oct. 1982), pp. 928–951. ISSN: 0004-5411. DOI: 10.1145/322344.322346 (cit. on p. 117).

[159] I. Pavlov. *LZMA SDK (Software Development Kit)*. Accessed: August 9, 2025. URL: https://www.7-zip.org/sdk.html (cit. on pp. 117, 119, 138).

[160] G. Kramer et al. *Informationstheorie: Komprimierung nach Lempel, Ziv und Welch*. Online. Accessed: October 9, 2025. 2021. URL: https://www.lntwww.de/Informationstheorie/Komprimierung_nach_Lempel,_Ziv_und_Welch (cit. on p. 118).

[161] B. Kwon, M. Gong, and S. Lee. “EDA-78: A Novel Error Detection Algorithm for Lempel-Ziv-78 Compressed Data”. In: *Wireless Personal Communications* 111.4 (Apr. 2020), pp. 2177–2189. ISSN: 1572-834X. DOI: 10.1007/s11277-019-06979-7 (cit. on p. 119).

[162] L. P. Deutsch. *DEFLATE Compressed Data Format Specification version 1.3*. RFC 1951. May 1996. DOI: 10.17487/RFC1951 (cit. on p. 118).

[163] L. P. Deutsch and J.-l. Gailly. *ZLIB Compressed Data Format Specification version 3.3*. RFC 1950. May 1996. DOI: 10.17487/RFC1950 (cit. on pp. 119, 138).

[164] Y. Collet. *LZ4 - Extremely fast compression*. Accessed: August 9, 2025. 2011. URL: https://lz4.org/ (cit. on pp. 119, 138, 142).

[165] Y. Collet and M. Kucherawy. *Zstandard Compression and the 'application/zstd' Media Type*. RFC 8878. Feb. 2021. DOI: 10.17487/RFC8878 (cit. on pp. 119, 138).

[166] A. Robinson and C. Cherry. "Results of a prototype television bandwidth compression scheme". In: *Proceedings of the IEEE* 55.3 (1967), pp. 356–364. DOI: 10.1109/PROC.1967.5493 (cit. on p. 119).

[167] J. L. Bentley et al. "A locally adaptive data compression scheme". In: *Commun. ACM* 29.4 (Apr. 1986), pp. 320–330. ISSN: 0001-0782. DOI: 10.1145/5684.5688 (cit. on p. 119).

[168] F. Ghido. "QLFC - a compression algorithm using the Burrows-Wheeler transform". In: *Data Compression Conference*. 2005, pp. 459–. DOI: 10.1109/DCC.2005.75 (cit. on p. 119).

[169] M. Burrows and D. J. Wheeler. *A Block-sorting Lossless Data Compression Algorithm*. SRC Research Report 124. Palo Alto, CA: Digital Equipment Corporation, Systems Research Center, May 1994. URL: https://www.cs.jhu.edu/~langmea/resources/burrows_wheeler.pdf (cit. on p. 120).

[170] M. Schindler. "A fast block-sorting algorithm for lossless data compression". In: *Proceedings DCC '97. Data Compression Conference*. 1997, pp. 469–. DOI: 10.1109/DCC.1997.582137 (cit. on p. 121).

[171] M. Kufleitner. "On Bijective Variants of the Burrows-Wheeler Transform". In: *CoRR* abs/0908.0239 (2009). arXiv: 0908.0239. URL: http://arxiv.org/abs/0908.0239 (cit. on p. 121).

[172] J. Seward. *bzip2 and libbzip2, version 1.0.8: A program and library for data compression*. Accessed: October 10, 2025. 2019. URL: https://sourceware.org/bzip2/manual/manual.pdf (cit. on p. 121).

[173] I. Grebnov. *High performance data compression library*. Accessed: October 2, 2025. 2025. URL: http://libbsc.com/ (cit. on pp. 121, 143).

[174] CMS Collaboration. *CMSSW_14_2_0_pre1 release on GitHub*. 2023. URL: https://github.com/cms-sw/cmssw/releases/tag/CMSSW_14_2_0_pre1 (cit. on p. 123).

[175] *The LXPLUS Service*. Accessed: October 25, 2025. URL: https://lxplusdoc.web.cern.ch/ (cit. on p. 123).

[176] S. Rossi Tisbeni. *NGT Raw size impact analysis*. Accessed: September 22, 2025. URL: https://github.com/rsreds/NGTRawSizeImpact (cit. on p. 124).

[177] *Using edmEventSize*. Accessed: October 25, 2025. URL: https://twiki.cern.ch/twiki/bin/view/CMSPublic/SWGuideEdmEventSize (cit. on p. 125).

[178] *[NGT] Extend edmEventSize to measure leaf sizes - Pull Request #47248 - cms-sw/cmssw*. Accessed: October 25, 2025. URL: https://github.com/cms-sw/cmssw/pull/47248 (cit. on p. 125).

