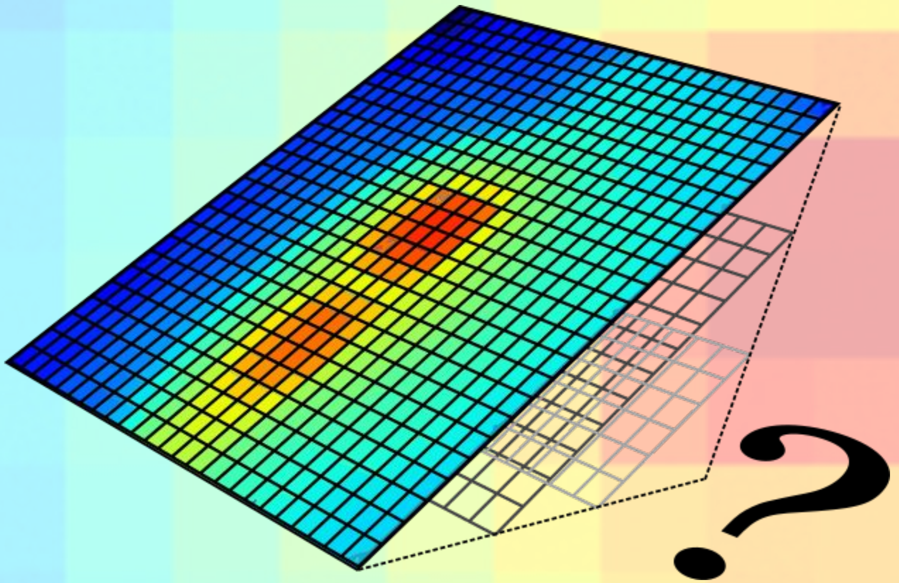


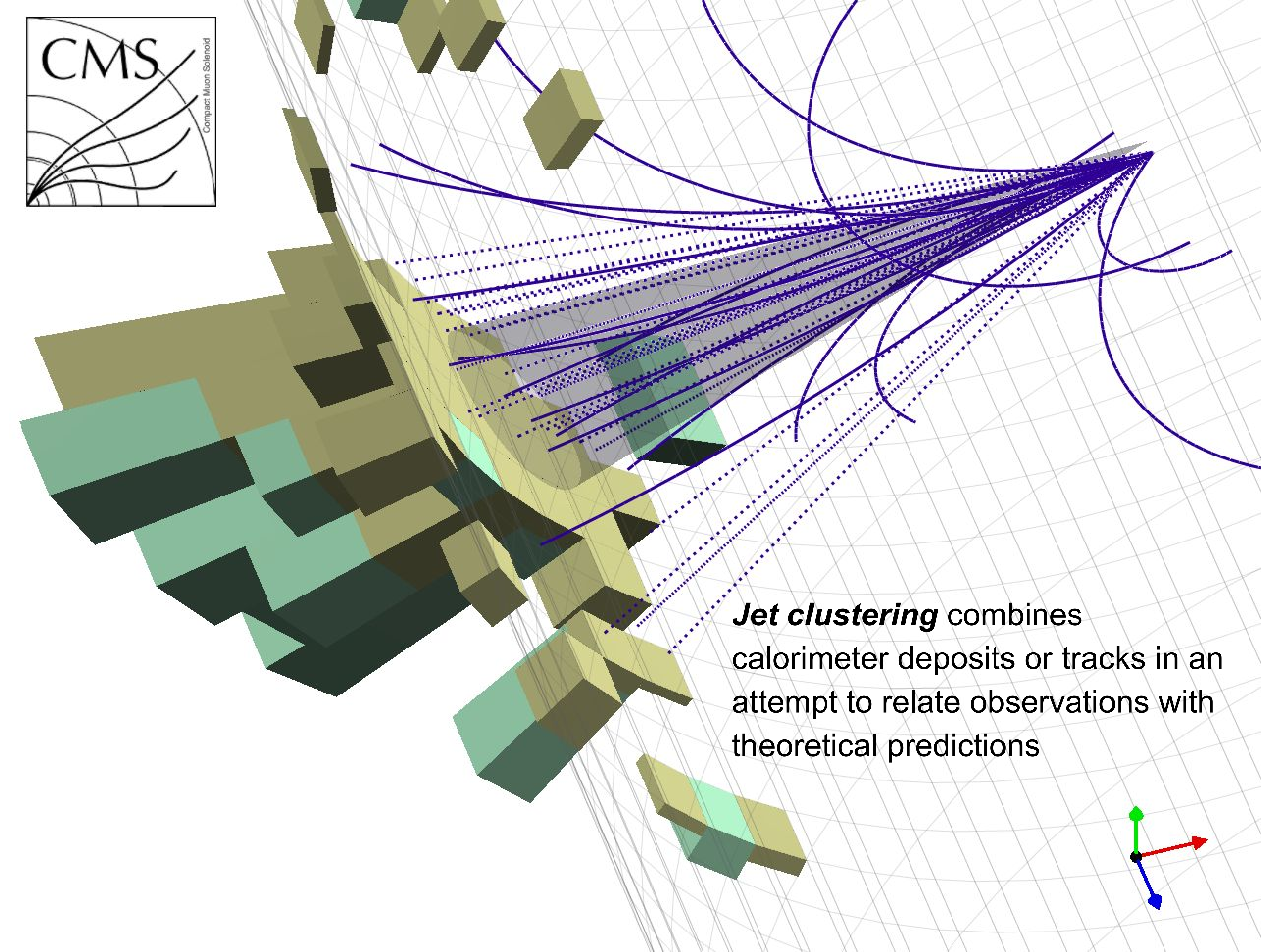
Deep Learning Jet Images

Noel Dawe

MLHEP 2017
Reading, UK

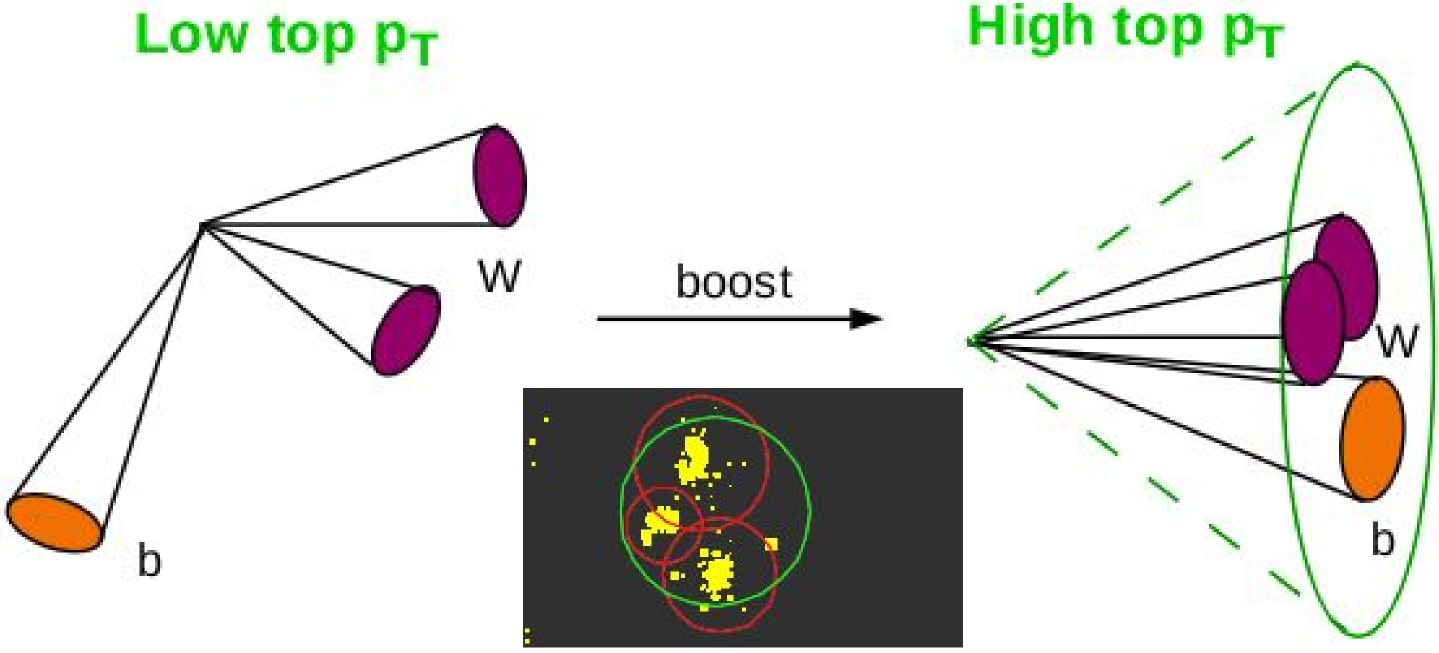
THE UNIVERSITY OF MELBOURNE

CoEPP
ARC Centre of Excellence for Particle Physics at the Terascale

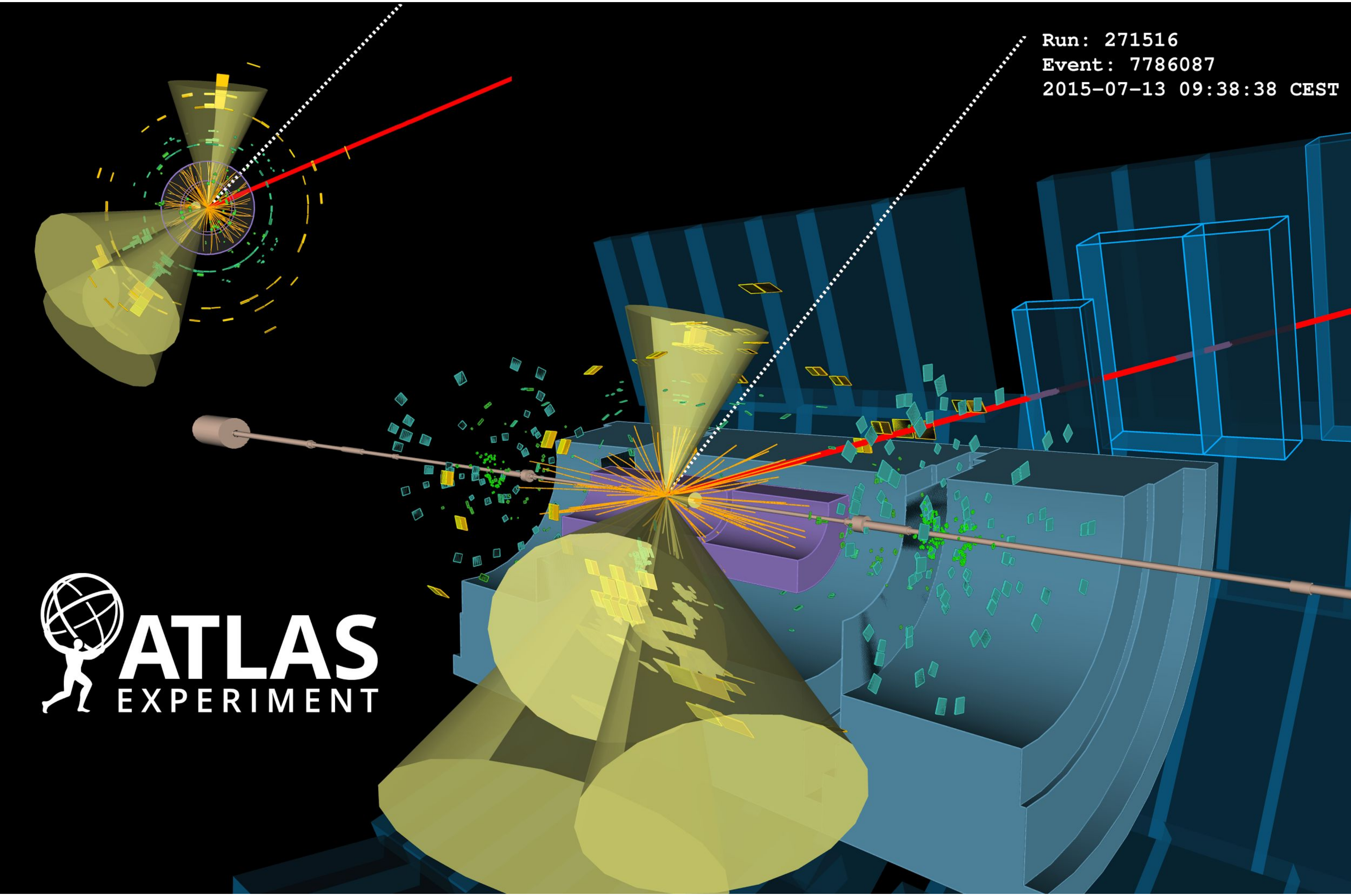


Jet clustering combines calorimeter deposits or tracks in an attempt to relate observations with theoretical predictions

