Implementation of a pipeline to evaluate the performance of the GGTF algorithm for IDEA

Andrea De Vita, Dolores Garcia, Brieuc Francois

4th September 2024

Overview

The track finder described by Dolores (Slide FCC Full Sim Meeting - 7th August) has been tested on Python, but there is no pipeline that returns a root file that can be used to evaluate the performance:

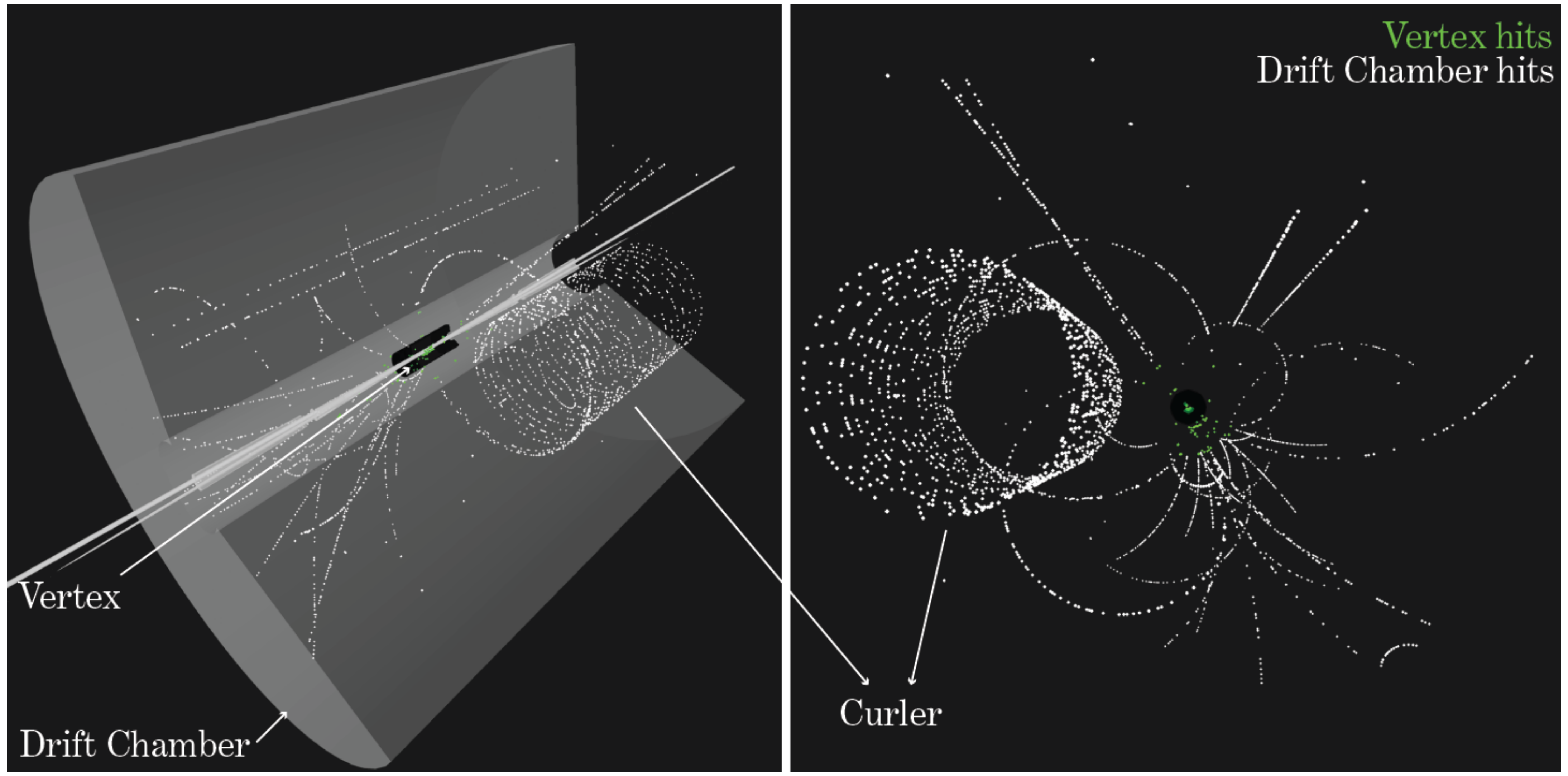
1. **Implementation of a Track Finding gaudi::functional** that returns a track collection given a collection of hits.

2. **Implementation of an evaluation step** that returns a quick estimate of tracking efficiency and a table of parameters to calculate tracking efficiency as a function of particle properties (such as p_T, θ etc...).

FUTURE CIRCULAR COLLIDER

Complete Pipeline

From to Simulation to Evaluation



The complete pipeline consists of several steps:

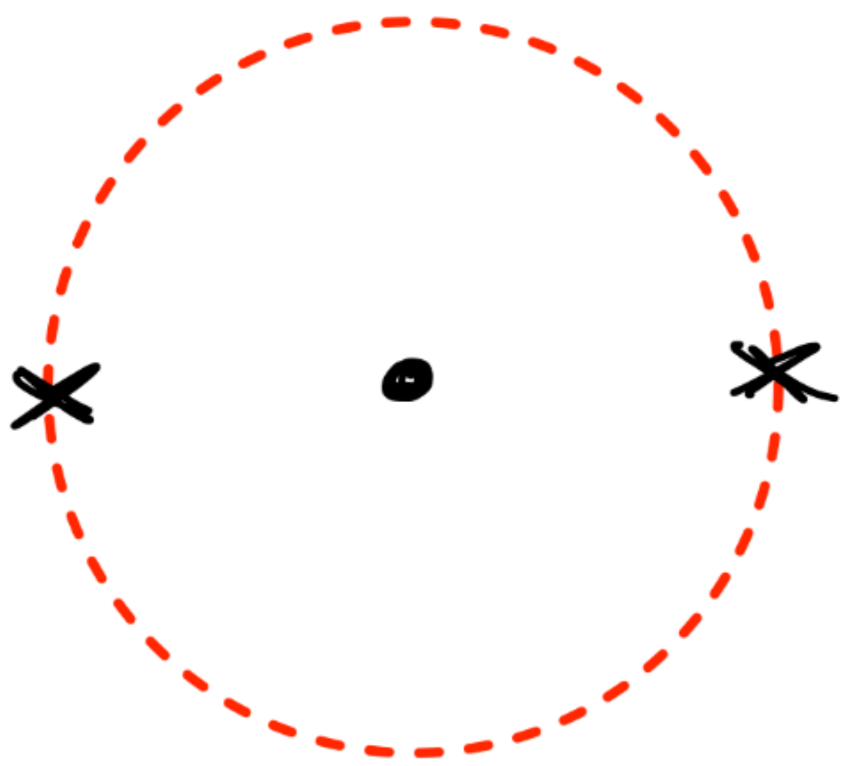
1. Idea detector simulation (IDEA_o1_v02.xml)
2. Digitizer v01 (moving to Digitizer v02)
3. Generalised geometric track finding algorithm
4. Evaluation step (tracking efficiency)

