ABSTRACT

RNTUPLE FOR ATLAS ANALYSIS WORKFLOWS

Fatima Rodriguez, M.S.
Department of Physics
Northern Illinois University, 2025
Hector de la Torre Perez, Director

RNTuple is the new data storage format set to replace TTree at the start of the High-Luminosity LHC. An investigation was conducted to evaluate how analysis workflows for ATLAS researchers will change with RNTuple, using reading speed, writing speed, disk space, and memory consumption as metrics. In this study, improvements in all metrics were observed using converted RNTuple inputs from ATLAS Open Data, compared to their TTree equivalents. Additionally, RNTuples produced with the LZ4 compression algorithm were generated and compared with those produced using ZSTD. Finally, two new versions of the Analysis Grand Challenge (AGC) using ATLAS Open Data were completed for TTree and RNTuple inputs with RDataFrame in Python. This constitutes the first implementation of an end-to-end analysis completed using RNTuple.

NORTHERN ILLINOIS UNIVERSITY
DE KALB, ILLINOIS

DECEMBER 2025

RNTUPLE FOR ATLAS ANALYSIS WORKFLOWS

BY

FATIMA RODRIGUEZ

A THESIS SUBMITTED TO THE GRADUATE SCHOOL

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE

MASTER OF SCIENCE

DEPARTMENT OF PHYSICS

Thesis Director:
Hector de la Torre Perez

16

ACKNOWLEDGEMENTS

I would first like to thank my advisors: Hector, Peter, Walter, and Serhan. I feel incredibly fortunate to have had outstanding advisors over the years. I am deeply grateful for your guidance, support, and patience throughout this project. I am also very grateful for my family: Haydee, Jose Alfredo, Saul and Vivianita Rodriguez. I am very proud to be your daughter and older sister. Thank you all for encouraging me to be brave and to pursue this passion. Los quiero mucho! Many thanks to my fanbase: Mariela, Audrey, Jorge, Susan, Syriah, the Garcia Family, Emily, et al. I am so lucky to be surrounded by your beautiful friendships! Thank you all for hyping me up throughout my career. Thank you to my big sister and mentora, Dr. Arcelia Hermosillo! Meeting you was significant not only for my career but also for the friendship that I will always cherish. Thank you so much to Dr. Brendan Kiburg, Dr. Saskia Charity, and Fermilab friends for bringing me to Illinois and solidifying my passion in high energy physics! Big shout out to Net Force, the PGSA, and colleagues. It was an honor to serve as your volleyball coach! Thank you for going along with my social event ideas. As one can see, it took many communities for this accomplishment. Additional shout outs to those that propelled me here and continue to support others like me: Cal NERDs, Hispanic Engineers and Scientists, and the Latin American Graduate Student Association. Finally, I would like to acknowledge $(CP)^2$ for funding the work done in this thesis.

36

DEDICATION

Para mis padres, con toda mi gratitud por su amor y apoyo.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1
INTRODUCTION

Our current understanding of the building blocks of our universe is summarized with one model, called the Standard Model (SM) [1]. From the way we power our cities, to the particles that hold them together, the SM explains how the basic building blocks of matter interact, governed by fundamental forces: electromagnetism, the strong force and the weak force. Yet, questions remain about the SM, such as is there a unification theory that includes gravity? Why are there only three generations of fundamental particles? What is the nature of dark matter and dark energy, and how does it fit within the SM? What about the origin of the matter-antimatter asymmetry? Is the SM complete or do other exotic particles exists? Over the years, experimental particle physicists and engineers have built technology to test the SM, either by performing precision measurements of particles and their behaviors, or by colliding particles and measuring their outputs. As a result, we have increased our confidence in the SM theory, but continue to search for answers for these remaining questions through experimental discovery.

A Toroidal LHC Apparatus (ATLAS) [2] is a particle physics experiment designed to detect the high-energy particle collisions from the Large Hadron Collider (LHC) [3]. At the LHC, collisions take place at a rate of more than a billion interactions per second, which is a combined data volume of about 60 million megabytes per second [4]. In order to extend its discovery potential, the LHC will have a major upgrade to increase the number of instantaneous collision rate. This upgrade, called the High-Luminosity LHC (HL-LHC) [5], will require a new data storage format that can handle this increase in data.

RNTuple [6] is the new ROOT [7] data storage format that will be in use at the start of the HL-LHC [8]. RNTuple takes advantage of modern C++ techniques, which have shown to improve read speed ability and memory usage when compared to its predecessor, TTree, and other data storage formats such as HDF5 and Parquet [9]. RNTuple is currently under heavy development. Its base format has only recently left the experimental stage and many tools and capabilities built around it are still evolving.

This thesis investigates the performance of RNTuple for ATLAS analysis workflows. This chapter will provide a more detailed introduction of the SM and physical quantities relevant to this thesis. An introduction to the ATLAS experiment and its detector technology is provided in Chapter 2. In Chapter 3, the ATLAS software and computing system is introduced along with an explanation of the RNTuple and TTree format. Performance studies conducted for RNTuple and how they compare with TTree will be presented in Chapter 4. In Chapter 5, the Analysis Grand Challenge (AGC) [10] is presented along with its RNTuple implementation. Finally, conclusions are given in Chapter 6.

1.1 Phenomenology at the LHC

The SM is a quantum field theory that explains and categorizes all observed fundamental particles by their properties and interactions. Quantum field theory (QFT) is the main theoretical tool for describing particle interactions by combining special relativity and quantum mechanics. Due to this combination, QFT is a probabilistic theory where each particle has an associated field that permeates all of space; therefore, forces are simply the interactions between these different fields. For example, the electromagnetic force is the interaction between the electromagnetic field and charged matter fields, which fall under quantum electrodynamics (QED). In sum, the SM encompasses all known elementary particle interactions, except for gravity, through a collection of quantum field theories: QED, the Glashow-Weinberg-Salam theory of electroweak processes, and quantum chromodynamics.

The four groups of particles shown in Figure 1.1: quarks, leptons, gauge bosons, and scalar bosons, can be further categorized as *fermions* or *bosons* because of a fundamental property called spin. Similar to the Earth, particles carry orbital angular momentum and spin angular momentum; however, for particles, spin is an intrinsic property. All bosons carry an integer spin; while, fermions carry half-integer spin.

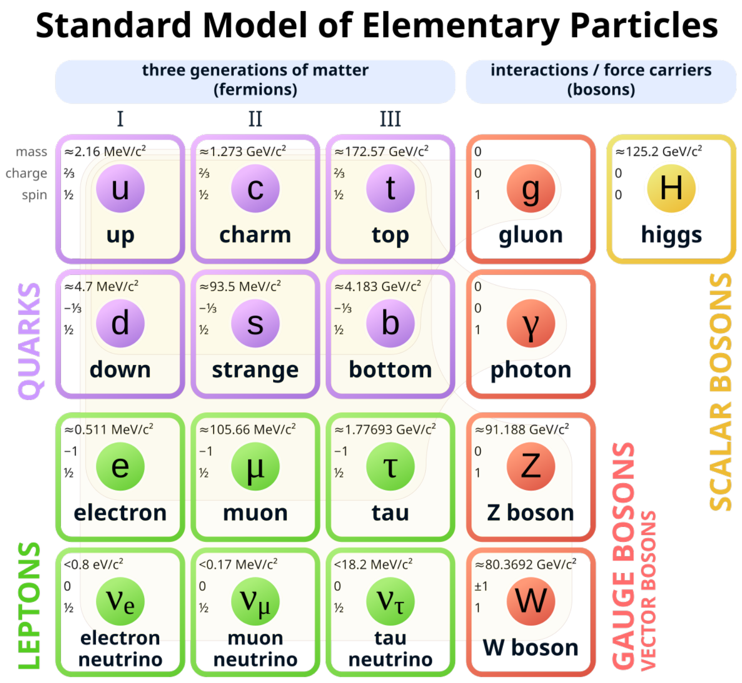

Standard Model of Elementary Particles
three generations of matter (fermions)
interactions / force carriers (bosons)
I
II
III
mass
charge
spin
≈2.16 MeV/c² ⅔ ½ u up
≈1.273 GeV/c² ⅔ ½ c charm
≈172.57 GeV/c² ⅔ ½ t top
0 0 1 g gluon
≈125.2 GeV/c² 0 0 H higgs
≈4.7 MeV/c² −⅓ ½ d down
≈93.5 MeV/c² −⅓ ½ s strange
≈4.183 GeV/c² −⅓ ½ b bottom
0 0 1 γ photon
≈0.511 MeV/c² −1 ½ e electron
≈105.66 MeV/c² −1 ½ μ muon
≈1.77693 GeV/c² −1 ½ τ tau
≈91.188 GeV/c² 0 1 Z Z boson
<0.8 eV/c² 0 ½ νe electron neutrino
<0.17 MeV/c² 0 ½ νμ muon neutrino
<18.2 MeV/c² 0 ½ ντ tau neutrino
≈80.3692 GeV/c² ±1 1 W W boson
QUARKS
LEPTONS
GAUGE BOSONS
VECTOR BOSONS
SCALAR BOSONS

Figure 1.1: Particle content of the Standard Model [11].

Fermions are the particles that make up matter. Each fermion has an antiparticle with the same mass and lifetime as the particle itself, but are oppositely charged. The three charged leptons (e, μ, τ) are massive, while their corresponding neutrinos (ν_e, ν_μ, ν_τ), are treated as massless with neutral charge. Quarks combine to form composite particles, such as protons and neutrons, which are collectively called hadrons. There are six flavors or types of quarks (up, down, strange, charm, top, and bottom), each of which carries an intrinsic property called color charge (red, green and blue).

Bosons mediate the interactions between fermions. Gluons interact with quarks through the strong nuclear force. Photons and the $W^{\pm}Z$ bosons interact with leptons (and quarks), giving rise to the electromagnetic and weak nuclear forces. The Higgs boson is separately categorized as a scalar boson because unlike the spin-1 vector bosons, it has spin 0 and is responsible for giving other elementary particles their mass.

In sum, there are a total of 12 leptons including their antiparticles, 12 quarks, also including their antiparticles, 5 vector bosons, and 1 scalar boson, which makes a total of 30 fundamental particles. Together, these particles and their interactions form the complete framework of the SM of particle physics.

Collider experiments probe the SM by studying the products of collisions between fundamental particles. In colliders, two particle beams are accelerated to reach high energies and brought together for collision. Each collision is called an event and specific interactions or transformations are called processes. Processes are governed by conservation laws, such as conservation of energy and charge, which follow the interactions and rules described within the SM. Around collision points, particle detectors are built to detect the particles produced from events. These detectors are complex and composed of different parts that allow particles to interact with by either ionizing material or by depositing energy such that it produces a signal. The measured signals are then used to reconstruct and classify the particles and processes. Through QFT, the rate of a process, called cross-sections, can be predicted via the kinematics of the particles involved, their properties, and the properties of the process. Experimentally, cross-sections can be calculated via Equation 1.1, where N is the number of events for the process being measured and L is the instantaneous luminosity, defined in Equation as 1.2.

$$\sigma = \frac{N}{\int L dt} \tag{1.1}$$

$$L = f\frac{n_1 n_2}{4\pi\sigma_x\sigma_y} \tag{1.2}$$

f is the frequency of collisions, n_1 and n_2 are the number of particles in the colliding bunches. σ_x and σ_y are the root-mean-squared horizontal and vertical beam sizes. Figure 1.2 displays the predicted cross-sections for certain processes and the required center of mass energies for those processes to be observed. Processes with smaller cross-sections are considered rare-processes because they have a lower probability of being observed. Increasing the probability of these rare-processes would require an increase of energy.

1.2 Physics Quantities

This section will cover some relevant physics quantities used in this thesis.

1.2.1 Invariant Mass

Invariant mass is a quantity that characterizes a system's total energy and momentum independent of the overall motion of the system [13]. Due to special relativity, space and time coordinates are linked, but dependent on a frame of reference. Lorentz transformations are used to convert coordinates from one reference frame to another, and four-vectors are used to simplify these transformations [14]. A four-vector represents a physical quantity in space-time. For example, the position four-vector includes the spatial coordinates (x, y, z) and time, while the four-momentum vector includes the energy and the momentum

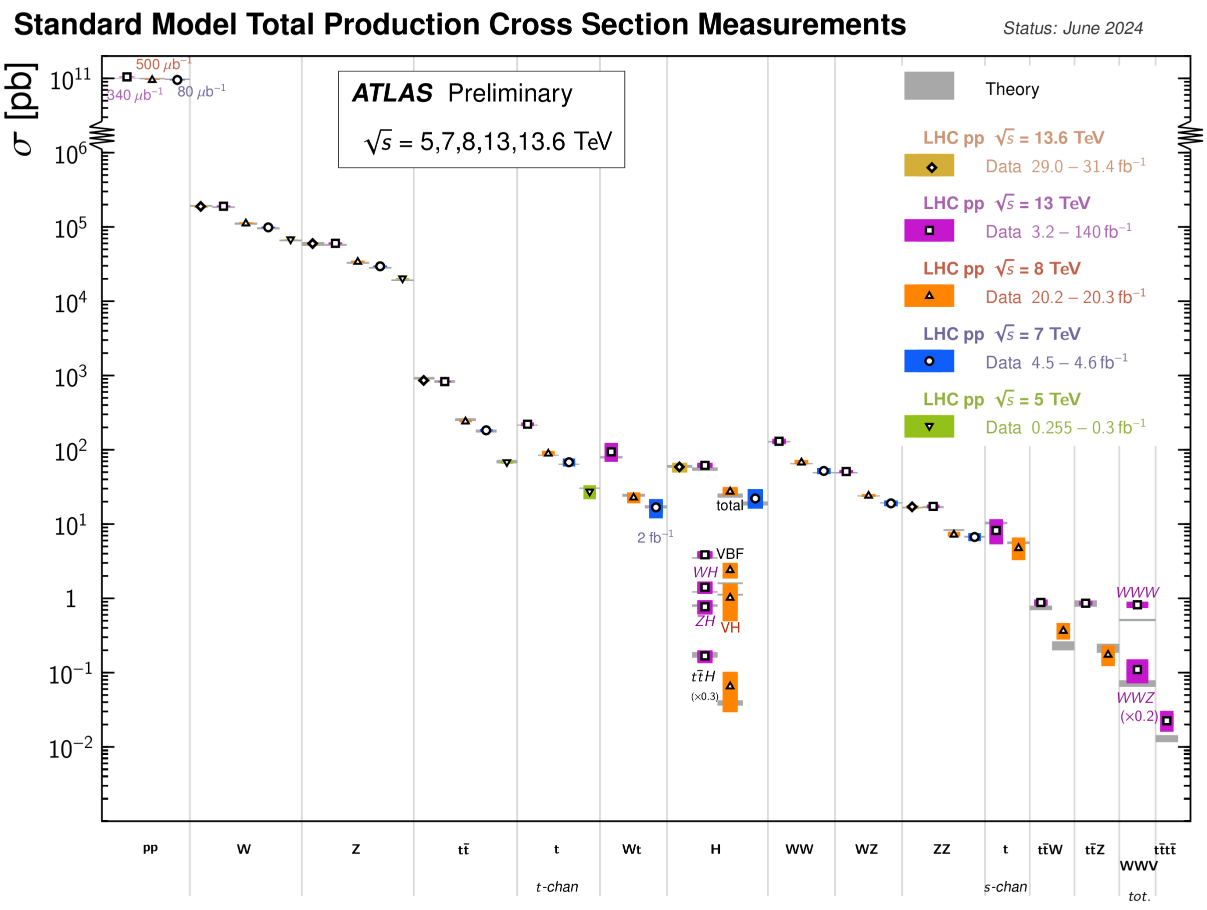


Figure 1.2: Summary of several Standard Model cross-section measurements. The associated references can be found in [12]. The measurements are corrected for branching fractions, compared to the corresponding theoretical expectations.

coordinates in the x, y, and z directions. Four-vectors provide a convenient framework for calculating invariant quantities such as the invariant mass of a resonance that has decayed into other particles.