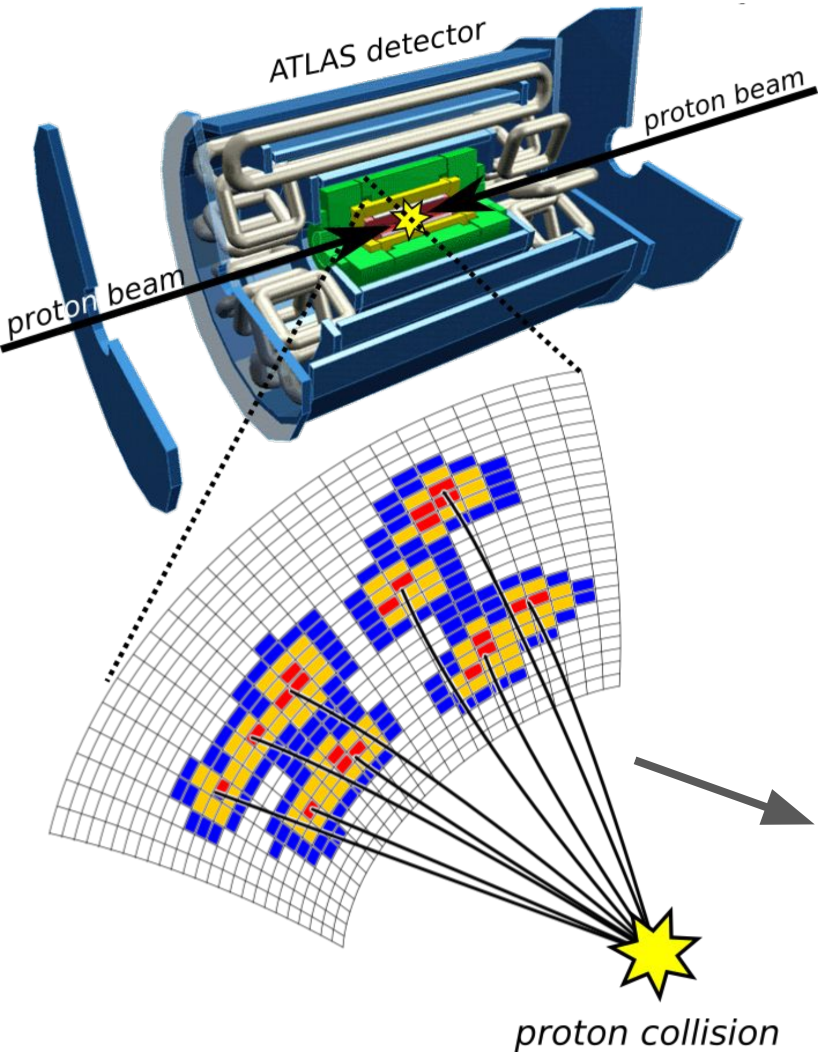
Jet Substructure



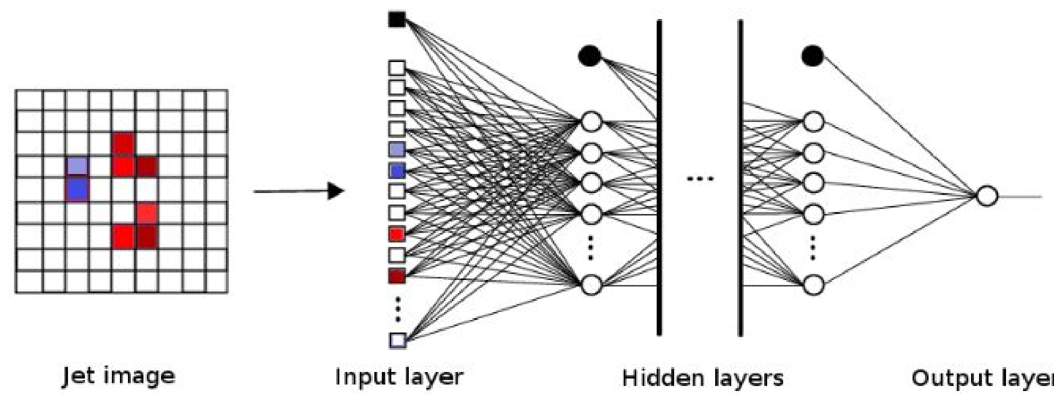
The energy distribution within the jet (substructure) can reveal information about the process that initiated the jet

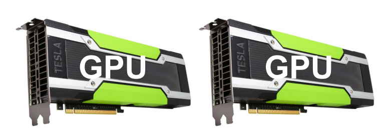

Run: 271516
Event: 7786087
2015-07-13 09:38:38 CEST
ATLAS
EXPERIMENT

Machine Learning Jet Substructure



Apply **deep neural networks** common in computer vision applications to distinguish different sources of jets using ***"jet images"***

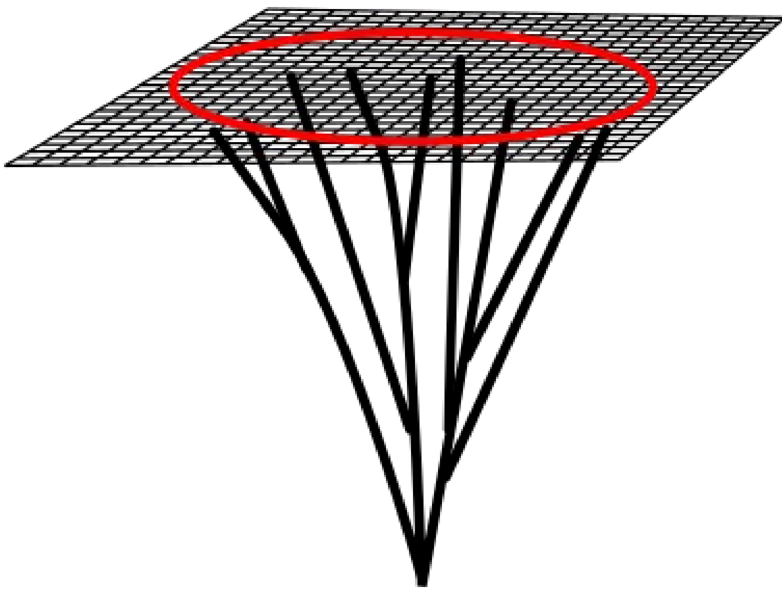




Flattening the calorimeter into a 2D image...

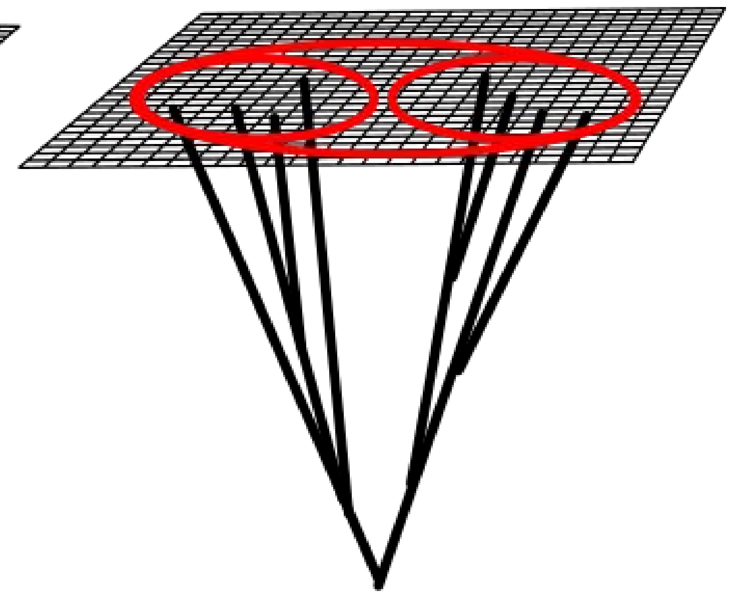
Challenge: Boosted hadronic W decays vs QCD jets

