

Research Open Data Limitations

Mariana Vivas Albornoz (DESY), on behalf of the ATLAS Open Data team
ATLAS Open Data Tutorial
25 November 2025



Why a “limitations” section?

Our samples are targeted to a specific types of analyses. We want to highlight this to **prevent dead ends or false positives**.

If your idea needs missing ingredients we want you to know that:

- You can [contact us](#)
- Consider [short-term association](#).



Want to discuss? Check the [CERN Open Data forum](#) or open a new topic by clicking on one of the categories below!

An account is required.

 Software

 Datasets

 Documentation

 Contributing

If you still need assistance, write us at [atlas-outreach-opendata-support\[at\]cern\[dot\]ch](mailto:atlas-outreach-opendata-support@cern.ch)

Let's start with general information

To use our samples, there are some things you should have in mind, to avoid inconsistencies.

- Always **use data listed in the Good Runs List (GRL)**: This is the only way we have to assure you of the quality of the data.
- **Some signals/variations/systematics are not publicly released**: if you need a specific sample, you can request it.
- **Producing large bespoke MC yourself is possible but discouraged**, it is resource-heavy, and easy to mismatch conditions.

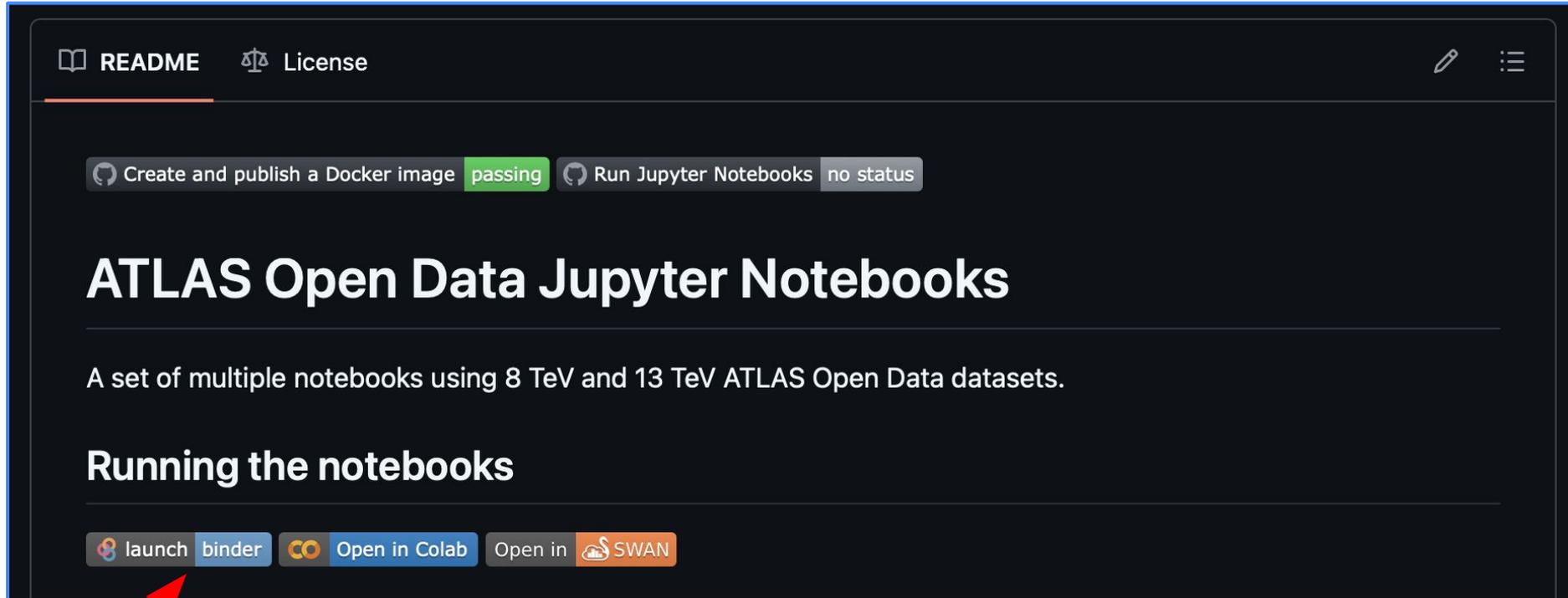
Let's start with general information

To use our samples, there are some things you should have in mind, to avoid inconsistencies.

- Always **use data listed in the Good Runs List (GRL)**: This is the only way we have to assure you of the quality of the data.
- **Some signals/variations/systematics are not publicly released**: if you need a specific sample, you can request it.
- **Producing large bespoke MC yourself is possible but discouraged**, it is resource-heavy, and easy to mismatch conditions.

Good run list and good lumiblocks

Open the [notebooks' repository](#), and click on launch binder.