The invariant mass of oppositely charged muons and electrons is calculated for studies in Chapter 4. A lepton selection with transverse momentum greater than 25 GeV is applied first to suppress background, followed by the pairing of oppositely charged leptons. Their invariant mass is then calculated using Equation 1.3, where p_x, p_y, p_z is momentum in the x, y, z directions and E is energy. The peak of the invariant mass distribution, Figure 1.3, returns the Z boson mass at 75.1338 GeV.

$$m = \sqrt{\sum E^2 - \sum p_x^2 - \sum p_y^2 - \sum p_z^2} \tag{1.3}$$

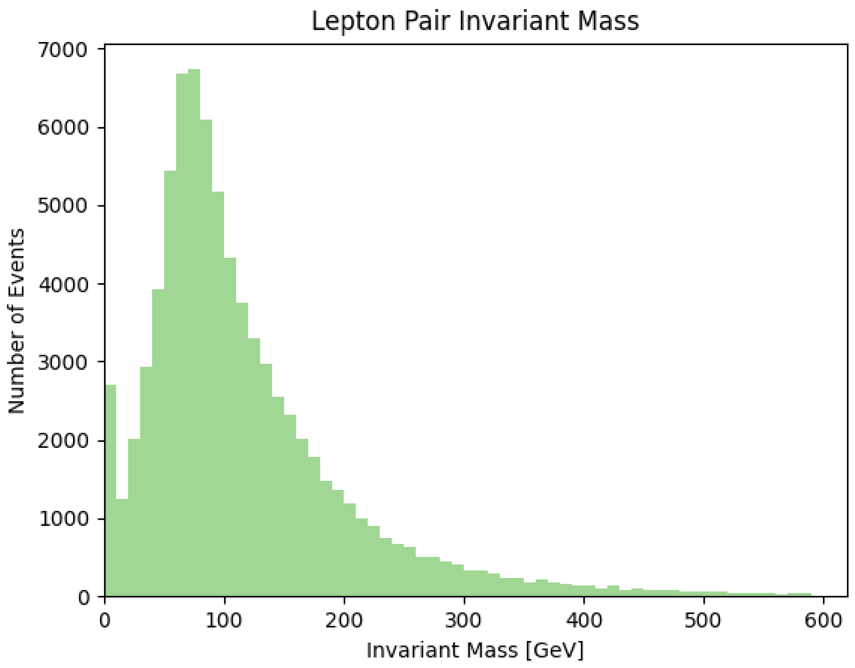


Figure 1.3: Invariant Mass distribution of oppositely charged lepton pairs using data highlighted in Chapter 3.

1.2.2 Jets

Jets are energy deposits in the detector that are grouped together to represent quarks and gluons, collectively known as partons [15]. Partons cannot be observed in isolation due to their color charge property, causing them to be permanently bound inside hadrons (a composite subatomic particle made of two or more quarks [16]). As a result, jets are used as physical proxies for partons and are reconstructed using various algorithms to different types of objects.

In Chapter 5, a selection of b-tagged jets is applied for the AGC. A b-tagged jet is a jet that is identified as being originated by a bottom or anti-bottom quark. B-tagging algorithms use a combination of tracking and vertexing variables that exploit the long lifetime of B-hadrons. In the studies presented in this thesis, a machine learning algorithm based on a deep neural network (DL1) is used. The output of the algorithm are the p_b, p_c, and p_u variables that are combined in Equation 1.4, where f_c is a constant equal to $f_c = 0.018$. The final b-tagging discriminate is defined as D_{DL1}. A jet is considered as b-tagged if D_{DL1} is above the threshold value of 2.456, corresponding to an efficiency of 77% [17].

$$D_{DL1} = \log\left(\frac{p_b}{f_c \times p_c + (1 - f_c) \times p_u}\right) \tag{1.4}$$

CHAPTER 2
THE ATLAS EXPERIMENT

ATLAS was designed to be a general-purpose experiment, optimized to search for the Higgs boson, top quark decays, and supersymmetry. In July 1997, the ATLAS Experiment was approved and by November 2008, ATLAS was the largest detector ever constructed at 44 meters long and 25 meters in diameter. By November 2009, ATLAS recorded its first proton-proton collision and by December 2010, ATLAS was first to observed the production of top quark pairs, which are the heaviest known elementary particle with a strong coupling to the Higgs boson. By July 2012, both ATLAS and the Compact Muon Spectrometer (CMS) experiment successfully observed the Higgs boson [18, 19]. ATLAS is projected to continue operation until 2041 to continue searching for standing questions from the SM.

This chapter will provide an brief description of the LHC and the Run 2 ATLAS detector, relevant to the data used in the remainder of this study.

2.1 The Large Hadron Collider

The LHC is a two-ring-superconducting-hadron accelerator and collider built outside of Geneva, Switzerland at the Conseil Europeen pour la Recherche Nucleaire (CERN). It was approved for construction in 1996 to search for beyond the SM physics at energies larger than 10 TeV. It's approval was heavily influenced by the cost-saving idea of reusing the existing 26.5 km tunnels from the Large Electron-Positron (LEP) collider [20]. The LHC has four main collision points that house the ATLAS, CMS, Large Hadron Collider beauty (LHCb) [21], and A Large Ion Collider Experiment (ALICE) [22]. ATLAS and CMS are the two high-energy experiments located at diametrically opposite straight sections. LHCb is a low luminosity experiment dedicated to investigate the difference between matter and anti-matter by detecting b quarks. ALICE is an ion experiment dedicated to studying quark-gluon plasma forms.

The LHC is initially supplied with protons from the injector complex, which is a sequence of accelerators shown in Figure 2.1. The three main components within each of these accelerators are magnets, vacuum chambers, and radiofrequency (RF) cavities. Superconducting magnets are responsible for guiding the beams, and vacuum chambers ensure that particles do not interact with external residual gas molecules. RF cavities are metallic chambers located inside the beam vacuum. They are designed to resonate at specific frequencies to provide small energy boosts when particles pass through.

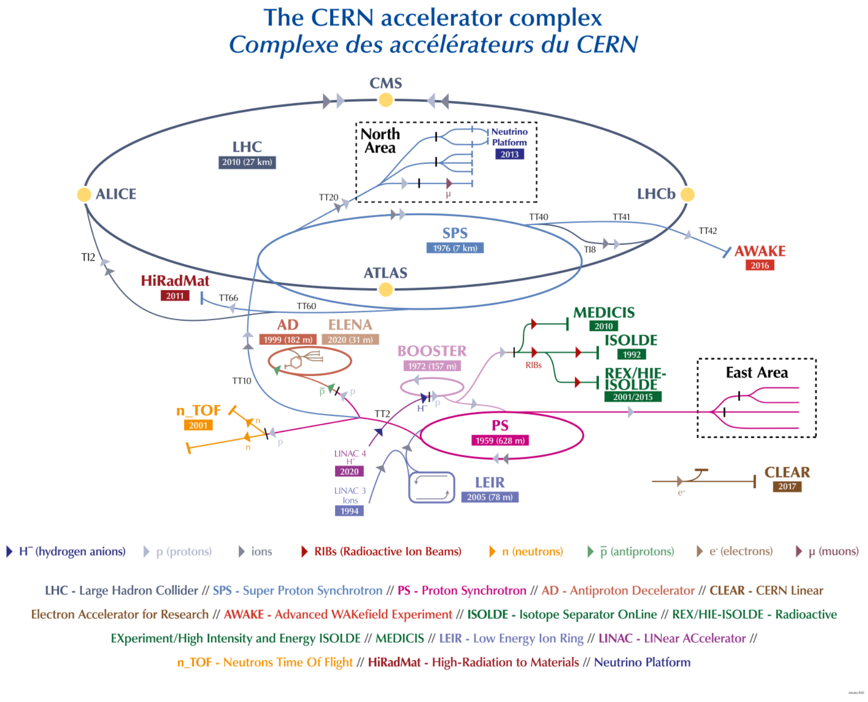

The CERN accelerator complex
Complexe des accélérateurs du CERN
CMS
North Area
Neutrino Platform 2013
LHC 2010 (27 km)
ALICE
LHCb
TT20
TT40
TT41
TT42
AWAKE 2016
SPS 1976 (7 km)
TI8
TI2
ATLAS
HiRadMat 2011
TT66
TT60
AD 1999 (182 m)
ELENA 2020 (31 m)
BOOSTER 1972 (157 m)
MEDICIS 2010
ISOLDE 1992
REX/HIE-ISOLDE 2001/2015
RIBs
East Area
TT10
n_TOF 2001
TT2
PS 1959 (628 m)
LINAC 4 H⁻ 2020
LINAC 3 Ions 1994
LEIR 2005 (78 m)
CLEAR 2017
H⁻ (hydrogen anions)
p (protons)
ions
RIBs (Radioactive Ion Beams)
n (neutrons)
p̄ (antiprotons)
e⁻ (electrons)
μ (muons)
LHC - Large Hadron Collider // SPS - Super Proton Synchrotron // PS - Proton Synchrotron // AD - Antiproton Decelerator // CLEAR - CERN Linear Electron Accelerator for Research // AWAKE - Advanced WAKefield Experiment // ISOLDE - Isotope Separator OnLine // REX/HIE-ISOLDE - Radioactive EXperiment/High Intensity and Energy ISOLDE // MEDICIS // LEIR - Low Energy Ion Ring // LINAC - LINear ACcelerator // n_TOF - Neutrons Time Of Flight // HiRadMat - High-Radiation to Materials // Neutrino Platform

Figure 2.1: The CERN accelerator complex [23].

During Run 2, the LHC collided protons at a center of mass energy of $\sqrt{s} = 13$ TeV with a combined integrated luminosity of $L = 140.07\ fb^{-1}$ [24]. Beams were delivered in bunches with bunch separation of 25 ns, corresponding to a bunch crossing frequency of 40 MHz.

2.2 The ATLAS Apparatus

The ATLAS detector, shown in Figure 2.2, consists of a collection of subsystems confined in a 46 m long, 25 m in diameter cylinder, 100 m below ground. The first subsystem is the Inner Detector (ID) [25], which is responsible for tracking charged-particles. A calorimeter system follows and measures the energy loss of the particles passing through the detector [26]. The final subsystem is the Muon Spectrometer (MS) [27], which measures the deflection of muons within a magnetic field using a trigger and high precision tracking chambers. Additionally, a first-level and high-level trigger system is implemented to select interesting events and record them to disk [28].

ATLAS uses a cylindrical coordinate system (r, η, ϕ) for detector design, reconstruction and data analysis. The polar coordinates, (r, ϕ), point in the plane towards the center of the LHC ring and upwards. The pseudorapidity, η, is defined in Equation 2.1, where θ is the polar angle and equal to the true rapidty defined in Equation 2.2.

$$\eta = -\ln\left(\tan\frac{\theta}{2}\right) \tag{2.1}$$

$$y = \frac{1}{2}\ln\left(\frac{E + p_z}{E - p_z}\right) \tag{2.2}$$

The ID tracks particles in the range $|\eta| < 2.5$, the calorimeter system covers $|\eta| < 4.9$, and the MS detects muon in the $|\eta| < 2.7$ range.

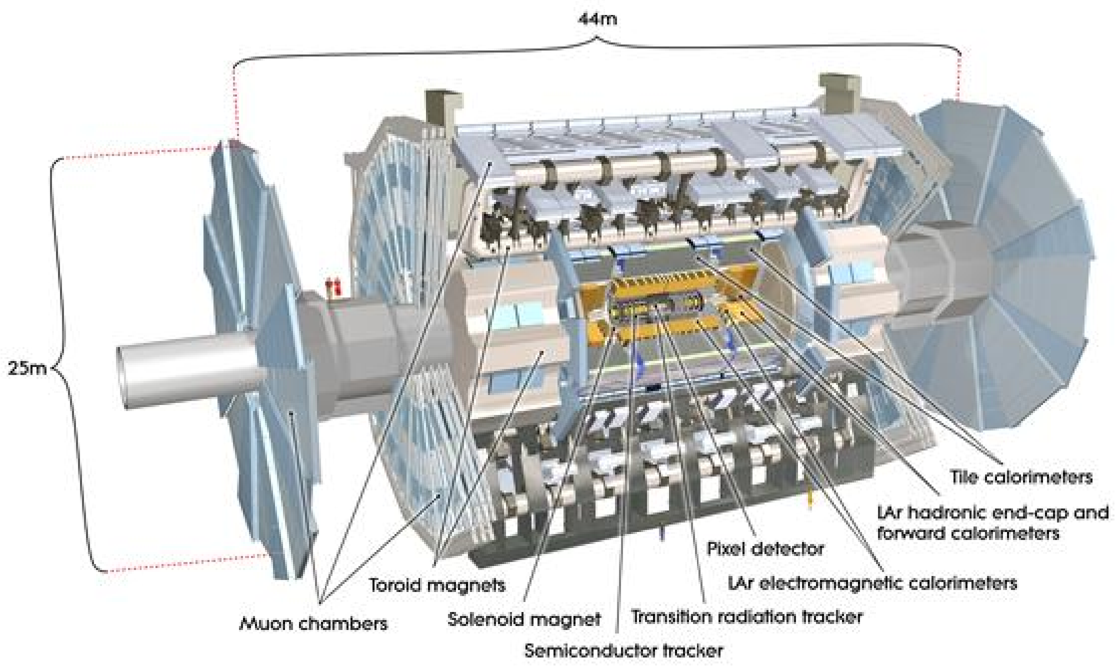

44m
25m
Tile calorimeters
LAr hadronic end-cap and forward calorimeters
Pixel detector
LAr electromagnetic calorimeters
Toroid magnets
Solenoid magnet
Transition radiation tracker
Muon chambers
Semiconductor tracker

Figure 2.2: Computer generated image of the whole ATLAS detector [29].

2.2.1 The Inner Detector

The main components of the ID are the Pixel Detector, Semiconductor Tracker (SCT), and the Transition Radiation Tracker (TRT). This layout is provided in Figure 2.3. The Pixel Detector is first to pick up the energy deposits of the collisions at a precision of 10 μm. Their signals determine the origin and momentum of the particles. The SCT surrounds the Pixel Detector, which measures particle tracks with a precision of up to 25 μm. The TRT is the final layer that provides particle type information, in combination with the other information gained in the ID.

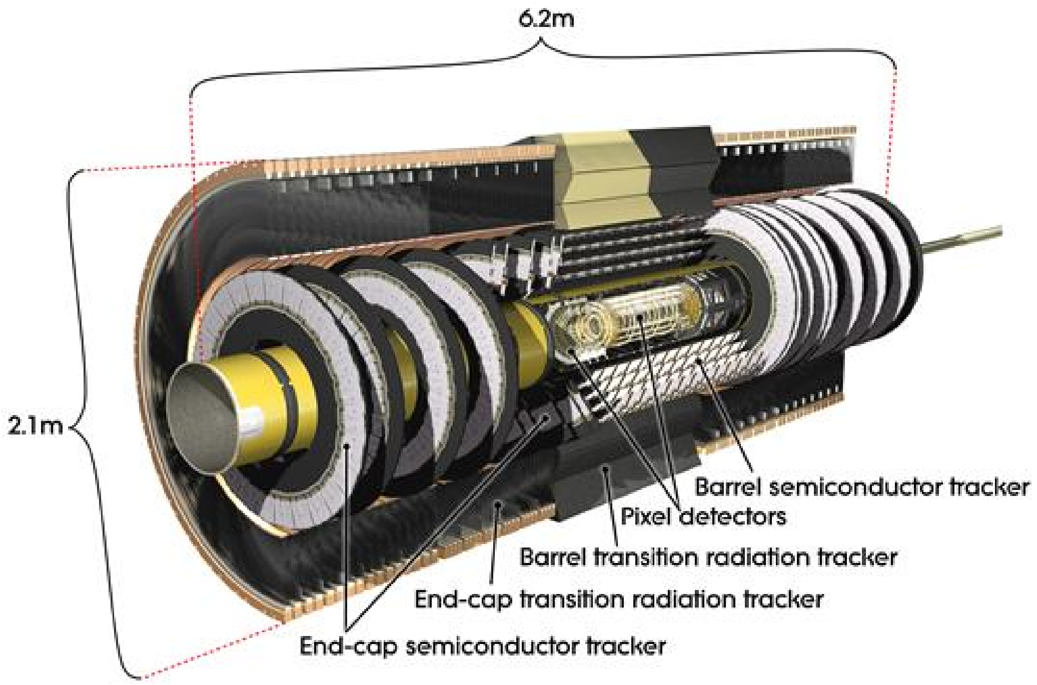

6.2m
2.1m
Barrel semiconductor tracker
Pixel detectors
Barrel transition radiation tracker
End-cap transition radiation tracker
End-cap semiconductor tracker

Figure 2.3: Computer generated image of the ATLAS inner detector [29].

