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, all metrics were measured 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

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.

DEDICATION

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

TABLE OF CONTENTS

Page

LIST OF TABLES vi

LIST OF FIGURES. vii

Chapter

1 INTRODUCTION 1

1.1 Phenomenology at the LHC 2

1.2 Physics Quantities. 5

1.2.1 Invariant Mass 5

1.2.2 Jets 8

2 THE ATLAS EXPERIMENT 9

2.1 The Large Hadron Collider 9

2.2 The ATLAS Apparatus 10

2.2.1 The Inner Detector 13

2.2.2 Calorimeter Systems 14

2.2.3 Muon Spectrometer 14

2.2.4 Magnet System. 15

2.2.5 ATLAS Trigger System 16

2.3 HL-LHC. 17

3 ATLAS SOFTWARE AND COMPUTING 18

3.1 ATLAS Open Data 19

3.2 ROOT 20

3.2.1 ROOT Compression Algorithms. 21

Chapter Page

3.2.2 TTree Data Structure . . . 22
3.2.3 RNTuple Data Structure . . . 24
3.3 TTree vs. RNTuple API . . . 26
3.3.1 Native C++ Event Loops . . . 26
3.3.2 RDataFrame in C++ and Python . . . 27
3.3.3 Uproot . . . 27
4 RNTUPLE VS. TTREE PERFORMANCE . . . 30
4.1 Readability Speed . . . 31
4.2 Writing Speed . . . 32
4.3 Output Sizes . . . 35
4.4 Memory Consumption . . . 36
4.5 `LZ4` Compression Algorithm Study . . . 36
4.6 Performance Discussion . . . 38
5 ANALYSIS GRAND CHALLENGE: RNTUPLE VS. TTREE . . . 41
5.1 RDataFrame Analysis Workflow . . . 41
5.1.1 Event Selections . . . 42
5.2 AGC Performance Studies . . . 44
6 CONCLUSION . . . 49
REFERENCES . . . 51

LIST OF TABLES

Table Page

4.1 File Size and Avg. Event Size of UnFiltered Output 35

4.2 File Size and Avg. Event Size of Filtered Output 36

LIST OF FIGURES

Figure		Page
1.1	The Standard Model	3
1.2	Summary of SM Cross-section Measurements	6
1.3	Invariant Mass Distribution of Oppositely Charged Electron Pairs	7
2.1	The CERN Accelerator Complex	11
2.2	The ATLAS Detector	12
2.3	ATLAS Inner Detector Schematics	13
2.4	ATLAS Calorimeter System	15
2.5	ATLAS Muon Spectrometer	16
3.1	Data Chain	19
3.2	Data Processes	20
3.3	TTree Data Structure	23
3.4	TTree vs. RNTuple Mapping	25
3.5	Native C++ Event Loop: TTree vs. RNTuple	28
3.6	Reading Multiple Inputs in RDataFrame Python: TTree vs. RNTuple	29
3.7	Applying an RDataFrame Filter in C++ : TTree vs. RNTuple	29
4.1	Per-file Disk Size Ratios RNTuple:TTree	31
4.2	Distribution of Total Loading Times	32
4.3	Writing a Two Column Output Algorithm Using RDataFrame in C++	33
4.4	Distribution of Total Writing Times	34

Figure Page

4.5 Peak Memory Usage: Writing Two Column Output 37
4.6 Distribution of Loading Times Using LZ4 Inputs 38
4.7 Distribution of Writing Times Using LZ4 Inputs. 39
4.8 Per-file Disk Size Ratios of RNTuples LZ4:ZSTD 40

5.1 Top and Anti-top Quark Collision. 43
5.2 The Trijet Mass Prefit 44
5.3 H_T. 45
5.4 Distribution of AGC Execution Times. 46
5.5 Distribution of AGC Execution Times using LZ4 Inputs 47
5.6 AGC Peak Memory Usage 48

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 predicting the electron magnetic dipole moment [2], to the Higgs boson [3], 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 do they 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 direct searches for physics beyond the SM. 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) [4] is a particle physics experiment designed to detect the high-energy particle collisions from the Large Hadron Collider (LHC) [5]. 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 [6]. 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) [7], will require a new data storage format that can handle this increase in data.

RNTuple [8] is the new ROOT [9] data storage format that will be in use at the start of the HL-LHC [10]. RNTuple takes advantage of modern C++ techniques, which have shown to improve read speed ability and memory usage when compared to its predecessor, TTree [11], and other data storage formats such as HDF5 [12] and Parquet [13, 14]. 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) [15] 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 par-

ticle interactions, except for gravity, through a collection of quantum field theories: QED, the Glashow-Weinberg-Salam theory of electroweak processes [16, 17, 18], 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. All bosons carry an integer spin; while, fermions carry half-integer spin.

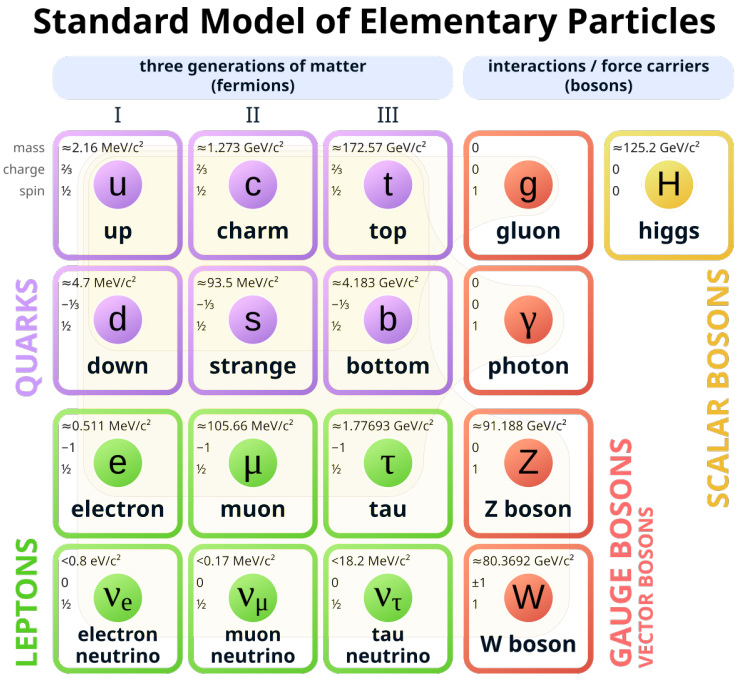


Figure 1.1: Particle content of the Standard Model [19].

Fermions are the particles that make up matter. Each fermion has an antiparticle with the same mass and lifetime as the particle itself, but the antiparticle is 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 [20, 21] is responsible for giving other elementary particles their mass and for electroweak symmetry breaking. It has spin 0 and is categorized separately from the spin-1 vector bosons as a scalar boson.

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 crossing of particle beams 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, and 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 its cross section, 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 [22]. 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 [23]. 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

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.

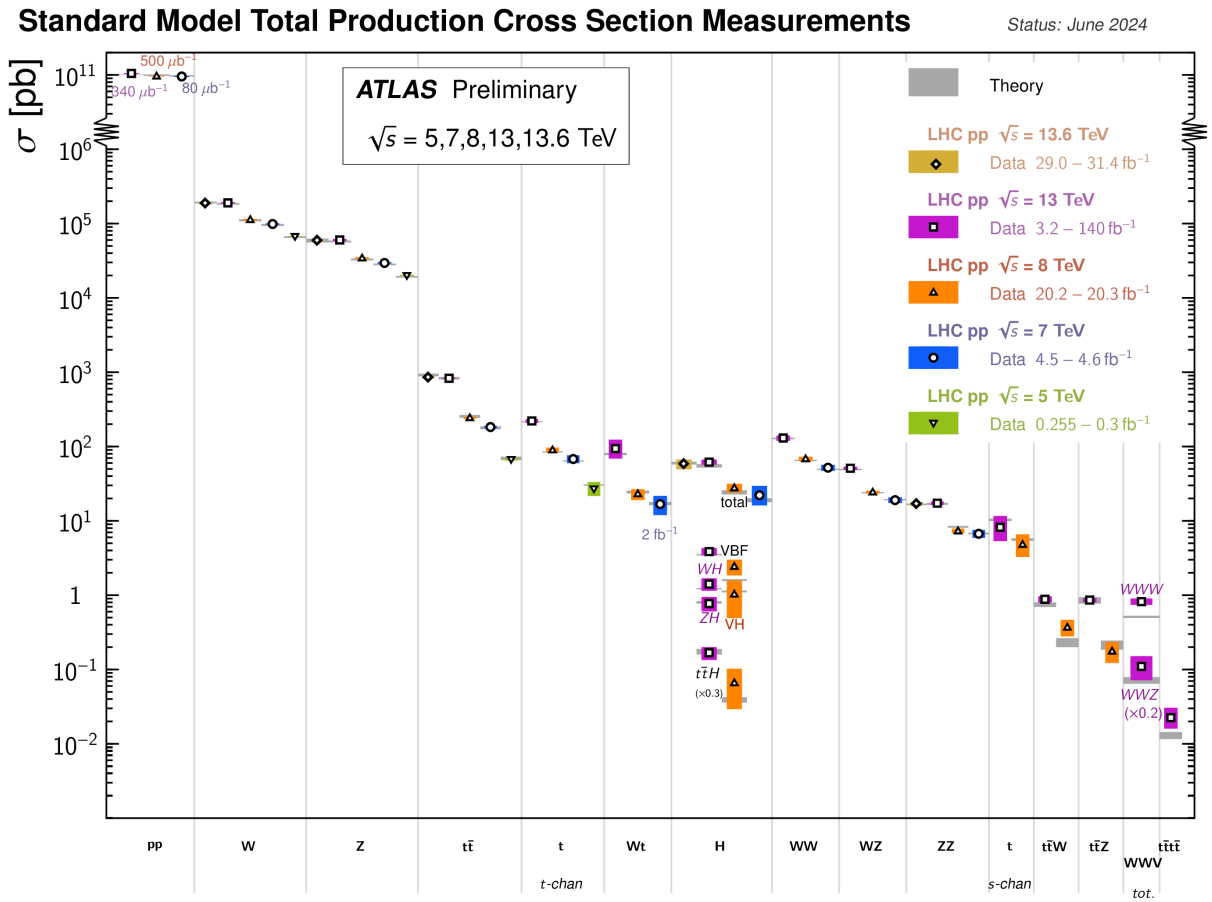


Figure 1.2: Summary of several Standard Model cross-section measurements by the ATLAS Collaboration [24]. The measurements are corrected for branching fractions, compared to the corresponding theoretical expectations.

As an example, a Z boson can decay into a pair of oppositely charged electrons or oppositely charged muons. The ATLAS detector will record the leptons' momentum in x, y, and z directions (p_x, p_y, p_z), their energy (E), and other kinematic information. Using this information, the invariant mass of the lepton pair can be calculated using Equation 1.3 to ultimately confirm their origin. Figure 1.3 displays an example distribution of invariant mass values for oppositely charged electron pairs using an electroweak boson sample from ATLAS Open Data [25]. The peak of the distribution returns the expected Z boson mass at 91.25 GeV.