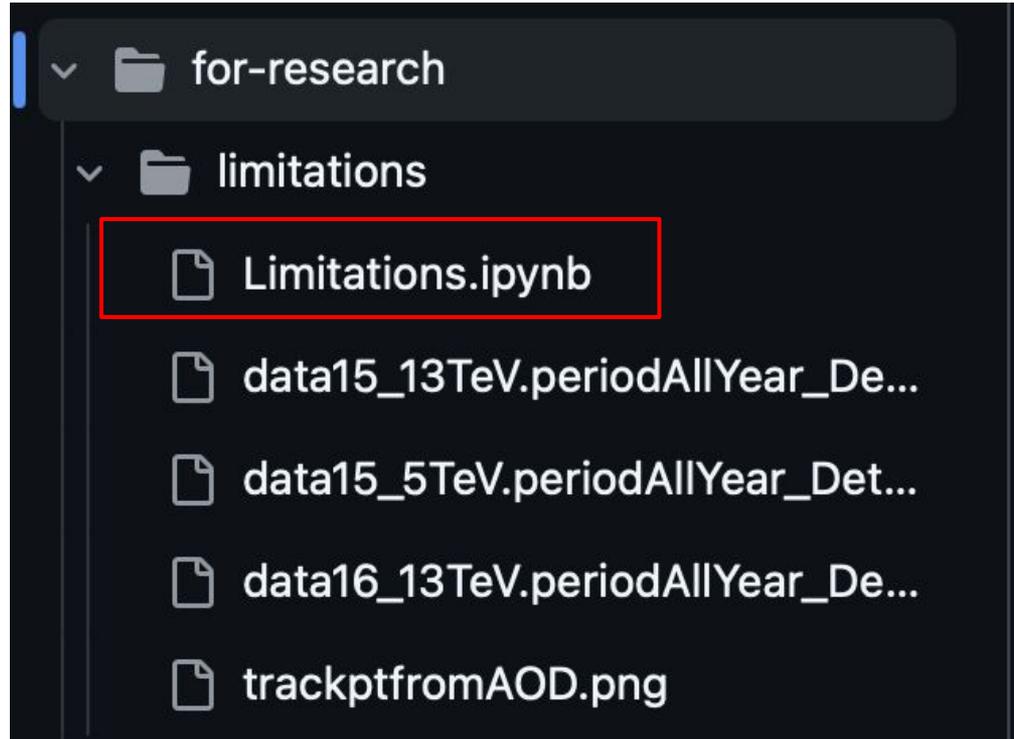


The screenshot shows a GitHub repository page for 'ATLAS Open Data Jupyter Notebooks'. At the top, there are tabs for 'README' and 'License'. Below the tabs, there are two status indicators: 'Create and publish a Docker image' with a green 'passing' label, and 'Run Jupyter Notebooks' with a grey 'no status' label. The main heading is 'ATLAS Open Data Jupyter Notebooks', followed by a description: 'A set of multiple notebooks using 8 TeV and 13 TeV ATLAS Open Data datasets.' Below this is a section titled 'Running the notebooks'. At the bottom of this section, there are four buttons: 'launch binder' (with a red arrow pointing to it), 'Open in Colab', 'Open in SWAN', and a 'launch' button with a logo.

⚠️ We need Delphes, so this won't run on colab or swan.

Good run list and good lumiblocks

Open the [notebooks' repository](#), and click on launch binder.



⚠ We need Delphes, so this won't run on colab or swan.

Good run list and good lumiblocks

PHYS_StandardGRL_All_Good_25ns Good Run List

Query

Period: data15_13TeV.AllYear; Defect: PHYS_StandardGRL_All_Good_25ns; Defect tag: DetStatus-v89-pro21-02; Ignoring None

RunList

276262,276329,276336,276416,276511,276689,276778,276790,276952,276954,278880,278912,278968,279169,279259,279279,279284

StreamListInfo

Lumi blocks

276262

LB : 72-84, LB : 86-110, LB : 118-288, LB : 292-379,

276329

LB : 147-558,

276336

LB : 4-15, LB : 36-54,

Good run list and good lumiblocks

PHYS_StandardGRL_All_Good_25ns Good Run List

Query

Period: data15_13TeV.AllYear; Defect: PHYS_StandardGRL_All_Good_25ns; Defect tag: DetStatus-v89-pro21-02; Ignoring None

RunList

276262,276329,276336,276416,276511,276689,276778,276790,276952,276954,278880,278912,278968,279169,279259,279279,279284

StreamListInfo

Lumi blocks

276262

LB : 72-84, LB : 86-110, LB : 118-288, LB : 292-379,

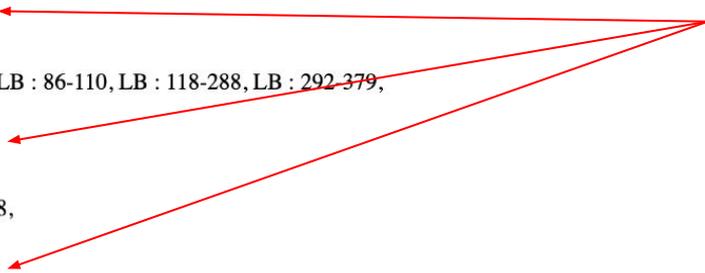
276329

LB : 147-558,

276336

LB : 4-15, LB : 36-54,

Good runs



Good run list and good lumiblocks

PHYS_StandardGRL_All_Good_25ns Good Run List

Query

Period: data15_13TeV.AllYear; Defect: PHYS_StandardGRL_All_Good_25ns; Defect tag: DetStatus-v89-pro21-02; Ignoring None

RunList

276262,276329,276336,276416,276511,276689,276778,276790,276952,276954,278880,278912,278968,279169,279259,279279,279284

StreamListInfo

Lumi blocks

276262

LB : 72-84, LB : 86-110, LB : 118-288, LB : 292-379,

276329

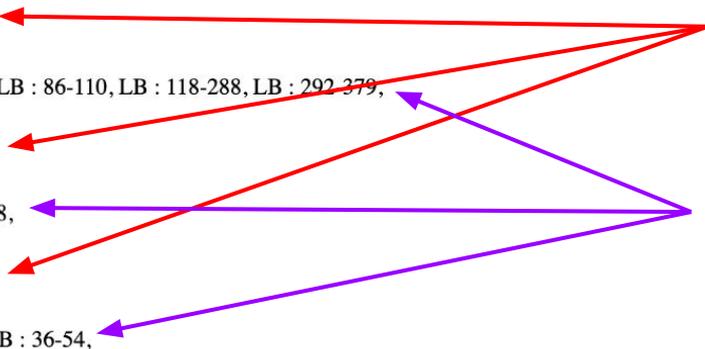
LB : 147-558,

276336

LB : 4-15, LB : 36-54,

Good runs

Good lumiblocks



Good run list and good lumiblocks

The .xml files look like this:

```
1 <?xml version="1.0"?>
2 <!DOCTYPE LumiRangeCollection SYSTEM "http://atlas-runquery.cern.ch/LumiRangeCollection.dtd">
3 <!--This document is created by GoodRunsListWriter.-->
4 <?xml-stylesheet type="text/xsl" href="http://atlasdqm.web.cern.ch/atlasdqm/grlview/grl.xsl" title="grlview" ?>
5 <LumiRangeCollection>
6 <NamedLumiRange>
7 <Name>PHYS_HeavyIonP_All_Good</Name>
8 <Version>2.1</Version>
9 <Metadata Name="ARQEquivalentQuery">find run data15_5TeV.periodAllYear and dq PHYS_HeavyIonP_All_Good DEFECTS#DetStat
10 <Metadata Name="Query">Period: data15_5TeV.AllYear; Defect: PHYS_HeavyIonP_All_Good; Defect tag: DetStatus-v105-pro22
11 <Metadata Name="RunList">286361,286364,286367,286411,286474</Metadata>
12 <LumiBlockCollection>
13 <Run>286361</Run>
14 <LBRange Start="143" End="280"/>
15 </LumiBlockCollection>
16 ☐LumiBlockCollection☐
17 <Run>286364</Run>
18 <LBRange Start="141" End="618"/>
19 </LumiBlockCollection>
20 <LumiBlockCollection>
21 <Run>286367</Run>
22 <LBRange Start="4" End="75"/>
23 <LBRange Start="78" End="149"/>
```