Background



QCD jet
quark/gluon

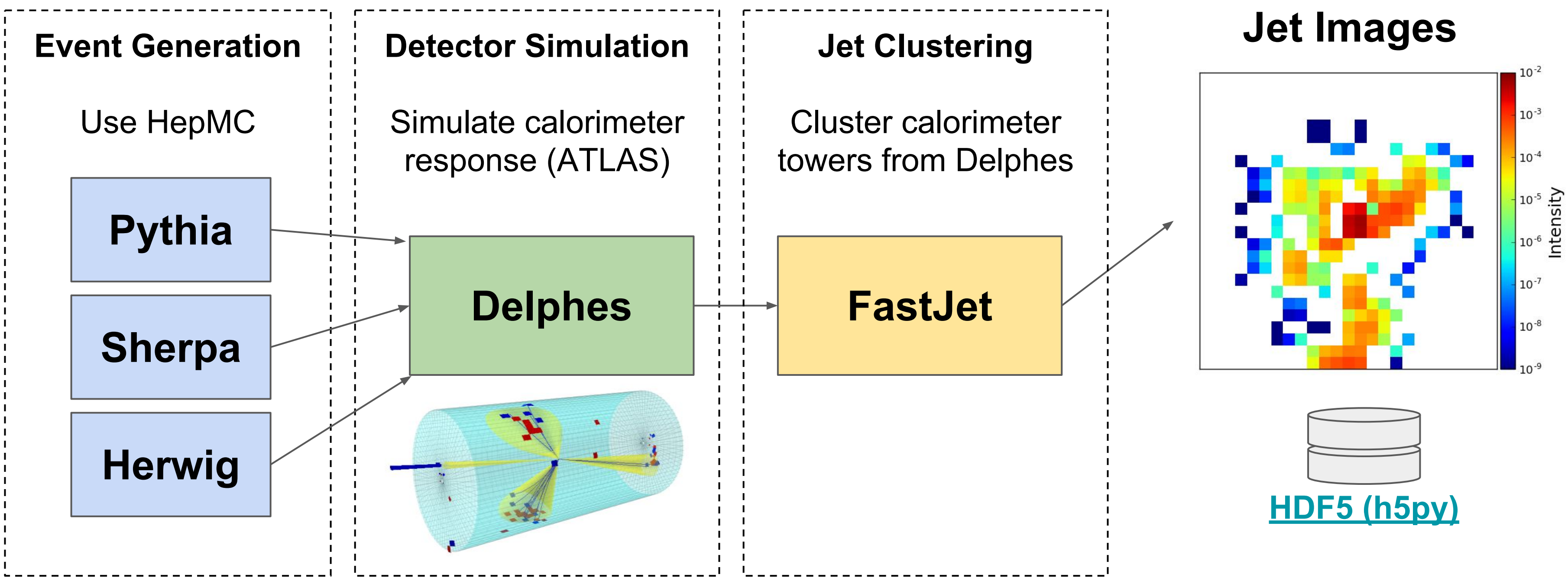
Signal



W jet

Two subjets with separation scaling as $\boldsymbol{2\, m_W / p_T}$

Creating Jet Image Data



Each stage is a Python generator function that yields a numpy array

Jet images can be produced and used "on-the-fly" or saved to disk for later use

Heavy use of Cython for interfacing NumPy and the above software

See the code: https://github.com/deepjets/deepjets

numpythia: Interfacing NumPy & PYTHIA

https://github.com/ndawe/numpythia

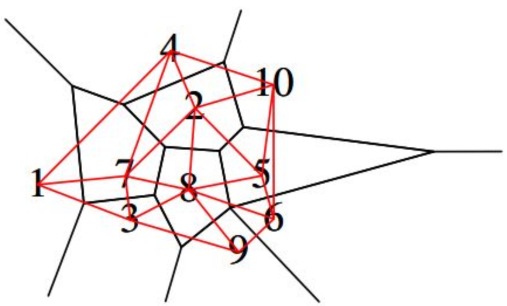
pip install --user -v numpythia

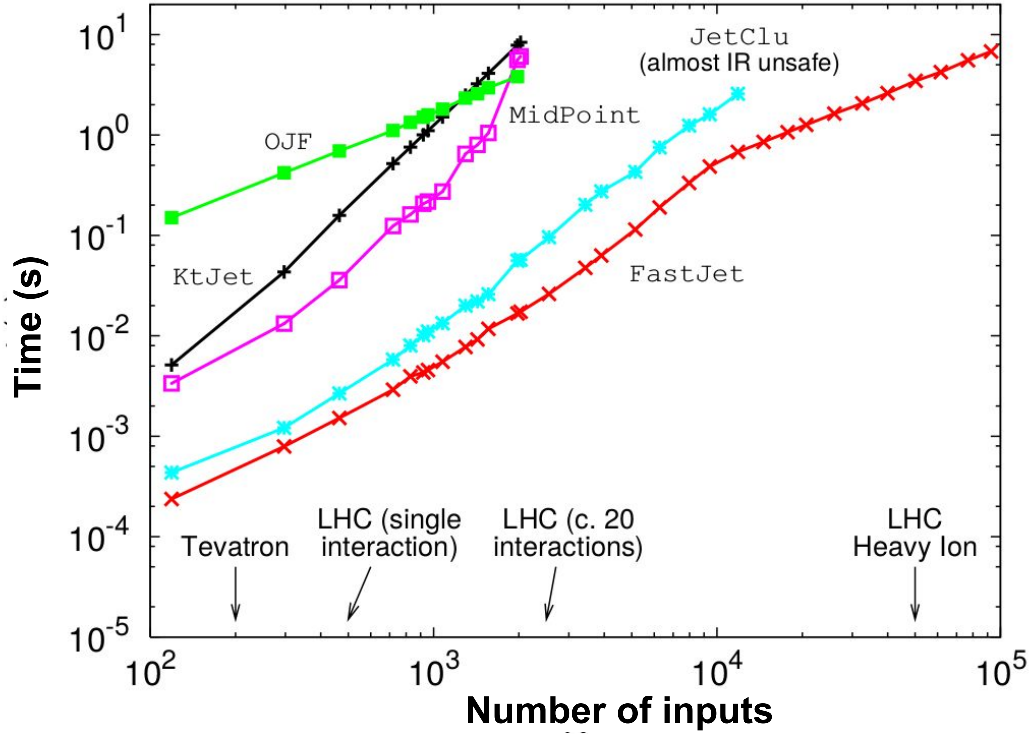
Only depends on NumPy. Latest PYTHIA and HepMC included.

```
from numpythia import Pythia, hepmc_write
from numpythia import STATUS, HAS_END_VERTEX, ABS_PDG_ID
from numpythia.testcmnd import get_cmnd

pythia = Pythia(get_cmnd('w'), random_state=1)
selection = ((STATUS == 1) & ~HAS_END_VERTEX &
             (ABS_PDG_ID != 12) & (ABS_PDG_ID != 14) & (ABS_PDG_ID != 16))

# generate events while writing to ascii hepmc
for event in hepmc_write('events.hepmc', pythia(events=1)):
   # get visible final state particles as a numpy array
   array = event.all(selection)
```

Jet Clustering with FastJet

- C++ library implementing all widely used jet algorithms.
- Huge performance improvement over previous implementations O(N ln(N))





http://fastjet.fr/
https://arxiv.org/abs/hep-ph/0512210

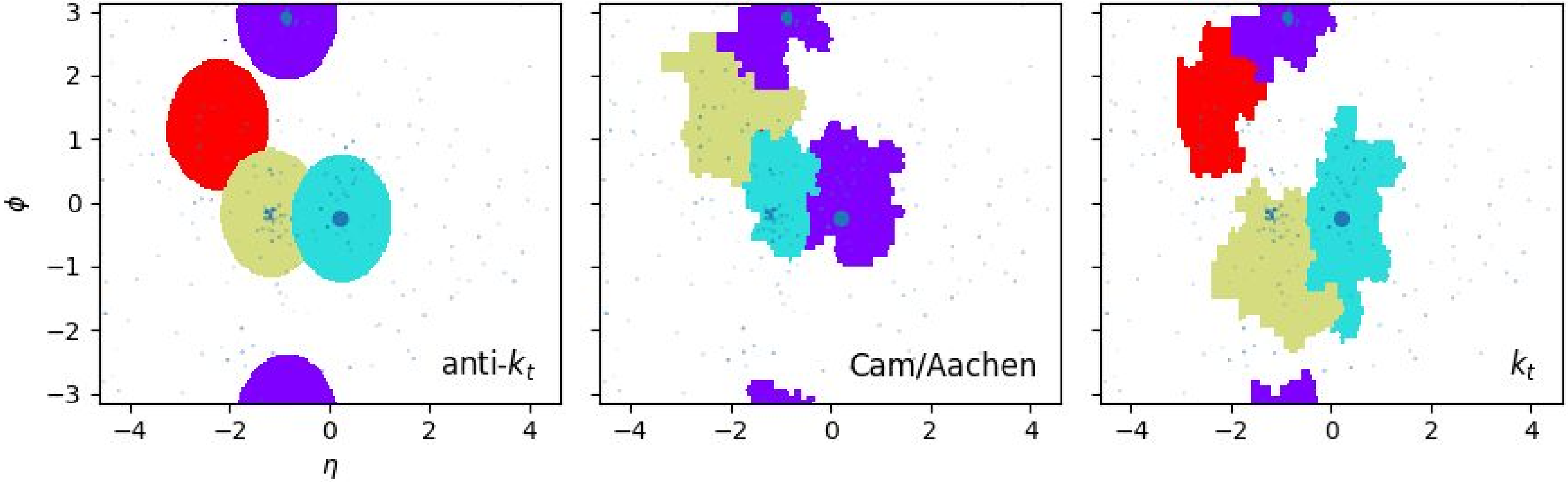
pyjet: Interfacing NumPy & FastJet

https://github.com/ndawe/pyjet

pip install --user pyjet

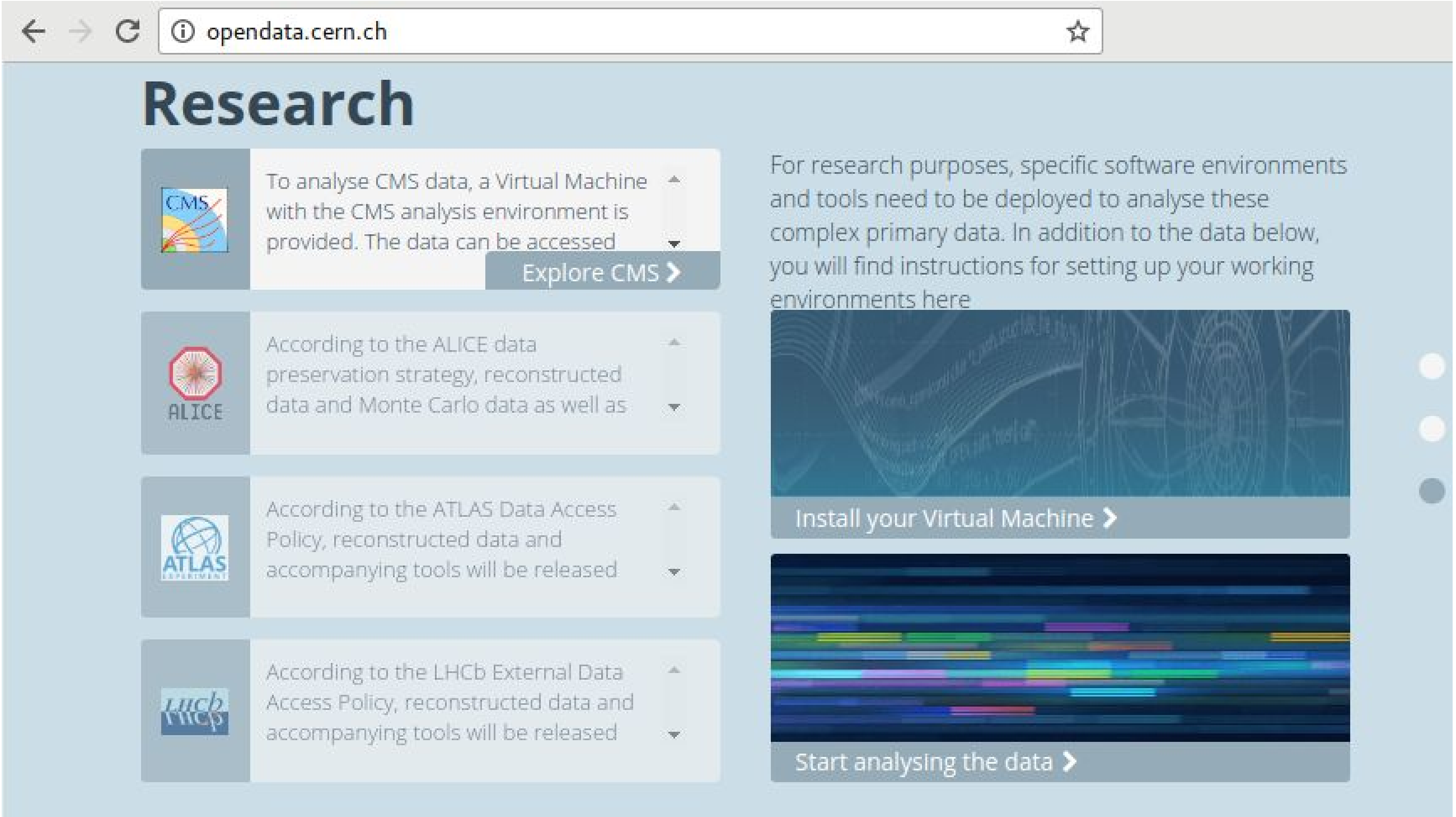
Only depends on NumPy. The standalone FastJet source is included.

```
from pyjet import cluster
from pyjet.testdata import get_event
vectors = get_event() # example numpy array of four-momenta
sequence = cluster(vectors, R=1.0, p=-1)
jets = sequence.inclusive_jets()  # list of PseudoJets
```