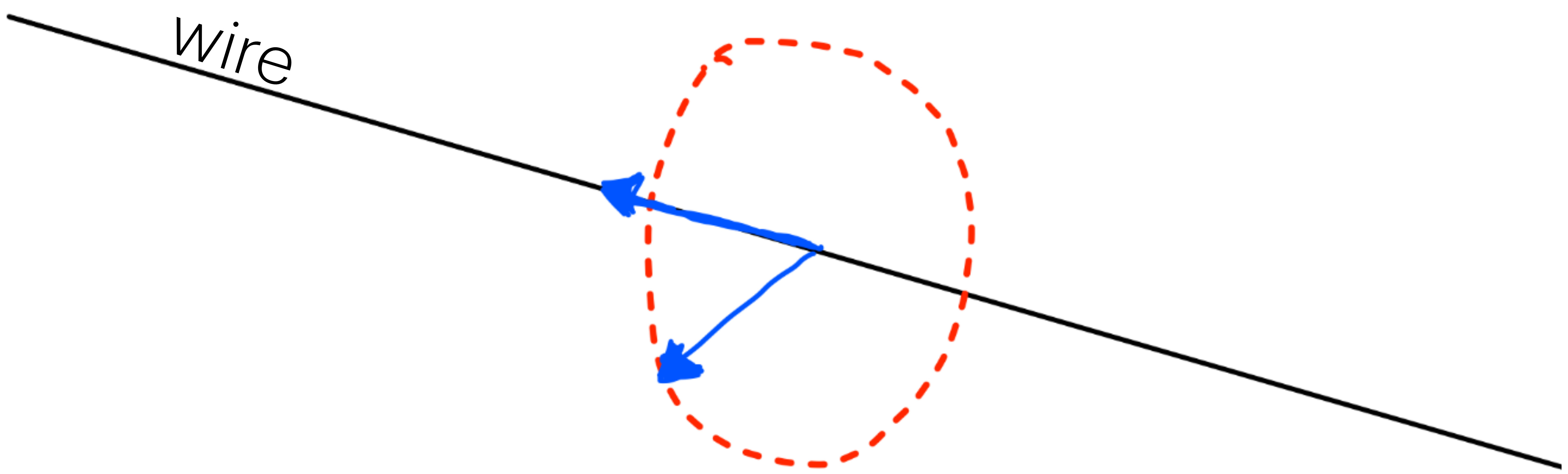
FUTURE CIRCULAR COLLIDER

Digitizer v01 vs Digitizer v02

In the following analysis, we used the digitizer_v01, which describes the drift chamber hits through two positions, **left** and **right** of the wire.

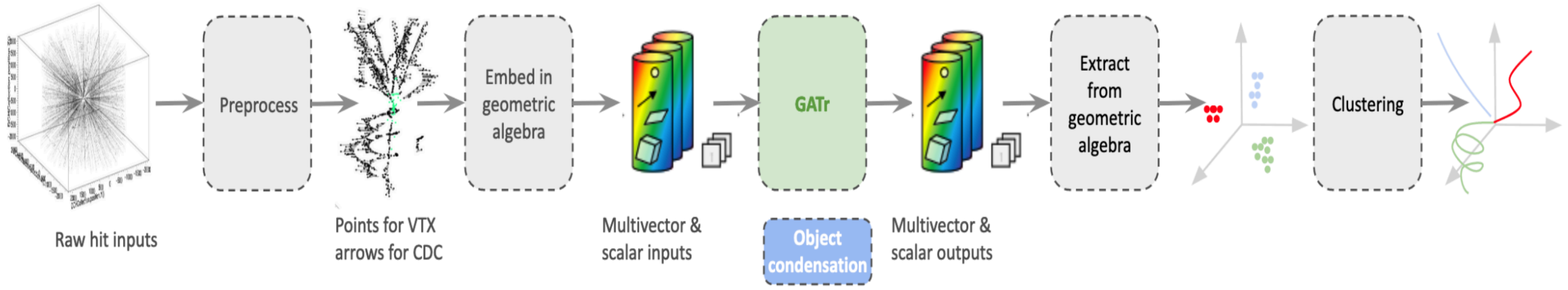
It will then be necessary to switch to digitiser_v02, for which the drift chamber hits are described by considering all possible positions on the **circumference** defined by a radius around the wire.





FUTURE CIRCULAR COLLIDER

Generalised geometric track finding algorithm



1. **ML step**: Track-finding approach based on a graph structure of inputs, on which geometric algebra transformations are applied. The final output is a set of pairs (beta, coordinates). Beta is a scalar used to define a potential that attracts results belonging to the same tracks and rejects those belonging to different tracks.

2. **Clustering step**: From the clusters in the embedding space, tracks can be obtained using the HDBSCAN or DBSCAN clustering algorithm.

Track Finder Implementation

The GGTF algorithm is implemented within the gaudi framework using a k4FWCore::MultiTransformer.

A transformer / MultiTransformer is an example of gaudi::functional, which consists in a general building block that is multithreading friendly.