Good run list and good lumiblocks

```
[3]: import xml.etree.ElementTree as ET

def grl_parser(xml_file):
    # We are going to use this function to parse the GRL .xml file

    tree = ET.parse(xml_file)
    root = tree.getroot()

    # Dict: run_number -> list of (start_lb, end_lb)
    runs_to_lbs = {}

    # find the beginning and end of the file
    named = root.find("NamedLumiRange")
    # each run+LBs is in a LumiBlockCollection
    for lbcoll in named.findall("LumiBlockCollection"):
        run = int(lbcoll.find("Run").text) # get the run number

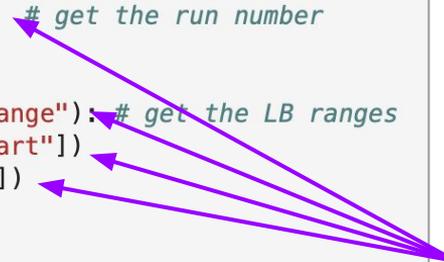
        lb_ranges = []
        for lbrange in lbcoll.findall("LBRange"): # get the LB ranges
            start = int(lbrange.attrib["Start"])
            end = int(lbrange.attrib["End"])
            lb_ranges.append((start, end))

        # map run number to its LB ranges
        runs_to_lbs[run] = lb_ranges

    return runs_to_lbs
```

We are going to use this to parse the file. Not super relevant as this can be parsed in your preferred way

These are the tags that let us know where the information is



Good run list and good lumiblocks

The GRL for 2015 run

```
[4]: # let's pretty print the GRL dictionary
import pprint
pp = pprint.PrettyPrinter()
grl15_dict = grl_parser("data15_13TeV.periodAllYear_DetStatus-v89-pro21-02_Unknown_PHYS_StandardGRL_All_Good_25ns.xml")
pp.pprint(grl15_dict)
```

```
{276262: [(72, 84), (86, 110), (118, 288), (292, 379)],
276329: [(147, 558)],
276336: [(4, 15), (36, 54)],
276416: [(373, 418), (445, 508)],
276511: [(185, 476)],
276689: [(198, 214), (216, 464)],
276778: [(177, 193)],
276790: [(251, 286)],
```

Good runs

Good LBs

Good run list and good lumiblocks

We are going to use the 2015 pp data and open it with uproot.

```
[5]: # Get a list with all the data samples
data_urls = atom.get_urls('data', protocol='https', cache=True)
```

```
[6]: # Let's look at the first file
tree = uproot.open({data_urls[0]: "CollectionTree"})
print(f'The file in "{data_urls[0]}" \ncontains the following run number:')
runnumber = tree["EventInfoAuxDyn.runNumber"].array()
print(runnumber)
```

```
The file in "simplecache::https://opendata.cern.ch/eos/opendata/atlas/rucio/data15_13TeV/DA0D_PHYSLITE.37001626._000001.pool.root.1"
contains the following run number:
[266904, 266904, 266904, 266904, 266904, ..., 266904, 266904, 266904, 266904]
```

The run number is found inside
the EventInfo container

Good run list and good lumiblocks

You can easily compare the run number with the good runs in the GRL:

```
[7]: # Convert the run numbers, which are the keys in our GRL dict, to a list  
grl15 = list(grl15_dict.keys())  
runnumber[0] in grl15
```

```
[7]: False
```

Good run list and good lumiblocks

And compare the lumiblocks within a loop:

```
tree = uproot.open({data_urls[3007]: "CollectionTree"})
# Get all the event info
runnumber = tree["EventInfoAuxDyn.runNumber"].array()
# Check unique run numbers
unique_runnumber = np.unique(ak.to_numpy(runnumber))
print("Unique run numbers:", unique_runnumber)
lumiblock = tree["EventInfoAuxDyn.lumiBlock"].array()
# Check unique lumiblocks
unique_lbs = np.unique(ak.to_numpy(lumiblock))
print("Unique lumiblocks:", unique_lbs)
eventnumber = tree["EventInfoAuxDyn.eventNumber"].array()
```

Run number

Lumiblock

Good run list and good lumiblocks

And compare the lumiblocks within a loop:

```
if runnumber[0] not in grl15:
    print("Run number not in GRL 2015!")
else:
    print("Run number is in GRL 2015!")
    lumiblos_grl15 = grl15_dict[runnumber[0]]
    # We will record the event number (as IDs) for the good events
    good_eventnumbers = []
    # Loop over all lumiblocks
    for ev, lb in zip(eventnumber, lumiblock):
        in_good_range = False
        for start, end in lumiblos_grl15:
            if start <= lb <= end:
                in_good_range = True
                break # no need to check other ranges after found

    # if in good range, save event number
    if in_good_range:
        good_eventnumbers.append(ev)
```

Check run number

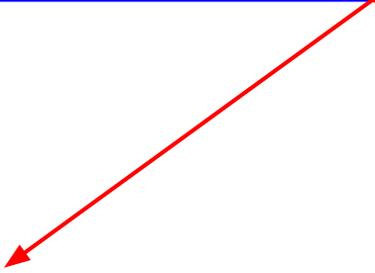
We are going to save the event number of the events with good runs and good LBs

Loop through lumiblocks

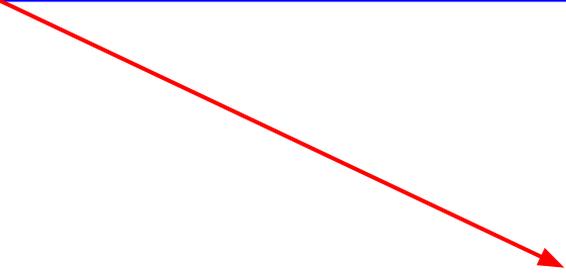
Check that the lumiblock fall in the good lumiblocks range

Good run list and good lumiblocks

How it's done is not so important, just remember to **always check that you are only using good runs and good lumiblocks** in your analysis.