What other use is there for jet clustering with numpy arrays?

opendata.cern.ch



Jet clustering CMS data *without CMSSW*

```
from pyjet import cluster
from root_numpy import root2array, stretch

branches=[
   'recoPFCandidates_particleFlow__RECO.obj.pt_',
   'recoPFCandidates_particleFlow__RECO.obj.eta_',
   'recoPFCandidates_particleFlow__RECO.obj.phi_',
   'recoPFCandidates_particleFlow__RECO.obj.mass_',
]
filename = ("root://eospublic.cern.ch//eos/opendata/cms/Run2011A/DoubleMu/AOD/"
            "12Oct2013-v1/10000/000D143E-9535-E311-B88B-002618943934.root")
events = root2array(filename, "Events", branches=branches, stop=1)  # one event

for event in events:
   flattened_event = stretch(event.reshape(-1))
   sequence = cluster(flattened_event, R=0.5, p=-1)
   jets = sequence.inclusive_jets(ptmin=3)  # you get the same jets as CMS!
```

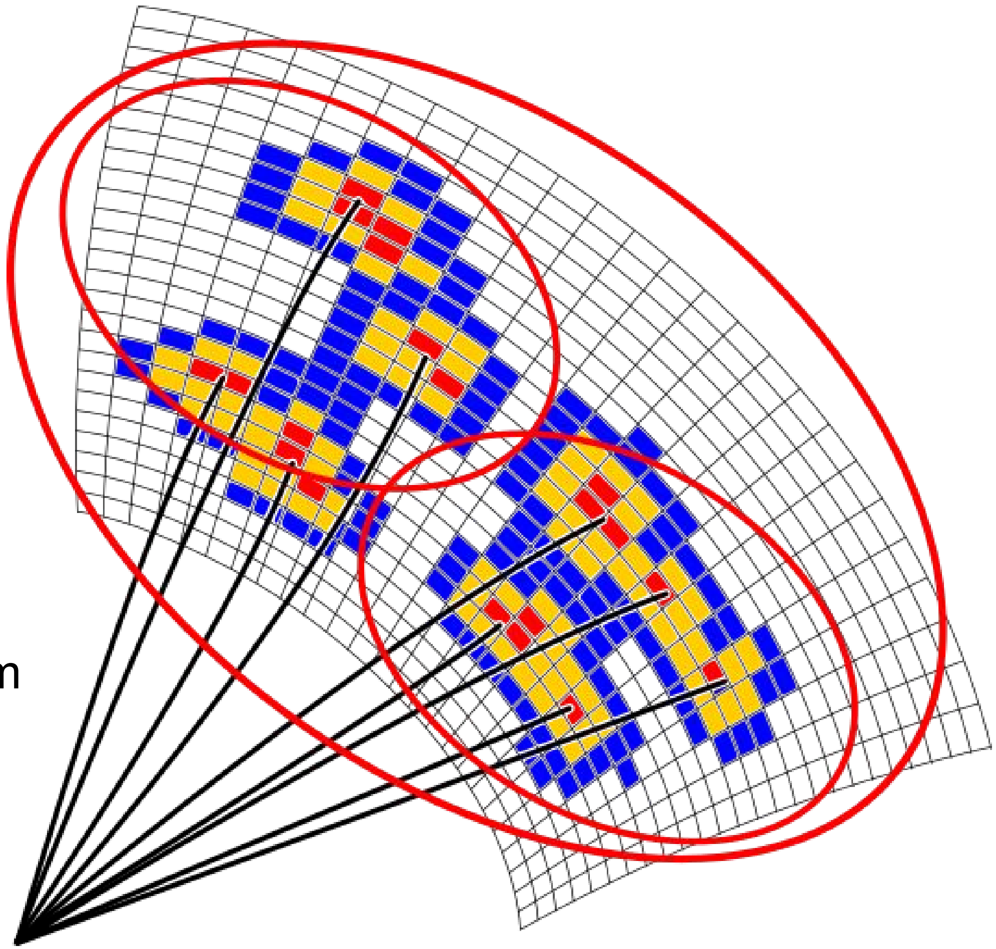
Constructing Jet Images

1. Cluster Delphes towers with **anti-k_T R = 1.0 and take highest p_T jet**

2. Run k_T clustering with **R = 0.3** on the jet's constituents to construct ***subjets***

3. Discard all subjets with less than 5% of the original jet momentum

4. Define the ***trimmed jet*** to be the sum of the remaining subjets

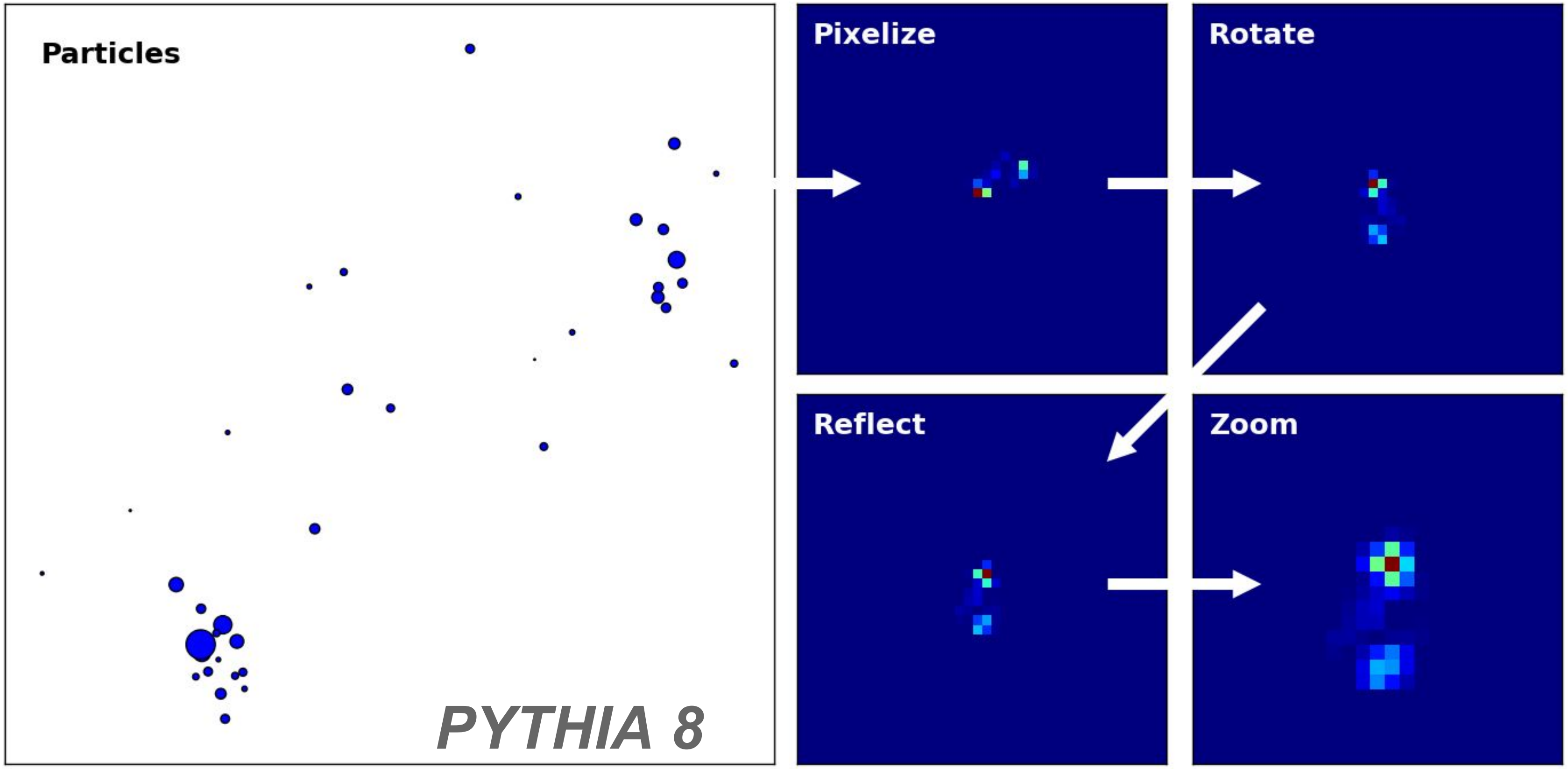


Jet images will only contain constituents of the trimmed jet

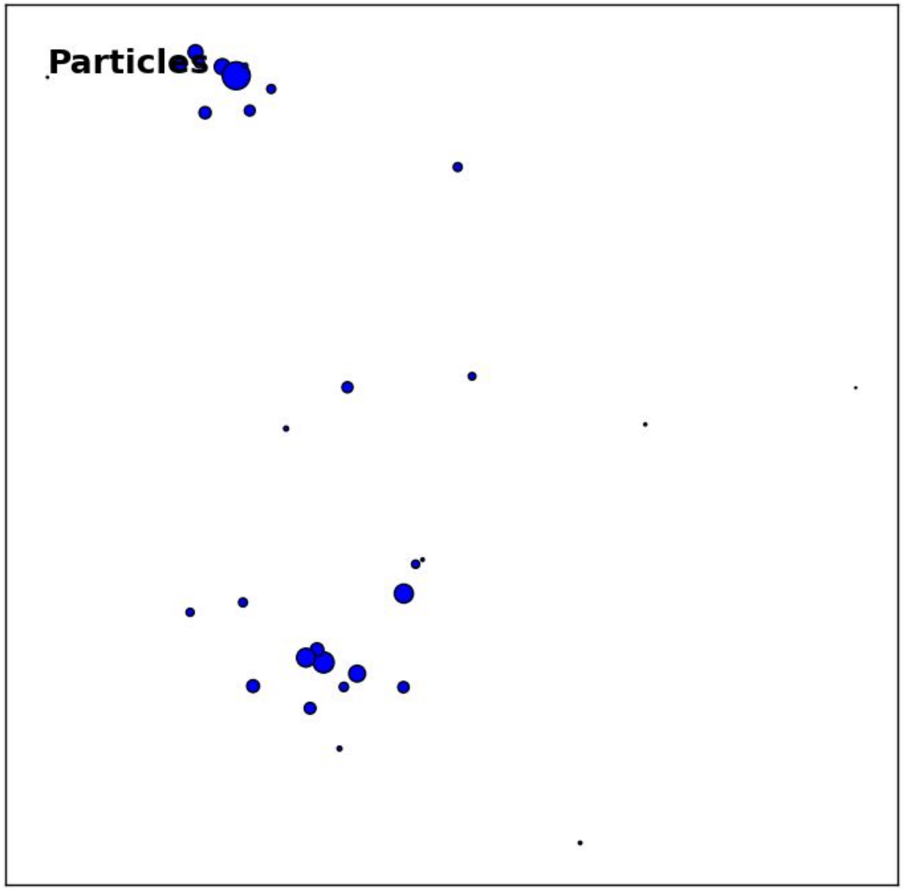
Use events with 250 < p_T < 300 GeV and 50 < m < 110 GeV