- **Inputs**: digitalised Drift Chamber hits (extension::DriftChamberDigi) and digitalised Vertex hits (extension::TrackerHit3D)
- **Outputs**: collection of Tracks (extension::TrackCollection)

Track Finder - General Structure

```
struct GGTF_tracking_dbscan final : k4FWCore::MultiTransformer< std::tuple<TrackColl>(
                        const DCHitColl&
                        const VertexHitsColl&
                        const VertexHitsColl&
                        const VertexHitsColl&)>

{
    GGTF_tracking_dbscan(const std::string& name, ISvcLocator* svcLoc) : MultiTransformer ( name, svcLoc,
            {

                KeyValues("inputHits_CDC", {"inputHits_CDC"}),
                KeyValues("inputHits_VTXIB", {"inputHits_VTXIB"}),
                KeyValues("inputHits_VTXD", {"inputHits_VTXD"}),
                KeyValues("inputHits_VTXOB", {"inputHits_VTXOB"})

            },
            {
                KeyValues("outputTracks", {"outputTracks"})

            }) {}

    StatusCode initialize(){

           // CODE
           return StatusCode::SUCCESS;}


    std::tuple< <INSERT OUTPUTS> > operator()( <INSERT INPUTS> ) const override {

          // CODE
           return std::make_tuple(std::move(*output_tracks));}

    private:

    // PROPERTIES
};
```

Track Finder - ML Implementation

The first step of the track finder is the execution of the **ONNX model**.

In order to execute the model, **initialize()** must be used to create the inference session (<Ort::Session>) and define its options, such as memory management. Furthermore, during the initialization phase, the model is imported from the corresponding ONNX file.

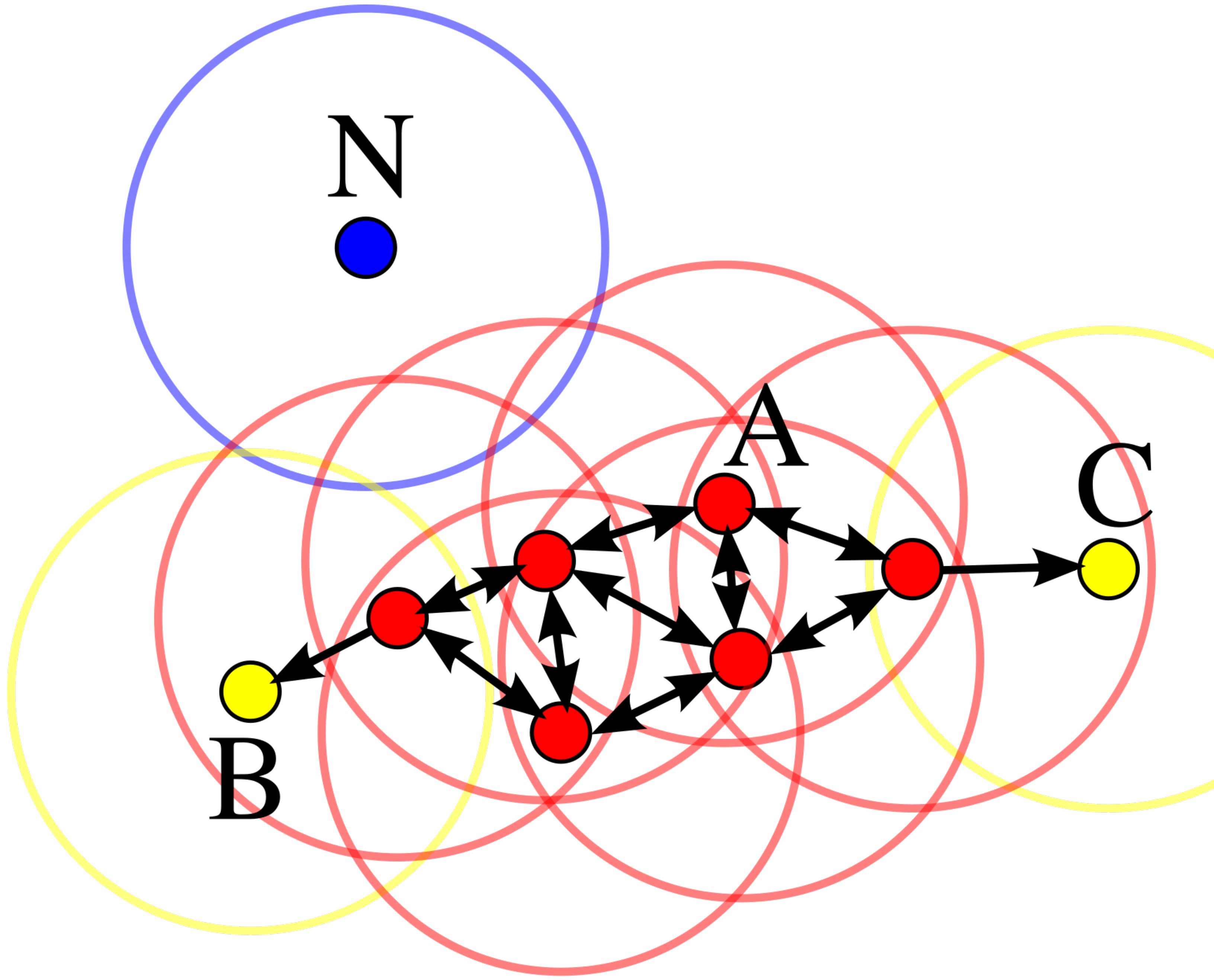
Within the execution phase **operator()**, the model receives as input a tensor containing all hits and it returns the coordinates (pos, beta) in the embedding space.

Track Finder - ML Implementation

```
StatusCode initialize() {

        fInfo = Ort::MemoryInfo::CreateCpu(OrtArenaAllocator, OrtMemTypeDefault);

        auto envLocal = std::make_unique<Ort::Env>(ORT_LOGGING_LEVEL_WARNING, "ONNX_Runtime");
        fEnv          = std::move(envLocal);

        fSessionOptions.SetIntraOpNumThreads(1);
        fSessionOptions.SetGraphOptimizationLevel(GraphOptimizationLevel::ORT_DISABLE_ALL);
        fSessionOptions.DisableMemPattern();

        auto sessionLocal = std::make_unique<Ort::Session>(*fEnv, “model.onnx”, fSessionOptions);
        fSession          = std::move(sessionLocal);

        Ort::AllocatorWithDefaultOptions allocator;
        const auto input_name = fSession->GetInputNameAllocated(0, allocator).release();
        const auto output_names = fSession->GetOutputNameAllocated(0, allocator).release();

        fInames.push_back(input_name);
        fOnames.push_back(output_names);

        return StatusCode::SUCCESS;

}
```

```
auto output_model_tensors = fSession->Run(Ort::RunOptions{nullptr},
            fInames.data(), input_tensors.data(), fInames.size(), fOnames.data(), fOnames.size());
```

Track Finder - Clustering



The second step of the track finder consists of the clustering algorithm, which is implemented with **DBSCAN** (Github Repository by Eleobert).