2.2.2 Calorimeter Systems

Calorimeters are detectors that measure the energies and positions of charged and neutral electromagnetically or strongly interacting particles. They consists of highly-dense materials that force particles to deposit their energy. That energy is then converted into a measurable signal using layers of "active" media. The calorimeter systems consists of two types of calorimeters as shown in Figure 2.4: electromagnetic and hadronic. Electromagnetic calorimeters are used to measure charged particles like electrons, positrons, and photons. Hadronic calorimeters are designed to detect hadrons, such as quarks, protons, and neutrons.

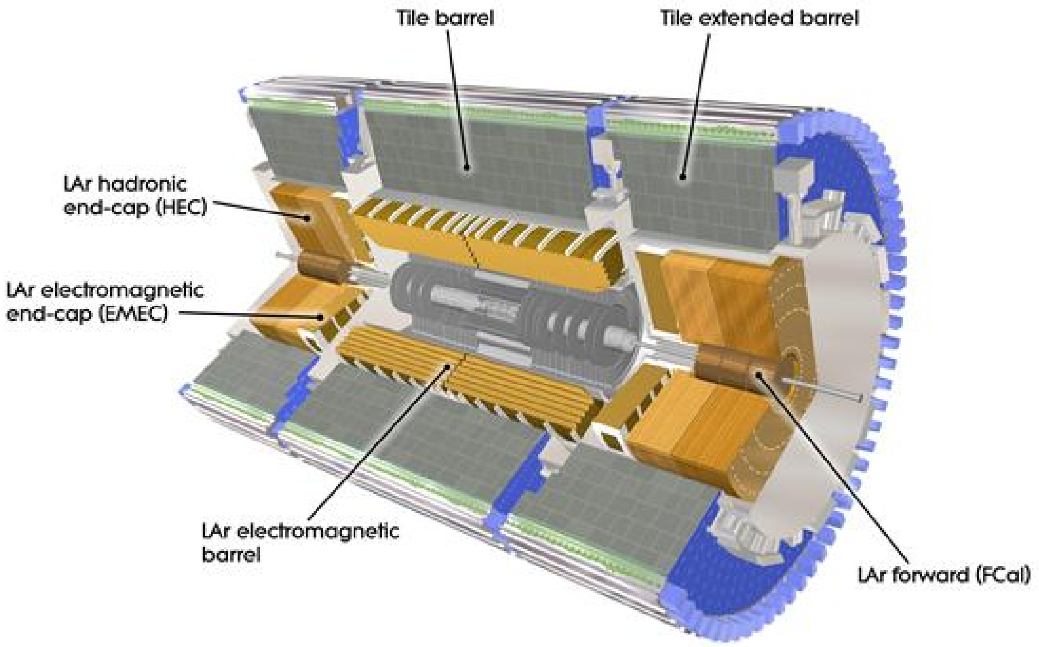

Tile barrel
Tile extended barrel
LAr hadronic end-cap (HEC)
LAr electromagnetic end-cap (EMEC)
LAr electromagnetic barrel
LAr forward (FCal)

Figure 2.4: Computer generated image of the ATLAS Liquid Argon [29].

2.2.3 Muon Spectrometer

The muon spectrometer, shown in Figure 2.5, is the outer part of the ATLAS detector, designed to measuring the momentum of muons. Muons are minimally ionizing particles, meaning they can travel to the edge and beyond the ATLAS detector. The magnetic field that bends their directories is generated by superconducting air-core toroidal magnets, located at the two end caps and one in the center barrel. Three stations of precision chambers, consisting of layers of Monitored Drift Tubes (MDTs) detect the deflection of the muon trajectories in the magnetic field. The MDTs allow muons to knock out electrons from gas when passing through, to produce a signal. Two chambers sit surrounding the central region and ends of the experiment: the Resistive Plate Chambers (RPCs) and Thin Gap Chambers (TGCs). They both detect muons when they ionise the gas mixtures to generate signal.

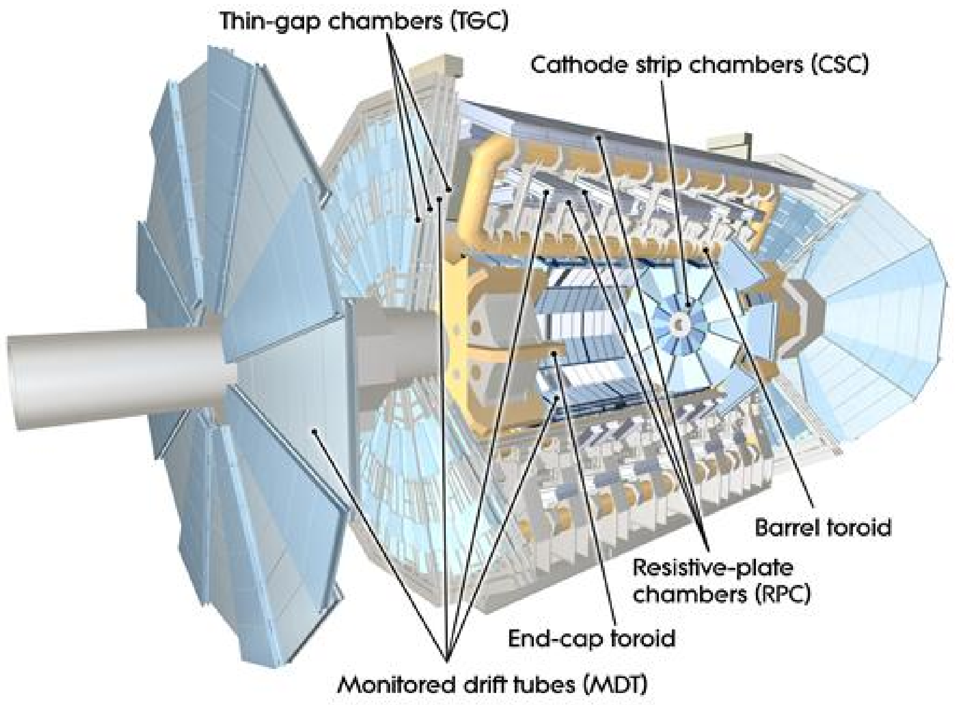

Thin-gap chambers (TGC)
Cathode strip chambers (CSC)
Barrel toroid
Resistive-plate chambers (RPC)
End-cap toroid
Monitored drift tubes (MDT)

Figure 2.5: Computer generated image of the ATLAS Muons subsystems [29].

2.2.4 Magnet System

The two main magnet systems are the Central Solenoid Magnet and the Toroid Magnets. Generally, superconducting magnets are required to bend the directories of charged particles, allowing for the ATLAS detector to to measure their momentum and charge. The Central Solenoid Magnet provides a 2 Tesla magnetic field surrounding the inner detector. The Toroid Magnets are located at the ends of the experiment, and a massive toroid magnet surrounds the center of the experiment. As mentioned in the previous section, the magnets at the ends of the experiment are to bend muons for the Muon spectrometer.

2.2.5 ATLAS Trigger System

The ATLAS Trigger system is a collection of electronics that make rapid decisions of saving certain events into disk. There are two trigger subsystems that help selectively read out and store data from interesting physics events. The first level of the trigger system, called the L1 trigger, uses reduced-granularity information from the calorimeters and muon system to search for signatures of these events. The maximum L1 accept rate is 100 kHz, meaning all processing for an event must be completed within that time window. The second level of the trigger system, called the High Level Trigger, which is a software-based system that performs a more thorough reconstruction of the events passed in L1 to then finally pass to a data storage system for offline analysis.

2.3 HL-LHC

The HL-LHC was proposed in 2010 to extend the discovery potential of the LHC by increasing its instantaneous luminosity (rate of collisions) by a factor of five beyond the original design value and the integrated luminosity (total number of collisions) by a factor ten. Increasing the total number of collisions will increase the probability for ATLAS and CMS to observe rare processes at higher precision [5]. The HL-LHC configuration relies on innovations in accelerator technology such as cutting edge 11 to 12 Tesla superconducting magnets, novel magnet designs, compact superconducting RF cavities for beam rotation with phase control, new technologies and materials for beam collimation, and high-current superconducting links with almost zero energy dissipation.

The ATLAS experiment will also require an upgrade following the HL-LHC. New subdetectors will be installed such as the Inner Tracker [30], the High Granularity Timing

Detector [31], and additional Muon chambers [32]. There will also be different electronics upgrades such as the Liquid Argon Calorimeter [33], the Tile Calorimeter [34], the Muon Spectrometer [35], and the Trigger and Data Acquisition (TDAQ) system [36].

CHAPTER 3

ATLAS SOFTWARE AND COMPUTING

The data collected from the ATLAS data acquisition system must be compared to a set of simulated data. This dataset aims to mimic the different physics processes: it's production by the colliding beams, the evolution of the collision products within the detector and materials, and the detector's response to ultimately interpret efficiencies and background processes. Except for collision data, the output of all these data processing steps are stored in ROOT files. It starts off with Monte Carlo (MC) simulations, which is a computational technique that uses random sampling to generate events. Given these events, the interactions within the detector and the detector's response is simulated. This reconstructed product is called an Analysis Object Data (AOD), which are then cleaned by compressing the data and cutting any unnecessary events or columns into a finalized product called Derived AOD (DAOD). The products produced at each step are then stored into a compressed binary file, called a ROOT file, and are validated using different software tools. These tools collectively encompass the software framework called Athena [37]. The flow of this process is display in Figure 3.1. This chapter will provide an introduction to ATLAS Open Data [38], ROOT, and its application programming interface (API) for TTree and RNTuple formats.

3.1 ATLAS Open Data

ATLAS Open Data is a publicly available dataset produced by the ATLAS collaboration. It's composed of MC simulations of particle collisions within the ATLAS detector and detector data measurements. The data used as inputs for the remainder of this study are MC

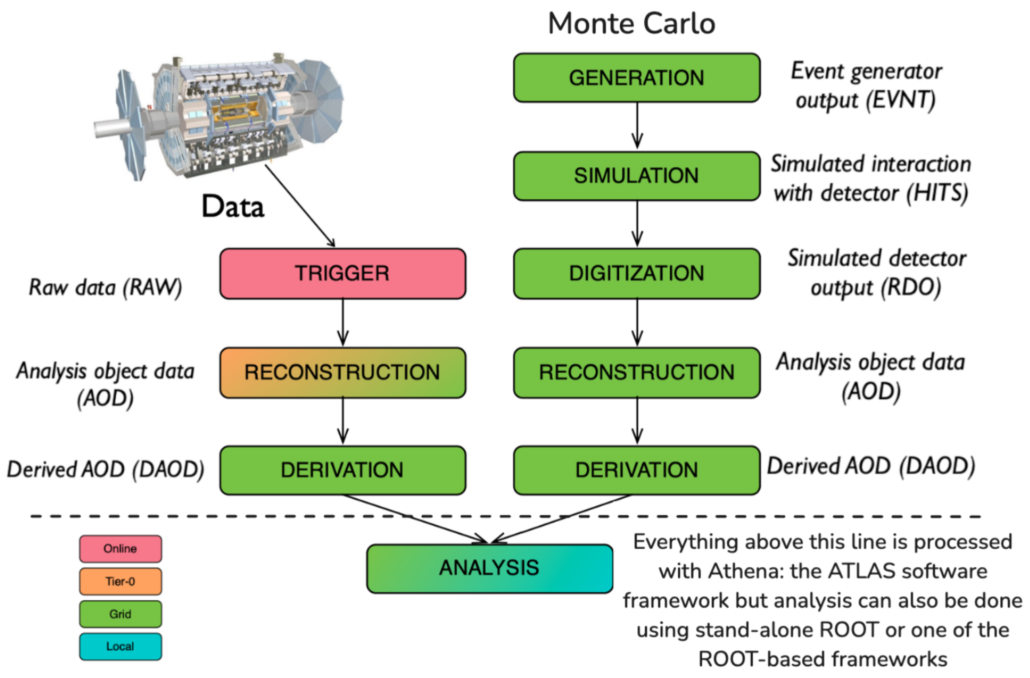
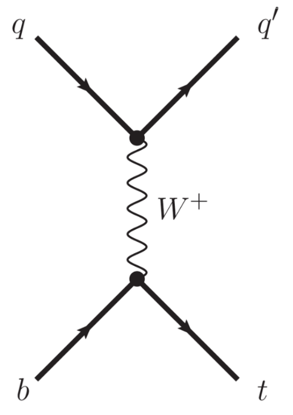

Monte Carlo
GENERATION
Event generator output (EVNT)
SIMULATION
Simulated interaction with detector (HITS)
DIGITIZATION
Simulated detector output (RDO)
RECONSTRUCTION
Analysis object data (AOD)
DERIVATION
Derived AOD (DAOD)
Data
TRIGGER
Raw data (RAW)
RECONSTRUCTION
Analysis object data (AOD)
DERIVATION
Derived AOD (DAOD)
ANALYSIS
Everything above this line is processed with Athena: the ATLAS software framework but analysis can also be done using stand-alone ROOT or one of the ROOT-based frameworks
Online
Tier-0
Grid
Local

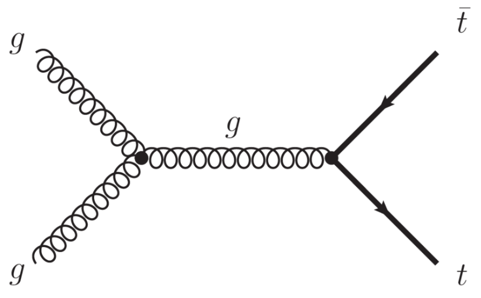
Figure 3.1: ATLAS data chain-processing for data and Monte Carlo simulation [39].

simulations of top nominal samples from Run 2 [40, 41]. They are simulated processes that produce single top quarks, matter-antimatter $t\bar{t}$ pairs, and W boson production in association with jets. Representative diagrams for these processes are shown in Figure 3.2.

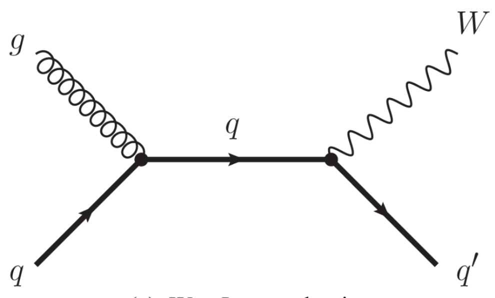
The inputs are all provided in PHYSLITE format, which contains already-calibrated objects directly from an AOD or PHYS product [42]. Those objects include jets, electrons, muons, photons, taus and their kinematics, such as transverse momentum, mass, charge, eta, and phi. Each event contains a number of physical objects that depends on the underlying process, resulting in a multidimensional dataset. A full description of the variables can be found in [43].



(a) Single top quark production.



(b) Matter-antimatter $t\bar{t}$ pair production.



(c) W + Jets production.

Figure 3.2: The representative diagrams of the processes used in this thesis.

3.2 ROOT

ROOT is a unified software package developed for processing, analyzing, visualizing and ultimately storing the massive high-energy physics datasets into ROOT files. Previously, high-energy experiments used FORTRAN-based libraries; however, an upgrade was needed to handle the scales and complexities of the data from the LHC. ROOT maintains an object-oriented structure, meaning it is organized around the data rather than the functions and logic. It's features include visualization tools such as histogramming, and statistical tools. ROOT can be used in C++ and python languages. Several sub packages exists for analysis such as RDataFrame.

3.2.1 ROOT Compression Algorithms

ROOT offers four different compression algorithms: `ZLIB`, `LZMA`, `LZ4`, and `ZSTD` [44]. Data compression allows users to store large files at reduced sizes without losing information from the original file. It can also increase data reading and writing speeds. There are generally two types of compression algorithms: lossless and lossy. Lossy algorithms reduce files at more depth and are irreversible processes. The four compression algorithms from ROOT are lossless algorithms, meaning they are reversible processes that reduce bits by eliminating statistical redundancy.

There are advantages and disadvantages in each of the four algorithms. `LZ4` focuses on compression and decompression speed, yet provides large files. `LZMA` provides higher compression at the cost of significantly slower reading speeds. `ZLIB` is an older version of `ZSTD`. Both provide a balance between compression and reading speeds; however, `ZSTD` has been shown to perform better in all metrics in comparison to ZLIB [45]. The input data applied for this study were all produced with `ZSTD`. A comparison study between `ZSTD` and `LZ4` is performed for RNTuple versions of the inputs, shown in Chapter 4.