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

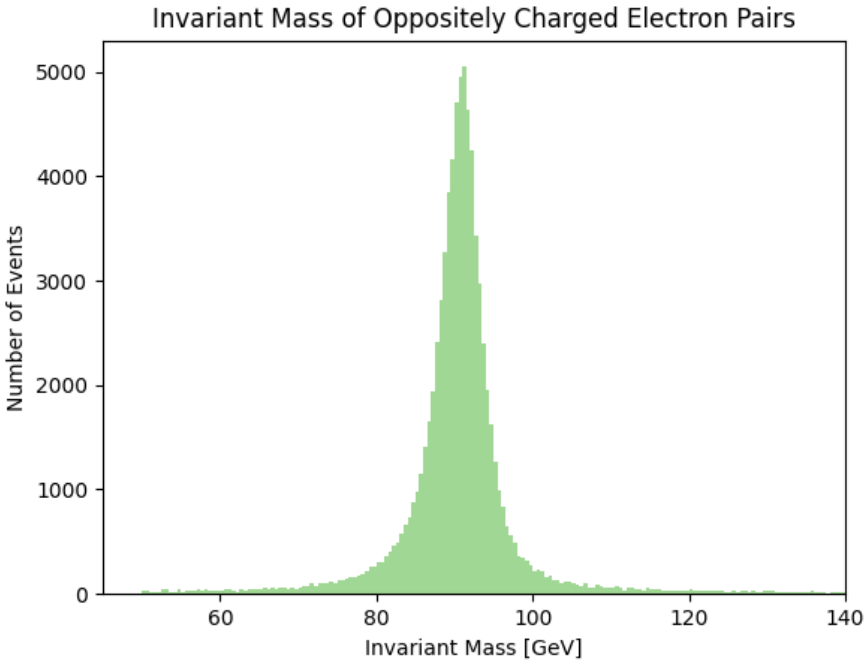


Figure 1.3: Invariant Mass distribution of oppositely charged electron pairs using electroweak boson sample from ATLAS Open Data [25].

In Chapter 4, an invariant mass calculation is performed using oppositely charged electron and muon pairs. This calculation was done as an initial benchmark to compare RNTuple versus its predecessor TTree.

1.2.2 Jets

Jets are energy deposits in the detector that are grouped together to represent quarks and gluons, collectively known as partons [26]. Due to color charge, quarks are permanently confined within hadrons [27]. When a parton is produced in a collision, it undergoes a parton shower and hadronization chain that produces a collimated grouping of hadrons that shares properties with the original parton. As a result, jets are used as physical proxies for partons and are reconstructed using various algorithms to different types of objects, such as energy deposits or tracks.

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% [28].

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

CHAPTER 2

THE ATLAS EXPERIMENT

ATLAS is 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 [29]. By November 2009, ATLAS recorded its first proton-proton collision [30] and by December 2010, ATLAS observed its first top quark pairs, which are the heaviest known elementary particle with a strong coupling to the Higgs boson [31]. By July 2012, both ATLAS and the Compact Muon Spectrometer (CMS) experiment successfully observed the Higgs boson [20, 21]. ATLAS is projected to continue operation until 2041 to continue searching for standing questions from the SM.

This chapter will provide a 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 Européen pour la Recherche Nucléaire (CERN) [32]. It was approved for construction in 1996 to search for beyond the SM physics at energies larger than 10 TeV. Its approval was heavily influenced by the cost-saving idea of reusing the existing 26.5 km tunnels from the Large Electron-Positron (LEP) collider. The LHC has four main collision points that house the ATLAS, CMS, Large Hadron Collider beauty (LHCb) [33], and A Large Ion Collider Experiment (ALICE) [34]. 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.

During Run 2, the LHC collided protons at a center of mass energy of $\sqrt{s} = 13$ TeV with a combined integrated luminosity of $L = 140\ fb^{-1}$ [35]. 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) [36], which is responsible for tracking charged-particles. A calorimeter system follows and measures the energy loss of the particles passing through the detector [37]. The final subsystem is the Muon Spectrometer (MS) [38], 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 [39].

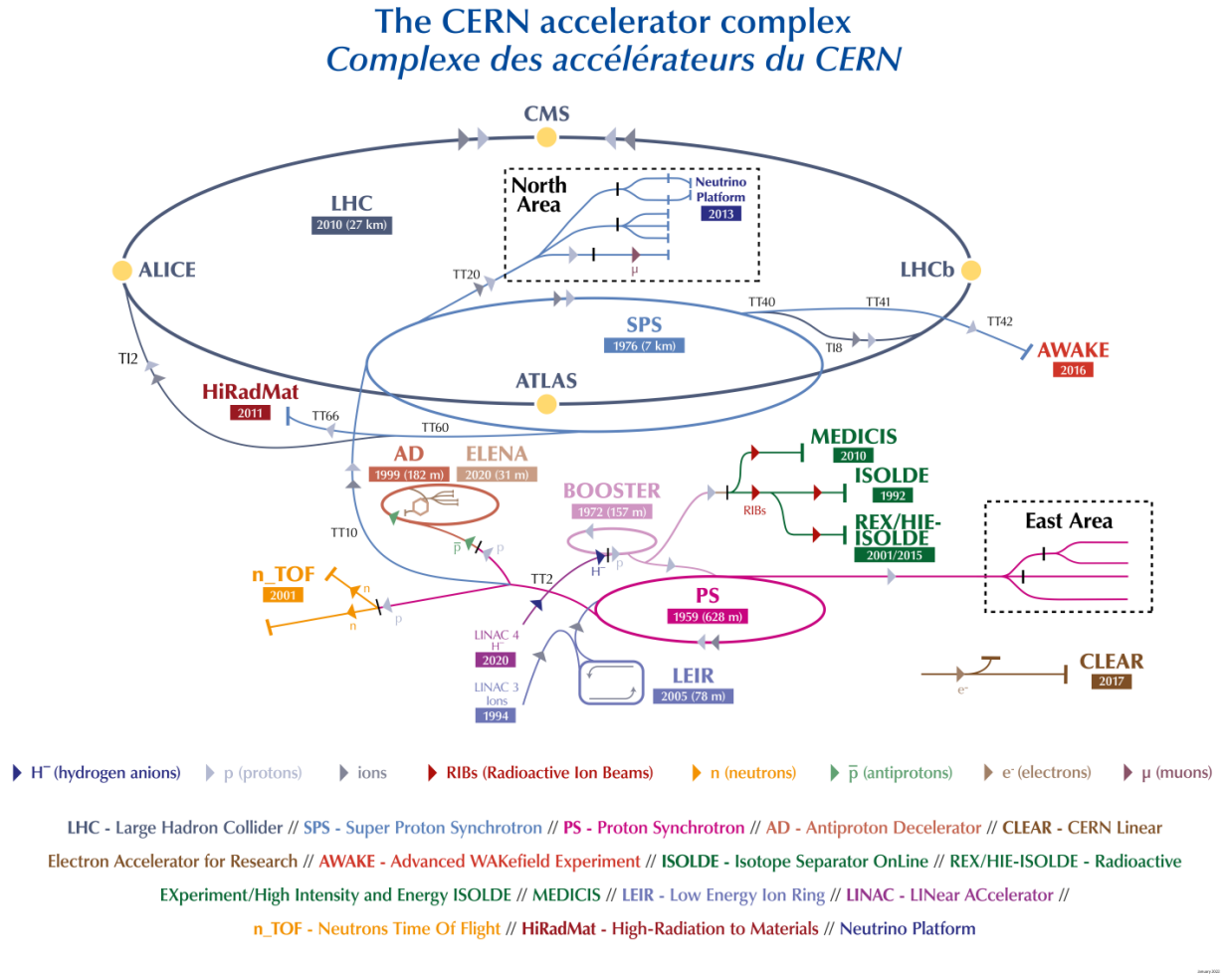


Figure 2.1: The CERN accelerator complex [40].

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.

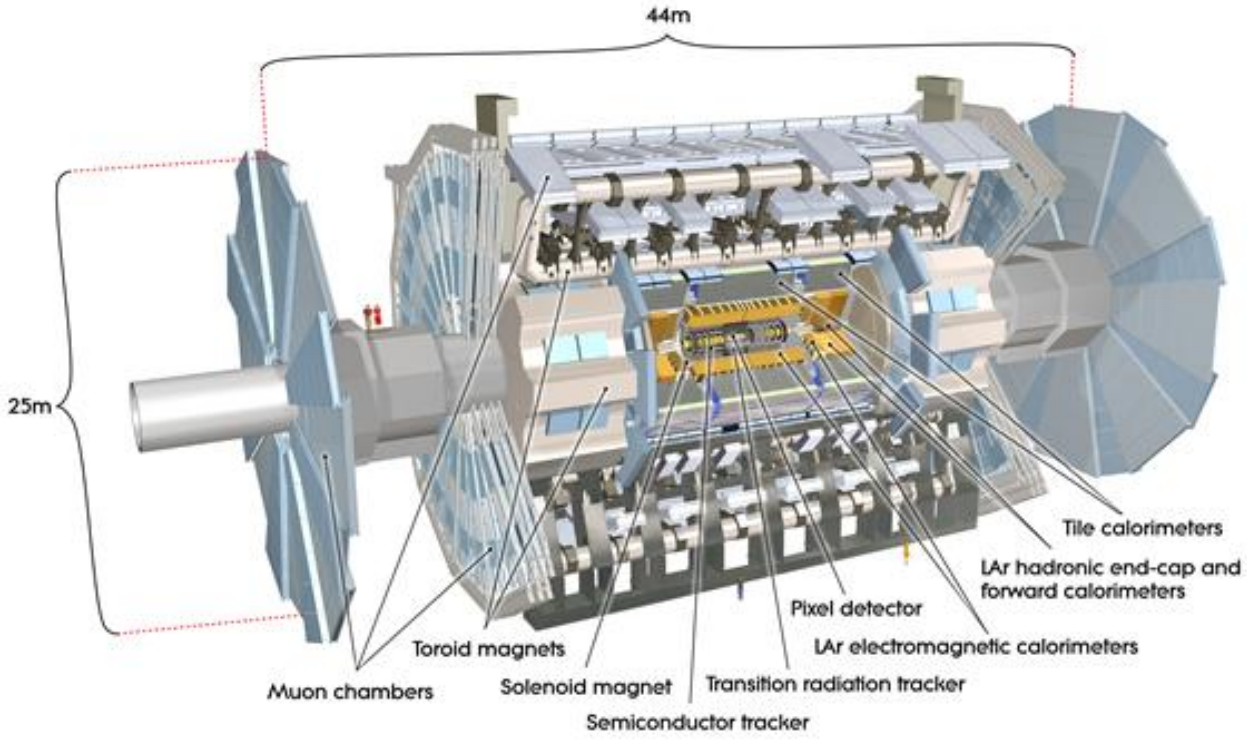


Figure 2.2: Computer generated image of the whole ATLAS detector [41].

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.