DBSCAN uses a definition of clusters based on the notion of **density**: if a point has a minimum number of points (min_points) within a certain epsilon distance (ϵ), it is classified as a **core point**. If a point is not a core point and it is not close to a core point, then it is classified as **noise**.

Starting with the core points, the clusters are expanded until all points are classified as noise or belonging to a cluster.

FUTURE CIRCULAR COLLIDER

Results

Track finder - OutputFile

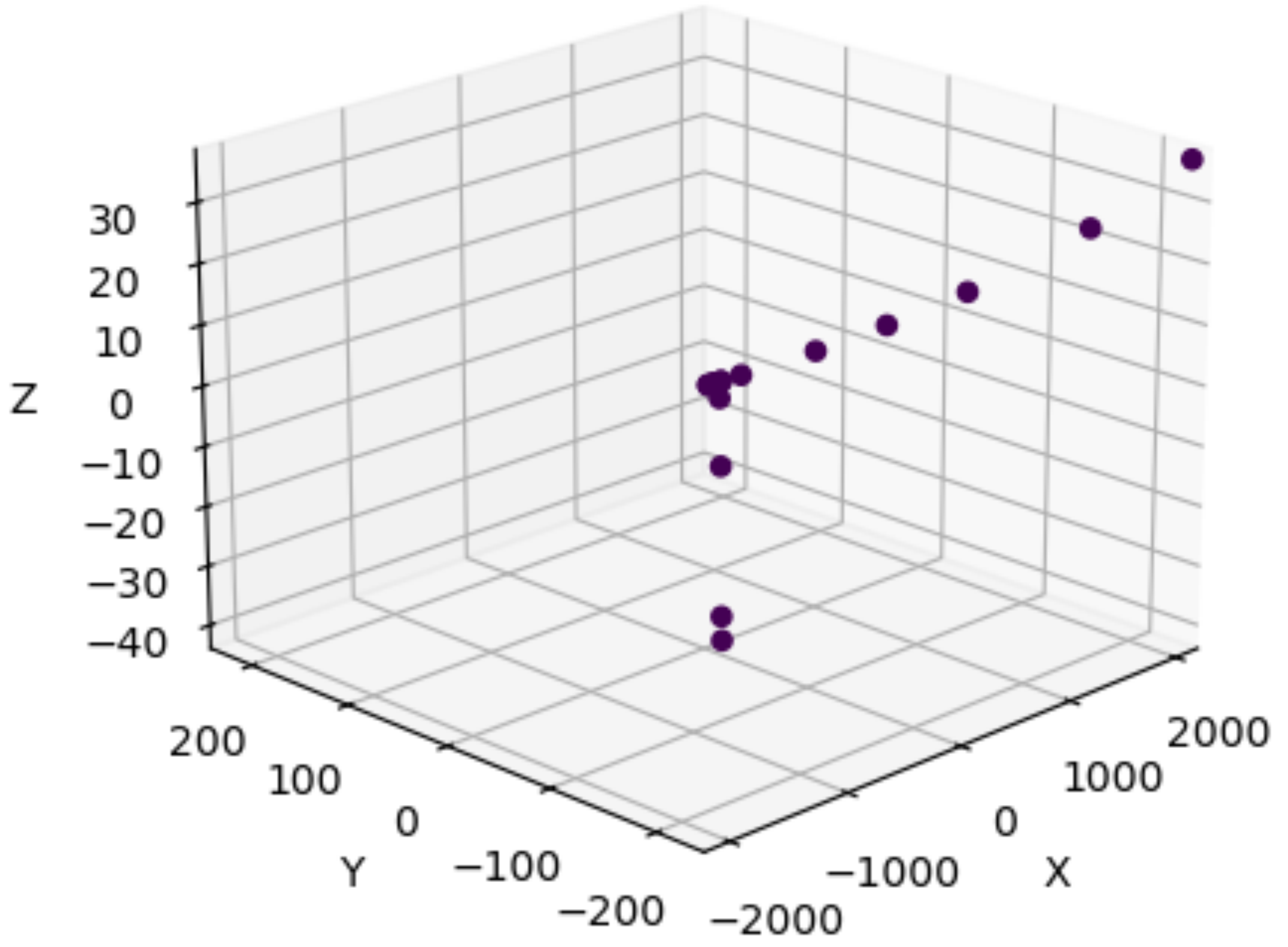
Name	ValueType
CDCHDigis	extension::DriftChamberDigi
CDCHDigisAssociation	extension::MCRecoDriftChamberDigiAssociation
CDCHHits	edm4hep::SimTrackerHit
Tracks	extension::Track
EventHeader	edm4hep::EventHeader
leftHitSimHitDeltaDistToWire	double
leftHitSimHitDeltaLocalZ	double
MCParticles	edm4hep::MCParticle
rightHitSimHitDeltaDistToWire	double
rightHitSimHitDeltaLocalZ	double
VTXD_links	edm4hep::TrackerHitSimTrackerHitLink
VTXDCollection	edm4hep::SimTrackerHit
VTXDDigis	edm4hep::TrackerHit3D
VTXIB_links	edm4hep::TrackerHitSimTrackerHitLink
VTXIBCollection	edm4hep::SimTrackerHit
VTXIBDigis	edm4hep::TrackerHit3D
VTXOB_links	edm4hep::TrackerHitSimTrackerHitLink
VTXOBCollection	edm4hep::SimTrackerHit
VTXOBDigis	edm4hep::TrackerHit3D