3.2.2 TTree Data Structure

ROOT provides a data structure called TTree to store large amounts of columnar data efficiently. Usually scientific data is stored in what we call row-oriented formats such as a spreadsheet or CSV table. This format is well organized if one wants to access a single event, but viewing a single column then becomes inefficient, especially with large datasets. A TTree is columnar based, meaning it consists of a list of independent columns, called branches. Examples of branches can be event IDs or particle kinematics such as momentum in the x, y, z coordinates. Branches can hold integers, strings and std::vector data types. Buffers are automatically allocated behind each branch. Buffers are temporary storage areas for the independent binary version of the object. This is done to efficiently handle the writing and reading of the data to and from disk. Each branch has one or more baskets, which manages the in-memory buffer. In other words, a basket holds the values of a branch for a number of consecutive events. When a buffer is full, it is optionally compressed and then the corresponding basket is written to disk, leading to the creation of a new basket to hold the next entries. ROOT allows users to change buffersize parameters of the branch for personalized optimization. Figure 3.3 shows a more detailed flowchart of the TTree data structure.

3.2.3 RNTuple Data Structure

RNTuple is the new columnar data format that will be implemented at the start of the HL-LHC. It's design continues to be columnar based, as its predecessor TTree, but it now uses modern storage technologies for better performance characteristics in data compactness, scalability, and read and write speed. For this reason, RNTuple classes are backwards-incompatible to TTree both on the file format level and API level [47]. It's binary format version follows an *epoch.major.minor.path* scheme, where *epoch* indicates backward-incompatible changes, *major* indicates forward-incompatible changes, *minor* indicates new optional format features, and *patch* indicates backported features from newer format versions. The remainder of this study uses the first public release of RNTuple 1.0.0.0.

RNTuple organizes data using an internal BLOB-based data layout and an external metadata schema. A BLOB (binary large object) is a collection of binary data stored as a single entity. For example, instead of embedding data directly into a database, data can

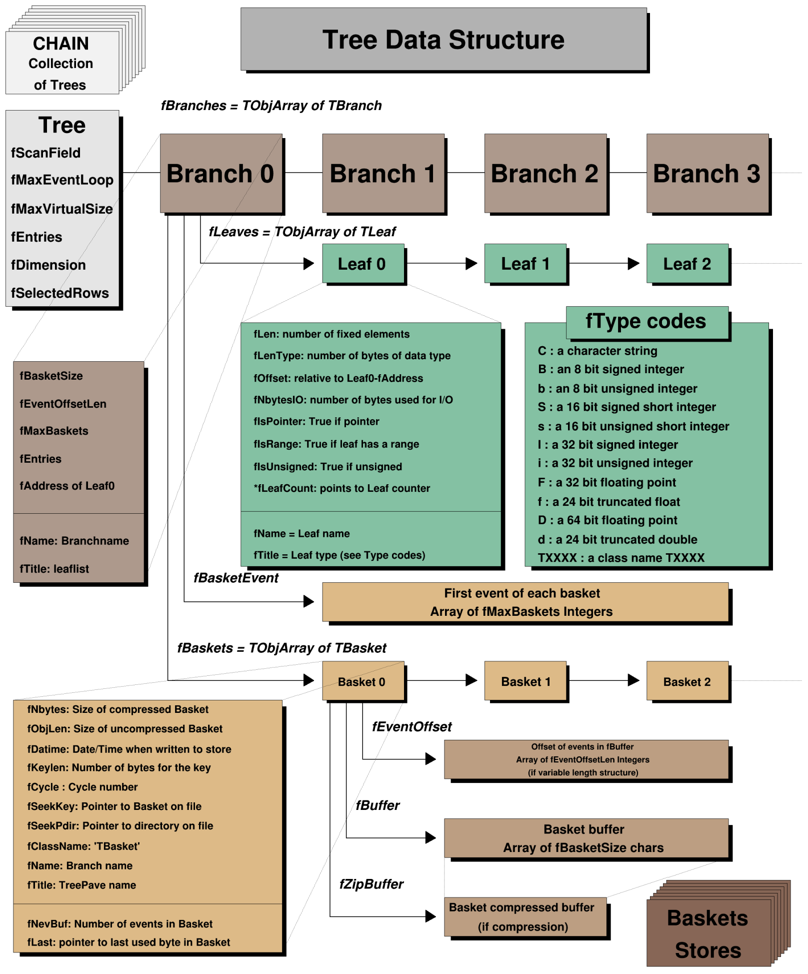

CHAIN
Collection
of Trees
Tree Data Structure
fBranches = TObjArray of TBranch
Tree
fScanField
fMaxEventLoop
fMaxVirtualSize
fEntries
fDimension
fSelectedRows
Branch 0
Branch 1
Branch 2
Branch 3
fLeaves = TObjArray of TLeaf
Leaf 0
Leaf 1
Leaf 2
fBasketSize
fEventOffsetLen
fMaxBaskets
fEntries
fAddress of Leaf0
fName: Branchname
fTitle: leaflist
fLen: number of fixed elements
fLenType: number of bytes of data type
fOffset: relative to Leaf0-fAddress
fNbytesIO: number of bytes used for I/O
fIsPointer: True if pointer
fIsRange: True if leaf has a range
fIsUnsigned: True if unsigned
*fLeafCount: points to Leaf counter
fName = Leaf name
fTitle = Leaf type (see Type codes)
fType codes
C : a character string
B : an 8 bit signed integer
b : an 8 bit unsigned integer
S : a 16 bit signed short integer
s : a 16 bit unsigned short integer
I : a 32 bit signed integer
i : a 32 bit unsigned integer
F : a 32 bit floating point
f : a 24 bit truncated float
D : a 64 bit floating point
d : a 24 bit truncated double
TXXXX : a class name TXXXX
fBasketEvent
First event of each basket
Array of fMaxBaskets Integers
fBaskets = TObjArray of TBasket
Basket 0
Basket 1
Basket 2
fNbytes: Size of compressed Basket
fObjLen: Size of uncompressed Basket
fDatime: Date/Time when written to store
fKeylen: Number of bytes for the key
fCycle : Cycle number
fSeekKey: Pointer to Basket on file
fSeekPdir: Pointer to directory on file
fClassName: 'TBasket'
fName: Branch name
fTitle: TreePave name
fNevBuf: Number of events in Basket
fLast: pointer to last used byte in Basket
fEventOffset
Offset of events in fBuffer
Array of fEventOffsetLen Integers
(if variable length structure)
fBuffer
Basket buffer
Array of fBasketSize chars
fZipBuffer
Basket compressed buffer
(if compression)
Baskets
Stores

Figure 3.3: Example of the TTree Data Structure [46].

be stored as a BLOB along with a unique identifier for later retrieval. This is beneficial for managing large unstructured data [48]. RNTuple uses a similar approach internally: Data is organized by columns of a single type and are attached to *fields*, which describes a serialized C++ type. Columns are partitioned into *pages*. Pages are compressed individually, similar to TTree baskets. *Clusters* are sets of pages that contain all the data belonging to an entry range. *Envelopes* are data blocks that contain metadata, such as field and columns types, cluster descriptions, and page locations. Overall, this structure allows for random-access of individual events without decompressing the entire dataset and for "fast merging" or concatenating RNTuples. A simplified diagram of the RNTuple structure in comparison to TTree is shown in Figure 3.4.

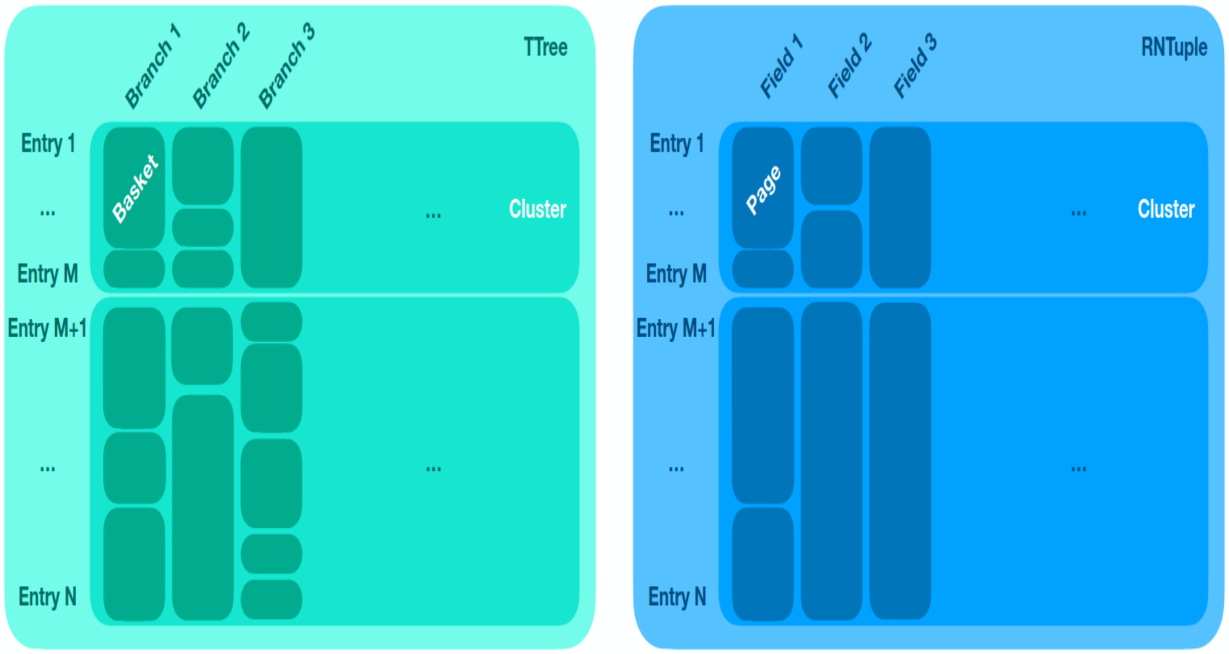


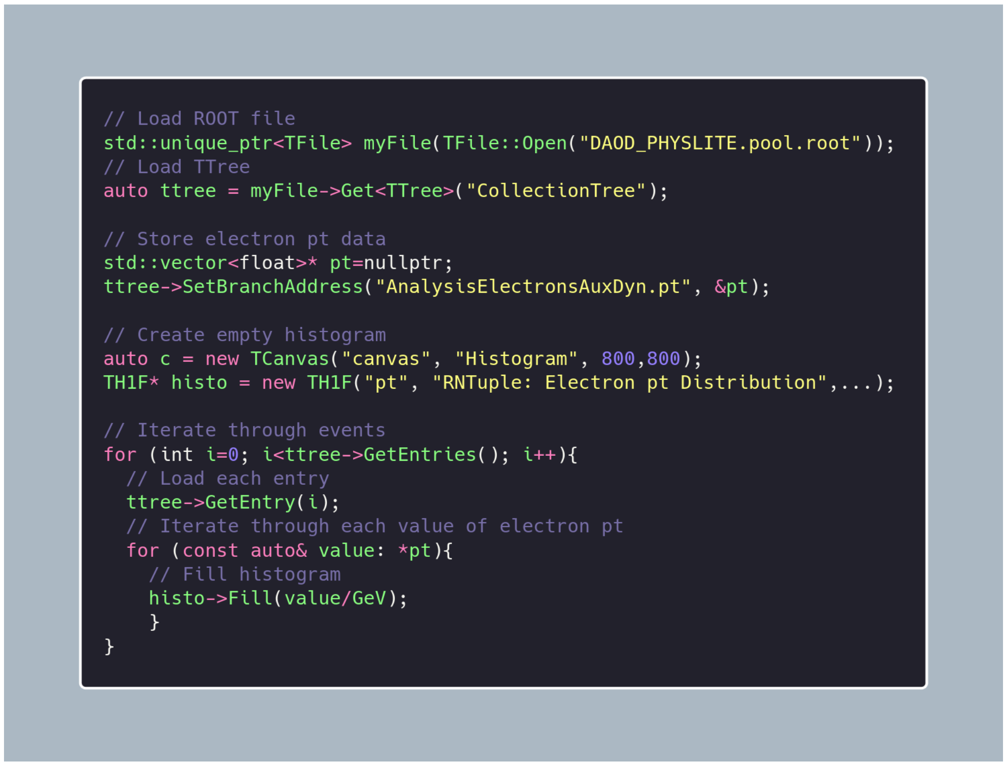
Figure 3.4: TTree Structure vs. RNTuple Structure [49].

3.3 TTree vs. RNTuple API

TTree's API is natively compatible with C++, RDataFrame analysis workflows in Python and C++, and the uproot library [50]. At this stage, RNTuple's API is best compatible with RDataFrame analysis workflows and hand-written event loops. It currently has limited capabilities with the uproot library. For example, functions in uproot that read or concatenate multiple RNTuples are still under development, as are the functions for writing RNTuple outputs. As part of this project, an effort was made to test and document the capabilities, usage and best practices of RNTuple in a physics analysis context during its development. The corresponding code repository, including the aforementioned documentation can be found in [51]. The sections below will provide examples of RNTuple's API in comparison to TTree's.

3.3.1 Native C++ Event Loops

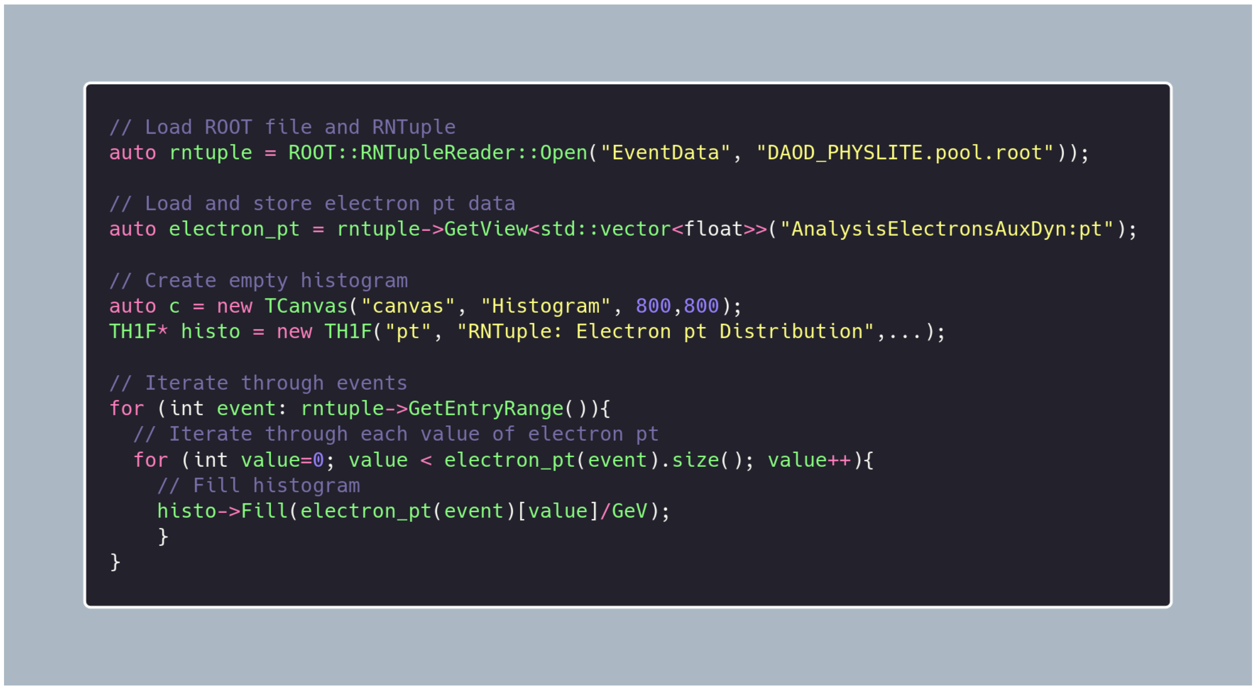
Due to the multidimensional nature of particle physics data, event loops are common algorithms used in data analysis workflows. It is a process that continuously iterates through the large datasets to apply specific analysis steps to each event. As seen in Figure 3.5, users must iterate through TTree in order to load branches and define an empty pointer object to store their entries.



```
// Load ROOT file
std::unique_ptr<TFile> myFile(TFile::Open("DAOD_PHYSLITE.pool.root"));
// Load TTree
auto ttree = myFile->Get<TTree>("CollectionTree");

// Store electron pt data
std::vector<float>* pt=nullptr;
ttree->SetBranchAddress("AnalysisElectronsAuxDyn.pt", &pt);

// Create empty histogram
auto c = new TCanvas("canvas", "Histogram", 800,800);
TH1F* histo = new TH1F("pt", "RNTuple: Electron pt Distribution",...);

// Iterate through events
for (int i=0; i<ttree->GetEntries(); i++){
  // Load each entry
  ttree->GetEntry(i);
  // Iterate through each value of electron pt
  for (const auto& value: *pt){
    // Fill histogram
    histo->Fill(value/GeV);
    }
}
```

Figure 3.5: Native C++ event loop using TTree. This script loads a PHYSLITE ROOT file containing a TTree titled "CollectionTree", and plots the distribution electron transverse momenta.