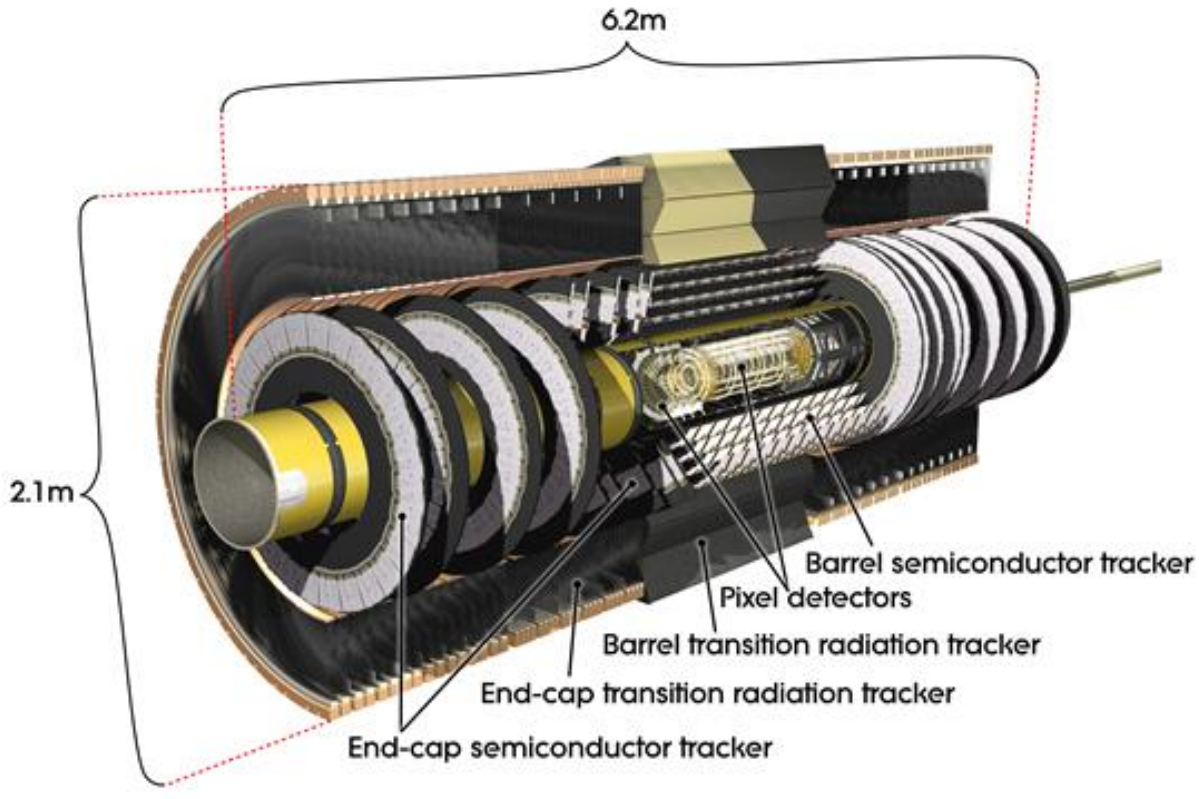


Figure 2.3: Computer generated image of the ATLAS inner detector [41].

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.

2.2.3 Muon Spectrometer

The muon spectrometer, shown in Figure 2.5, is the outer part of the ATLAS detector, designed to measure 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 ionize the gas mixtures to generate signal.

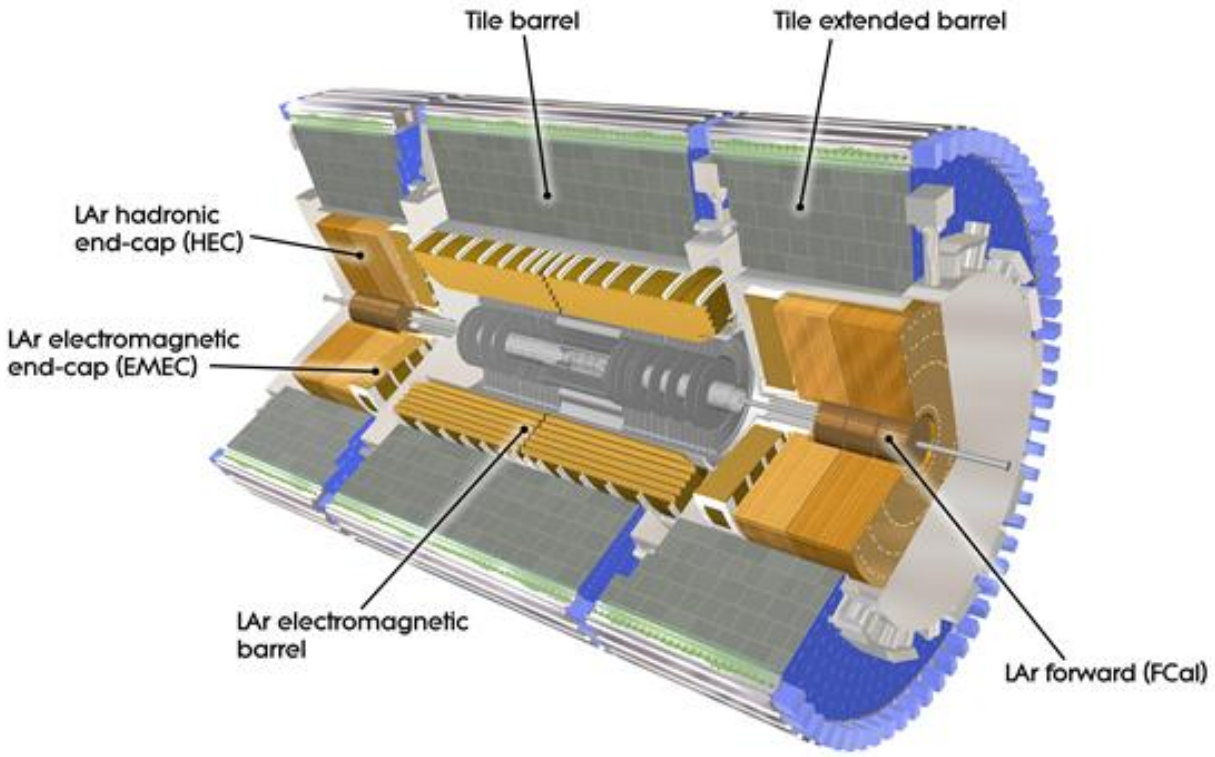


Figure 2.4: Computer generated image of the ATLAS calorimeter [41].

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 MS.

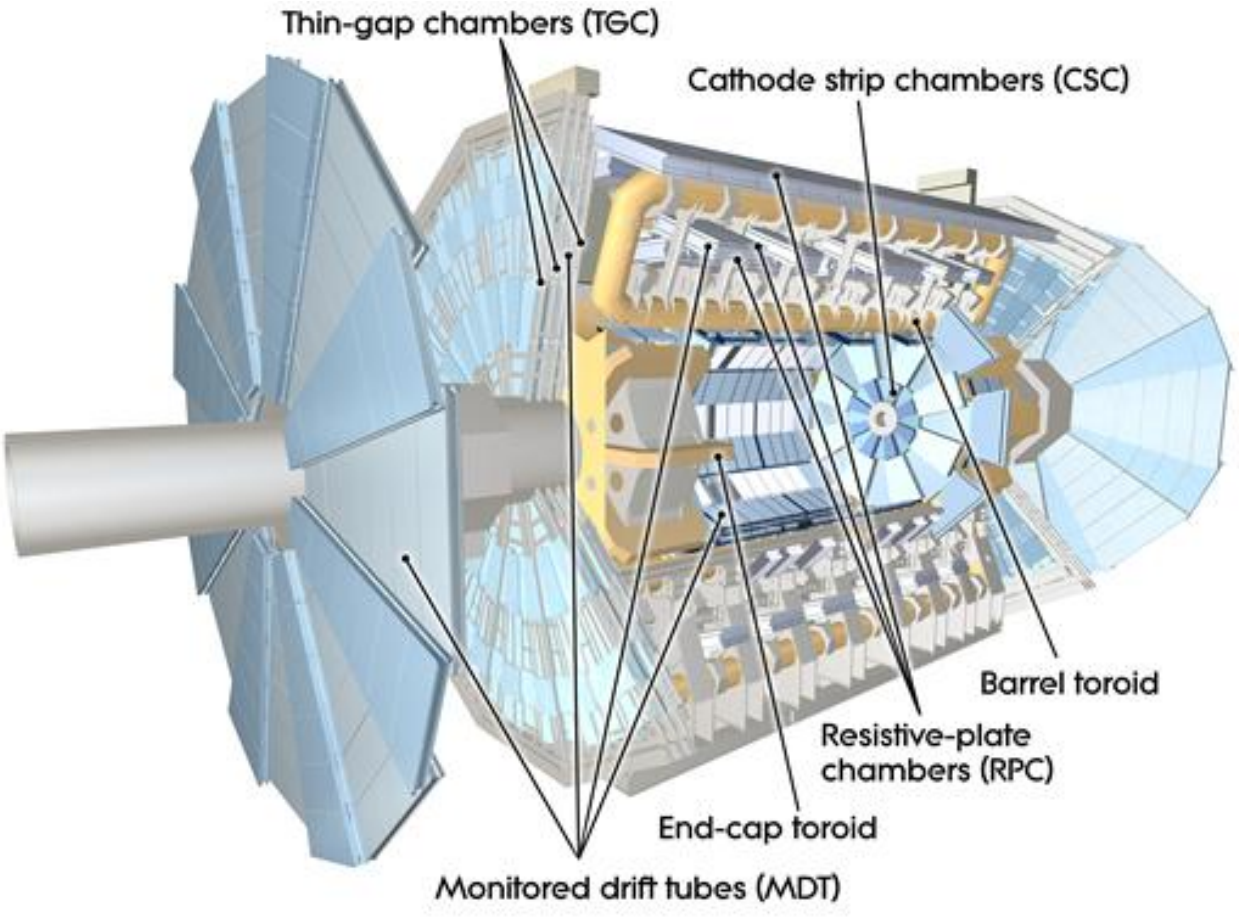


Figure 2.5: Computer generated image of the ATLAS Muons subsystems [41].

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, 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 [7]. 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 sub-detectors will be installed such as the Inner Tracker [42, 43], the High Granularity Timing Detector [44], and additional Muon chambers [45]. There will also be different electronics upgrades such as the Liquid Argon Calorimeter [46], the Tile Calorimeter [47], and the Trigger and Data Acquisition (TDAQ) system [48].

CHAPTER 3

ATLAS SOFTWARE AND COMPUTING

The data collected from the detector must be compared to a set of simulated data in order to interpret efficiencies and background processes. These data sets aim to mimic different physics processes such as the events produced by the collider beams, the evolution of the collision products within the detector and materials, and the detector's response. The preparation of simulation starts off with Monte Carlo (MC) simulation, 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 are 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. The software framework that encompasses the tools needed to produce, validate, and analyze all of these types of samples is called Athena [49]. The flow of this process, compared to the similar process followed by collision data, is shown in Figure 3.1.

The studies for this thesis use data at the analysis level. This chapter will introduce ATLAS Open Data [50], ROOT, and the data structures of TTree versus RNTuple. It will also provide a comparison between the application programming interface (API) for the TTree and RNTuple formats.