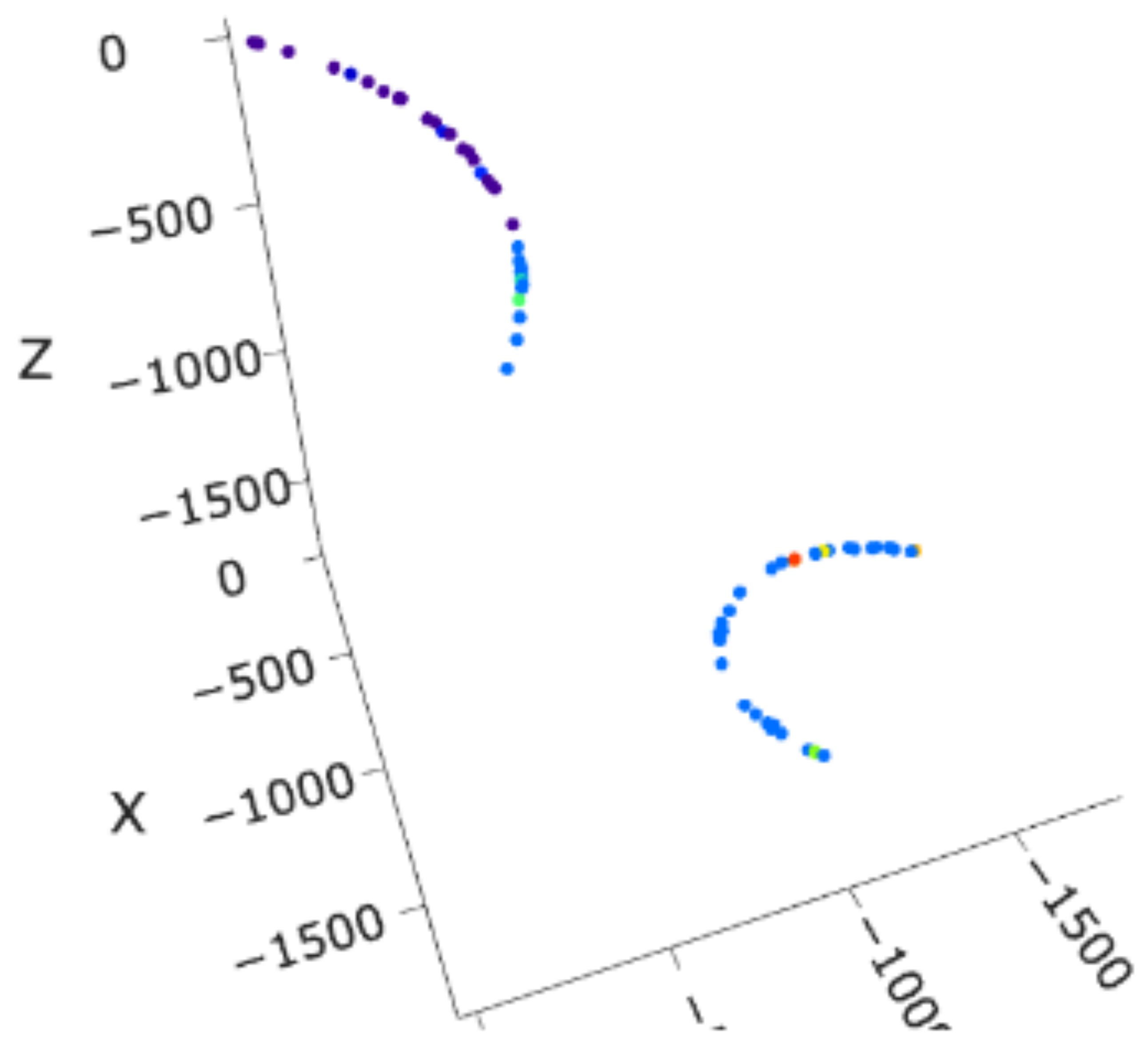
Results

Track finder - ddism behaviour

If we use the **ddism** command without any cut on the kinetic energy, what we get is that some hits are not assigned to the original particle.

If we add a cutoff to the kinetic energy, such as **SIM.part.minimalKineticEnergy “0.001*MeV”**, we obtain a correct classification of the hits, but hits in-between tracks are also assigned to other particles, which makes the definition of purity non-trivial.