The RNTuple interface uses smart pointers, which simulates a pointer while providing automatic memory management [52]. This feature shortens the amount of code necessary to read and load data by a couple of lines. For example `RNTupleReader::Open` simultaneously loads the ROOT file and the RNTuple, as seen in Figure 3.6. The function `GetView` also simultaneously loads and stores a field.



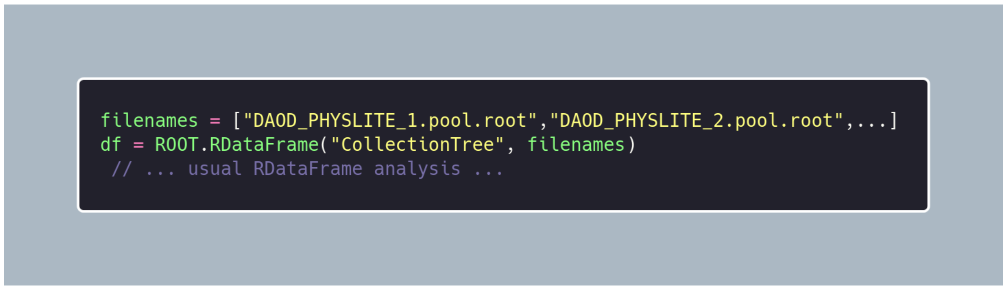
```
// Load ROOT file and RNTuple
auto rntuple = ROOT::RNTupleReader::Open("EventData", "DAOD_PHYSLITE.pool.root"));

// Load and store electron pt data
auto electron_pt = rntuple->GetView<std::vector<float>>("AnalysisElectronsAuxDyn:pt");

// Create empty histogram
auto c = new TCanvas("canvas", "Histogram", 800,800);
TH1F* histo = new TH1F("pt", "RNTuple: Electron pt Distribution",...);

// Iterate through events
for (int event: rntuple->GetEntryRange()){
  // Iterate through each value of electron pt
  for (int value=0; value < electron_pt(event).size(); value++){
    // Fill histogram
    histo->Fill(electron_pt(event)[value]/GeV);
    }
}
```

Figure 3.6: This is the RNTuple version of Figure 3.5.

3.3.2 RDataFrame in C++ and Python

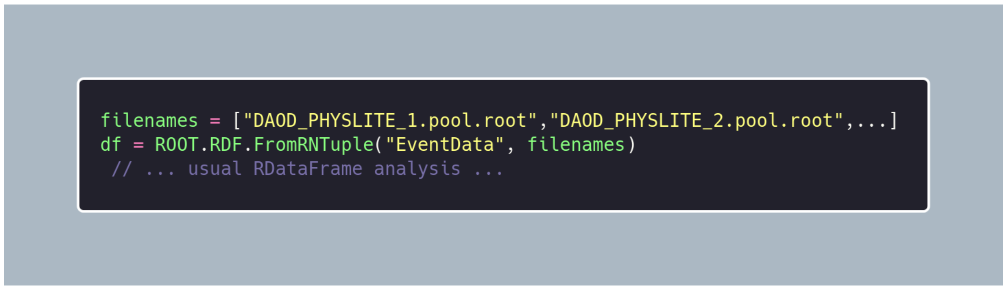
Analysis done with RDataFrame will mostly remain unmodified with RNTuple, as shown in Figure 3.7, with the exception of filtering. Due to RNTuple's internal data structure, sub fields such as `"AnalysisElectronsAuxDyn:pt"` are separated by their field, `"AnalysisElectronsAuxDyn"` by a column, instead of a period. This slight change confuses the filtering function in

467 RDataFrame, but can be bypassed by assigning an alias name. Figure 3.8 provides an example of how to read multiple inputs and apply a filter using RDataFrame in C++.



```
filenames = ["DAOD_PHYSLITE_1.pool.root","DAOD_PHYSLITE_2.pool.root",...]
df = ROOT.RDataFrame("CollectionTree", filenames)
 // ... usual RDataFrame analysis ...
```

(a) Reading multiple TTree inputs.

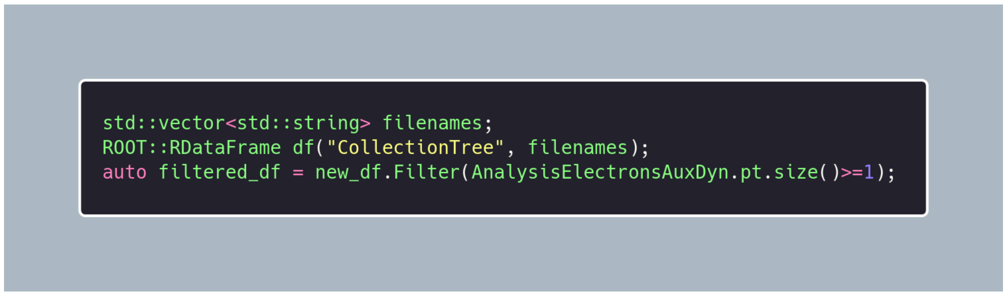


```
filenames = ["DAOD_PHYSLITE_1.pool.root","DAOD_PHYSLITE_2.pool.root",...]
df = ROOT.RDF.FromRNTuple("EventData", filenames)
 // ... usual RDataFrame analysis ...
```

(b) Reading multiple RNTuple inputs.

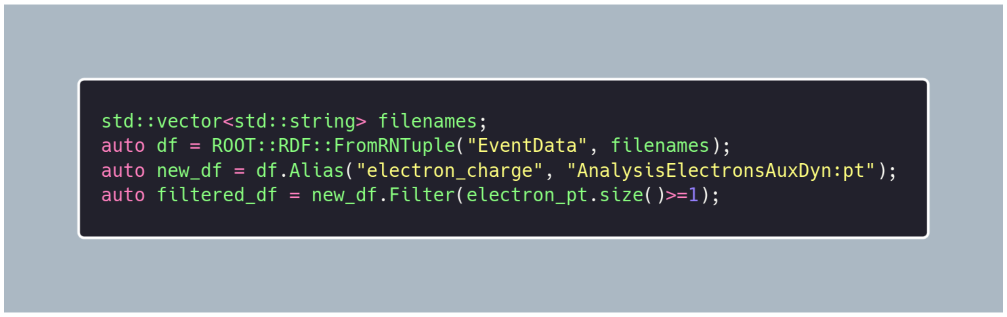
Figure 3.7: Examples of how to load multiple inputs into an RDataFrame in Python.

468



```cpp
std::vector<std::string> filenames;
ROOT::RDataFrame df("CollectionTree", filenames);
auto filtered_df = new_df.Filter(AnalysisElectronsAuxDyn.pt.size()>=1);
```

(a) Reading multiple TTree inputs.



```cpp
std::vector<std::string> filenames;
auto df = ROOT::RDF::FromRNTuple("EventData", filenames);
auto new_df = df.Alias("electron_charge", "AnalysisElectronsAuxDyn:pt");
auto filtered_df = new_df.Filter(electron_pt.size()>=1);
```

(b) Reading multiple RNTuple inputs.

Figure 3.8: Examples of how to load multiple inputs into an RDataFrame and create a new filtered dataframe in C++.

CHAPTER 4

RNTUPLE VS. TTREE PERFORMANCE

In this chapter, RNTuple performance is analyzed using RDataFrame and compared to TTree. First, 92 TTrees stored in `DAOD_PHYSLITE` files from ATLAS Open Data were converted to RNTuples using its default compression algorithm setting, ZSTD. An average size reduction of about 47% was observed between the converted RNTuples and the original TTrees, as shown in Figure 4.1. Speed tests were performed for loading and outputting RNTuples in comparison to TTrees using `std::chrono::high_resolution_clock::now()`. Each performance study contains two versions: a TTree version that uses TTree inputs and an RNTuple version that uses RNTuple inputs. A comparison of peak memory consumption was also performed using both sets of inputs. The entirety of this analysis was repeated for RNTuple inputs that were converted with LZ4 compression algorithm.

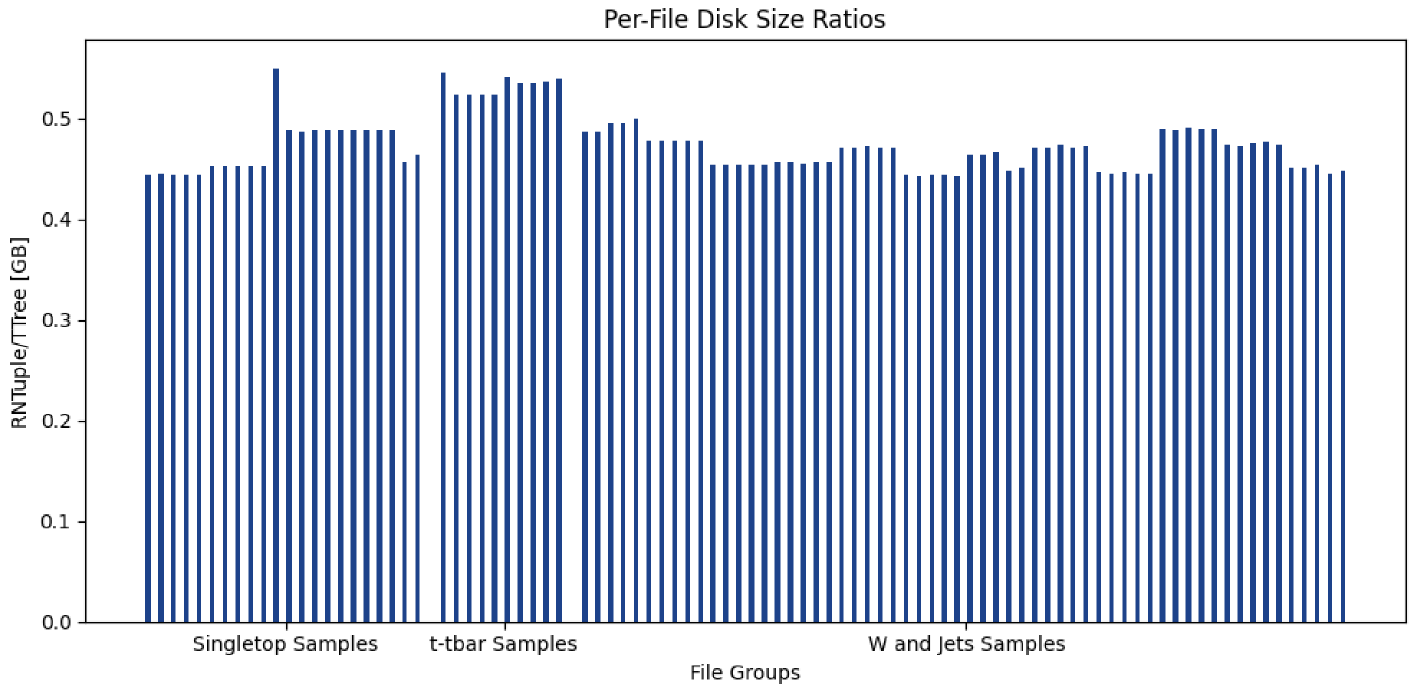


Figure 4.1: The RNTuple:TTree file size ratios over number of events per file.

4.1 Readability Speed

The total loading times for 92 RNTuples and their TTree equivalence were measured 100 times to ensure consistency. Loading multiple RNTuples in RDataFrame follows an identical procedure in both TTree and RNTuple versions (seen previously in 3.3.2). The timer began at the start of the script and was stopped after calculating the sum of the column `"AnalysisElectronsAuxDyn:pt"`. This was done to ensure that the data was being loaded and read by RDataFrame. The measured times were recorded onto a text file and are shown in Figure 4.2. In comparison to TTree, this study finds RNTuple to be 2.38 times faster at loading a column of data.

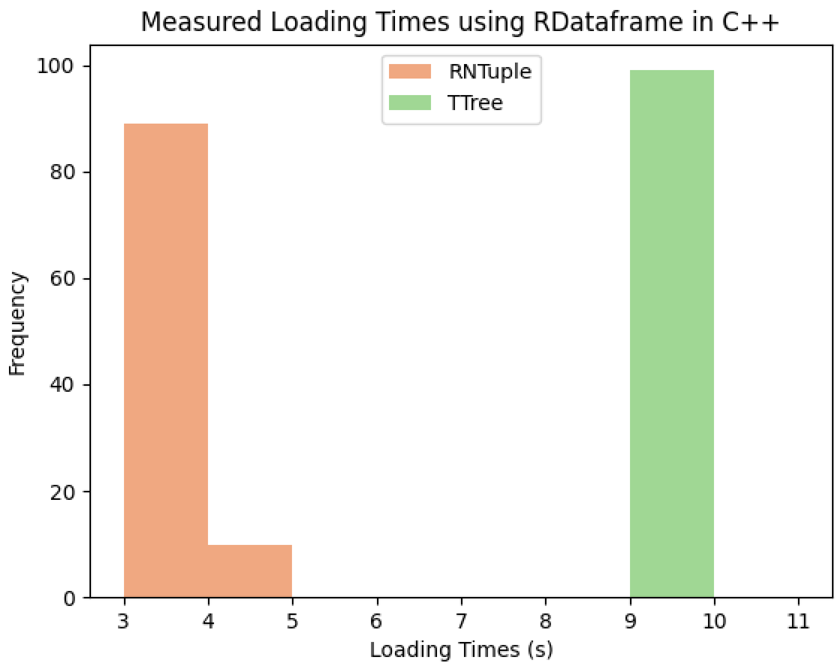
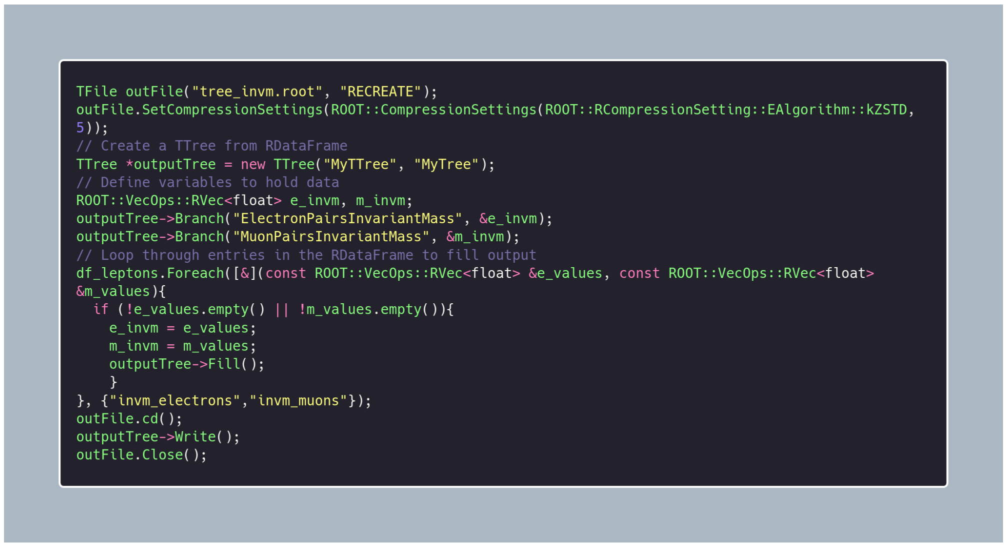


Figure 4.2: Total loading times measured for TTree and RNTuple using RDataFrame in C++.