[179] Carrot Search s.c. *Carrot Search Circles, API reference — get.carrotsearch.com*. Accessed: September 23, 2025. URL: https://get.carrotsearch.com/circles/latest/api/ (cit. on p. 125).

[180] S. Rossi Tisbeni et al. *Task 3.3 2024 Report - Reduction of the RAW data size for HLT*. Annual Report. NextGen, Oct. 2024 (cit. on pp. 128–132, 185, 186, 188–190).

[181] J. Blomer et al. *RNTuple Binary Format Specification 1.0.0.0*. Tech. rep. Geneva: CERN, 2024. URL: https://cds.cern.ch/record/2923186 (cit. on pp. 135, 136).

[182] *Kubernetes Design and Architecture*. Accessed: October 25, 2025. URL: https://github.com/kubernetes/design-proposals-archive/blob/main/architecture/architecture.md (cit. on p. 136).

[183] S. Rossi Tisbeni. *NextGen3.3 Compression Benchmarks*. Accessed: October 1, 2025]. 2025. URL: https://gitlab.cern.ch/srossiti/nextgen33_compression_benchmarks (cit. on p. 136).

[184] CMS Collaboration. *CMSSW FWIO/RNTuple Module*. Accessed: July 24, 2025. 2025. URL: https://github.com/cms-sw/cmssw/tree/CMSSW_15_1_RNTUPLE_X/FWIO/RNTuple (cit. on p. 136).

[185] *Performance Comparison of Lossless Compression Algorithms on CMS Data using ROOT TTree and RNTuple*. 2025. URL: https://cds.cern.ch/record/2941436 (cit. on pp. 137, 139–141).

[186] S. Rossi Tisbeni and S. Donato. *Performance of Lossless Data Compression Algorithms in the CMS Experiment*. Sept. 2025. DOI: 10.17181/8vvy6-3r471 (cit. on pp. 137, 138).

[187] P. Skibinski. *Lzbench - in-memory benchmark of open-source compressors*. Accessed: October 2, 2025. 2025. URL: https://github.com/inikep/lzbench (cit. on p. 141).

[188] N. C.
bibinitperiod Affiliates. *NVIDIA nvCOMP*. Accessed: October 2, 2025. 2024. URL: https://docs.nvidia.com/cuda/nvcomp/index.html (cit. on p. 142).

[189] N. Corporation. *NVIDIA/nvcomp at branch-2.3*. Accessed: October 2, 2025. 2022. URL: https://github.com/NVIDIA/nvcomp/tree/branch-2.3 (cit. on p. 142).

[190] *lzbench Compression Benchmark*. Accessed: October 2, 2025. URL: https://morotti.github.io/lzbench-web/ (cit. on p. 143).

[191] *SiStripClusterTools*. Accessed: October 2, 2025. 2024. URL: https://github.com/cms-sw/cmssw/blob/master/DataFormats/SiStripCluster/interface/SiStripClusterTools.h#L30 (cit. on p. 147).

[192] *hltSiStripClusterizerForRawPrime*. Accessed: October 2, 2025. 2024. URL: https://github.com/saswatinandan/cmssw/blob/saswati/RecoLocalTracker/SiStripClusterizer/test/prehlt.py#L353-L368 (cit. on p. 147).

[193] K. M. Dziedziniewicz et al. "CMS conditions database web application service". In: *J. Phys.: Conf. Ser.* 219 (2010), p. 072048. DOI: 10.1088/1742-6596/219/7/072048 (cit. on p. 153).

[194] S. Pigazzini. *Automatic data processing for prompt calibration of the CMS ECAL*. Tech. rep. Geneva: CERN, 2023. URL: https://cds.cern.ch/record/2853679 (cit. on pp. 154, 157).

[195] *Automation Framework Hackathon 2.0*. Accessed: July 23, 2025. CERN, 2023. URL: https://indico.cern.ch/event/1340676/?print=1 (cit. on pp. 155, 156).

[196] T. Reis. *Automating the CMS ECAL calibration workflows for optimal performance in LHC Run 3*. Sept. 2025. URL: https://indico.cern.ch/event/1488410/contributions/6562854/ (cit. on p. 155).

[197] P. Rebello Teles. "Attaining the optimal performance of the CMS lead tungstate Electromagnetic Calorimeter in the Run3 of LHC". In: *EPS-HEP 2025*. Accessed: July 23, 2025. Marseille, France, 2025. URL: https://indico.in2p3.fr/event/33627/contributions/155324/ (cit. on p. 155).

[198] S. Rossi Tisbeni. *Automation Session*. Nov. 2024. URL: https://indico.cern.ch/event/1468895/#151-automation-session (cit. on p. 155).

[199] M. Albrow et al. *CMS-TOTEM Precision Proton Spectrometer*. Tech. rep. CERN, 2014. URL: https://cds.cern.ch/record/1753795 (cit. on p. 155).