Variables are in the EventInfo tree



GRLs are in the repository

This applies to the proton-proton and heavy ion detector data.

Proton-proton collisions data

The scope of the proton-proton data is **limited by the group of samples that is public.**

Public samples cover **standard backgrounds and generic systematics** but key systematics may be missing, as well as many signal/variation grids that are not released.

🏠 > The Data > Data for research > Proton-proton Collisions > Limitations

Limitations

Although these data are all intended for use in scientific research, they are not without limitations. It is important to keep in mind the limitations of these datasets when using them in order to avoid false-positive results or dead ends. Although

We encourage [reaching out](#) in case you are missing a sample. Requests will be considered case by case.

PHYSLITE format

Proton-proton collisions are stored in a PHYSLITE format file.

The PHYSLITE format is aimed to support “standard” object selections and workflows. Non-standard object definitions and reconstruction are generally not supported. For example, **not all charged tracks are available**; neither are all calo clusters or PFlow objects.

Let's look at the tracks. We are going to use the dataset with id 410470, which is a ttbar sample.

```
[11]: # We are using 410470.PhPy8EG_A14_ttbar_hdamp258p75_nonallhad
      urls_sample = atom.get_urls(410470, protocol='https', cache=True)
      filename = urls_sample[0]
      #filename = 'DAOD_PHYSLITE.37620644._000012.pool.root.1' # Local
      tree = uproot.open({filename: "CollectionTree"})
```

PHYSLITE format

```
[13]: indet_n = tree["InDetTrackParticlesAuxDyn.phi"].array()
counts = ak.num(indet_n, axis=1) # tracks per event
counts_np = ak.to_numpy(counts)

# bin edges: ..., 0.5, 1.5, 2.5, ...
bins = np.arange(counts_np.max() + 2) - 0.5

# number of events per multiplicity bin
hist, edges = np.histogram(counts_np, bins=bins)

N_ev = len(counts_np)
bin_centers = 0.5 * (edges[1:] + edges[:-1])

# normalize: 1/N_ev * dN/dn (Δn = 1 so this is just fraction of events)
y = hist / N_ev

plt.figure(figsize=(6, 4))

# step plot vs bin centroid
plt.step(bin_centers, y, where="mid")

plt.yscale("log")
plt.ylim(1e-6, 1)
plt.xlabel(r"$n_{\mathrm{trk}}$")
plt.ylabel(r"$\frac{1}{N_{\mathrm{ev}}} \cdot \frac{dN_{\mathrm{ev}}}{dn_{\mathrm{trk}}}$")

plt.tight_layout()
plt.show()
```

We get a variable
from:
InDetTrackParticles

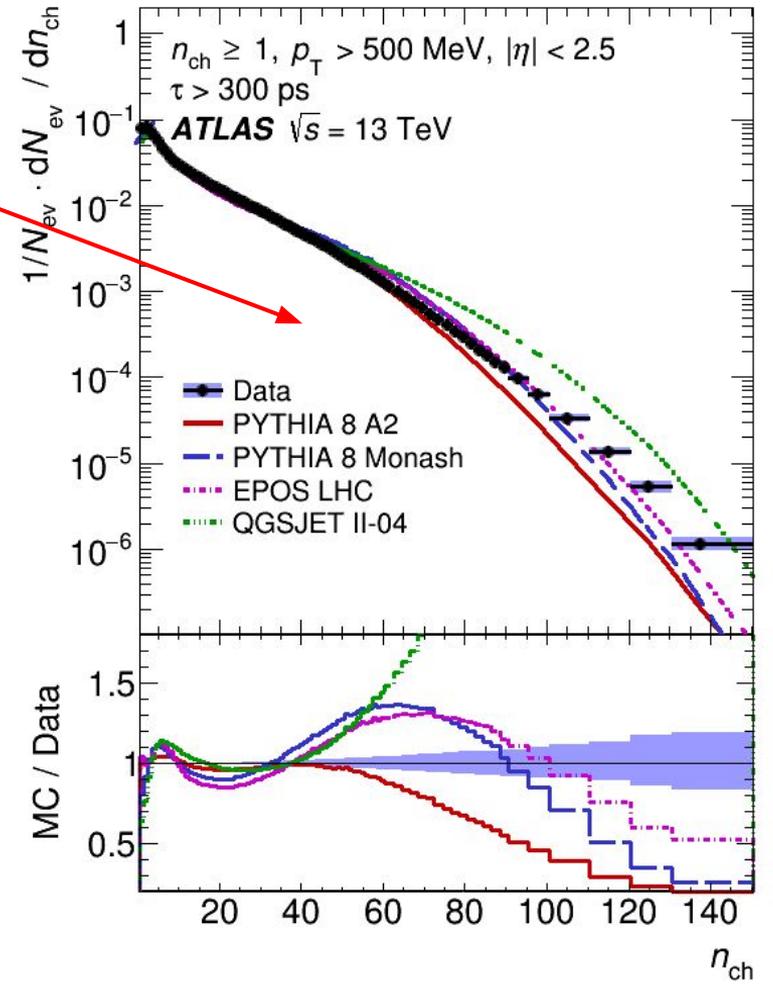
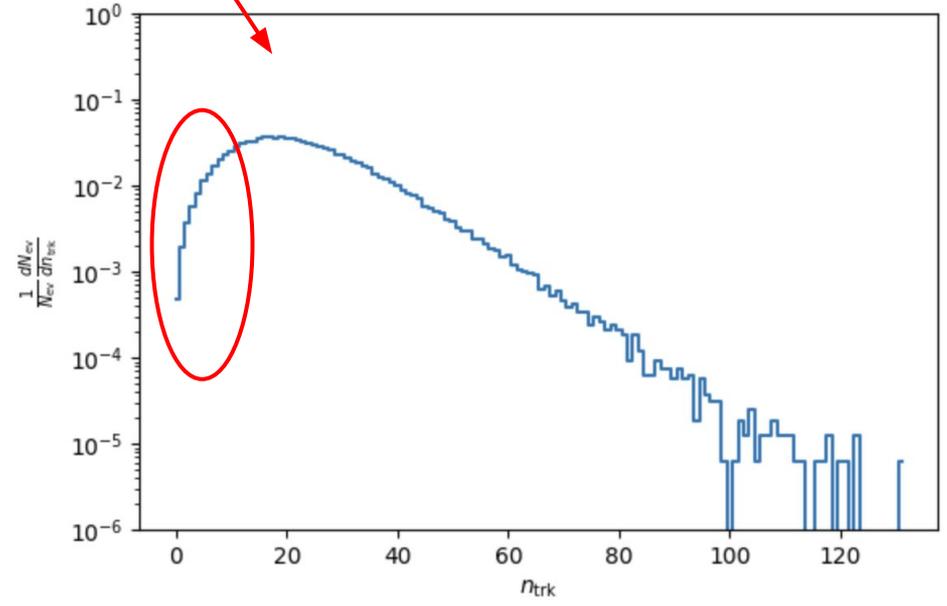
Count how many
tracks we have