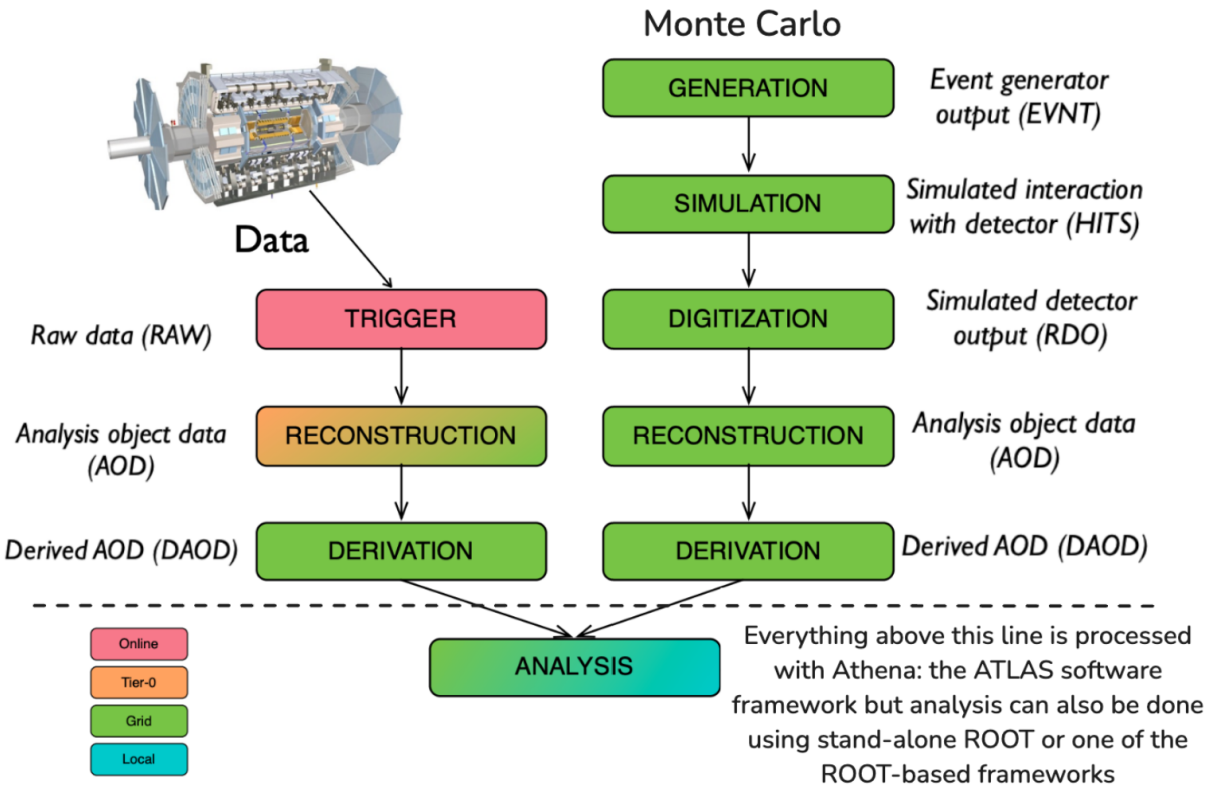
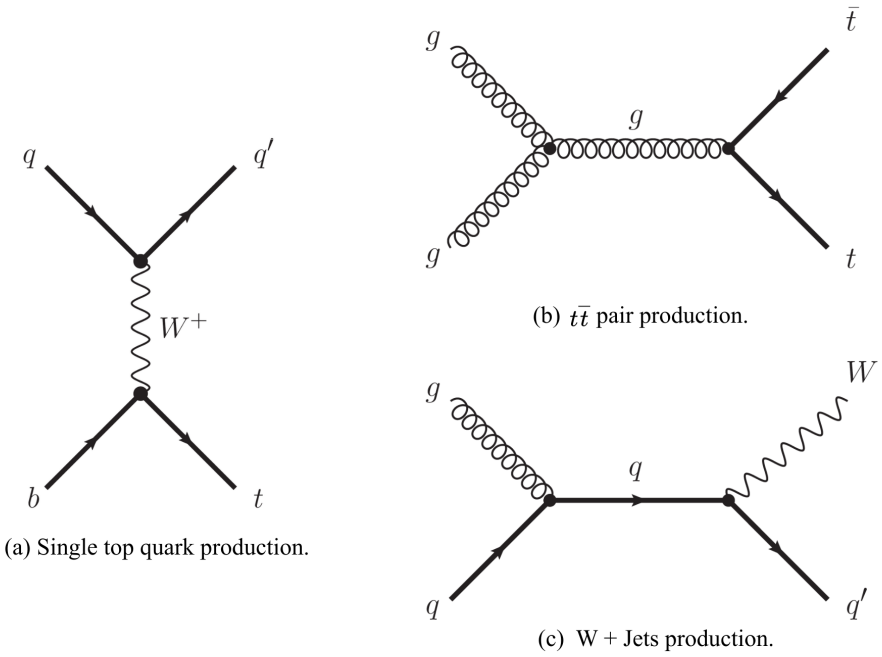


Figure 3.1: ATLAS data chain-processing for data and Monte Carlo simulation [51].

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 simulations of top samples from Run 2 [52]. They are simulated samples of 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.



(a) Single top quark production.

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

(c) W + Jets production.

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

The inputs are all provided in DAOD_PHYSLITE format, which contains already-calibrated objects directly from an AOD or PHYS product [53]. Those objects include jets, electrons, muons, photons, taus and their properties, such as 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 [54].

3.2 ROOT

ROOT is a unified software package developed for processing, analyzing, visualizing and ultimately storing the massive high-energy physics datasets. 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 [55]. ROOT maintains an object-oriented

structure, meaning it is organized around the data rather than the functions and logic. Its features include visualization tools such as histogramming, and statistical tools. ROOT can be used in C++ and Python languages. ROOT's declarative analysis interface (RDataFrame [56]) is used extensively in this thesis, both in Python and in C++.

3.2.1 ROOT Compression Algorithms

ROOT offers four different compression algorithms: `ZLIB`, `LZMA`, `LZ4`, and `ZSTD` [57]. 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 can reduce file sizes with larger compression factors, but are irreversible processes, meaning that information is lost during compression. 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 offers smaller compression factors and thus larger file sizes. `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 [58]. The default compression algorithm used to generate RNTuple samples is `ZSTD`. In Chapter 4, a performance study using RNTuples produced with `LZ4` is shown for comparison.

3.2.2 TTree Data Structure

ROOT provides a columnar-based data structure called TTree to efficiently store data. The columnar-based format allows users to access independent columns of data, such as event IDs or particle kinematics, versus accessing information per event. Internally, the physical data of each column are stored as Binary Large Objects (BLOBs), which are compressed blocks of serialized values. These BLOBs are stored in containers called TBaskets, each holding the data for a consecutive range of entries. Each column of data has a corresponding TBranch object that stores metadata. The metadata describes the column, such as the column's data type and serialization rules, and if the columns has subcomponents, which are represented by associated TLeaf objects. For I/O (Input/Output) efficiency, TTrees group consecutive entries into units called clusters.

The TBranch also stores metadata containing the list of file offsets and sizes of all of its baskets. ROOT uses these offsets and sizes to implicitly address the TBaskets and ultimately, decompress it and return the corresponding column data to the user. Figure 3.3 shows a more detailed flowchart of the TTree data structure.

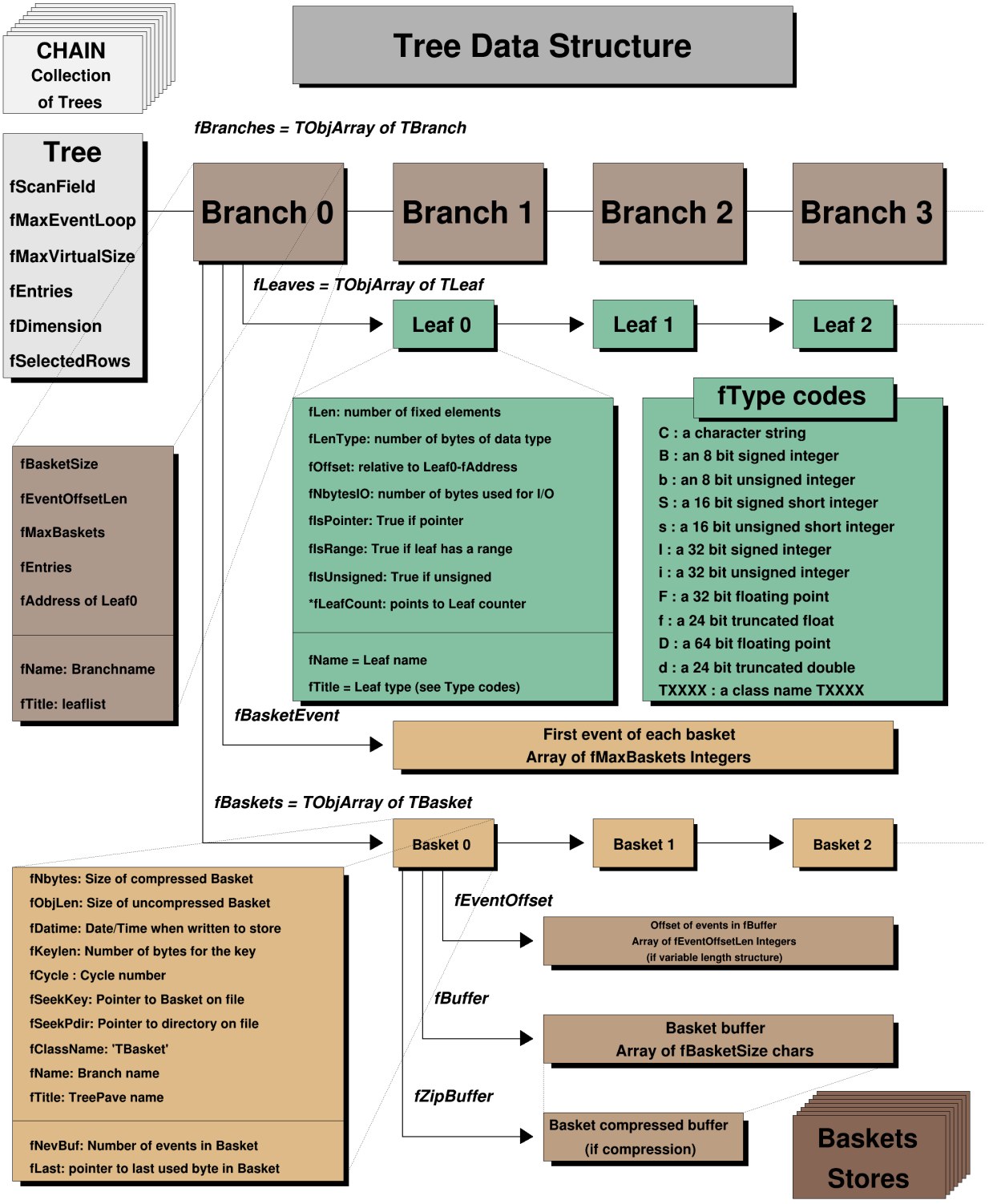


Figure 3.3: Representation of the TTree Data Structure [59].

3.2.3 RNTuple Data Structure

RNTuple is the new columnar data format that will be implemented at the start of the HL-LHC. Its 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 [60]. Its 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 separates logical schema from physical layout explicitly. Unlike TTree, RNTuple has an *anchor*, which is a top-level metadata object that describes the schema, columns, and how the physical data is stored. Each column has a corresponding *field*, which is an RNTuple object that maps the column to its physical storage location. Each column's data is stored in fixed-size units called *pages*. Pages are the containers that hold the actual serialized BLOBs. Pages are grouped into larger units called *clusters*, which define the granularity at which metadata and I/O decisions are made. Each cluster contains an *envelope*, which are metadata blocks describing the physical locations of all the pages in that cluster. Inside envelopes are *RBLOB* keys, which are the lowest-level object that hold the physical offset and size of a single page or multiple pages of the same cluster. Using RBLOB keys, RNTuple can explicitly locate, read, and decompress pages back to the user. A flow chart of this process in comparison to TTree is shown in Figure 3.4

Overall, this structure allows for more efficient random-access of individual events in comparison to TTree. For TTree, metadata is spread-out and stored in each individual TBranch;

therefore, its access pattern is more tightly coupled to sequential iteration. For RNTuple, the separation of the logical schema and physical layout allows for explicit addressing of the data and improves data compactness.