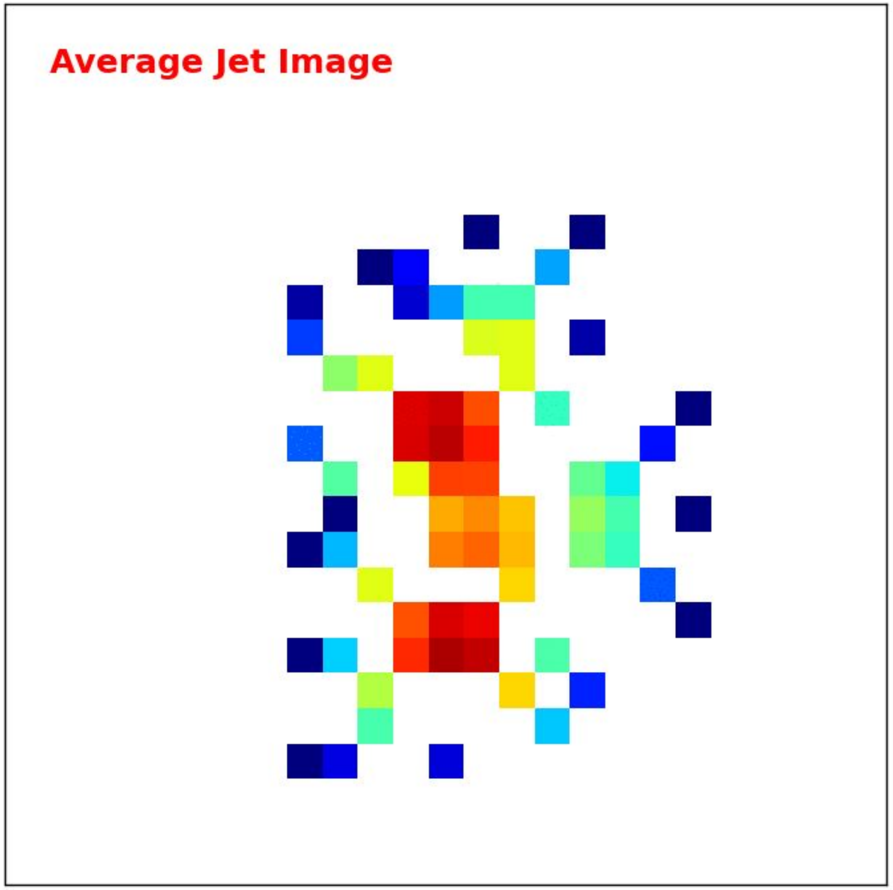
Constructing Jet Images

- Sum transverse energy of calorimeter towers in grid of 0.1 x 0.1 in η-φ space
- Perform translations, rotations and reflections in η-φ space
- Zoom the image to minimise p_T dependence
- Crop at 25 x 25 pixels and normalise



Animating the average hadronic W jet image





I just created 10TB of jet images in HDF5 files

How do I efficiently handle this data?

I can train a network in small batches

so I don’t need all data in RAM for that…

But how do I compute various statistics on the jet images?

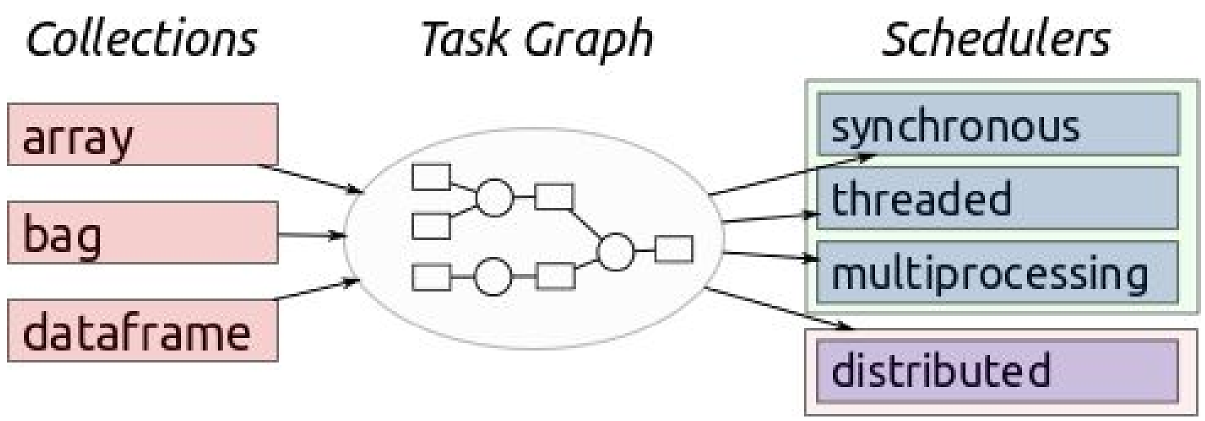
Introducing Dask

Dynamic task scheduling with "Big Data" collections dask.pydata.org

pip install --user dask

“How does Dask compare with Spark?”

Dask is lightweight, integrates nicely with the Python ecosystem,
and is well-suited for a single machine with many cores or a small cluster



This:

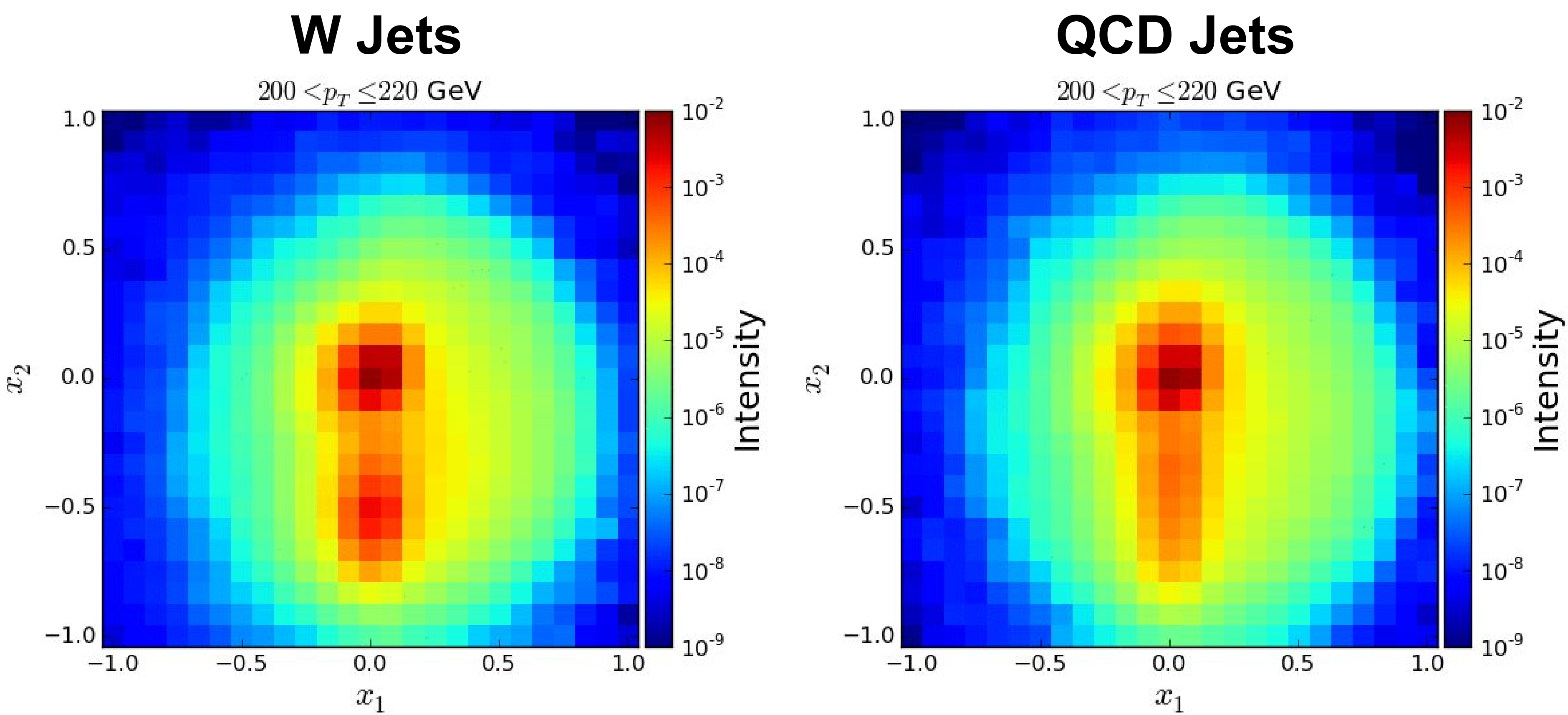
```python
import numpy as np
f = h5py.File('myfile.hdf5')
x = np.array(f['/small-data'])
x - x.mean(axis=1)
```

Becomes this:

```python
import dask.array as da
f = h5py.File('myfile.hdf5')
x = da.from_array(f['/big-data'], chunks=(1000, 1000))
x - x.mean(axis=1).compute()
```

```
da.tensordot(images, w, axes=(0, 0)).compute() / w.sum())
```

(Images are weighted such that the p_T distribution is flat)



Images zoomed by: p_T / 2 m_W

Building Deep Networks

Building networks with Keras is super easy:

```
model = Sequential()
model.add(MaxoutDense(256, input_shape=(625,), nb_feature=5))
model.add(MaxoutDense(128, nb_feature=5))
model.add(Dense(64))
model.add(Activation('relu'))
model.add(Dense(25))
model.add(Activation('relu'))
model.add(Dense(1))
model.add(Activation('sigmoid'))
model.compile(loss='binary_crossentropy', optimizer=Adam)
```

Note: MaxoutDense is deprecated in Keras 2.0
Use Dense and layers.merge.Maximum instead

Keras can use TensorFlow or Theano. Runs on CPU or GPU:

```
import theano.sandbox.cuda
theano.sandbox.cuda.use('gpu0')
```

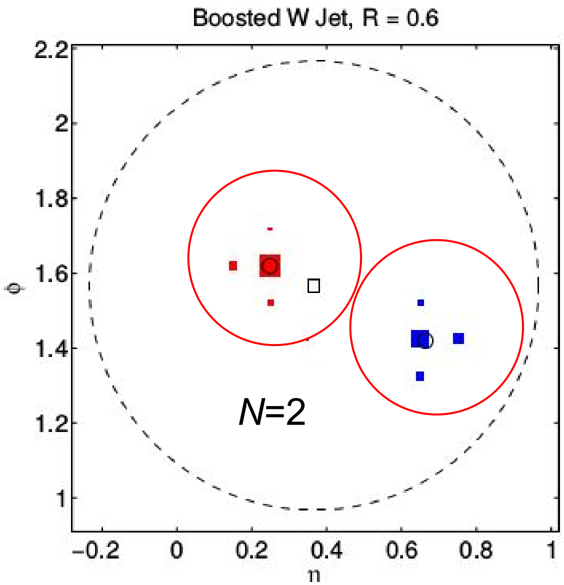
gpu0

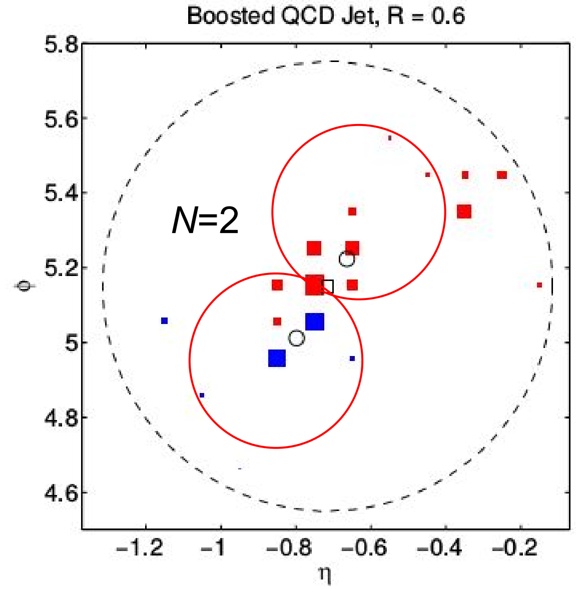
gpu1

...