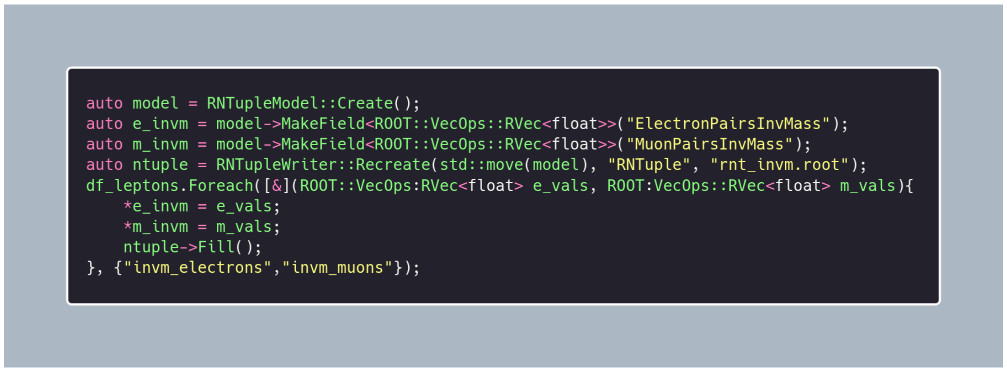
4.2 Writing Speed

Writing speed was measured by performing an invariant mass calculation and outputting a new dataset with two columns: `"ElectronPairsInvMass"` and `"MuonPairsInvMass"`. The timer began at the start of an invariant mass calculation and stopped after creating a new dataset. A TTree was written for the TTree version and an RNTuple was written for the RNTuple version. The quick function that outputs a TTree in RDataFrame, `df.Snapshot(...)`, is currently not developed to output an RNTuple yet; therefore, for consistency, both versions of the script uses the RDataFrame function `df.ForEach(...)` to loop through events and fill in the new columns. This procedure for RNTuple and TTree versions is shown in Figure 4.3.



```cpp
TFile outFile("tree_invm.root", "RECREATE");
outFile.SetCompressionSettings(ROOT::CompressionSettings(ROOT::RCompressionSetting::EAlgorithm::kZSTD,
5));
// Create a TTree from RDataFrame
TTree *outputTree = new TTree("MyTTree", "MyTree");
// Define variables to hold data
ROOT::VecOps::RVec<float> e_invm, m_invm;
outputTree->Branch("ElectronPairsInvariantMass", &e_invm);
outputTree->Branch("MuonPairsInvariantMass", &m_invm);
// Loop through entries in the RDataFrame to fill output
df_leptons.Foreach([&](const ROOT::VecOps::RVec<float> &e_values, const ROOT::VecOps::RVec<float>
&m_values){
  if (!e_values.empty() || !m_values.empty()){
    e_invm = e_values;
    m_invm = m_values;
    outputTree->Fill();
    }
}, {"invm_electrons","invm_muons"});
outFile.cd();
outputTree->Write();
outFile.Close();
```

(a) TTree Version.



```cpp
auto model = RNTupleModel::Create();
auto e_invm = model->MakeField<ROOT::VecOps::RVec<float>>("ElectronPairsInvMass");
auto m_invm = model->MakeField<ROOT::VecOps::RVec<float>>("MuonPairsInvMass");
auto ntuple = RNTupleWriter::Recreate(std::move(model), "RNTuple", "rnt_invm.root");
df_leptons.Foreach([&](ROOT::VecOps:RVec<float> e_vals, ROOT:VecOps::RVec<float> m_vals){
    *e_invm = e_vals;
    *m_invm = m_vals;
    ntuple->Fill();
}, {"invm_electrons","invm_muons"});
```

(b) RNTuple Version.

Figure 4.3: TTree vs. RNTuple writing algorithms using the RDataFrame function `df.ForEach(...)` in C++.

Although the procedures are the same, the RNTuple version takes up significantly less code due to the RNTuple API. With TTree, an empty vector has to be created before writing a branch. With RNTuple, the `ROOT::RNTupleModel` class has the function `MakeField`, which creates a new field given a name and a corresponding value managed by a shared pointer. The function `RNTupleWriter::Recreate()` simultaneously creates the RNTuple and the output ROOT file. In the TTree version, both the TTree and the output file need to be defined separately.

The total output times were measured 100 times and were recorded in a text file. The results in Figure 4.4 show that writing with RNTuple is 1.51 times faster than with TTrees.

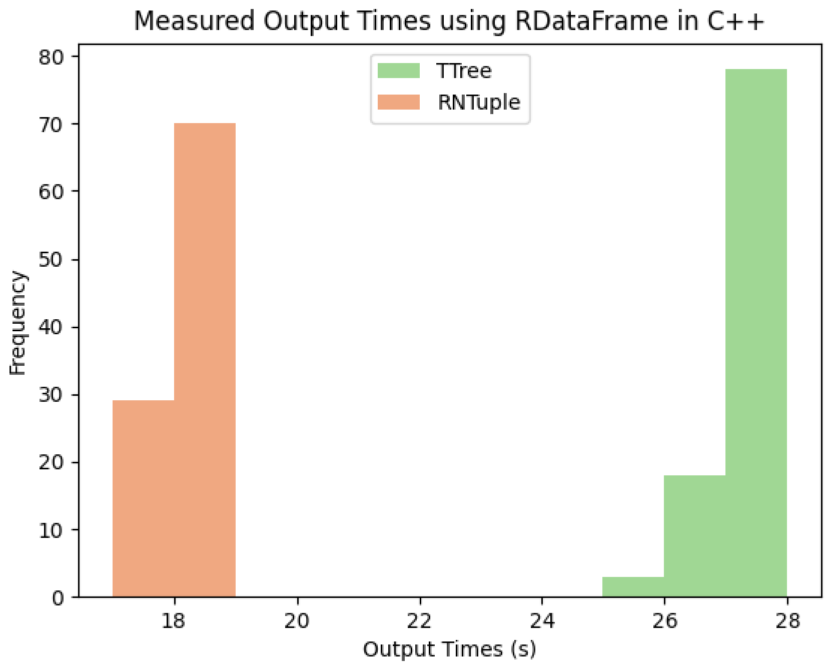


Figure 4.4: Total writing times measured for TTree and RNTuple using RDataFrame in C++.

4.3 Output Sizes

The output file sizes were measured to determine whether RNTuple maintains a consistent size-reduction behavior at this stage of the analysis. By error, the outputs initially produced contained empty events; however, this brought some insights on RNTuple when compared to "cleaned" outputs that filtered out empty events. The results shown in Table 4.1, reveal that RNTuple provides a 99% event size reduction to TTree when the outputs written include empty events. This implies that RNTuple is handling repeated bits significantly better than

TTree. Table 4.2 reveals a 63% reduction from RNTuple when eliminating the empty events. The latter result is considered more practical or realistic for an analysis; yet, these results open an opportunity to write data and approach analysis workflows differently.

Table 4.1: File size and averaged compressed event size for TTree and RNTuple outputs with empty events. The total number of unfiltered events written is 9,045,000 events.

DataFormat	File Size [bytes]	Average Compressed Event Size [bytes/event]
TTree	48 086 740	5.31
RNTuple	447 414	0.049

Table 4.2: File size and averaged compressed event size for TTree and RNTuple outputs without empty events. The total number of filtered events is 77,411 events.

DataFormat	File Size [bytes]	Average Compressed Event Size [bytes/event]
TTree	791 428	10.23
RNTuple	288 529	3.73

4.4 Memory Consumption

Peak memory usage was measured using Python versions of the writing scripts used in 4.2. Using the command `usr/bin/time`, memory usage was measured 100 times for both TTree and RNTuple versions. For this test study, results shown in Figure 4.5 demonstrate no significant difference between RNTuple and TTree.

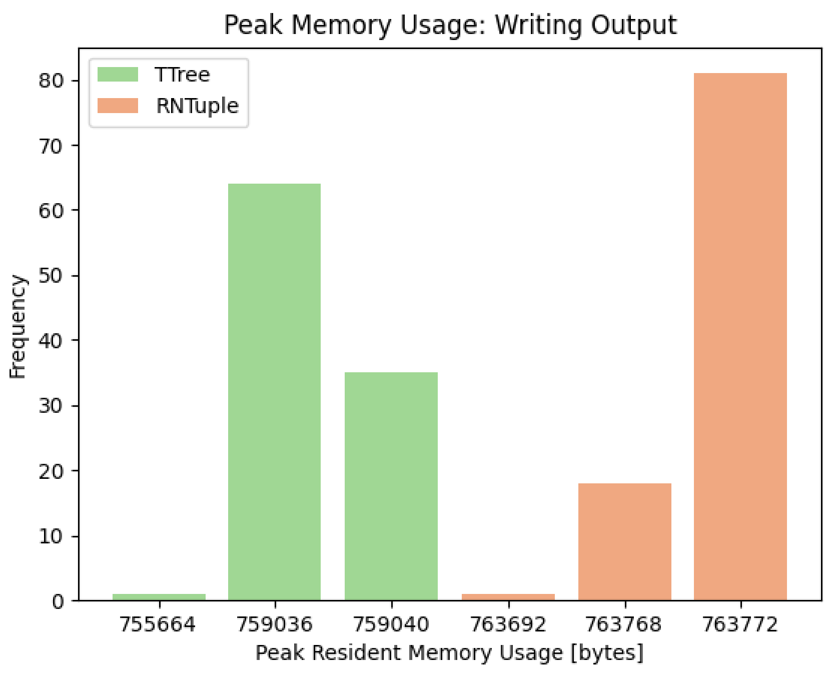


Figure 4.5: Peak memory usage while producing an output with two columns. Measurements were taken 100 times for each version.

4.5 LZ4 Compression Algorithm Study

Studies have shown that LZ4 improves reading and writing speeds for TTree, but at the cost of larger files [44]. This section will investigate if this behavior is consistent with RNTuple by repeating the loading and writing measurements. The same 92 ATLAS Open Data files were used to produce RNTuple equivalents with the LZ4 compression algorithm specified. Time measurements for loading the electron transverse momenta column are shown in Figure 4.6, and the time measurements for writing an RNTuple output are shown in Figure 4.7. There are no significant differences between reading RNTuples produces from LZ4 or ZSTD algorithms; however, there is a 2 second difference between writing an RNTuple with LZ4 versus ZSTD. The ratio of the LZ4 RNTuple sizes over the RNTuples produced with

ZSTD are shown in Figure 4.8. They reveal that the LZ4 algorithm increases the RNTuple file sizes by an average of 14% from ZSTD.

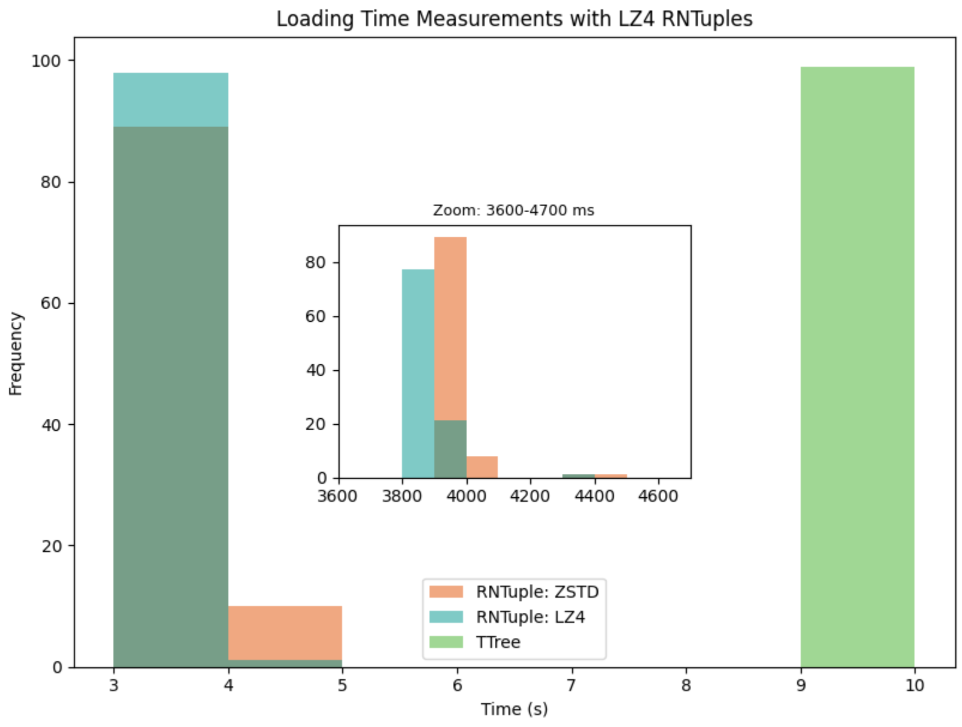


Figure 4.6: Loading time measurements for RNTuples produced by the LZ4 and ZSTD algorithms, and for TTree. The RNTuples composed with ZSTD and LZ4 only differ by a couple of milliseconds.

4.6 Performance Discussion

Common analysis steps used in RDataFrame workflows have improved with RNTuple compared to its TTree predecessor. Converting TTree to RNTuple showed immediate reductions in file size on disk. Reading and writing speeds increased without any significant cost to memory usage. Additionally, the RNTuple API reduces lines of code, making it more user-friendly than the TTree API.

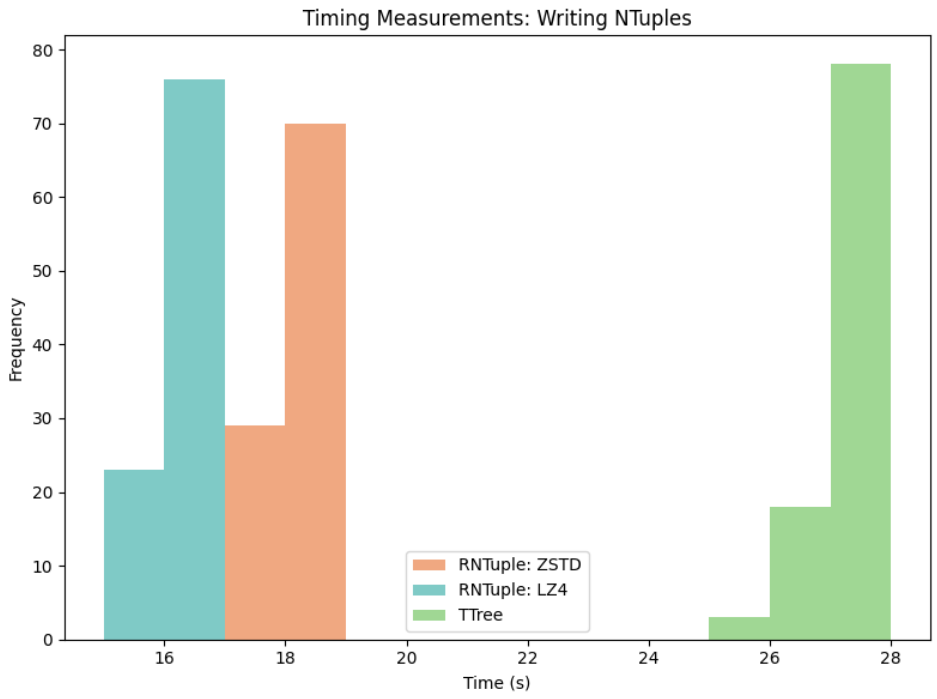


Figure 4.7: Writing time measurements for RNTuples produced by the LZ4 and ZSTD algorithms, and TTree.

The comparisons between RNTuples produced with LZ4 versus ZSTD provide a first look at how RNTuple behaves with different compression algorithms. RNTuples produced with LZ4 show improved reading and writing speeds, though not at significant levels and at the cost of increased disk size. Given this study, producing RNTuples with ZSTD is recommended.