[200] *PPS: performance in Run 3 and efficiency of the pixel detector*. 2024. URL: https://cds.cern.ch/record/2890102 (cit. on p. 155).

[201] T. Ostafin. "Improvement of the Timing Calibration in the CMS PPS Timing Detectors". In: *Acta Phys. Pol. B Proc. Suppl.* 18.5 (2025), 5–A26. DOI: 10.5506/APhysPolBSupp.18.5-A26 (cit. on p. 155).

[202] A. J. Peters and L. Janyst. "Exabyte Scale Storage at CERN". In: *Journal of Physics: Conference Series* 331.5 (Dec. 2011), p. 052015. DOI: 10.1088/1742-6596/331/5/052015 (cit. on p. 156).

[203] E. A. Sindrilaru et al. "EOS developments". In: *J. Phys.: Conf. Ser.* 898.6 (2017). DOI: 10.1088/1742-6596/898/6/062032 (cit. on p. 156).

[204] A. Bellora and T. D. Ostafin. "An automation framework for the calibration of the CMS Precision Proton Spectrometer". In: *Conference on Computing in High Energy and Nuclear Physics (CHEP 2024)*. Accessed: July 23, 2025. Zurich, Switzerland, 2024. URL: https://indico.cern.ch/event/1338689/contributions/6018635/ (cit. on p. 156).

[205] D. Thain, T. Tannenbaum, and M. Livny. "Distributed computing in practice: the Condor experience." In: *Concurrency - Practice and Experience* 17.2-4 (2005), pp. 323–356 (cit. on p. 157).

[206] InfluxData. *InfluxDB*. Accessed: July 22, 2025. URL: https://www.influxdata.com/products/influxdb/ (cit. on p. 157).

[207] Jenkins project. *Jenkins*. Accessed: July 22, 2025. URL: https://www.jenkins.io/ (cit. on p. 158).

[208] GitLab Inc. *GitLab*. Accessed: July 22, 2025. URL: https://docs.gitlab.com/ (cit. on p. 158).

[209] D. Merkel. "Docker: lightweight linux containers for consistent development and deployment". In: *Linux journal* 2014.239 (2014), p. 2 (cit. on p. 158).

[210] CERN Batch Service. *Apptainer (formerly Singularity) with HTCondor container universe at CERN*. Accessed: July 23, 2025. URL: https://batchdocs.web.cern.ch/containers/singularity.html (cit. on p. 158).

[211] L. Promberger et al. "CernVM-FS at Extreme Scales". In: *EPJ Web Conf.* 295 (2024), p. 04012. DOI: 10.1051/epjconf/202429504012 (cit. on p. 158).

[212] E. Bocchi et al. "CernVM-FS powered container hub". In: *EPJ Web Conf.* 251 (2021), p. 02033. DOI: 10.1051/epjconf/202125102033 (cit. on p. 158).

[213] Grafana Labs. *Grafana*. Accessed: July 22, 2025. URL: https://grafana.com/docs (cit. on p. 158).

[214] J. G. Layter. *The CMS muon project*. Technical design report. CMS. Geneva: CERN, 1997. URL: https://cds.cern.ch/record/343814 (cit. on p. 163).

[215] M. Tytgat et al. "Measurement of the background in the CMS muon detector in pp-collisions at $\sqrt{s} = 13$ TeV". In: *The European Physical Journal C* 84.9 (Sept. 2024). ISSN: 1434-6052. DOI: 10.1140/epjc/s10052-024-13077-x (cit. on p. 164).

[216] *Zero bias and HF-based minimum bias triggering for pp collisions at 14 TeV in CMS*. Tech. rep. CERN, 2007. URL: https://cds.cern.ch/record/1152558 (cit. on p. 164).

[217] A. J. Bell. "Beam & Radiation Monitoring for CMS". In: *2008 IEEE Nuclear Science Symposium Conference Record*. 2008, pp. 2322–2325. DOI: 10.1109/NSSMIC.2008.4774822 (cit. on p. 164).

[218] M. Giffels, Y. Guo, and D. Riley. *Data Bookkeeping Service 3 - Providing event metadata in CMS*. Tech. rep. Geneva: CERN, 2014. DOI: 10.1088/1742-6596/513/4/042022 (cit. on p. 165).

[219] J.-M. André et al. *Presentation layer of CMS Online Monitoring System*. Tech. rep. Geneva: CERN, 2019. DOI: 10.1051/epjconf/201921401044 (cit. on p. 166).

[220] I. P. Trobo. *GitLab application experiencing a DOS attack*. CERN Service Portal Service Incident. Accessed: October 25, 2025. Sept. 2025. URL: https://cern.service-now.com/service-portal?id=outage&n=OTG0157579 (cit. on p. 167).