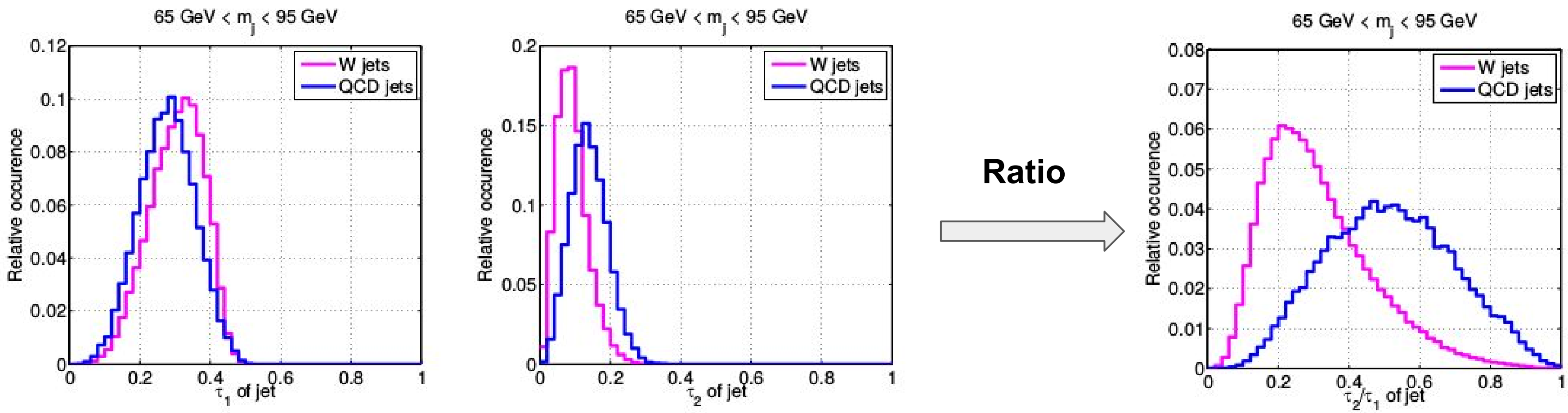
An NVIDIA Tesla K80 trained our network in ***6 minutes*** (6 hours on CPU) on 3M signal and 3M background images in batches of 100 up to 100 epochs

Benchmark: N-Subjettiness

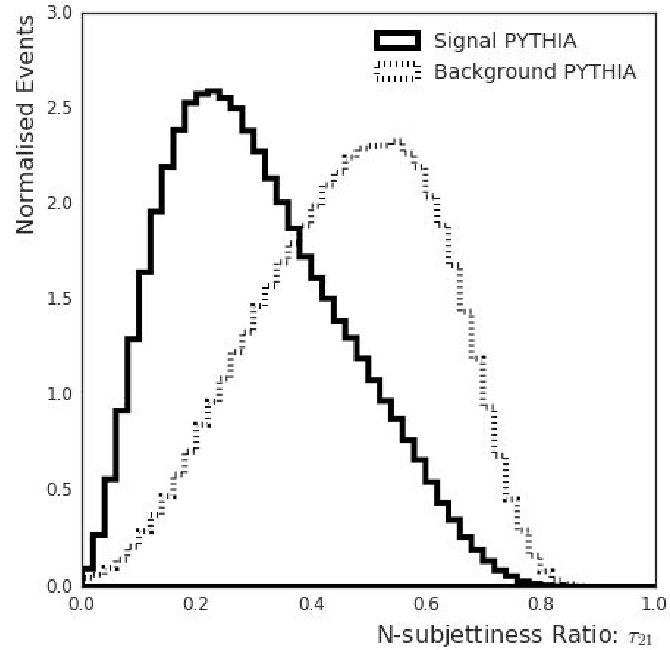


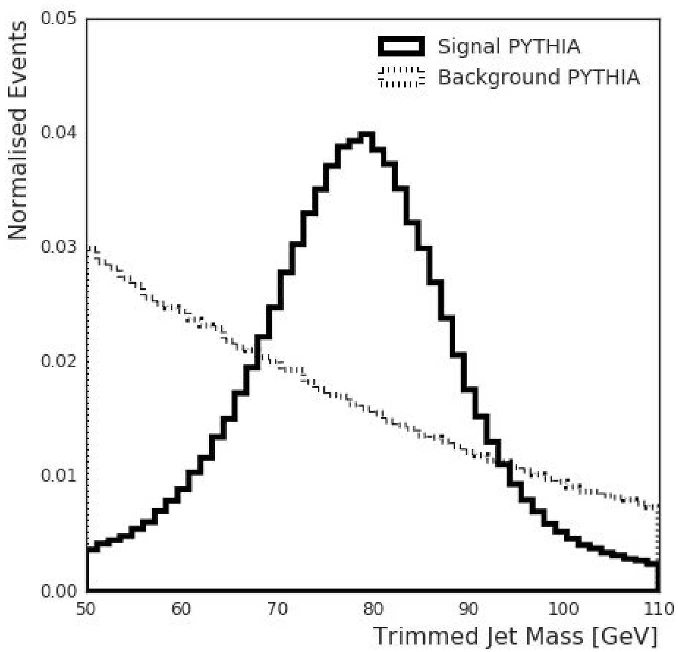


$$\tau_N = \frac{1}{d_0} \sum_k p_{T,k} \min\{\Delta R_{1,k}, \Delta R_{2,k}, \cdots, \Delta R_{N,k}\} \qquad d_0 = \sum_k p_{T,k} R_0,$$