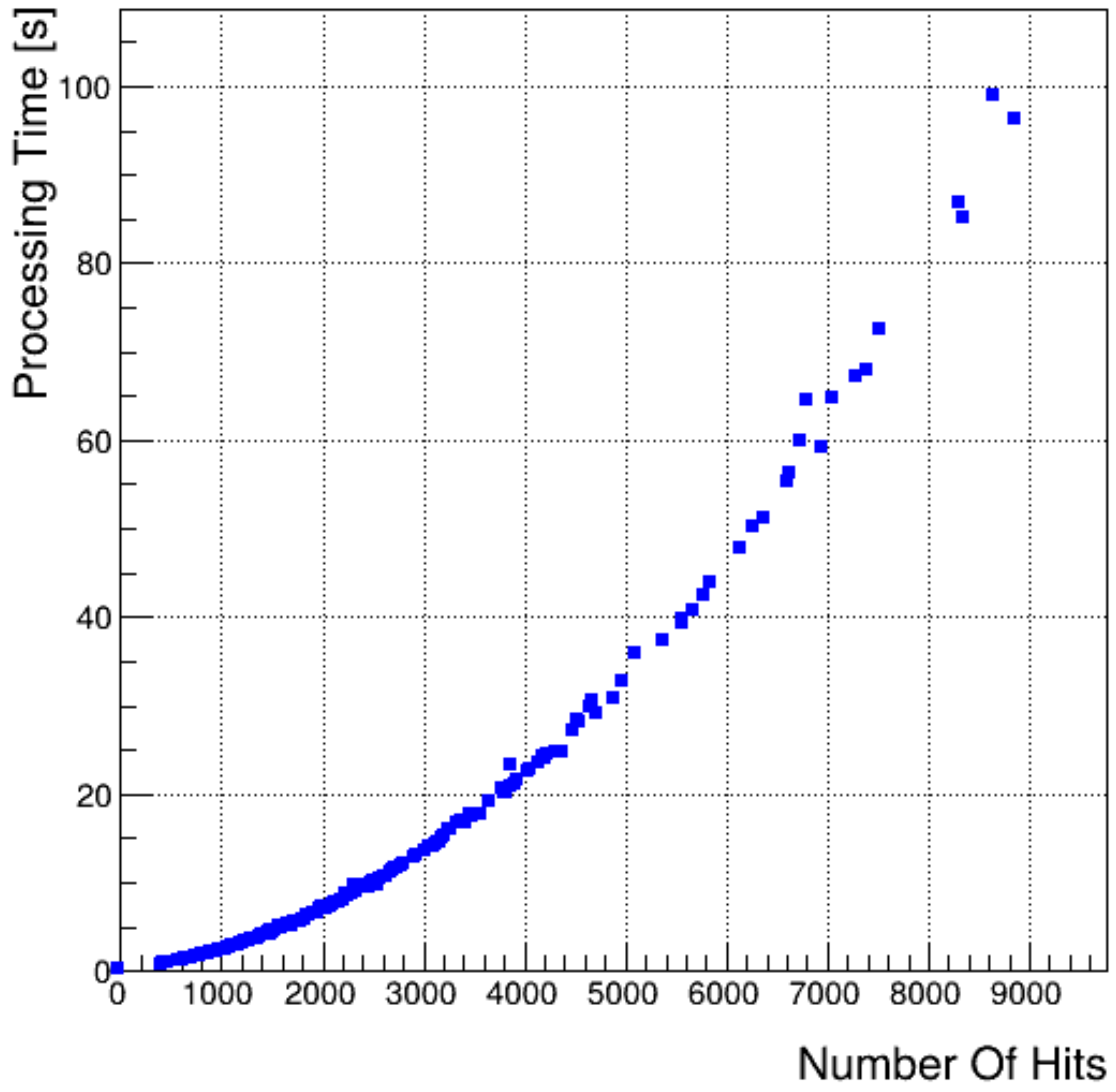


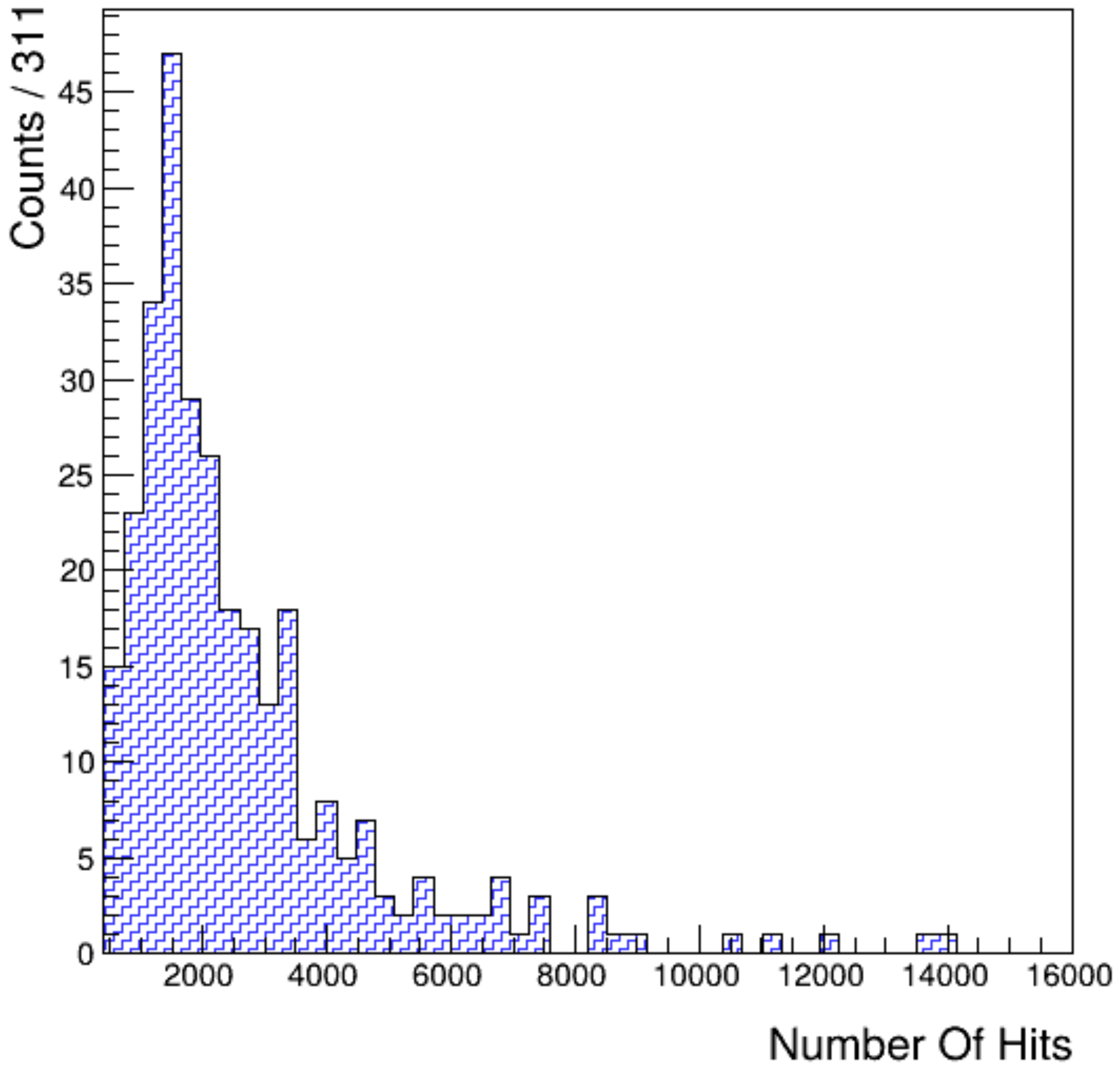


FUTURE CIRCULAR COLLIDER

Results

Track finder - processing time





FUTURE CIRCULAR COLLIDER

Track Finder - Issues

The tracker can only be used for events that contain less than 20000 hits, as otherwise there is excessive memory consumption (**Out of Memory - OOM**). Possible solutions are the use of machines with more available resources or the use of gpu even in the inference phase.

Sometimes the datatype **extension::TrackerHit** causes a segmentation violation when trying to access the hits saved in the output file.

Tracking Efficiency

Introduction

It is important to find a definition of tracking efficiency that is consistent with the design of the detector.

IDEA is based on a tracker with **drift chambers** and **vertex detectors**. For this reason, the number of hits is considerably higher than in CLD, since the drift chamber contributes significantly to this number.

By analogy of design, the definition proposed by BELLE II is chosen, for which a particle is assigned to a track depending on **purity** and **efficiency** values.

Moreover, the number of particles being considered is reduced through **cuts on certain properties of the MC particles**.