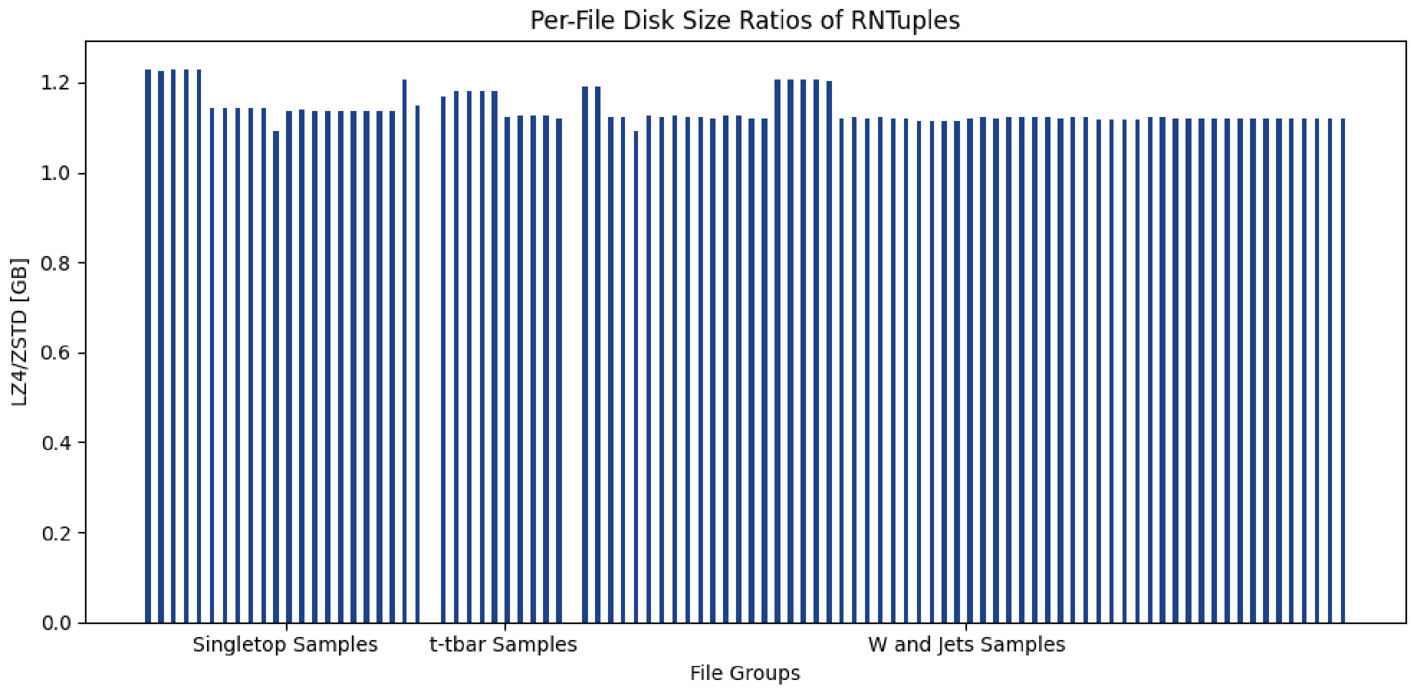


Figure 4.8: Per-file compression ratios of LZ4:ZSTD over total number of events.

CHAPTER 5

ANALYSIS GRAND CHALLENGE: RNTUPLE VS. TTREE

The Analysis Grand Challenge (AGC) is an analysis on top quark production meant to showcase an end-to-end analysis pipeline. Developed and organized by Iris-HEP, the AGC has several versions that showcase different cyber infrastructure and workflows, making it a great benchmark to test RNTuple. This section will describe the development of two new AGC versions that use ATLAS Open Data and RDataFrame: TTree and RNTuple versions. These versions were heavily influenced on the existing RDataFrame AGC repository that applies CMS open data and the uproot AGC repository that uses ATLAS Open Data. The full implementations of the AGC can be found in [53].

5.1 RDataFrame Analysis Workflow

The AGC is divided into two parts: an analysis script and a statistical script, both written in Python. The analysis scripts uses RDataFrame to apply preselections and output histograms of the top quark mass and the scalar sum of the transverse momenta, H_T, into a ROOT file. The statistical script performs a simple statistical analysis using the output ROOT file from the analysis script.

The inputs used for the AGC are the same 92 ROOT file from ATLAS Open Data, as described in 3.1. Specifically, there are 22 single top samples, 10 $t\bar{t}$ samples, and 60 W+jets samples.

5.1.1 Event Selections

To reconstruct the top quark mass, events are selected from top quark pair production with final states that include a single charged lepton corresponding to the signature of semileptonic $t\bar{t}$ events, as shown in Figure 5.1. The leptons must have p_t larger than 30 GeV and $|\eta|$ less than 2.1 events must include four jets, with two of the four being b-tagged. The other two jets are from the W boson decay. The top mass observable is then reconstructed by taking the invariant mass of the trijet with the largest transverse momentum, p_t. To plot the H_T observable, the selected events must have at least one b-tagged jet among the four jets and exactly one lepton.

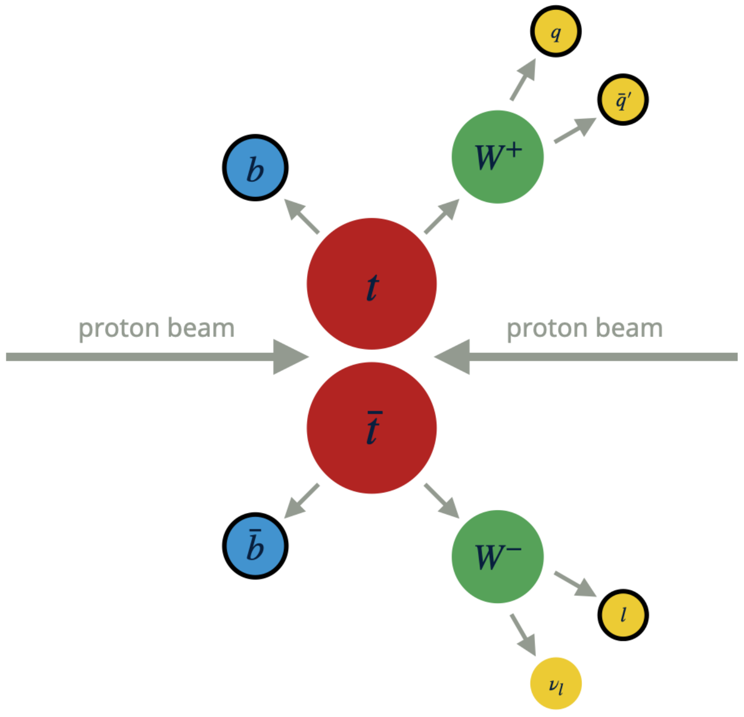


Figure 5.1: The schematic view of a top and anti-top quark collision [54]

The results of the newly developed AGC using ATLAS Open Data are shown in Figures 5.2 and 5.3. Both the RNTuple and TTree versions produced the same output, confirming that analysis performed in RDataFrame using RNTuple will remain largely unmodified. As previously mentioned, RNTuple only changes the structure of variable field names; therefore, alias variable names were applied to both TTree and RNTuple versions for consistency.

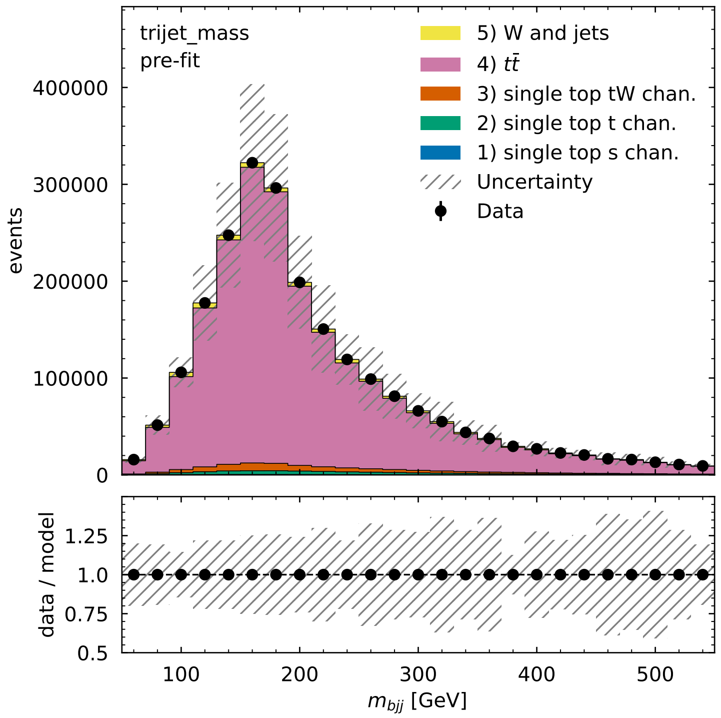


Figure 5.2: The trijet mass prefit. This result is the same for both RNTuple and TTree versions of the AGC.

5.2 AGC Performance Studies

A performance study evaluating execution speed and memory usage was conducted for both TTree and RNTuple versions of the AGC. Total execution times were measured 100 times for each version using the Python *time* library. Both versions used inputs produced with the ZSTD compression algorithm. As shown in Figure 5.4, RNTuple averaged 47.58 seconds to produce the top quark mass and H_T histograms into a ROOT file, while TTree averaged 71.75 seconds. RNTuple was approximately 1.51 times faster, consistent with

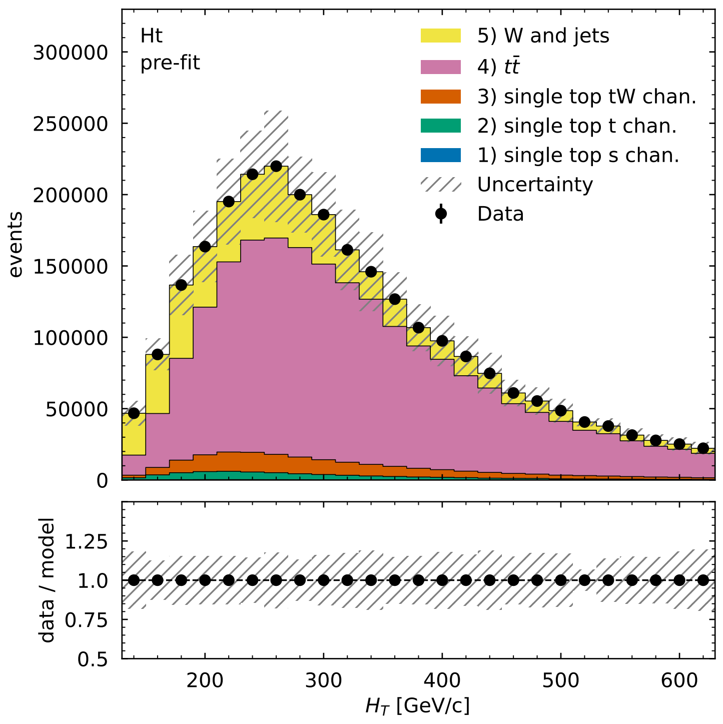


Figure 5.3: The H_T observable prefit. This result is the same for both RNTuple and TTree versions of the AGC.

previous time measurements shown in Chapter 4. The execution times were then remeasured using RNTuples produced with the LZ4 compression algorithm. As shown in Figure 5.5, LZ4 yields a slight improvement on the order of a few seconds, which is also consistent with previous results. Peak memory usage was measured using the inputs produces with ZSTD and with `/usr/bin/time`. As shown in Figure 5.6, RNTuple consumes slightly less memory usage than TTree when executing the AGC analysis script.

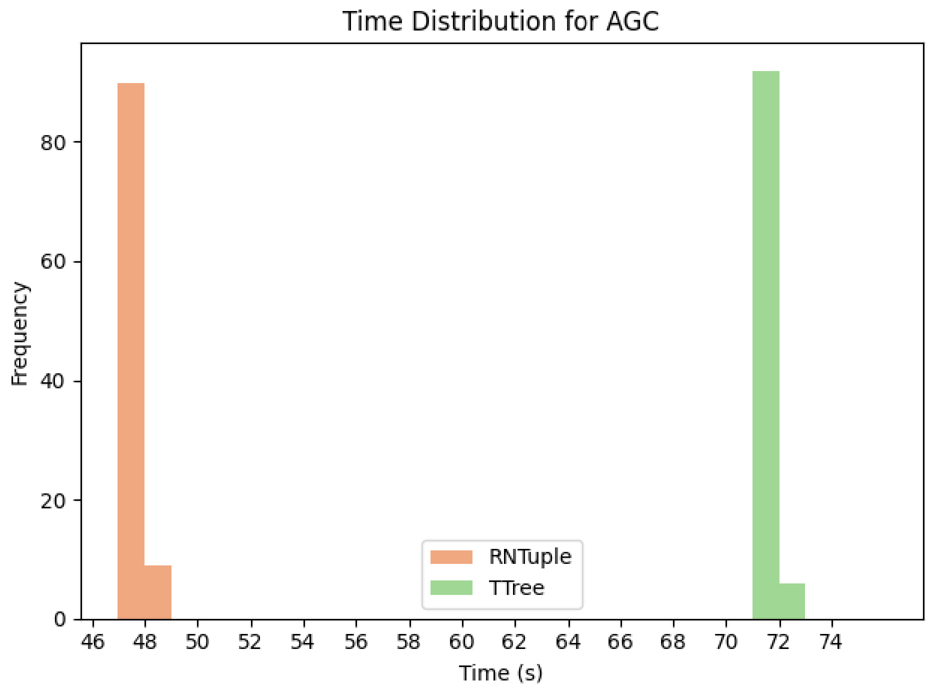


Figure 5.4: The total execution times of the AGC measured 100 times for TTree and RNTuple versions.

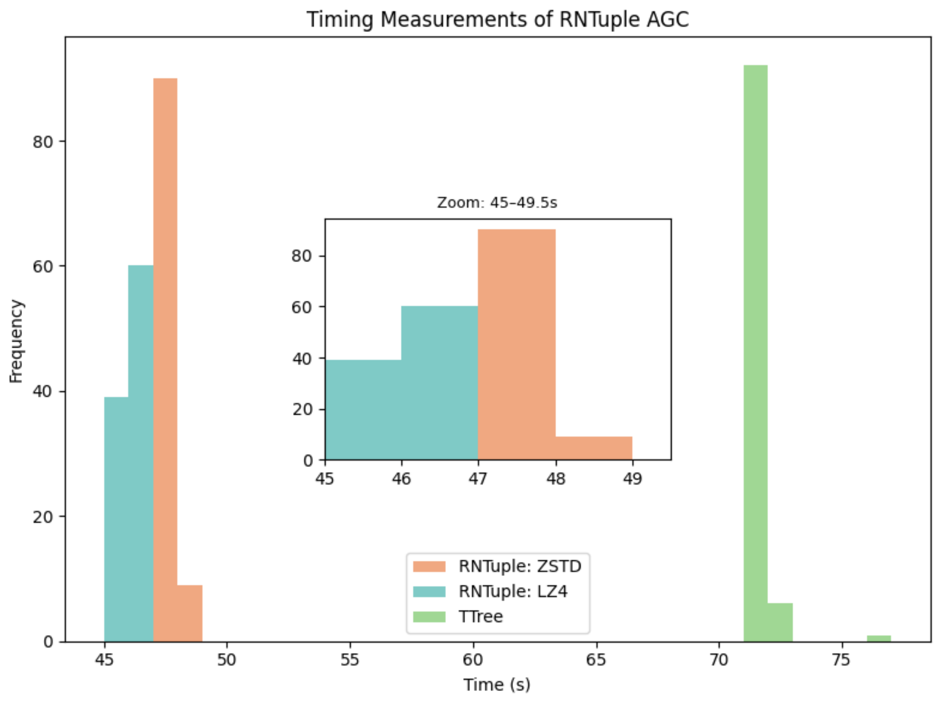


Figure 5.5: The total execution times of the AGC measured 100 times with RNTuples produced with the LZ4 compression algorithm.

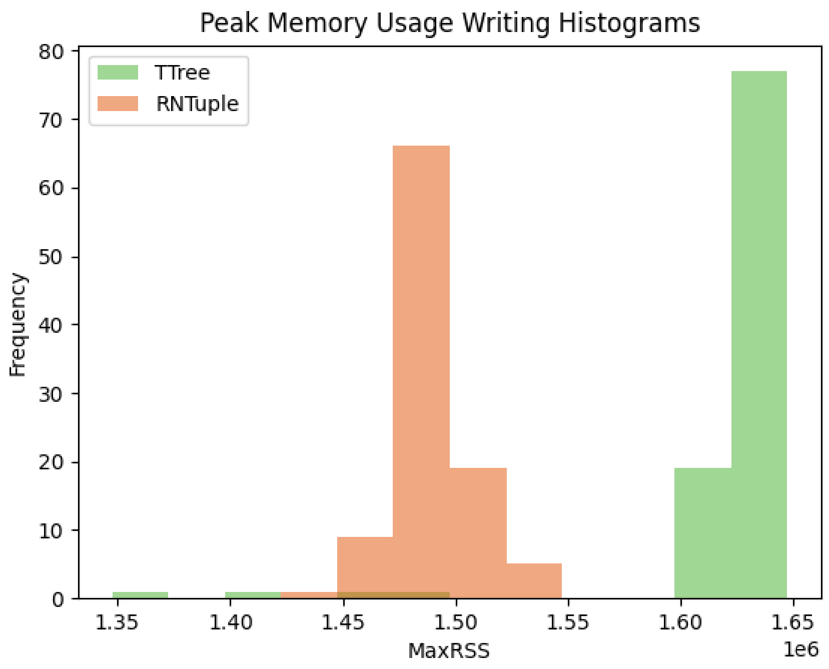


Figure 5.6: The peak memory usage when executing the AGC.

