Explorative Data Layout Analysis of Next-Generation High Energy Physics Datasets

Summer Student Project Report 2024

Ida Caspary[1]

Supervisor: Florine de Geus[2]

[1]University of Stirling/Imperial College London
[2]CERN/University of Twente

Overview

1.1 Introduction

A 27km-long tunnel underground, the Route Einstein to cycle, posters of worldview-changing discoveries around, CERN can be surreal for Bachelor's students. Participating in the CERN summer students programme means to live that curious world. Summer student projects allow a dive into a piece of the complex research web surrounding CERN High Energy Physics. This project - based on the exabytes of data generated for experiments and saved in the ROOT data format - focused on explorative layout analysis for the next-generation of data storage. This report tells what we tackled, the development process, as well as challenges, outcomes, and a view ahead.

The project's repository is at: github.com/IdaCy/RNTupleTTreeChecker.

1.2 Project Motivation

The extent of data generated by Large Hardon Collider experiments requires efficient data storage and analysis. The *TTree* data format was designed to enable this. However, TTree's design and implementation limits further optimisation using modern practices. It will be succeeded by the new data format of *RNTuple*. The transition will be a large change for many ROOT users, and changing the toolset may meet hesitance. Therefore, a tool to aid this change would support the understanding of parallels between the old and new data format and highlight potential troubles in conversion.

Objectives

2.1 ROOT Framework

The required environment for data analysis is given by the object-oriented framework of ROOT. It includes a large range of utilities for data storage, visualisation, and statistical analysis. The primary data structures - TTrees - store data in a hierarchical format that allows for efficient reading and writing.

2.2 The RNTuple Transition

TTree served the ROOT community well for decades. However, it was designed for the hardware and software of the late 1990s/early 2000s, and with the growth in data volumes and changes hardware and programming models, limitations arose. RNTuple is designed to overcome these limitations. It aims to improve input/output performance and reduce overall disk space requirements.

2.3 Development Goal

Many ROOT user may have expertise for specific behaviours of ROOT. There may be preferences for keeping the known, staying productive without the risk of unforeseen issues that arise when adopting a new technology.

A tool that allows direct comparisons between the well-known TTree format and the new RNTuple format could give the needed confidence for users to switch. It should inform them of structure consistency and identical data found in checked files. Any discrepancies that may have arisen on data format conversion should be highlighted. It should also prove the preservation of data accuracy in the new format.

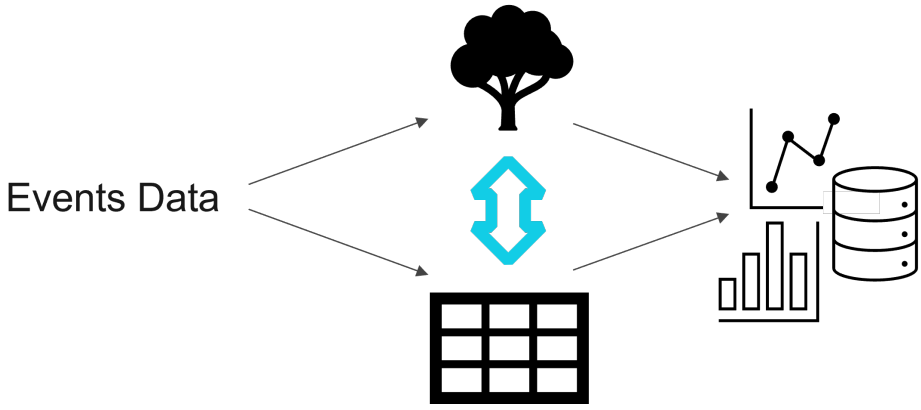

Figure 1: The Checker Tool Between TTree & RNtuple

2.4 Implementation

Tooling Scope

Given a TTree and an RNTuple, the new tool should compare the data formats - which gives it its name *RNTupleTTreeChecker*. For any two such data structures, it should perform specific checks to find similarities and differences.

The checks include:

- Structure: Entry Count, Field Count
- Schema: Field Names, Field Types
- Data: Consistency

Separating those checks allows for specific uses in various cases as required.

Structure

The tool is made up of two parts: The core file checking tool - *Checker* - and the Command Line Interface tool - *CheckerCLI*.