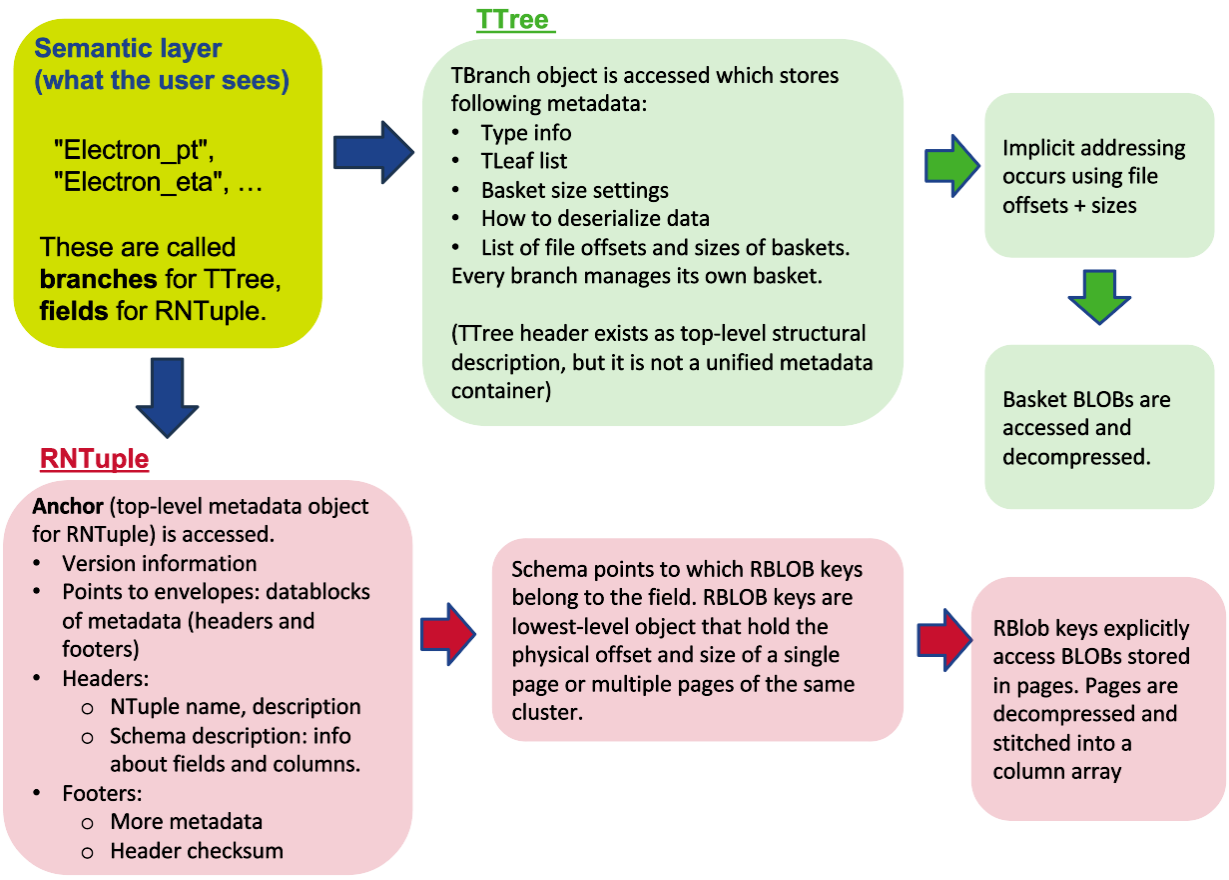


Figure 3.4: TTree versus RNTuple mapping systems.

3.3 TTree vs. RNTuple API

The first portion of this thesis aimed to test RNTuple API against TTree using ATLAS analysis workflows. Efforts were made to document the capabilities, usage and best practices of RNTuple during its development. The corresponding code repository, including the aforementioned documentation can be found in [61]. The sections below will provide examples of RNTuple's API in comparison to TTree's, using native C++ event loops, RDataFrame, and Uproot [62].

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 3.5a, users must iterate through TTree in order to load branches and define an empty pointer object to store their entries.

The RNTuple interface uses smart pointers, which simulate a pointer while providing automatic memory management [63]. 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 3.5b. The function `GetView` also simultaneously loads and stores a field.

3.3.2 RDataFrame in C++ and Python

RDataFrame is a high-level interface for analysis of data stored in TTree, RNTuple, CSV, and other data formats. RDataFrame can be used via C++ or Python languages. Analysis done with RDataFrame will mostly remain unmodified with RNTuple, as shown in Figure 3.6, 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 colon, instead of a period. This slight change confuses the filtering function in RDataFrame, but can be bypassed by assigning an alias name. Figure 3.7 provides an example of how to read multiple inputs and apply a filter using RDataFrame in C++.

3.3.3 Uproot

Uproot is a Python library designed to extract and manage data from ROOT files. Analysis done with Uproot will also remain unmodified with RNTuple. Users can load RNTuples and convert them into dictionaries of arrays just as with TTrees. Also, Uproot's added function, `uproot.mkrntuple`, allows users to write RNTuple outputs. Loading multiple RNTuple inputs using Uproot's functions `uproot.concatenate` and `uproot.iterate` also works with RNTuple.

```cpp
// 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);
    }
}
```

(a) Native C++ event loop using TTree.

```cpp
// 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);
    }
}
```

(b) Native C++ event loop using RNTuple.

Figure 3.5: These scripts load a DAOD_PHYSLITE file containing a TTree in (a) and an RNTuple in (b) to plot the distribution of electron transverse momenta. Transverse momentum is the momentum perpendicular of the colliding beams.

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

(a) Reading multiple TTree inputs.

```python
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 3.6: Examples of how to load multiple inputs into an RDataFrame in Python.

```python
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.

```python
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 3.7: 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 using the top simulation files form ATLAS Open Data. First, 92 TTrees stored in `DAOD_PHYSLITE` files were converted to RNTuples using its default compression algorithm setting, `ZSTD`. Conversion was performed using a dedicated Athena-based procedure typically used to convert one type of ATLAS sample into another. In sum, the 92 TTrees and RNTuples each contained 9,045,000 events; however, an average disk size reduction of about 47% was observed in the converted RNTuples compared to the original TTrees, as shown in Figure 4.1. This set of data, which collectively occupied about 110.8 GB of disk space when stored in TTrees, now occupied about 52.5 GB when stored in RNTuples. This reduction depends on the type and size of the converted sample, but it's relatively consistent. RNTuple encodes each column's values into uniform typed streams, which are then compressed independently into a page. For TTree, baskets mix heterogeneous data types, making compression less efficient. The disk size reduction when converting TTrees to RNTuple can be explained primarily due to this split-encoding process, along with clearer separation between logical schema and physical I/O with RNTuple.

Speed tests were performed for loading and outputting RNTuples in comparison to TTrees using `std::chrono::high_resolution_clock::now()`. This method is part of the standard C++ date and time library. 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.

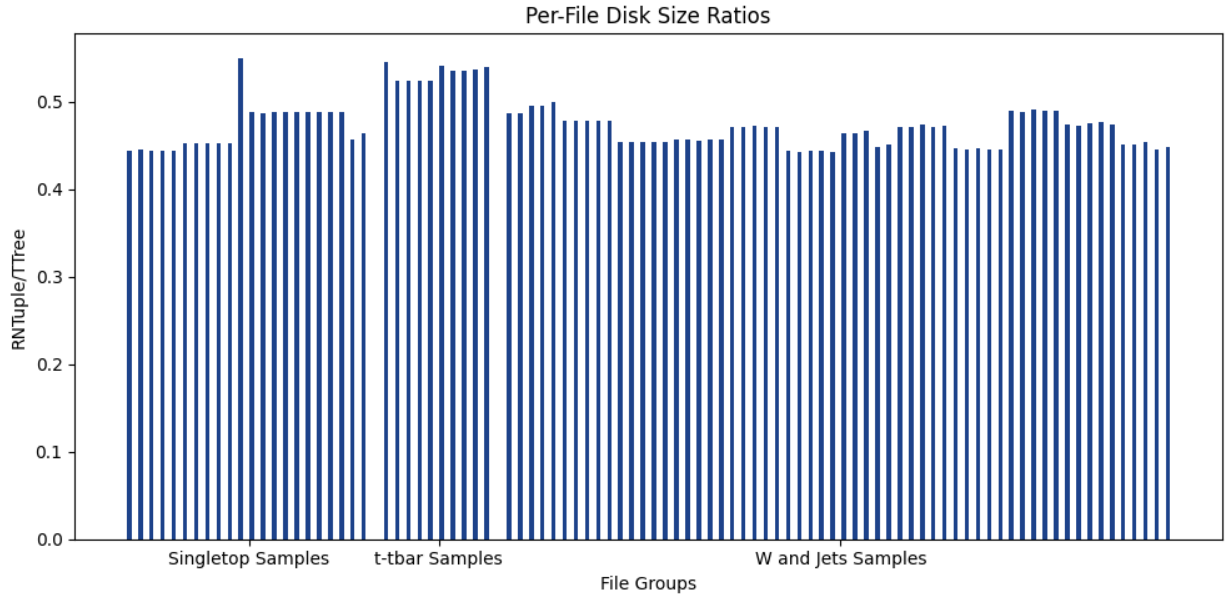


Figure 4.1: The RNTuple:TTree file size ratios for all samples.

4.1 Readability Speed

The total loading times for 92 RNTuples and their TTree equivalents 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.

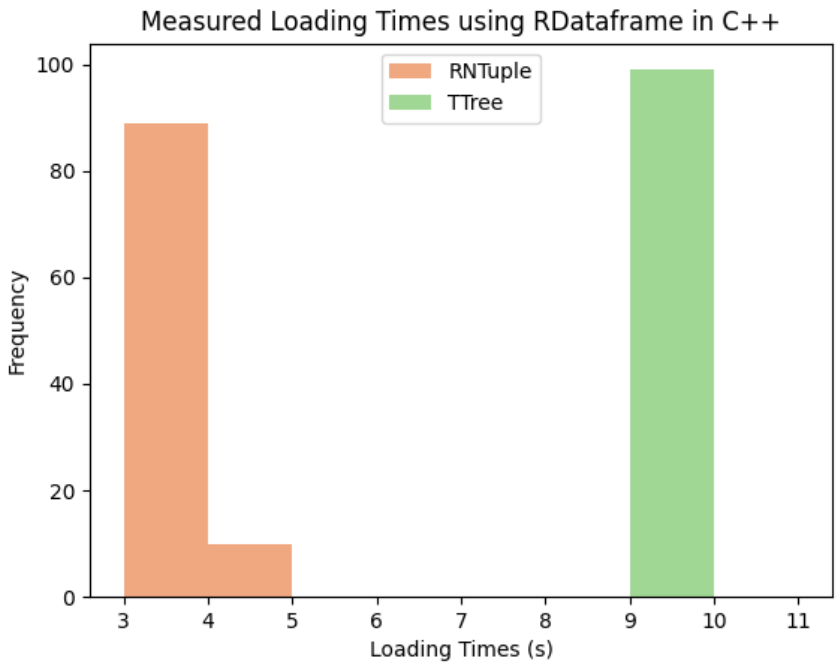


Figure 4.2: Total loading times measured for TTree and RNTuple using RDataFrame in C++.

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.