CHAPTER 6

CONCLUSION

RNTuple performance meets expectations, demonstrating improvements in reading and writing speed and disk space usage, with little variance on memory consumption. An average size reduction of 47% was observed when converting the TTree inputs to RNTuples using ZSTD compression algorithm. Using RDataFrame in C++, the average time execution for loading one column of data was found to be 2.4 times faster using RNTuple inputs versus TTree. The average time execution for writing a set with two columns was about 1.5 times faster for RNTuple than TTree. The peak memory usage measured while writing the two column output also improved by an average of 4,767 bytes. These improvements were achieved while preserving the RDataFrame workflow in C++ and requiring only minimal code changes, demonstrating a seamless transition from TTree to RNTuple.

An initial evaluation of RNTuple behavior using the ZSTD and LZ4 compression algorithms was conducted. Performance tests repeated with RNTuple inputs produced using the LZ4 compression algorithm show a 14% file size increase with no significant improvements in reading or writing speeds. Inputs produced with LZ4 exhibited only millisecond-level improvements when loading a field compared to RNTuple inputs produced with ZSTD. Writing speed also improved slightly, by about two seconds when using LZ4-produced RNTuple inputs versus those produced with ZSTD inputs. These minimal improvements in reading and writing speeds occur at the cost of larger file sizes. Given these result, the ZSTD compression algorithm is recommended when producing RNTuples.

The first implementations of the AGC using ATLAS Open Data were completed for both TTree and RNTuple inputs. The RNTuple version represents the first full end-to-end implementation of a physics analysis using RNTuple and demonstrates the feasibility and improvements of the format. The analysis script for the RNTuple AGC version remained largely unchanged from the TTree version, indicating a smooth analysis workflow transition within RDataFrame. For completeness, performance studies were repeated using the AGC and demonstrate consistent RNTuple performance. The total execution times for reading the 91 ATLAS Open Data inputs and writing the top mass and H_T histograms were improved by an average of about 24.17 seconds using RNTuple inputs; hence, the RNTuple version was 1.5 times faster than the TTree version. No significant improvements in speed were observed when executing the AGC with RNTuple inputs produced with LZ4. Overall, these consistent improvements in RNTuple performance reaffirm it as a promising data storage format for ATLAS at start of the HL-LHC.

REFERENCES

[1] Robert Mann. *An Introduction to Particle Physics and the Standard Model.* Taylor & Francis, 2010. ISBN: 978-1-4200-8300-2, 978-1-4200-8298-2, 978-0-429-14122-5. DOI: `10.1201/9781420083002` (cit. on p. 1).

[2] “ATLAS: Letter of intent for a general purpose p p experiment at the large hadron collider at CERN”. In: (Oct. 1992) (cit. on p. 1).

[3] “LHC Design Report Vol.1: The LHC Main Ring”. In: (June 2004). Ed. by Oliver S. Bruning et al. DOI: `10.5170/CERN-2004-003-V-1` (cit. on p. 1).

[4] ATLAS Collaboration. *Trigger and Data Acquisition System.* `https://atlas.cern/Discover/Detector/Trigger-DAQ`. Accessed: 2025-11-07. CERN / ATLAS Experiment, 2025 (cit. on p. 1).

[5] I. Zurbano Fernandez et al. “High-Luminosity Large Hadron Collider (HL-LHC): Technical design report”. In: 10/2020 (Dec. 2020). Ed. by I. Béjar Alonso et al. DOI: `10.23731/CYRM-2020-0010` (cit. on pp. 1, 16).

[6] Blomer, Jakob et al. “ROOT’s RNTuple I/O Subsystem: The Path to Production”. In: *EPJ Web of Conf.* 295 (2024), p. 06020. DOI: `10.1051/epjconf/202429506020`. URL: `https://doi.org/10.1051/epjconf/202429506020` (cit. on p. 2).

[7] CERN ROOT Team. *ROOT Reference Documentation — Master Version.* `https://root.cern.ch/doc/master/index.html`. Accessed: 2025-11-07. CERN, 2025 (cit. on p. 2).

[8] Jakob Blomer et al. “Evolution of the ROOT Tree I/O”. In: *EPJ Web of Conferences* 245 (2020). Ed. by C. Doglioni et al., p. 02030. ISSN: 2100-014X. DOI: `10.1051/epjconf/202024502030`. URL: `http://dx.doi.org/10.1051/epjconf/202024502030` (cit. on p. 2).

[9] Javier Lopez-Gomez and Jakob Blomer. “RNTuple performance: Status and Outlook”. In: *Journal of Physics: Conference Series* 2438.1 (Feb. 2023), p. 012118. ISSN: 1742-

6596. DOI: 10.1088/1742-6596/2438/1/012118. URL: http://dx.doi.org/10.1088/1742-6596/2438/1/012118 (cit. on p. 2).

[10] IRIS-HEP. *The Analysis Grand Challenge*. https://iris-hep.org/projects/agc.html. Accessed: 2025-11-07. IRIS-HEP, 2025 (cit. on p. 2).

[11] Wikimedia Foundation. *Elementary particle*. https://en.wikipedia.org/wiki/Elementary_particle. Accessed: 2025-11-07. Wikimedia Foundation, 2025 (cit. on p. 3).

[12] "Standard Model Summary Plots June 2024". In: (2024) (cit. on p. 6).

[13] ATLAS Collaboration. *Mass / Invariant mass — Glossary of Terms*. https://atlas.cern/glossary/mass. Accessed: 2025-11-07. CERN / ATLAS Experiment, 2025 (cit. on p. 5).

[14] Dave Van Wijk. *4-vectors and invariant mass cheat sheet*. https://atlas.cern/node/38632. Accessed: 2025-11-07. ATLAS Collaboration, CERN, 2024 (cit. on p. 5).

[15] ATLAS Collaboration. *Jets — Physics objects documentation, ATLAS OpenData*. https://opendata.atlas.cern/docs/documentation/physic_objects/jets. Accessed: 2025-11-07. CERN / ATLAS Experiment, 2025 (cit. on p. 7).

[16] CERN. *hadron — Tag page*. https://home.cern/tags/hadron. Accessed: 2025-11-07. CERN, 2025 (cit. on p. 7).

[17] Georges Aad et al. "ATLAS b-jet identification performance and efficiency measurement with $t\bar{t}$ events in pp collisions at $\sqrt{s} = 13$ TeV". In: *Eur. Phys. J. C* 79.11 (2019), p. 970. DOI: 10.1140/epjc/s10052-019-7450-8. arXiv: 1907.05120 [hep-ex] (cit. on p. 8).

[18] ATLAS Collaboration. "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 (Sept. 2012), pp. 1–29. ISSN: 0370-2693. DOI: 10.1016/j.physletb.2012.08.020. URL: http://dx.doi.org/10.1016/j.physletb.2012.08.020 (cit. on p. 9).

[19] ATLAS Collaboration. "Observation of a new boson at a mass of 125 GeV with the CMS experiment at the LHC". In: *Physics Letters B* 716.1 (Sept. 2012), pp. 30–61.

ISSN: 0370-2693. DOI: 10.1016/j.physletb.2012.08.021. URL: http://dx.doi.org/10.1016/j.physletb.2012.08.021 (cit. on p. 9).

[20] Lyndon Evans and Philip Bryant. “LHC Machine”. In: *Journal of Instrumentation* 3.08 (Aug. 2008), S08001. DOI: 10.1088/1748-0221/3/08/S08001. URL: https://doi.org/10.1088/1748-0221/3/08/S08001 (cit. on p. 9).

[21] S. Amato et al. “LHCb technical proposal: A Large Hadron Collider Beauty Experiment for Precision Measurements of CP Violation and Rare Decays”. In: (Feb. 1998) (cit. on p. 9).

[22] “ALICE: Technical proposal for a large ion collider experiment at the CERN LHC”. In: (Dec. 1995) (cit. on p. 9).

[23] CERN. *Theacceleratorcomplex.* https://home.cern/science/accelerators/accelerator-complex. Accessed: 2025-11-07. CERN, 2025 (cit. on p. 10).

[24] ATLAS Collaboration. “Luminosity determination in pp collisions at

$$\sqrt{s} = 13$$

TeV using the ATLAS detector at the LHC”. In: *The European Physical Journal C* 83.10 (Oct. 2023). ISSN: 1434-6052. DOI: 10.1140/epjc/s10052-023-11747-w. URL: http://dx.doi.org/10.1140/epjc/s10052-023-11747-w (cit. on p. 11).

[25] ATLAS Collaboration. “Alignment of the ATLAS Inner Detector in Run 2”. In: *The European Physical Journal C* 80.12 (Dec. 2020). ISSN: 1434-6052. DOI: 10.1140/epjc/s10052-020-08700-6. URL: http://dx.doi.org/10.1140/epjc/s10052-020-08700-6 (cit. on p. 11).

[26] Steffen Starz. “ATLAS Calorimeter system: Run-2 performance, Phase-1 and Phase-2 upgrades”. In: (2018). URL: https://cds.cern.ch/record/2628123 (cit. on p. 11).

[27] ATLAS Collaboration. “Performance of the ATLAS muon triggers in Run 2”. In: *Journal of Instrumentation* 15.09 (Sept. 2020), P09015–P09015. ISSN: 1748-0221. DOI: 10.1088/1748-0221/15/09/p09015. URL: http://dx.doi.org/10.1088/1748-0221/15/09/p09015 (cit. on p. 11).

[28] William Panduro Vazquez and on behalf of the ATLAS Collaboration. “The ATLAS Data Acquisition System in LHC Run 2”. In: *Journal of Physics: Conference Series* 898.3 (Oct. 2017), p. 032017. DOI: 10.1088/1742-6596/898/3/032017. URL: https://doi.org/10.1088/1742-6596/898/3/032017 (cit. on p. 11).

[29] ATLAS Collaboration. *ATLAS Schematics — Free-to-download schematics of the ATLAS detector*. https://atlas.cern/Resources/Schematics. Accessed: 2025-11-07. CERN / ATLAS Experiment, 2025 (cit. on pp. 12–15).

[30] “Technical Design Report for the ATLAS Inner Tracker Pixel Detector”. In: (2017). DOI: 10.17181/CERN.FOZZ.ZP3Q (cit. on p. 16).

[31] “A High-Granularity Timing Detector for the ATLAS Phase-II Upgrade: Technical Design Report”. In: () (cit. on p. 17).

[32] “Technical Design Report for the Phase-II Upgrade of the ATLAS Muon Spectrometer”. In: () (cit. on p. 17).

[33] “ATLAS Liquid Argon Calorimeter Phase-II Upgrade : Technical Design Report”. In: (). DOI: 10.17181/CERN.6QIO.YGHO (cit. on p. 17).

[34] “Technical Design Report for the Phase-II Upgrade of the ATLAS Tile Calorimeter”. In: () (cit. on p. 17).

[35] Junjie Zhu. “The Phase-II upgrade of the ATLAS Muon Spectrometer”. In: *PoS* LeptonPhoton2019 (2019), p. 070. DOI: 10.22323/1.367.0070 (cit. on p. 17).

[36] “Technical Design Report for the Phase-II Upgrade of the ATLAS TDAQ System”. In: (). DOI: 10.17181/CERN.2LBB.4IAL (cit. on p. 17).

[37] ATLAS Collaboration et al. “Athena”. In: *Zenodo* (2019) (cit. on p. 18).

[38] ATLAS Collaboration. *DAOD_PHYSLITE format 2015–2016 Open Data for Research from the ATLAS experiment*. Accessed: 2025-11-07. CERN Open Data Portal, 2024. DOI: 10.7483/OPENDATA.ATLAS.9HK7.P5SI. URL: https://opendata.cern.ch/record/300 (cit. on p. 18).

[39] J. Catmore and ATLAS Collaboration. *The ATLAS data processing chain: from collisions to papers*. CERN Indico event 472469. Presentation slides at ATLAS Collaboration meeting, CERN; available at https://indico.cern.ch/event/472469/contributions/1982677/attachments/1220934/1785823/intro_slides.pdf (accessed 2025-11-07). 2020 (cit. on p. 19).

[40] ATLAS Collaboration. *ATLAS DAOD_PHYSLITE format MC simulation top nominal samples*. Accessed: 2025-11-07. CERN Open Data Portal, 2024. DOI: 10.7483/OPENDATA.ATLAS.MM1Y.OOPH. URL: https://opendata.cern.ch/record/301 (cit. on p. 19).

[41] ATLAS Collaboration. *ATLAS DAOD_PHYSLITE format MC simulation electroweak boson nominal samples*. Accessed: 2025-11-07. CERN Open Data Portal, 2024. DOI: 10.7483/OPENDATA.ATLAS.K5SU.X65Y. URL: https://opendata.cern.ch/record/302 (cit. on p. 19).

[42] ATLAS Collaboration. *PHYSLITE – a new reduced common data format for ATLAS*. Presentation slides, ATL-SOFT-SLIDE-2023-158, CERN Document Server. Accessed: 2025-11-07. 2023. URL: https://cds.cern.ch/record/2857821/files/ATL-SOFT-SLIDE-2023-158.pdf (cit. on p. 19).

[43] ATLAS Collaboration. *PHYSLITE — Documentation on analysis variables for ATLAS Open Data*. https://atlas-physlite-content-opendata.web.cern.ch/. Accessed: 2025-11-07. 2025 (cit. on p. 19).

[44] Caterina Marcon et al. "Optimizing ATLAS data storage: the impact of compression algorithms on ATLAS physics analysis data formats". In: *EPJ Web Conf.* 295 (2024), p. 03027. DOI: 10.1051/epjconf/202429503027 (cit. on pp. 21, 35).

[45] Brian Bockelman and Oksana Shadura. *Zstd & LZ4*. Presentation slides at DIANA/HEP Workshop (FNAL Indico 16264). Accessed: 2025-11-07. 2021. URL: https://indico.fnal.gov/event/16264/contributions/36466/attachments/22610/28037/Zstd__LZ4.pdf (cit. on p. 21).

[46] CERN ROOT Team. *TTree Class Reference — ROOT v6-30*. Version 6-30; accessed 2025-11-09. CERN. 2024. URL: https://root.cern.ch/doc/v630/classTTree.html (cit. on p. 23).

[47] ROOT Team. *RNTuple Binary Format Specification 1.0.0.2.* Accessed: 2025-11-09. CERN / ROOT Project. 2024. URL: `https://root.cern/doc/master/md_tree_2ntuple_2doc_2BinaryFormatSpecification.html` (cit. on p. 22).

[48] Google Cloud. *What is binary large object (BLOB) storage?* 2025. URL: `https://cloud.google.com/discover/what-is-binary-large-object-storage` (visited on 11/09/2025) (cit. on p. 23).

[49] Alaettin Serhan Mete et al. *Persistifying the Complex Event Data Model of the ATLAS Experiment in RNTuple.* Tech. rep. ATL-SOFT-PROC-2024-002. Accessed: 2025-11-09. CERN / ATLAS Experiment, 2024. URL: `https://cds.cern.ch/record/2905189/files/ATL-SOFT-PROC-2024-002.pdf` (cit. on p. 24).

[50] scikit-hep / Uproot Developers. *uproot: ROOTI/O in pure Python and NumPy.* Version 5.6.8. Accessed: 2025-11-09. 2025. URL: `https://pypi.org/project/uproot/` (cit. on p. 24).

[51] Fatima Rodriguez and faarodri (GitLab). *RNTuple-for-analysis-workflows.* GitLab repository, CERN: `https://gitlab.cern.ch/faarodri/rntuple-for-analysis-workflows`. Accessed: 2025-11-11. 2025 (cit. on p. 25).

[52] Wikimedia Foundation. *Smart pointer.* `https://en.wikipedia.org/wiki/Smart_pointer`. Accessed: 2025-11-09. 2025 (cit. on p. 26).

[53] Fatima (faarodri) Rodriguez. *RNTuple for Analysis Workflows.* Version master. Accessed: 2025-11-11. 2025. URL: `https://gitlab.cern.ch/faarodri/rntuple-for-analysis-workflows` (cit. on p. 39).

[54] Alexander Held and Oksana Shadura. *End-to-end physics analysis with Open Data: the Analysis Grand Challenge (PyHEP 2022 talk notebook).* GitHub repository: `https://github.com/alexander-held/PyHEP-2022-AGC/blob/main/talk.ipynb`. Accessed: 2025-11-11. IRIS-HEP / HEP Software Foundation, 2022 (cit. on p. 40).