This allows for two different interfaces. The Checker is purely a tool made up of various functions that can draw values from TTrees and/or RNTuples. The CheckerCLI contains a complete set of checks accessible via the command line that call Checker functions and use returned values to create a comprehensive output stating results.

Functionality

Running the *RNTupleTTreeChecker*'s CheckerCLI triggers a series of tasks to receive the input, check validity, and use the Checker to compare the content of a TTree and an RNTuple. It may provide detailed outputs to the user. Figure 2 shows the workflow.

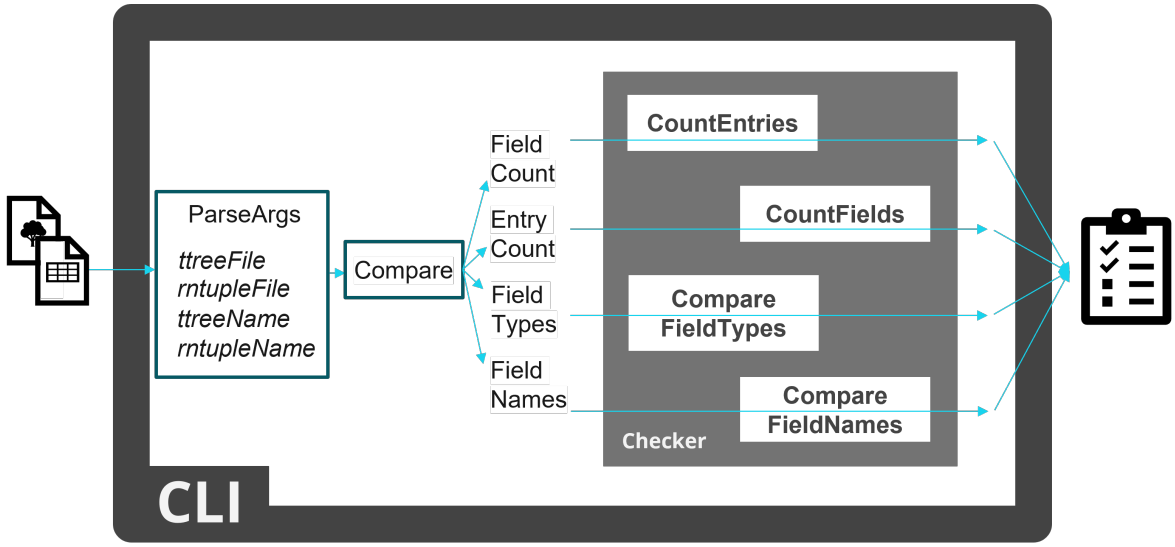

Figure 2: CheckerCLI Workflow: From User Command to Result Output

1. **Input:** A user invokes the *CheckerCLI* in the command line interface. It includes:
   - **-t** The TTree file
   - **-r** The RNTuple file
   - **-tn** The TTree name
   - **-rn** The RNTuple name

   For example:

   ```
   ./Checker -t ttreefile.root -r ../rootfile.root -tn tree_0
   -rn rntuple_0
   ```

   The TTree and RNTuple names are required as files may contain several TTrees or RNTuples.

2. **Argument Parsing:** The *CheckerCLI* first invokes the *ParseArgs* function to extract the necessary information from the command line input. With this, paths to the files and the names of the TTree and RNTuple can be passed to the corresponding checks.

3. **Execution of Checks:** Based on the extracted information, the *CheckerCLI* triggers specific checks such as *CountEntries* and *CompareFieldNames*. These checks comparing the structure, schema, and data consistency between the TTree and RNTuple.

4. **Internal Operations:** Each check uses the *Checker* tool, which draws the actual values from the TTree and RNTuple files.

   For instance, the *Checker* might retrieve the entry counts from both the TTree and RNTuple, or it might compare the field names and types to ensure they match across the two formats.

5. **Output:** Receiving the value results, the *CheckerCLI* compiles the results into consumable results for the user.