The rest is just
normalizing and
plotting a
histogram.

PHYSLITE format

PHYSLITE

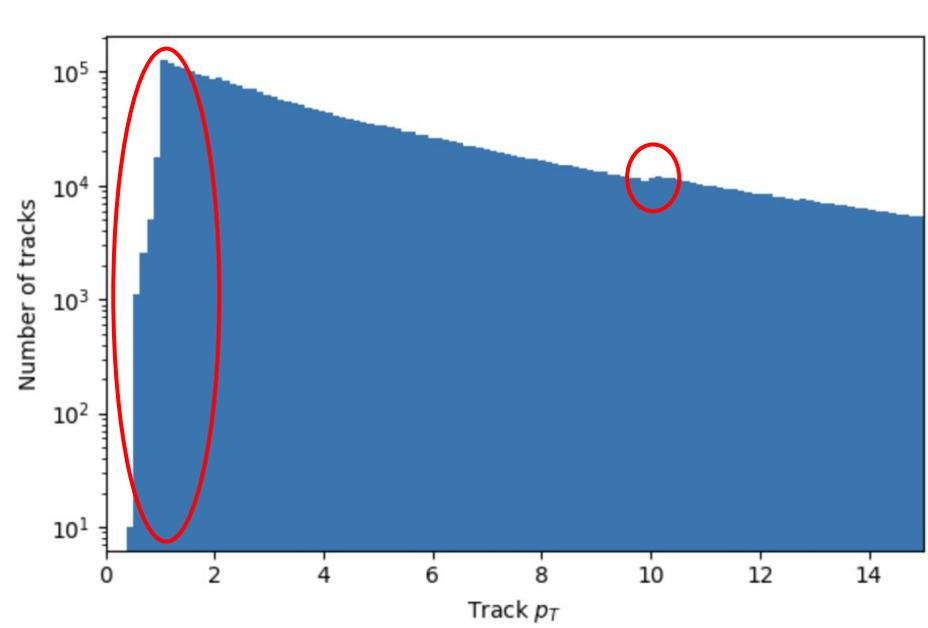
Multiplicity of charged-particle
[ref]



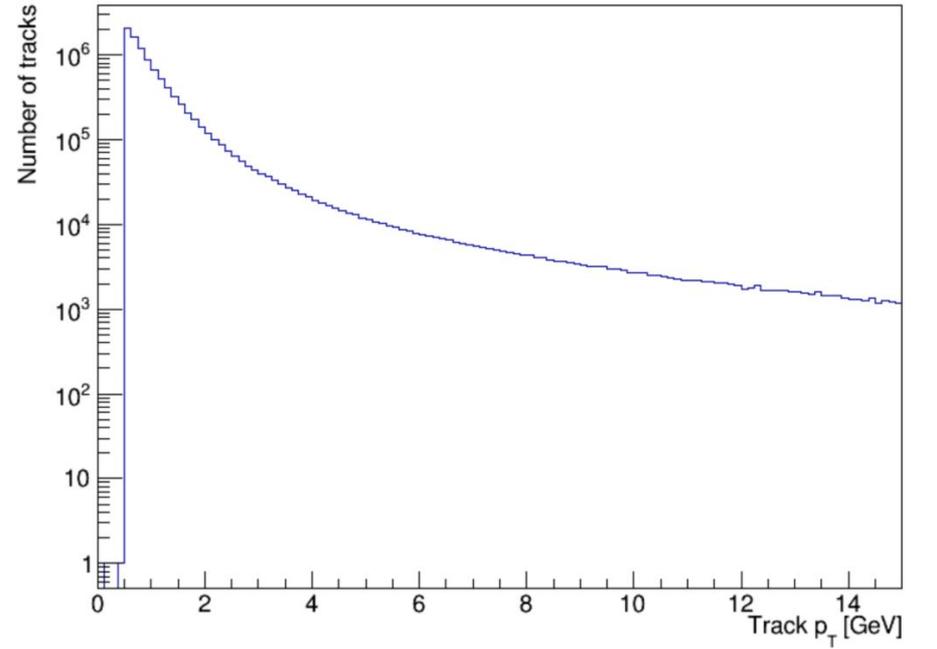
PHYSLITE format

Using QoverP and theta we can look at the p_T spectrum as well. PHYSLITE doesn't have all the tracks for tracks with p_T lower than 10 GeV

PHYSLITE



AOD (from where the PHYSLITE came from)



PHYSLITE format

We can also look at where the vertices that are stored actually sit in the detector.

```
[15]: # get coordinates
hs_x = tree["HardScatterVerticesAuxDyn.x"].array()
hs_y = tree["HardScatterVerticesAuxDyn.y"].array()
pv_x = tree["PrimaryVerticesAuxDyn.x"].array()
pv_y = tree["PrimaryVerticesAuxDyn.y"].array()
```

For that we use HardScatterVertices and PrimaryVertices, the x and y position.