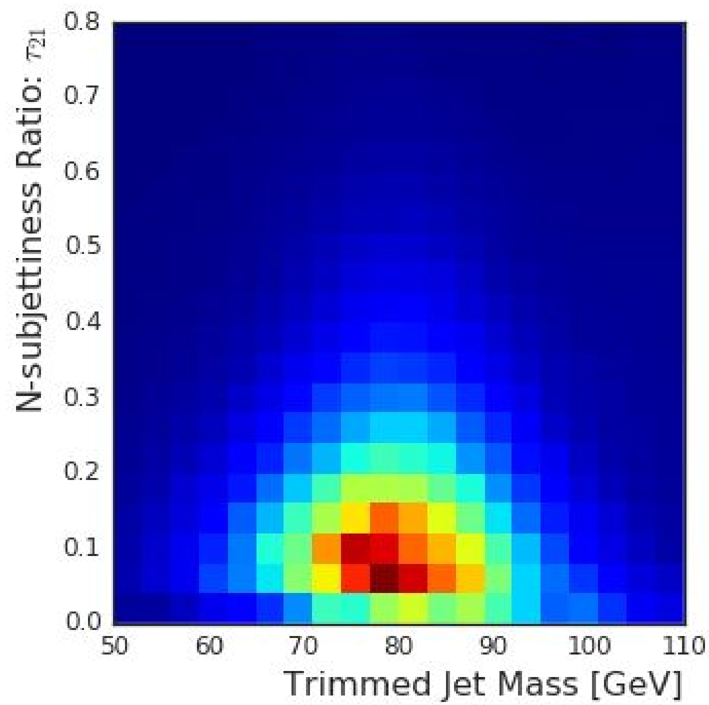


http://arxiv.org/abs/1011.2268v3 Jesse Thaler, Ken Van Tilburg

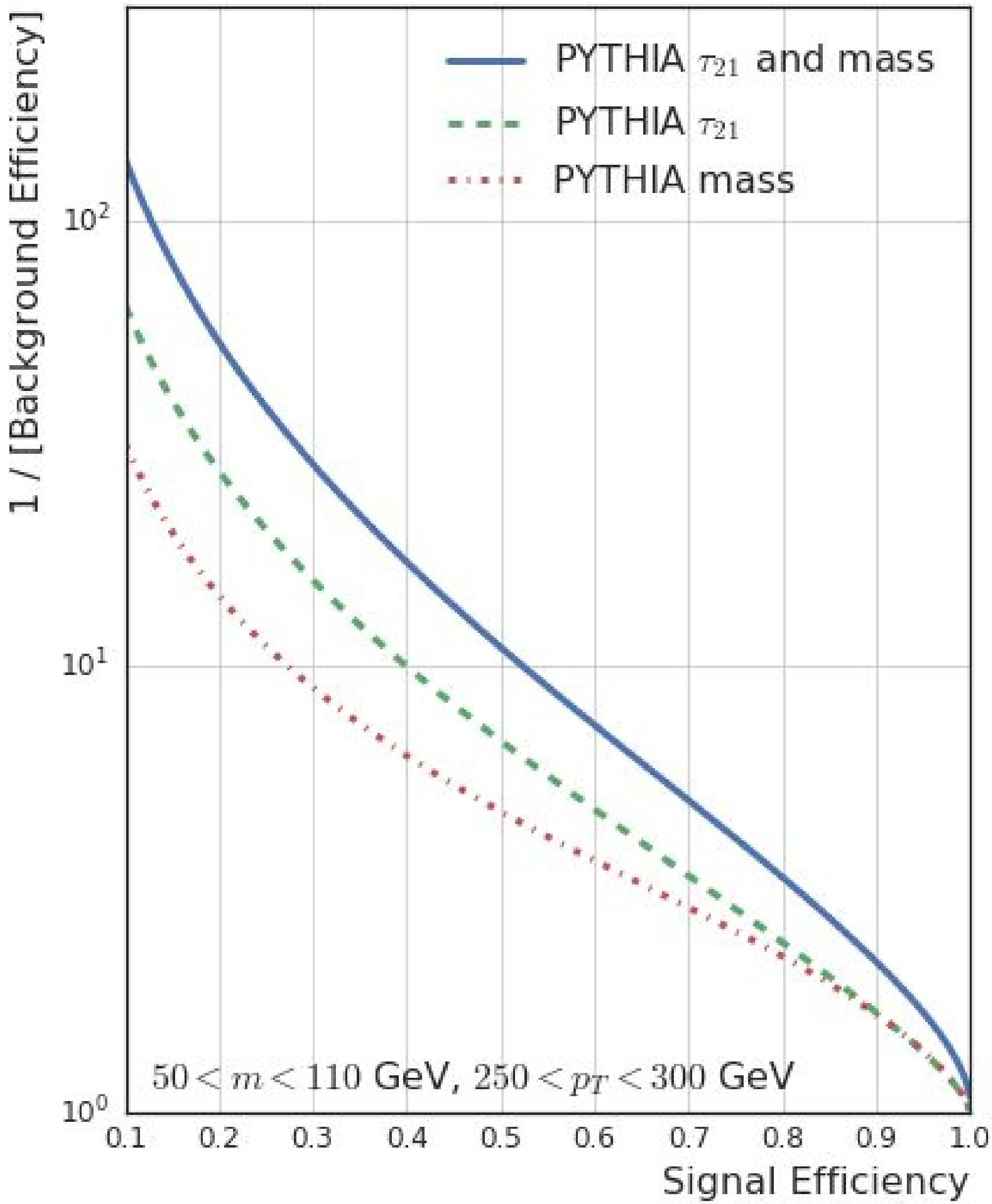




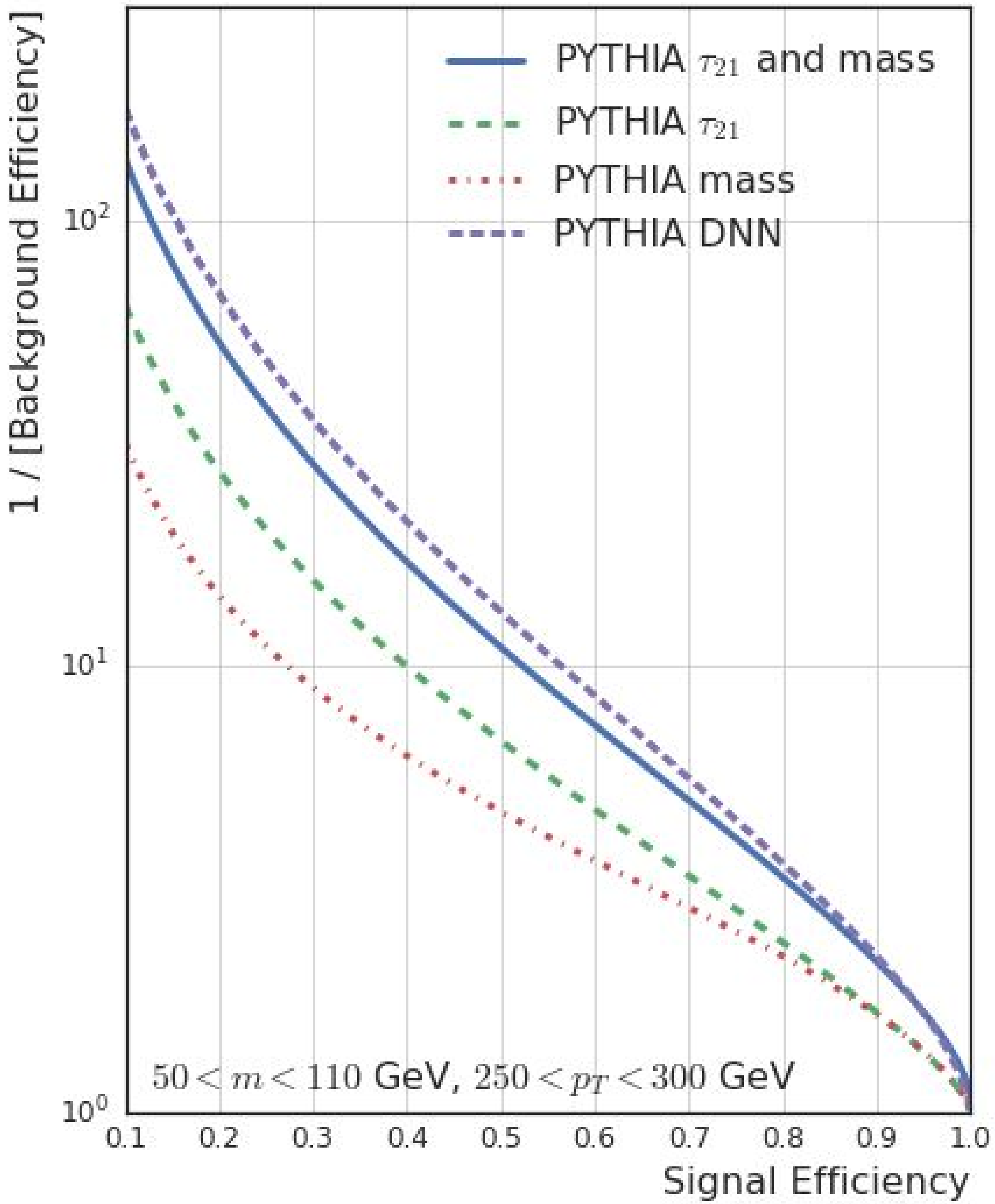
Construct a 2D likelihood ratio



… and compute a ROC curve



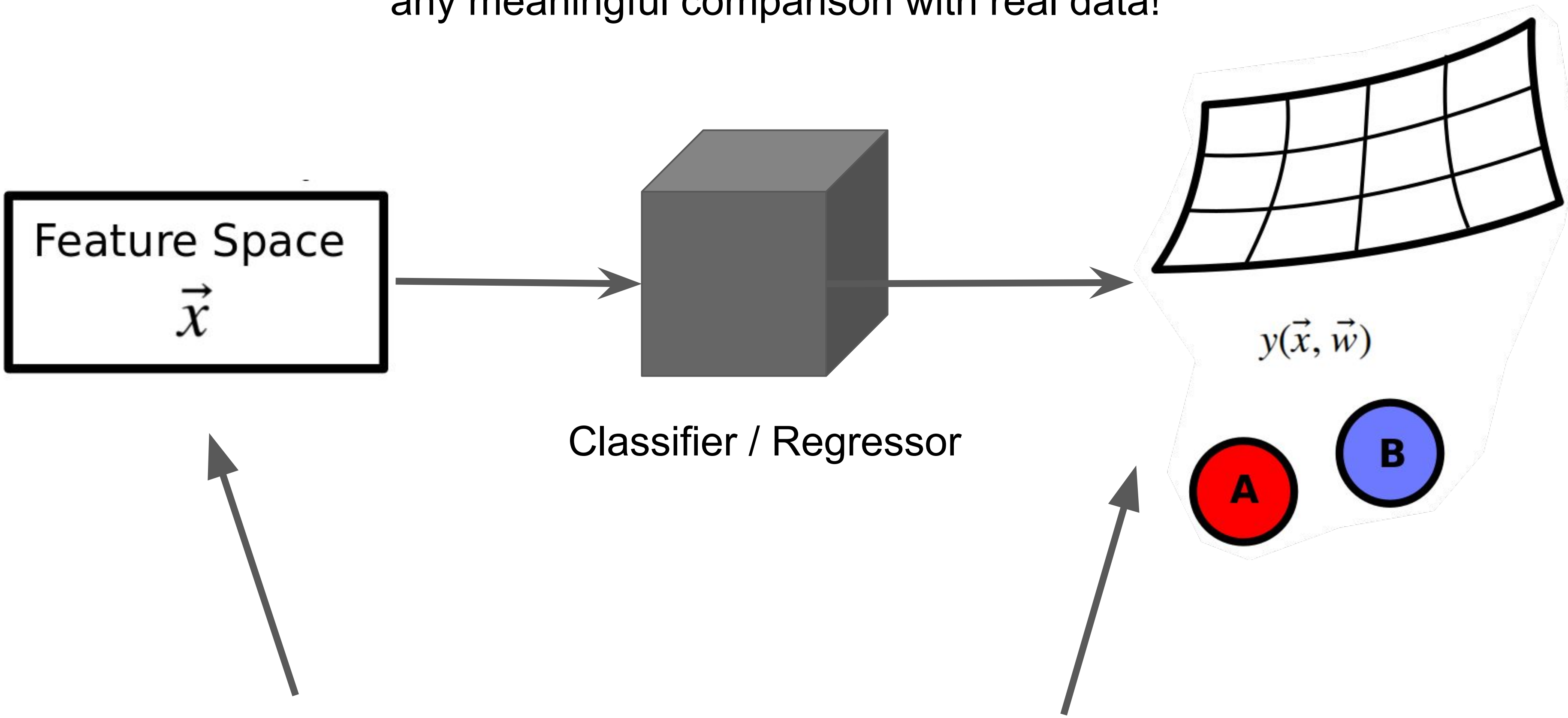
PYTHIA τ_{21} and mass
PYTHIA τ_{21}
PYTHIA mass
$50 < m < 110$ GeV, $250 < p_T < 300$ GeV
1 / [Background Efficiency]
10^2
10^1
10^0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0
Signal Efficiency



PYTHIA τ_{21} and mass
PYTHIA τ_{21}
PYTHIA mass
PYTHIA DNN
$50 < m < 110$ GeV, $250 < p_T < 300$ GeV
1 / [Background Efficiency]
10^2
10^1
10^0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0
Signal Efficiency

What about systematic uncertainties?

We need an estimate of systematic uncertainties to make any meaningful comparison with real data!



Uncertainties on the input features lead to uncertainties on the output

Comparing Generators

What uncertainties/differences are present across generators?

- **Numerical:** computational precision and statistical convergence
- **Parametric:** external to the MC generator: masses, couplings, PDFs
- **Perturbative:** truncation of expansion series
- **Phenomenological:** parameters deriving from non-perturbative models
- ***Algorithmic: the parton shower algorithm***

We focus on ***algorithmic*** differences by comparing **Pythia 8**, **Sherpa 2**, and **Herwig 7**

Comparing Generators: Parton Shower Algorithm

We focus on ***algorithmic*** differences by comparing **Pythia 8**, **Sherpa 2**, and **Herwig 7**

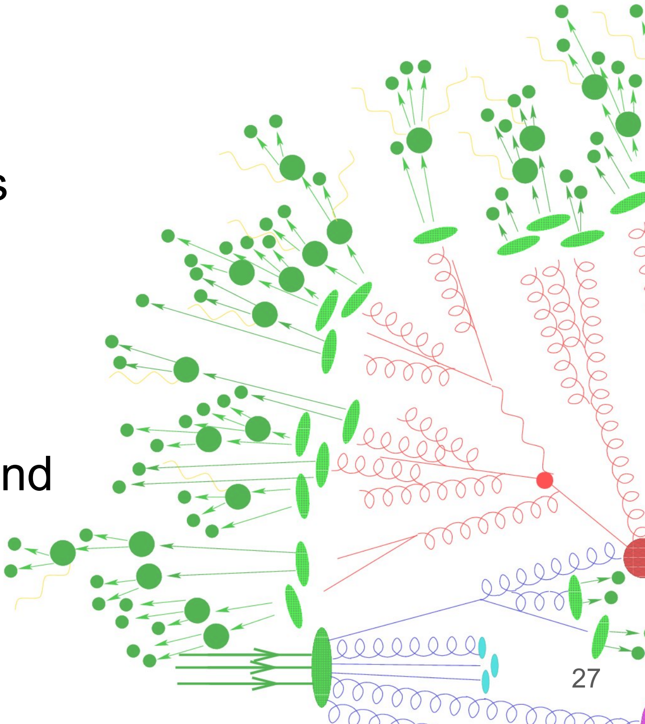
- Some algorithms consider 1→2 splittings with angular or p_T ordering in the shower evolution