   Each result specifies whether the check was successful or failed, along with detailed information about the comparison. Figure 3 and 4 show matches and mismatches.

```
idacy@IdaLab:~/development/RNTupleTTreeChecker/build$ ./Checker -t
../files /t.root -r ../files/rn.root -tn tree_0 -rn rntuple_0 -v

*** Entry Count ***
Number of entries: 10
TTree and RNTuple have the same entry count: TRUE

*** Field Count ***
Number of fields:  4
TTree and RNTuple have the same field count: TRUE

*** Field Names ***
TTree Field         |  RNTuple Field
------------------------------------
value               |  value
weight              |  weight
energy              |  energy
isNew               |  isNew

The fields have the same names: TRUE

*** Field Types ***
Type - TTree        |  Type - RNTuple      Field
-----------------------------------------
int                 |  int                 value
float               |  float               weight
double              |  double              energy
bool                |  bool                isNew

The fields have the same types: TRUE
```

Figure 3: CheckerCLI Output with Matching Data Formats

```
idacy@IdaLab:~/development/RNTupleTTreeChecker/build$ ./Checker -t
../files /t.root -r ../files/r.root -tn tree_0 -rn rntuple_3 -v

*** Entry Count ***
Number of entries: 10
TTree and RNTuple have the same entry count: TRUE

*** Field Count ***
Number of fields:  4
TTree and RNTuple have the same field count: TRUE

*** Field Names ***
TTree Field         |  RNTuple Field
------------------------------------
value               |  value
weight              |  weight
energy              |  No match
isNew               |  isNew
No match            |  mass

The fields have the same names: FALSE

*** Field Types ***
Type - TTree        |  Type - RNTuple      Field
-----------------------------------------
int                 |  int                 value
float               |  float               weight
double              |  Missing             energy
bool                |  bool                isNew
Missing             |  double              mass

Field type comparison yields match failure due to unmatching fields.
```

Figure 4: CheckerCLI Output with Non-Matching Data Formats

2.5 Checker Function

Format Access

The Checker class uses ROOT's internal mechanisms to traverse the branches and fields of the data structures.

For TTrees, the tool functions:

- Traverse the list of branches.
- Identify branches containing vectors of the desired data type.
- Extract the data and store it in a combined vector.

For RNTuples, the tool functions:

- Retrieve the descriptor for the RNTuple.
- Match the field type with the desired data type using regular expressions.
- Extract and compile the data into a vector.

Data Comparison

Histograms are created for data comparisons. They display the distribution of data found for the different data types. Figure 5 gives an example output.