Tracking Efficiency - Step 1

Reconstructable particles

A particle is defined as **reconstructable** if it satisfies the following conditions (these thresholds can be tuned in the steering file):

1. $p_T > 100$ MeV
2. $\cos(\theta) < 0.99$
3. Number of unique hits (Drift Chamber + Vertex) > 15
4. Number of Drift Chamber hits > 4
5. Generator Status == 1
6. Vertex < 50 mm

Tracking Efficiency - Step 2

Purity and efficiency - Assigned particles

Given a particle p and a track t, it is possible to define two quantities known as purity and efficiency:

$$Pur_{MC_p}^{TRACK_t} = \frac{\text{num of hits from } MC_p \text{ in } TRACK_t}{\text{num of hits of } TRACK_t}$$

$$Eff_{TRACK_t}^{MC_p} = \frac{\text{num of hits from } TRACK_t \text{ in } MC_p}{\text{num of hits of } MC_p}$$

A particle p is **assigned** to the track t if $Pur_{MC_p}^{TRACK_t} > 0.5$ and $Eff_{TRACK_t}^{MC_p} > 0.5$

FUTURE CIRCULAR COLLIDER

Tracking Efficiency - Step 3

Tracking efficiency and Fake Tracks

$$\text{Tracking efficiency} = \frac{\text{number of reconstructable and assigned particles}}{\text{number of reconstructable particles}}$$

Tracks with no assigned particles are considered **fake tracks**.

Particles that are assigned but are not reconstructable may indicate that it is possible to relax the definition of “reconstructable”.

Evaluation Implementation

The GGTF evaluation step is implemented within the gaudi framework using a k4FWCore::MultiTransformer.

- **Inputs**:
  1. Track collection (extension::TrackCollection)
  2. simHits (edm4hep::SimTrackerHitCollection)
  3. MC particle collection (edm4hep::MCParticleCollection)

- **Outputs**: table of features (see structure in the next slide) and number of fakes

Table of Features

Structure

```
Event: 49
     PDG        pt  costheta       phi  ...  assigned_track  isReconstructable  isAssigned  isRecoAndAssigned
0     11  0.000000  1.000000  0.000000  ...               0                  0           0                  0
1     11  0.000000  1.000000  0.000000  ...               0                  0           0                  0
2     22  0.000000  1.000000  0.000000  ...               0                  0           0                  0
3    -11  0.000000  1.000000  0.000000  ...               0                  0           0                  0
4    -11  0.000000  1.000000  0.000000  ...               0                  0           0                  0
..   ...       ...       ...       ...  ...             ...                ...         ...                ...
```

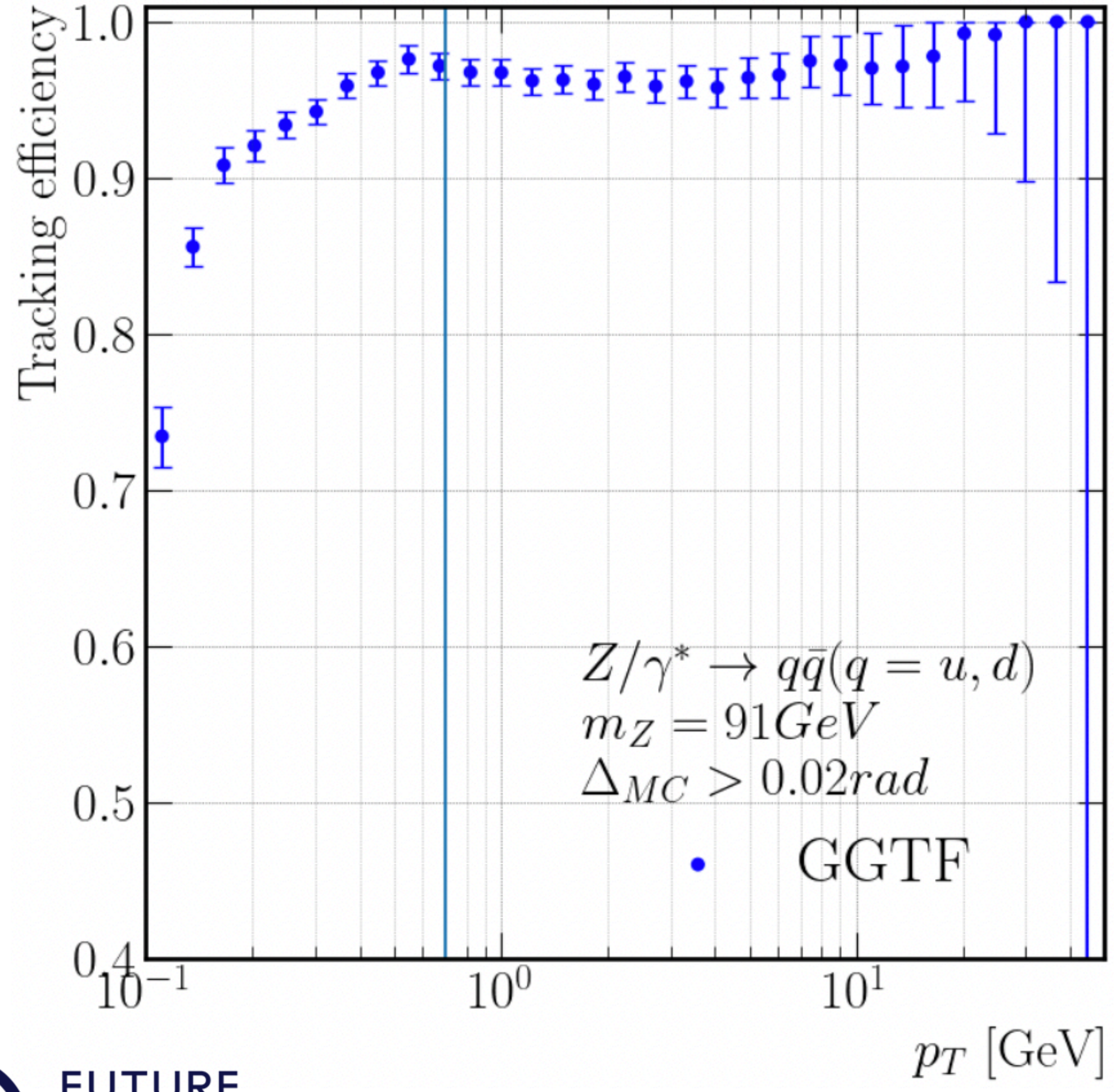
- **Assigned_track**: index of the track which the particle has been assigned to (0 if not assigned)
- **isReconstructable**: boolean value which is 1 if the particle is reconstructable and 0 otherwise
- **isAssigned**: boolean value which is 1 if the particle is assigned and 0 otherwise
- **Purity**: purity of the track which the particle has been assigned to (-1 if not assigned)
- **Efficiency**: efficiency of the particle with respect to the track which the particle has been assigned to (-1 if not assigned)