```
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.

```
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 RNTuple's classes. 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.

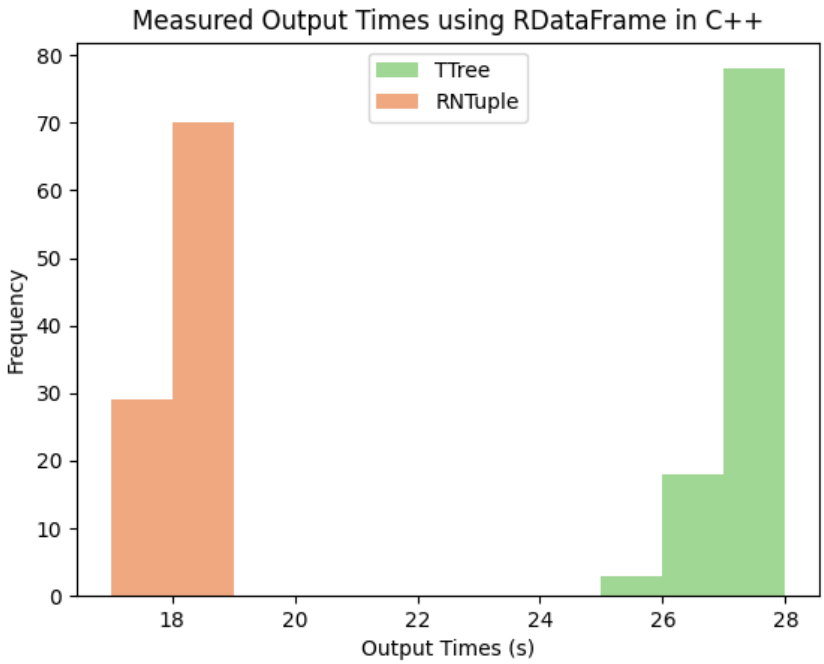


Figure 4.4: Total writing times measured for TTree and RNTuple using RDataFrame in C++.

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. Both outputs were produced using `ZSTD`. 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. In other words, RNTuple has more efficient compression handling. TTree can probably achieve the same level of compression, but would require the user to explicitly optimize TBasket compression settings. 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.

Data Format	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.

Data Format	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.

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 [57]. 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`.

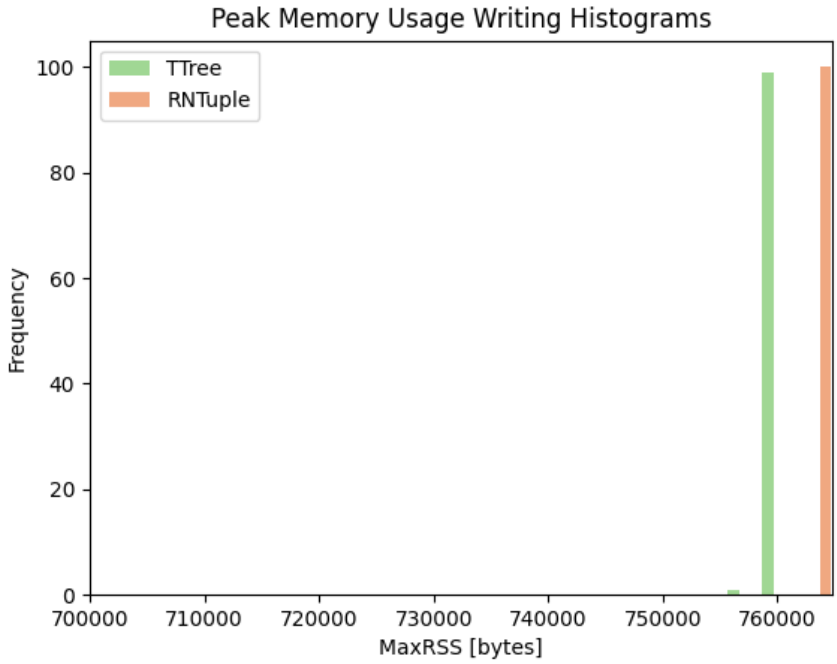


Figure 4.5: Peak memory usage while producing an output with two columns. Measurements were taken 100 times for each version.

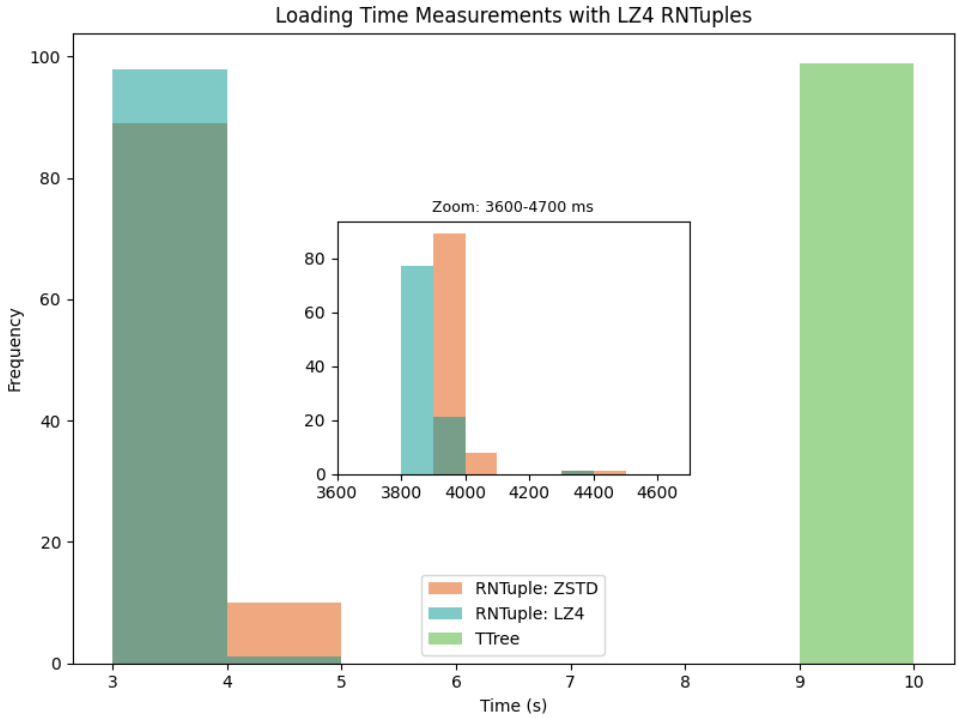


Figure 4.6: Loading time measurements for RNTuples produced by the `LZ4` and `ZSTD` algorithms, and for TTree. The RNTuples composed with `ZSTD` and `LZ4` only differ by a couple of milliseconds.

4.6 Performance Discussion

Converting TTrees to RNTuples showed immediate file size reductions. The file sizes on disk were reduced by about 47%, which is incredibly advantageous in preparation to the HL-LHC. This reduction is due to RNTuple's more efficient compression handling via split encoding. Additionally, clearer separation between metadata from physical payload in RNTuple's structure avoids repetition of structural information in the data; hence reducing overall disk usage.

Common analysis steps used in RDataFrame workflows improved with RNTuple compared to its TTree predecessor. Reading speed is 2.38 times faster than TTree, and writing speed is 1.51 times faster, without any significant cost to memory usage. This improvement

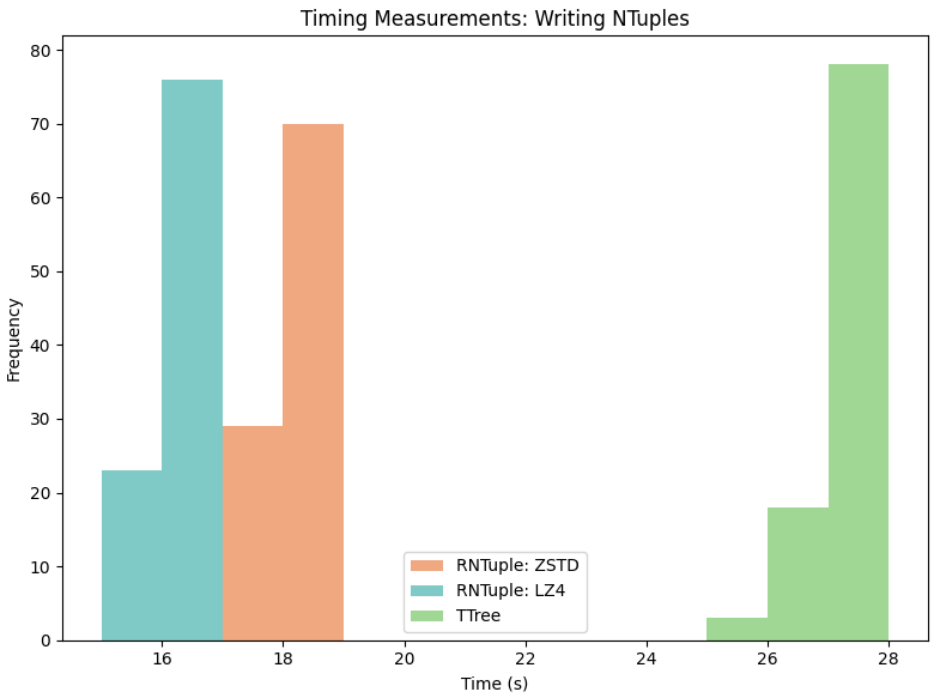


Figure 4.7: Writing time measurements for RNTuples produced by the `LZ4` and `ZSTD` algorithms, and TTree.

may be due to a combination of: finer grained page layout, simpler and safer access patterns, and better cache locality. As mentioned in Chapter 3, RNTuple stores and compresses physical data in pages, whereas TTree, stores physical data in TBaskets. Due to split-encoding, the data in pages are better-organized than data stored in TBaskets, allowing the I/O system to load the minimal data needed back to the user. Plus, the explicit mapping of data using RBLOB keys in RNTuple could also be contributing to the speed increase. In addition to speed improvements, the RNTuple API reduces lines of code, making it more user-friendly than the TTree API. Finally, the writing scripts revealed that RNTuple compresses repeated bits more efficiently than TTree. Instead of filtering out empty events, allowing padding with empty events may be a reasonable new approach for future analysis.

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`

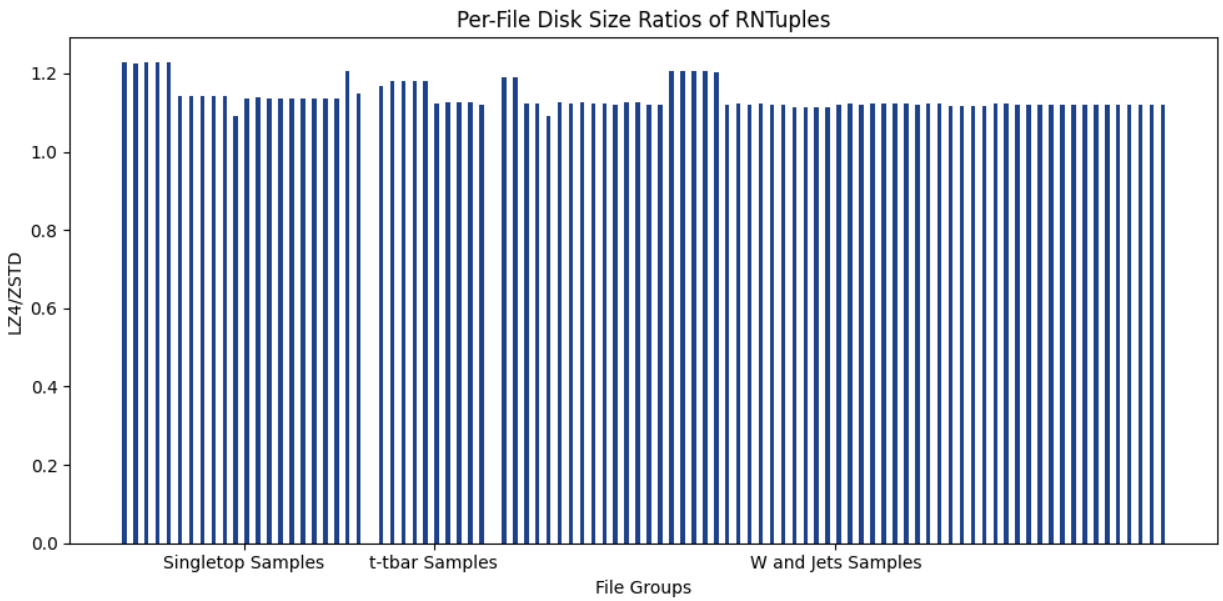


Figure 4.8: Per-file compression ratios of `LZ4:ZSTD` over total number of events.

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.

CHAPTER 5

ANALYSIS GRAND CHALLENGE: RNTUPLE VS. TTREE

The Analysis Grand Challenge (AGC) is a project organized by IRIS-HEP [64] designed to showcase an end-to-end analysis through a $t\bar{t}$ study. The AGC has several versions that demonstrate different cyber infrastructure and workflows, making it a great benchmark to test RNTuple. For the final part of this study, two new versions of the AGC were developed using ATLAS Open Data and RDataFrame—one with TTree inputs and another with RNTuple inputs. These versions were heavily influenced on the existing RDataFrame AGC repository that applies CMS open data [65] and the Uproot AGC repository that uses ATLAS Open Data [66]. The newly developed AGC versions that use ATLAS Open Data and RDataframe, using TTree and RNTuple inputs can be found in [67].

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 script 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 inputs are the same 92 ROOT files 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. These files were called from a local directory and stored within a large dictionary, with their corresponding metadata that include information, such as variation label. Samples with variation labeled "nominal" are the primary datasets representing the most accurate SM predictions, while those labeled "systematic" represent alternative simulated datasets

in which specific parameters have been intentionally varied from their nominal values. For this study, pseudodata was used instead of measured data to calculate systematic uncertainties. The pseudodata histogram is constructed by merging the histograms from all nominal variation samples, and stochastic noise is then introduced to the count in each bin with a Gaussian distribution. The specifics of this process is defined in a JSON file and processed using the Python library, Cabinetry [68] in the statistical script. The statistical script uses histograms produced by the analysis script to perform a simplified fit of the MC samples to the pseudodata. It remained unchanged relative to previous AGC versions and will not be discussed further in this thesis.

5.1.1 Event Selections