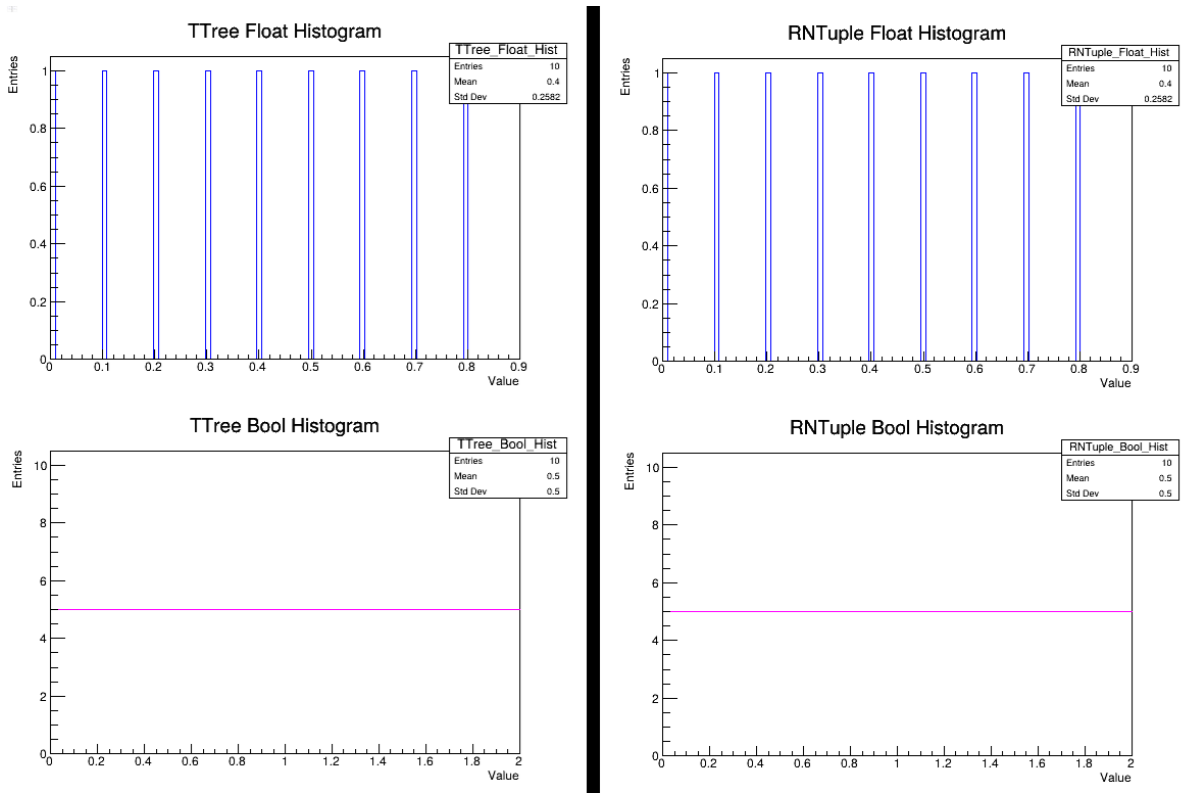

Figure 5: Example Histograms For Float And Bool

In addition, the following values are provided to the user:

- Field Count
- Mean
- Standard Deviation

Example outputs for printed histogram statistics with integer, float, double, and bool branches can be viewed in Figure 6.

```
*** Histograms ***
               | TTree Value    RNTuple Value
-------------------------------------------
Int Count      |  10             10
Int Mean       |  4.000000       4.000000
Int StdDev     |  2.581989       2.581989

Float Count    |  10             10
Float Mean     |  0.400000       0.400000
Float StdDev   |  0.258199       0.258199

Double Count   |  10             10
Double Mean    |  6.000000       6.000000
Double StdDev  |  3.872983       3.872983

Bool Count     |  10             10
Bool Mean      |  0.500000       0.500000
Bool StdDev    |  0.500000       0.500000

All histogram statistics match: TRUE
```

Figure 6: CheckerCLI Output Of Histogram Statistics

CheckerCLI Modes

In the CheckerCLI's default mode, only found discrepancies are printed. Figure 7 demonstrates the default output if no mismatch was found.

```
IdaLab:~/development/RNTupleTTreeChecker/build$ .
/Checker -t ../testfiles/mtt.root -r ../testfiles/mrn.root
 -tn tree_0 -rn rntuple_0

Check ran through successfully! No inconsistency found.

idacy@IdaLab:~/development/RNTupleTTreeChecker/build$
```

Figure 7: Successful Default Ouput

With the **-v** flag, the Checker can be run in the *verbose* mode. This mode prints the output comparisons of all the checks, as well as the histogram numbers.

Conclusion

3.1 Challenges

Having started from an intial class performing everything, the need for a separation of functionality between the *Checker* and the *CheckerCLI* arose. This needed significant work of taking the existing project apart, many design decisions, and various attempts with differently well working versions.

The *Checker* was designed as a reusable component, providing the core comparison functionality without being tied to a specific user interface. This choice ensures that the *Checker* can be directly embedded into other ROOT-based applications, offering flexibility for developers who need to perform automated checks on TTrees and RNTuples. On the other hand, the *CheckerCLI* was developed as a command-line interface for ease of use by end-users who might not need to dive into the code but require a quick, reliable comparison tool.

Additionally, feedback from immediate feedback provided new insights into potential extensions and improvements. Several of these ideas were implemented, such as enhancing the output to not only indicate discrepancies as *TRUE* or *FALSE* but also include a *NOT EXACTLY* state, represented by yellow, for cases where differences were detected but might be acceptable depending on the context. Other features like histogram comparisons and the verbosity flag (-v) for detailed output were also added. Comprehensive documentation was created.

3.2 Validation

The validation process was a critical aspect of ensuring that the *RNTupleTTreeChecker* performed as intended. The development phase included extensive testing with both macro-generated test TTrees and RNTuples to verify that the tool could handle different possible data structures.

Unittests were a key component of the project. These tests were focused on the *Checker* class, which is the core of the comparison functionality. By isolating this component, it was possible to comprehensively test each function until reliable.

Overall, the *RNTupleTTreeChecker* has been validated to be a generally complete and functional tool, with proven effectiveness in comparing TTrees and RNTuples. Its integration into ROOT or usage as a standalone tool can already support users in the transition from TTree to RNTuple, ensuring data integrity and consistency in the process.