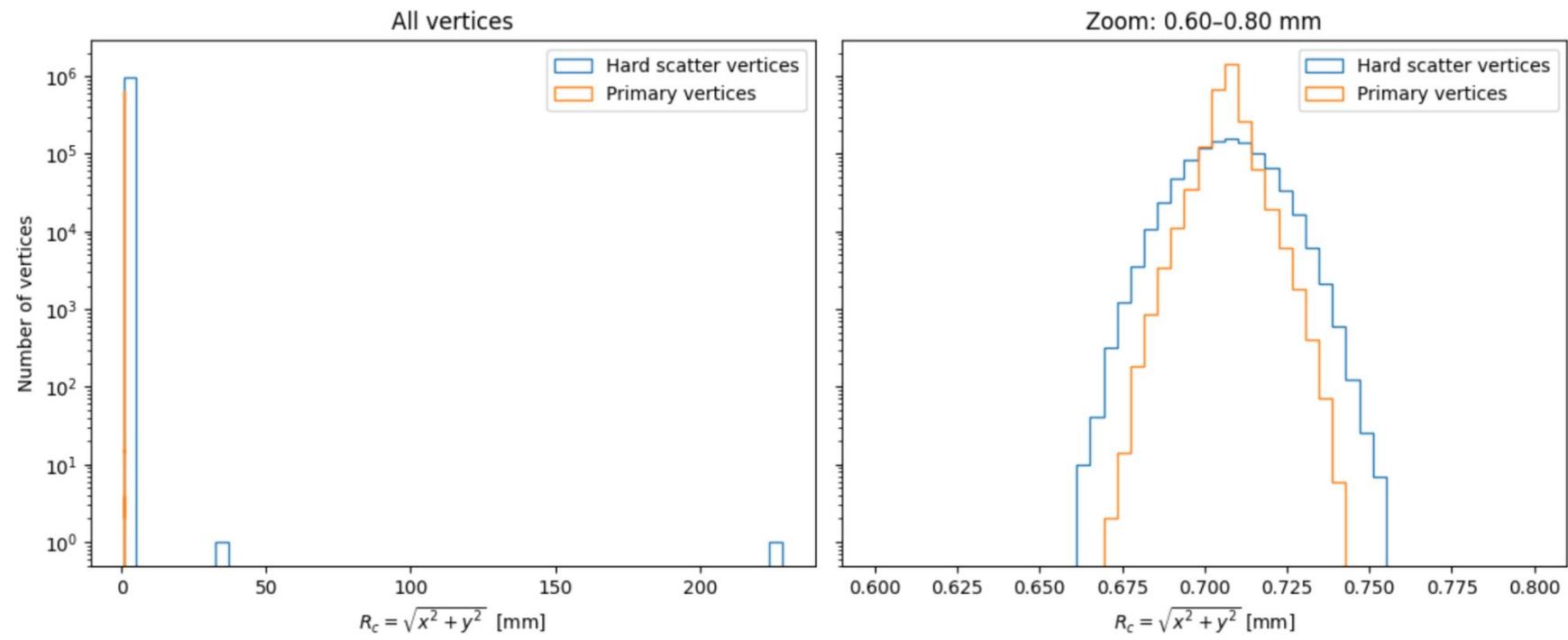
```
# Rc = sqrt(x^2 + y^2)
hs_rc = np.sqrt(hs_x**2 + hs_y**2)
pv_rc = np.sqrt(pv_x**2 + pv_y**2)
```

We calculate the distance from beam.

And we can plot!

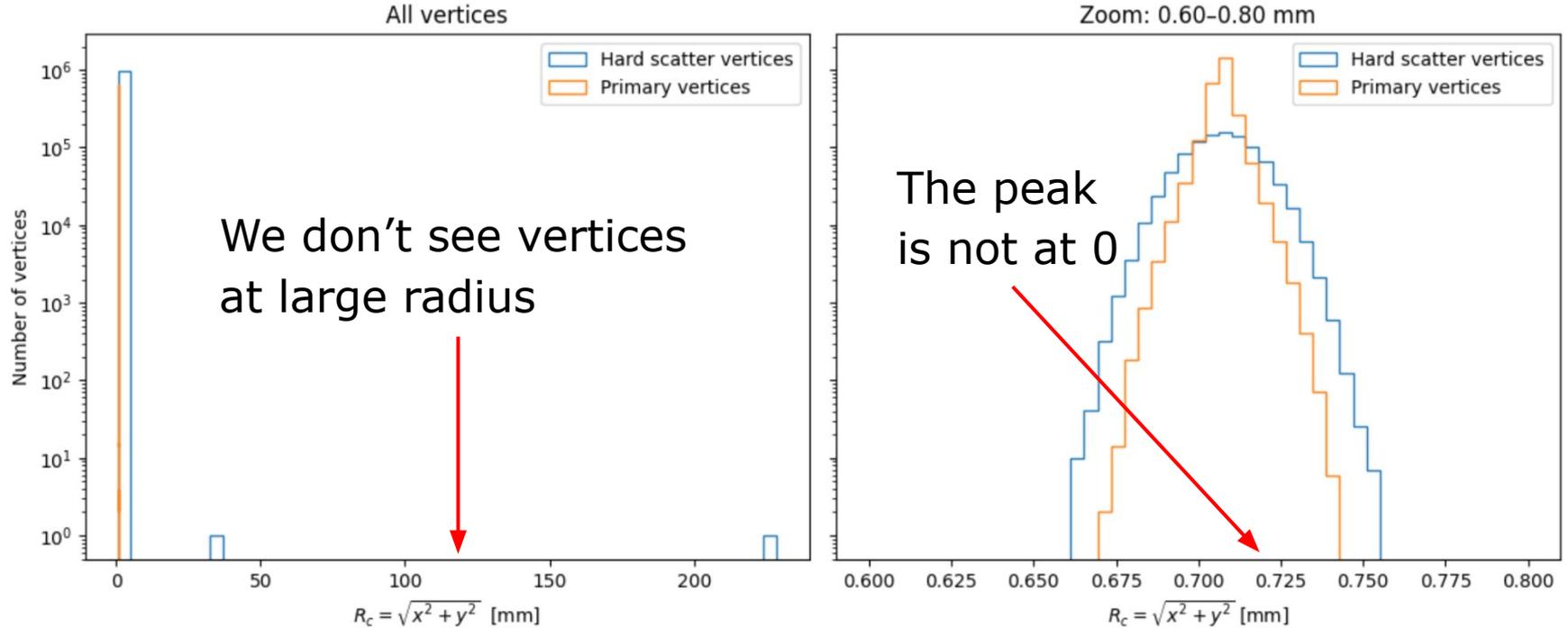
PHYSLITE format

We can also look at where the vertices that are stored actually sit in the detector.