The following event selections were applied in the analysis script of the AGC. 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. Events with at least four jets, two of the four being b-tagged are selected. Selected jets must have transverse momentum, p_t greater than 30 GeV and $|\eta|$ less than 2.4. Leptons must have p_t greater than 25 GeV and $|\eta|$ less than 2.1. The top mass observable is then reconstructed by taking the invariant mass of the trijet with the largest transverse momentum. 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.

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.

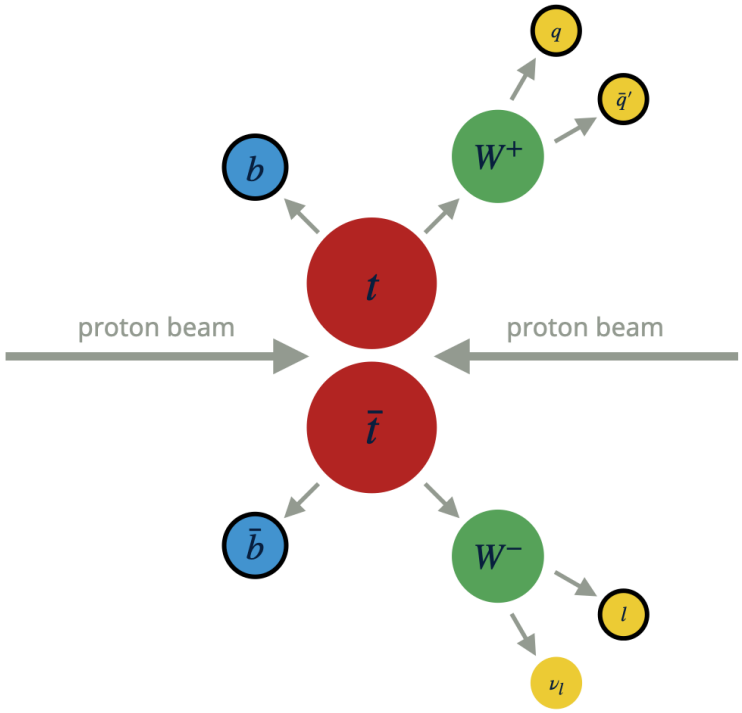


Figure 5.1: The schematic view of a top and anti-top quark collision [69]

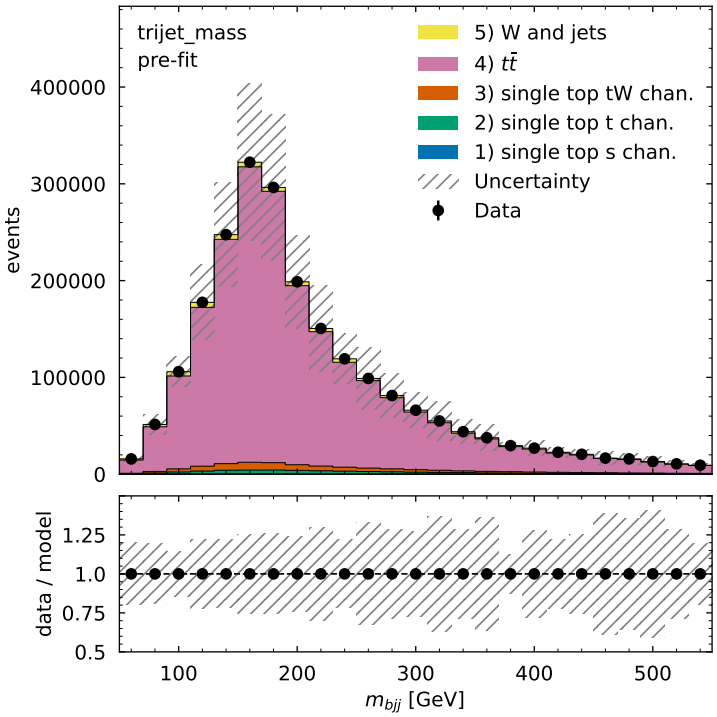


Figure 5.2: The trijet mass prefit. The uncertainty band corresponds to the statistical error of the MC sample, corresponding to a total of 9,045,000 events. This result is the same for both RNTuple and TTree versions of the AGC.

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 run the analysis script that produces the top quark mass and H_T histograms, while TTree averaged 71.75 seconds. RNTuple was approximately 1.51 times faster, consistent with 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

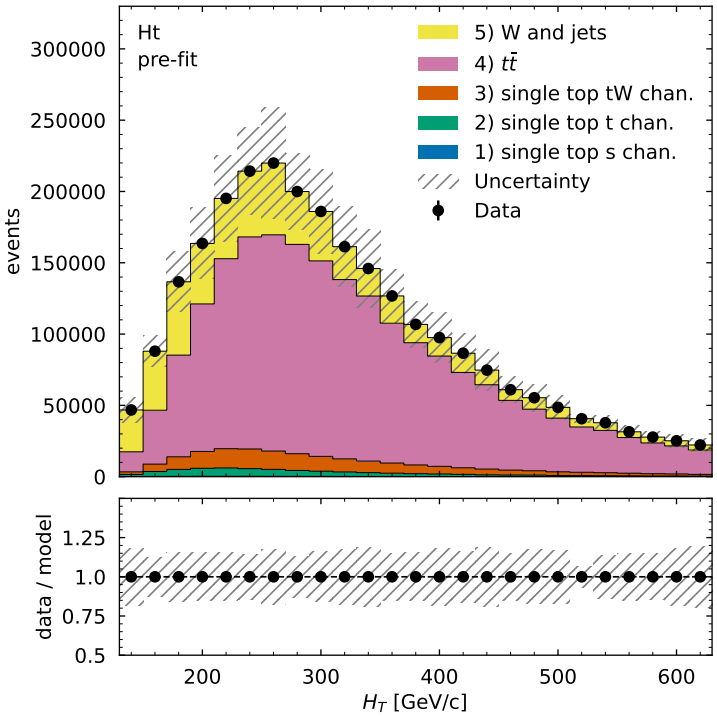


Figure 5.3: The H_T observable prefit. The uncertainty band corresponds to the statistical error of the MC sample, corresponding to a total of 9,045,000 events. This result is the same for both RNTuple and TTree versions of the AGC.

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. The statistical script uses histograms produced by the analysis script to perform a simplified fit of the MC samples to the pseudodata. It remained unchanged relative to previous AGC versions and will not be discussed further in this thesis.

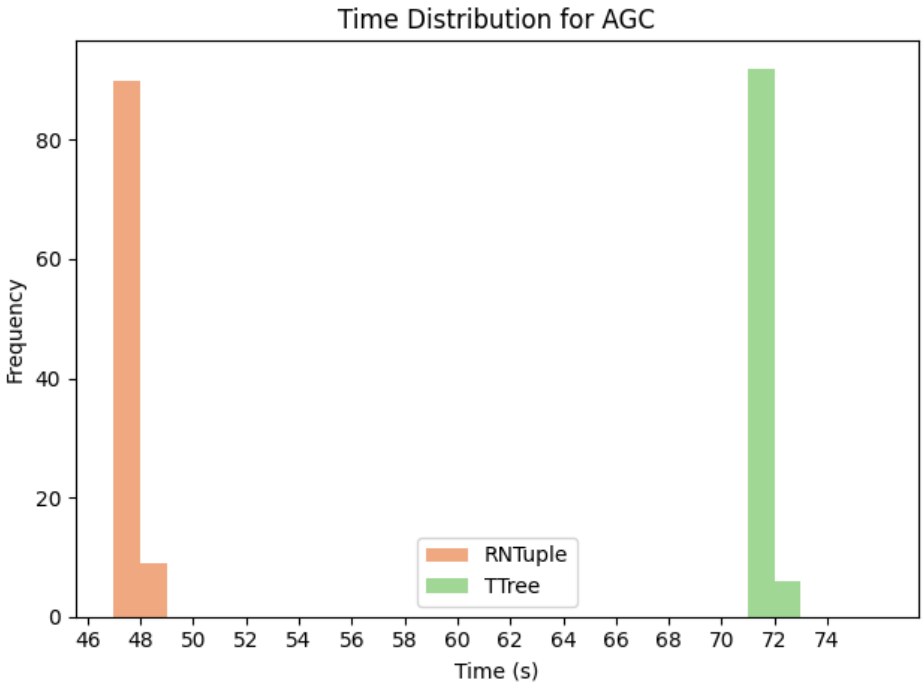


Figure 5.4: The total execution times of the AGC measured 100 times for TTree and RNTuple versions.

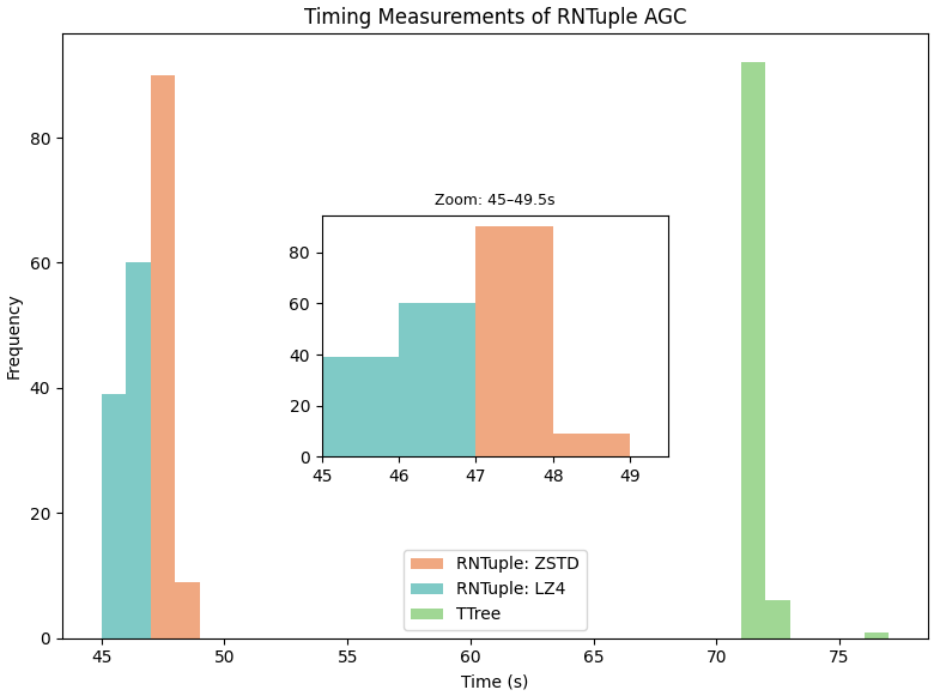


Figure 5.5: The total execution times of the AGC measured 100 times with RNTuples produced with the `LZ4` compression algorithm.

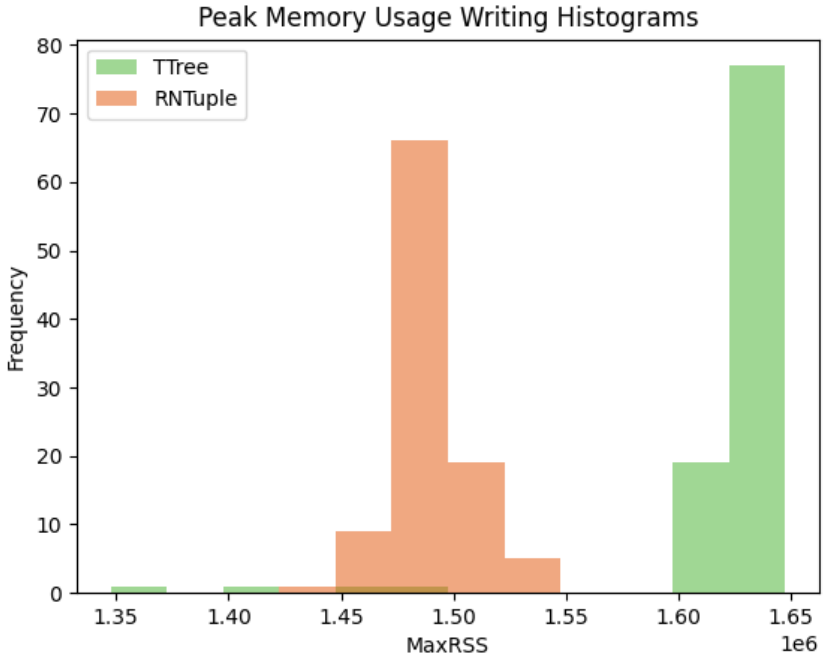


Figure 5.6: The peak memory usage when executing the AGC.

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. Through writing these outputs, it was revealed that RNTuple compresses repeated bits more efficiently than TTree. This opens up a potential new analysis approach that allows padding outputs with empty events. 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 implementations of the AGC using ATLAS Open Data and RDataFrame 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 92 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 reaffirms it as a promising data storage format for ATLAS analysis using RDataFrame workflows. Its minimal code modifications also promise a smooth transition toward 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. DOI: 10.1201/9781420083002 (cit. on p. 1).

[2] B. Odom et al. "New Measurement of the Electron Magnetic Moment Using a One-Electron Quantum Cyclotron". In: *Phys. Rev. Lett.* 97 (3 July 2006), p. 030801. DOI: 10.1103/PhysRevLett.97.030801. URL: https://link.aps.org/doi/10.1103/PhysRevLett.97.030801 (cit. on p. 1).

[3] Peter 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. URL: https://link.aps.org/doi/10.1103/PhysRevLett.13.508 (cit. on p. 1).

[4] ATLAS Collaboration. "The ATLAS Experiment at the CERN Large Hadron Collider". In: *JINST* 3 (2008), S08003. DOI: 10.1088/1748-0221/3/08/S08003 (cit. on p. 1).

[5] Lyndon Evans and Philip Bryant. "LHC Machine". In: *JINST* 3 (2008), S08001. DOI: 10.1088/1748-0221/3/08/S08001 (cit. on p. 1).

[6] 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).

[7] 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, 17).

[8] 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).

[9] 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).

[10] 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).

[11] CERN ROOT Team. *Trees — ROOT Manual*. 2025. URL: https://root.cern.ch/manual/trees/ (visited on 11/01/2025–11/30/2025) (cit. on p. 2).

[12] The HDF Group. *HDF5*. [software]. Version 1.10.0 Patch 1. 2017. URL: https://support.hdfgroup.org/ftp/HDF5/releases/hdf5-1.10/hdf5-1.10.0-patch1/ (cit. on p. 2).

[13] The Apache Parquet Project. *parquet-cpp*. [software]. Version 1.2.0. 2017. URL: https://github.com/apache/parquet-cpp/tree/apache-parquet-cpp-1.2.0 (cit. on p. 2).

[14] 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).

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

[16] S. L. Glashow. “Partial Symmetries of Weak Interactions”. In: *Nucl. Phys.* 22 (1961), pp. 579–588. DOI: 10.1016/0029-5582(61)90469-2 (cit. on p. 3).

[17] Steven Weinberg. “A Model of Leptons”. In: *Phys. Rev. Lett.* 19 (1967), pp. 1264–1266. DOI: 10.1103/PhysRevLett.19.1264 (cit. on p. 3).

[18] Abdus Salam. “Weak and electromagnetic interactions”. In: *Selected Papers of Abdus Salam*, pp. 244–254. DOI: 10.1142/9789812795915_0034. eprint: https://www.worldscientific.com/doi/pdf/10.1142/9789812795915_0034. URL: https://www.worldscientific.com/doi/abs/10.1142/9789812795915_0034 (cit. on p. 3).

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

[20] 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 pp. 4, 9).

[21] 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 pp. 4, 9).

[22] 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).

[23] 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).

[24] *Standard Model Summary Plots June 2024*. Tech. rep. All figures including auxiliary figures are available at https://atlas.web.cern.ch/Atlas/GROUPS/PHYSICS/PUBNOTES/ATL-PHYS-PUB-2024-011. Geneva: CERN, 2024. URL: https://cds.cern.ch/record/2903866 (cit. on p. 6).

[25] ATLAS Collaboration. *ATLAS DAOD_PHYSLITE format MC simulation electroweak boson nominal samples*. Accessed: 2025-12-10. CERN Open Data Portal, 2024. DOI: 10.7483/OPENDATA.ATLAS.K5SU.X65Y. URL: https://opendata.cern.ch/record/80010 (cit. on p. 7).