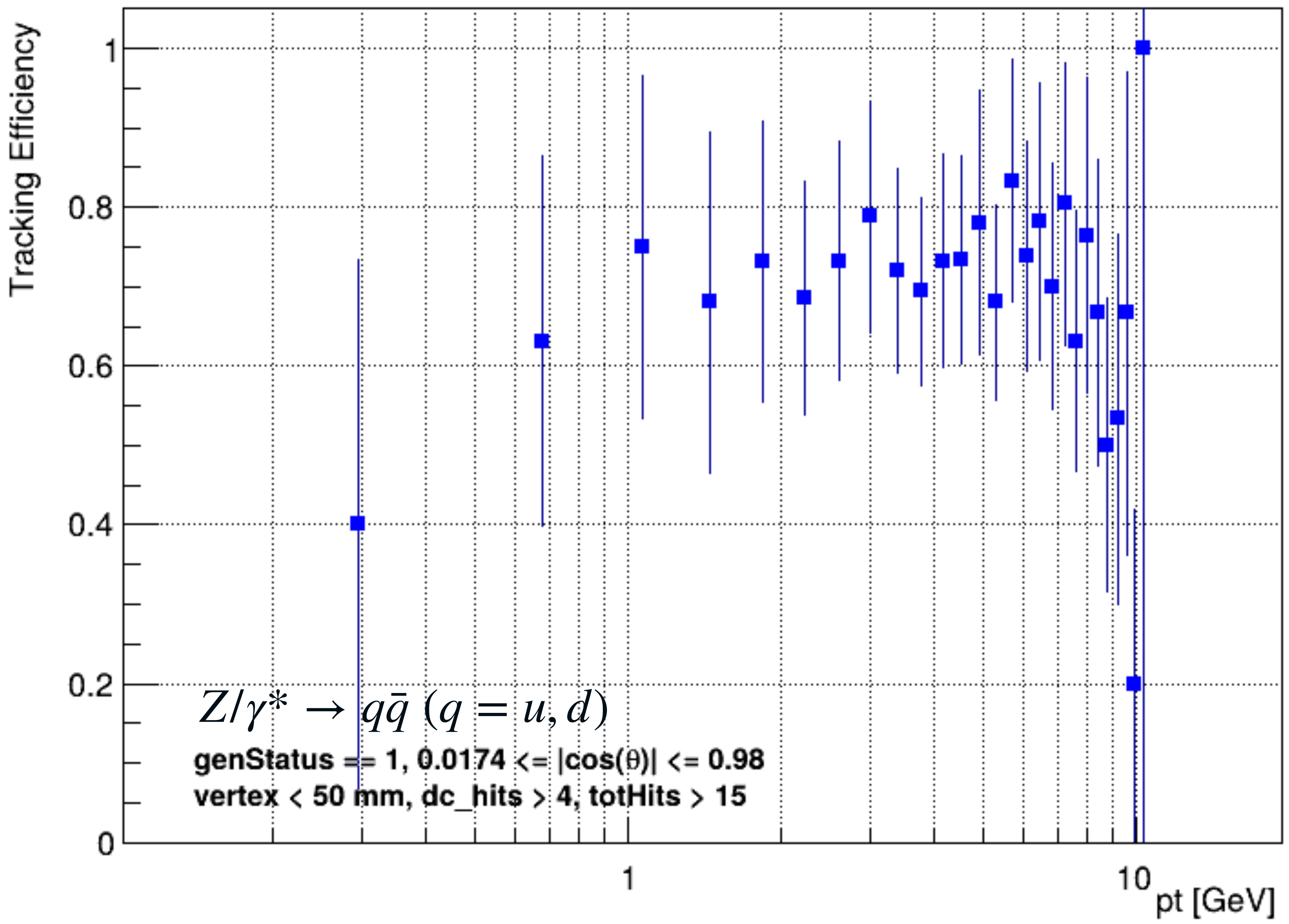
Evaluation - General Structure

```
struct GGTF_efficiency final :
        k4FWCore::MultiTransformer<std::tuple< <INSERT OUTPUTS > >( <INSERT OUTPUTS > )>
{
    GGTF_efficiency(const std::string& name, ISvcLocator* svcLoc) :
        MultiTransformer ( name, svcLoc,
            {

                KeyValues("InputCollectionTracks", {"inputTracks"}),
                KeyValues("InputCollectionParticles", {"inputMCparticles"}),
                KeyValues("inputHits_DC_sim", {"inputHits_DC_sim"}),
                KeyValues("inputHits_VTXIB_sim", {"inputHits_VTXIB_sim"}),
                KeyValues("inputHits_VTXD_sim", {"inputHits_VTXD_sim"}),
                KeyValues("inputHits_VTXOB_sim", {"inputHits_VTXOB_sim"})

            },
            {
                KeyValues("out_costheta", {"out_costheta"}),
                KeyValues("out_pt", {"out_pt"}),
                KeyValues("out_phi", {"out_phi"}),
                KeyValues("out_vertex", {"out_vertex"}),
                KeyValues("out_pdg", {"out_pdg"}),
                KeyValues("out_num_hits", {"out_num_hits"}),
                KeyValues("out_num_hits_driftChamber", {"out_num_hits_driftChamber"}),
                KeyValues("out_pur", {"out_pur"}),
                KeyValues("out_eff", {"out_eff"}),
                KeyValues("assigned_track_mc", {"assigned_track_mc"}),
                KeyValues("numberFakes", {"numberFakes"}),
                KeyValues("genStatus", {"genStatus"}),
                KeyValues("isReconstructable", {"isReconstructable"}),
                KeyValues("isAssigned", {"isAssigned"}),
                KeyValues("isRecoAndAssigned", {"isRecoAndAssigned"})

            }) {}


    std::tuple< <INSERT OUTPUTS > > operator()( <INSERT INPUTS > ) const override
    {
        // CODE
    }

    private:

      // PROPERTIES
};
```

Results

Performance for complex events - Python vs C++





FUTURE CIRCULAR COLLIDER

What's next

A pipeline is currently available, but there are some open problems:

1. A Track finder gaudi::functional is available for events with less than 20000 hits
2. Object within namespace "extension" have problems when writing to the rootfile, as reading the content causes segfault errors.
3. There is a difference in the performance of the ML model between inference with C++ and inference with Python. This is probably due to the way float and double are treated in C++ and Python (see open question on StackOverflow).
4. It is necessary to reduce the processing time of the model through a reduced model and / or through some optimisations.

FUTURE CIRCULAR COLLIDER