PHYSLITE format

We can also look at where the vertices that are stored actually sit in the detector.



LLP signatures that rely on reconstructing highly displaced vertices cannot be studied.

Heavy ions collisions data

For heavy ion collisions what you can do is also **constrained by the selected data** that have been released.

The open heavy ion data includes a single Monte Carlo sample, designed to represent **minimum bias heavy ion events** in the detector. **Its mainly aimed to support generic studies**, such as checking tracking performance and efficiencies.

🏠 > The Data > Data for research > Heavy Ion Collisions > Limitations

Limitations

Although these data are all intended for use in scientific research, they are not without limitations. It is important to keep in mind the limitations of these datasets when using them in order to avoid false-positive results or dead ends. Although some of the most important limitations are documented here, if you have any questions about whether a particular

HION14 format

The HION14 data format is **constructed to support "standard" track-based analyses**. It does not contain

- All physics objects (like jets or electrons)
- All calorimeter objects or particle flow objects.

Variables in Minbias-focused Data Format for Heavy Ion Collisions

List of Containers:

[CaloSums](#) | [CombinedMuonTrackParticles](#) | [EventInfo](#) | [ExtrapolatedMuonTrackParticles](#) | [InDetTrackParticles](#) | [Muons](#) |
[MuonSpectrometerTrackParticles](#) | [MuonTruthParticles](#) | [PrimaryVertices](#) | [TruthEvents](#) | [TruthParticles](#) |

As a result, **some analyses cannot be performed**. For example:

- Analyses requiring detailed calorimeter information.
- Analyses requiring jets in any way.
- Lepton-universality checks.
- Forward-detector analyses.

HION14 format

One way to see the limitations of the HION14 format is **examining the number of primary vertices per event**.

Primary vertices can be found in the PrimaryVertices tree. We are using the vertexType variable:

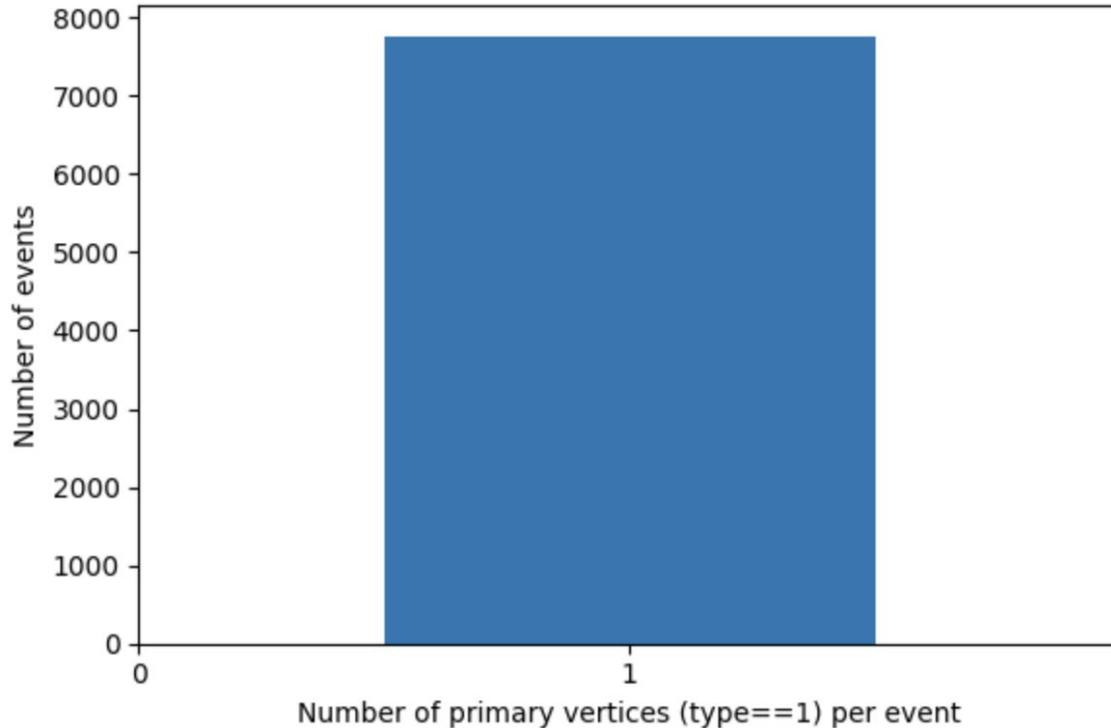
- 0: Dummy vertex (not used in vertex fit)
- 1: Primary Vertex
- 2: Secondary Vertex
- 3: Pile Up Vertex

```
[19]: # vertexType per event
      vtx_type = tree["PrimaryVerticesAuxDyn.vertexType"].array()

      # count how many vertices have type == 1 in each event (the "hard scatter" vertex)
      counts_type1 = ak.sum(vtx_type == 1, axis=1)
```

HION14 format

The HION14 format **only keeps a single hard-scatter vertex.**



- Vertex reconstruction performance cannot be studied
- Exploring rare events with multiple vertices is not possible
- Vertex multiplicity and structure as an additional observable to characterize the event or reject backgrounds is also not possible.

Event generation data

The samples are sufficient to **establish standard backgrounds and generically-applicable systematic uncertainties**. In some cases,

- Establishing key systematic uncertainties requires additional samples.
- Some signal/variation samples have not been released to reduce the total space required by the samples.