[26] 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. 8).

[27] CERN. *hadron — Tag page*. https://home.cern/tags/hadron. Accessed: 2025-11-07. CERN, 2025 (cit. on p. 8).

[28] ATLAS Collaboration. "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).

[29] CERN. *CERN Timeline — "143"*. https://timeline.web.cern.ch/timeline-header/143. Accessed: 2025-11-12. 2025 (cit. on p. 9).

[30] Manuela Cirilli, Michael Barnett, and ATLAS Collaboration. *First collisions in ATLAS*. ATLAS public news, CERN. Accessed: 2025-12-03. 2009. URL: https://atlas-public.web.cern.ch/updates/news/first-collisions-atlas (cit. on p. 9).

[31] ATLAS Collaboration. "Measurement of the top quark-pair production cross section with ATLAS in pp collisions at $\sqrt{s} = 7$ TeV". In: *The European Physical Journal C* 71.3 (Mar. 2011). ISSN: 1434-6052. DOI: 10.1140/epjc/s10052-011-1577-6. URL: http://dx.doi.org/10.1140/epjc/s10052-011-1577-6 (cit. on p. 9).

[32] 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).

[33] The LHCb Collaboration. "The LHCb Detector at the LHC". In: *Journal of Instrumentation* 3.08 (Aug. 2008), S08005. DOI: 10.1088/1748-0221/3/08/S08005. URL: https://doi.org/10.1088/1748-0221/3/08/S08005 (cit. on p. 9).

[34] ALICE Collaboration. "The ALICE experiment at the CERN LHC". In: *Journal of Instrumentation* 3 (2008), S08002. DOI: 10.1088/1748-0221/3/08/S08002 (cit. on p. 9).

[35] ATLAS Collaboration. "Luminosity determination in *pp* collisions at $\sqrt{s} = 13$ TeV using the ATLAS detector at the LHC". In: *Eur. Phys. J. C* 83.10 (2023), p. 982. DOI: 10.1140/epjc/s10052-023-11747-w. arXiv: 2212.09379 [hep-ex] (cit. on p. 10).

[36] 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. 10).

[37] 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. 10).

[38] 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. 10).

[39] 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. 10).

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

[41] 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, 13, 15, 16).

[42] ATLAS Collaboration. *ATLAS Inner Tracker Strip Detector: Technical Design Report*. ATLAS-TDR-025; CERN-LHCC-2017-005. 2017. URL: https://cds.cern.ch/record/2257755 (cit. on p. 17).

[43] ATLAS Collaboration. *ATLAS Inner Tracker Pixel Detector: Technical Design Report*. ATLAS-TDR-030; CERN-LHCC-2017-021. 2017. URL: https://cds.cern.ch/record/2285585 (cit. on p. 17).

[44] ATLAS Collaboration. *A High-Granularity Timing Detector for the ATLAS Phase-II Upgrade: Technical Design Report*. ATLAS-TDR-031; CERN-LHCC-2020-007. 2020. URL: https://cds.cern.ch/record/2719855 (cit. on p. 17).

[45] ATLAS Collaboration. *ATLAS Muon Spectrometer Phase-II Upgrade: Technical Design Report*. ATLAS-TDR-026; CERN-LHCC-2017-017. 2017. URL: https://cds.cern.ch/record/2285580 (cit. on p. 17).

[46] ATLAS Collaboration. *ATLAS LAr Calorimeter Phase-II Upgrade: Technical Design Report*. ATLAS-TDR-027; CERN-LHCC-2017-018. 2017. URL: https://cds.cern.ch/record/2285582 (cit. on p. 17).

[47] ATLAS Collaboration. *ATLAS Tile Calorimeter Phase-II Upgrade: Technical Design Report*. ATLAS-TDR-028; CERN-LHCC-2017-019. 2017. URL: https://cds.cern.ch/record/2285583 (cit. on p. 17).

[48] ATLAS Collaboration. *ATLAS TDAQ Phase-II Upgrade: Technical Design Report*. ATLAS-TDR-029; CERN-LHCC-2017-020. 2017. URL: https://cds.cern.ch/record/2285584 (cit. on p. 17).

[49] ATLAS Collaboration. *Athena — the ATLAS experiment software framework*. Version 21.2.129. 2019. DOI: 10.5281/zenodo.3932810. URL: https://doi.org/10.5281/zenodo.3932810 (cit. on p. 18).

[50] 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).

[51] 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).

[52] 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).

[53] 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. 20).

[54] 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. 20).

[55] René Brun et al. "PAW++ : Physics Analysis Workstation : User's Guide". In: (1999). CERN Program Library Long Writeups. URL: https://cds.cern.ch/record/2296392 (cit. on p. 20).

[56] CERN ROOT Team. *ROOT::RDataFrame Class Reference*. Accessed: 2025-12-11. CERN / ROOT Project. 2025. URL: https://root.cern/doc/master/classROOT_1_1RDataFrame.html (cit. on p. 21).

[57] 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, 36).

[58] 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).

[59] 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).

[60] 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. 24).

[61] Fatima Rodriguez. *RNTuple-for-analysis-workflows*. GitLab repository, CERN: https://gitlab.cern.ch/faarodri/rntuple-for-analysis-workflows. Accessed: 2025-11-11. 2025 (cit. on p. 26).

[62] scikit-hep / Uproot Developers. *uproot: ROOT I/O in pure Python and NumPy*. Version 5.6.8. Accessed: 2025-11-09. 2025. URL: https://pypi.org/project/uproot/ (cit. on p. 26).

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

[64] Institute for Research and Innovation in Software for High Energy Physics (IRIS-HEP). *IRIS-HEP — Institute for Research and Innovation in Software for High Energy Physics*. https://iris-hep.org/. Accessed: 2025-12-11. 2025 (cit. on p. 41).

[65] ROOT Project. *analysis-grand-challenge: Example workflows for the Analysis Grand Challenge*. https://github.com/root-project/analysis-grand-challenge. Accessed: 2025-12-11. 2025 (cit. on p. 41).

[66] IRIS-HEP. *AGC PhysLite: Main Notebook*. https://github.com/iris-hep/agc-physlite/blob/main/main_code.ipynb. Accessed: 2025-12-12. 2025 (cit. on p. 41).

[67] Fatima 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. 41).

[68] cabinetry developers. *cabinetry Documentation*. https://cabinetry.readthedocs.io/en/latest/. Accessed: 2025-12-11. 2025 (cit. on p. 42).

[69] 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. 43).