🏠 > The Data > Data for research > Event Generation Data > Limitations

Limitations

Although these data are all intended for use in scientific research, they are not without limitations. It is important to keep in mind the limitations of these datasets when using them in order to avoid false-positive results or dead ends. Although some of the most important limitations are documented here, if you have any questions about whether a particular

We encourage [reaching out](#) in case you are missing a sample.

HEPMC format

Contains all particles in the event record, **slightly modified from what the generator would normally provide**. In particular:

- Loops in the event record have been broken apart.
- The events have been checked for unexpected features (like stable gluons), and any such events have been removed. These can result from rare issues in specific event generators.

The data format is **without any detector simulation applied**. It is necessary to apply some smearing and inefficiencies before comparing them to reconstructed data. Standard programs like DELPHES should suffice for many cases.

Let's compare the output of DELPHES with MC data!

HEPMC format

We are using the dataset with ID 700793 in both event generation and the proton-proton data.

To use DELPHES we will do the same as what it is done in the event generation tutorial: download the file and run Delphes

```
[25]: %%bash -s {hepmc_file.name}  
DelphesHepMC2 ${CONDA_PREFIX}/cards/delphes_card_ATLAS.tcl delphes_output.root $1
```

```
# loop over the Delphes tree in chunks  
for arrays in tree.iterate(  
    filter_name=["Jet.PT", "Electron.Eta"],  
    step_size=1000,  
    library="ak",  
):
```

We iterate over the file we just got using Delphes. We are interested in the jet pT and electron eta.

HEPMC format

```
# iterate over all trees in chunks
for arrays in uproot.iterate(
    trees,
    filter_name=[
        "AnalysisJetsAuxDyn.pt",
        "AnalysisElectronsAuxDyn.eta",
    ],
    step_size=1000, # number of events per chunk
    library="ak",
):
```

⚠ Remember: to get the PHYSLITE file you have to change release in atlasopenmagic.

We do the same thing for the PHYSLITE file. Note the different naming in the variables.

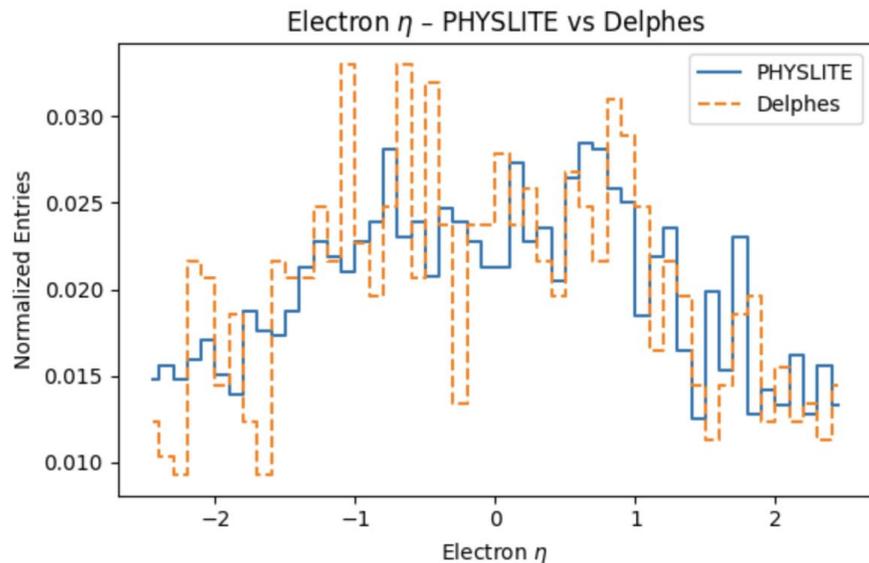
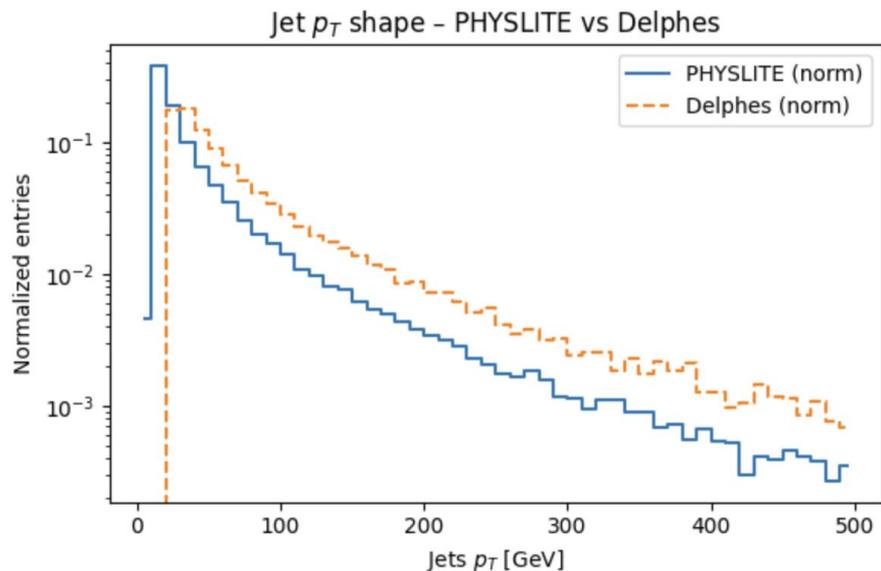
We also did this for the Delphes file



```
# flatten electrons over all events in the chunk
jet_pt = ak.to_numpy(ak.flatten(arrays["AnalysisJetsAuxDyn.pt"]))
el_eta = ak.to_numpy(ak.flatten(arrays["AnalysisElectronsAuxDyn.eta"]))
```

HEPMC format

The overall shape of the Delphes jet p_T and the electron η distribution is similar to the one from the MC simulation \rightarrow Fast simulation cannot exactly reproduce the behaviour of the reconstructed objects.



Summary

The research Open Data are designed to serve the **most general analysis needs**. Some analyses will be constrained by:

- Limited sample availability, and/or
- Missing information in the data formats.

We usually focus on **what you can do** with the releases, but it's important to check feasibility before committing to a project.

If you have questions:

- Start with the documentation in the [website](#).
- If it's not answered there, [reach out to us](#), especially if you're unsure whether something is possible or if you think you need more samples.

For more involvement and access all ATLAS samples, consider a [short-term association](#).