  Herwig angular and dipole showers
  Sherpa and Pythia (dipole)

- Other algorithms consider colour-connected partons that undergo 2→3 branchings (antenna showers)

  Pythia's new VINCIA shower plugin

- *Also:* different soft radiation from underlying event and parton-to-hadron fragmentation



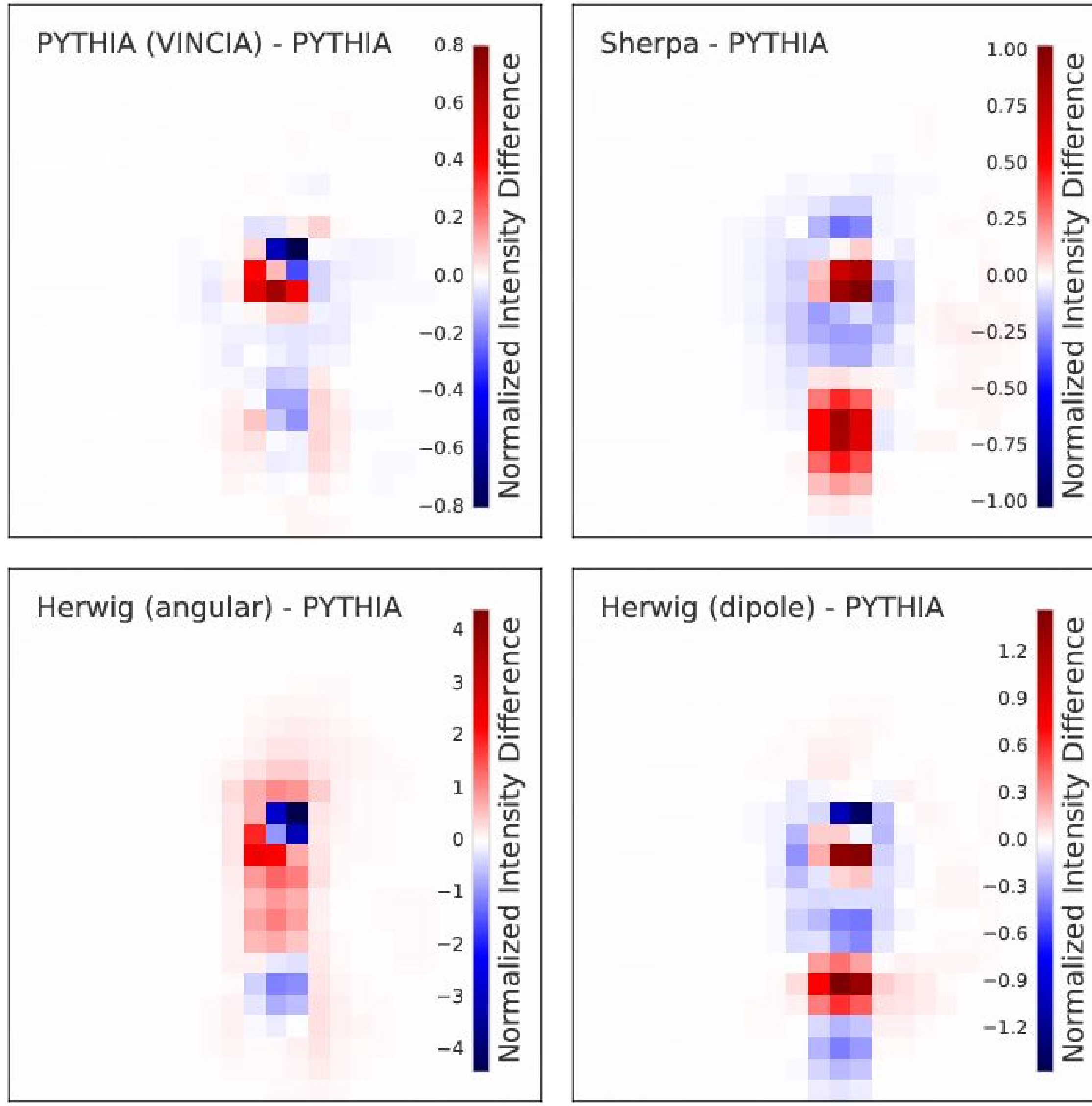
Comparing Generators: Jet Image Differences

1. Normalise Pythia, Sherpa, Herwig images

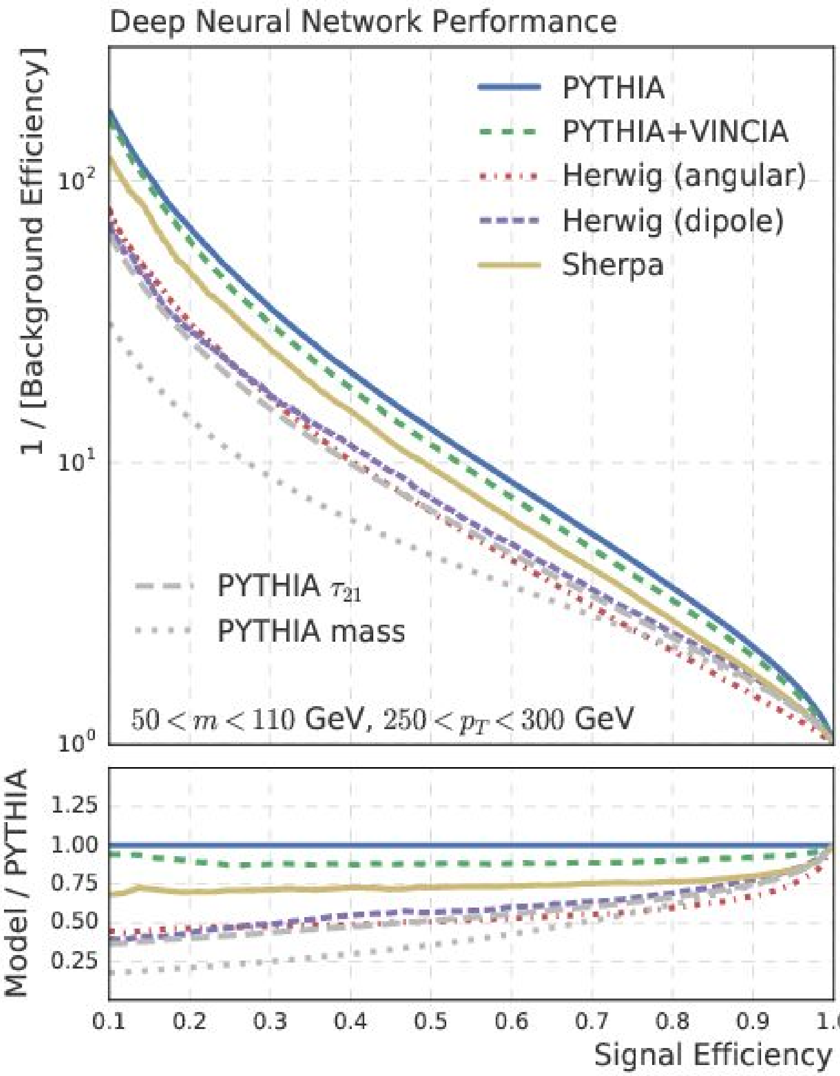
2. Subtract Pythia from each other model

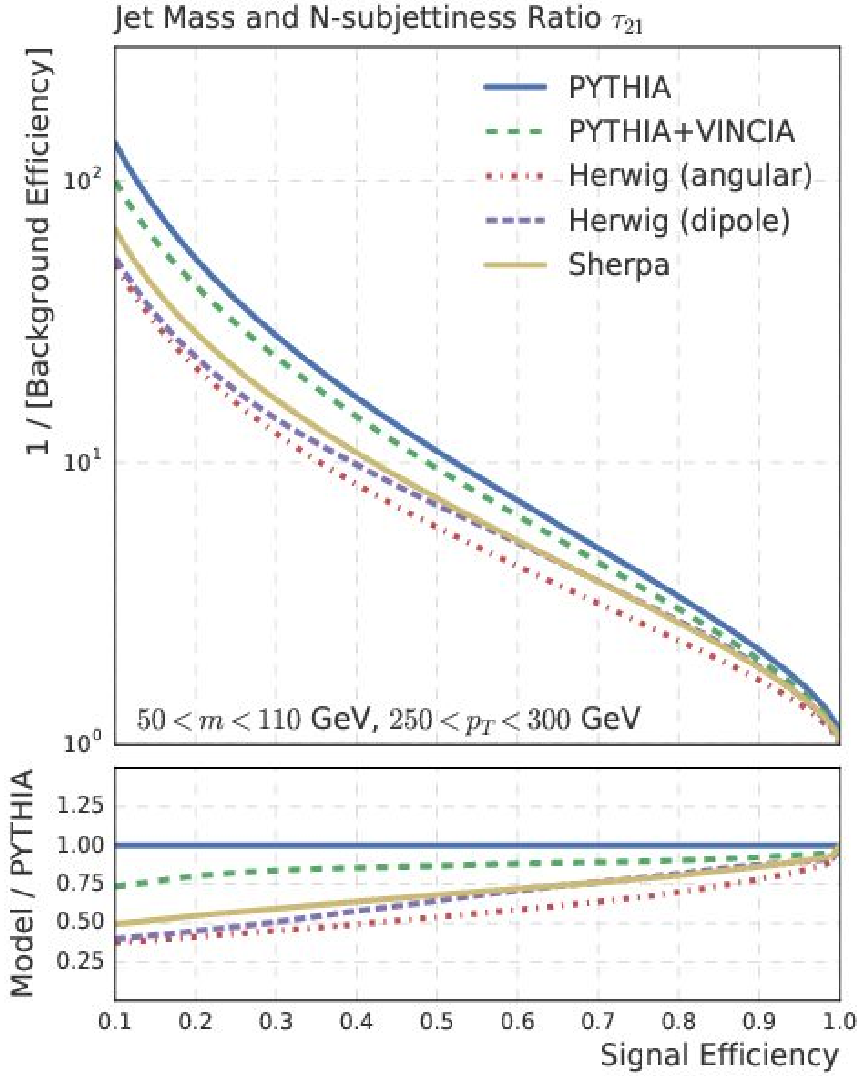
3. Observe regions of excess and deficit relative to Pythia

Are we learning about physics or Monte Carlo?



Comparing Generators: Network Performance





DNN slightly outperforms traditional techniques and appears to have ***uncertainties similar in size!***

What's next?

- **How does this all look in real data and where generators have been tuned to this data?**
  We have only compared generators "out of the box" here…

- **Do realistic jet uncertainties completely wash away any gain in performance?**

- **Can we uncover new observables that constrain differences between generators and data?**
  What additional information is the DNN learning?

Note:

- Please feel free to contribute to and use: github.com/deepjets/deepjets
- You are welcome to have access to our events, jets, and images
  Just ask: noel.dawe@cern.ch

Further Reading

Jet-Images -- Deep Learning Edition
Luke de Oliveira, Michael Kagan, Lester Mackey, Benjamin Nachman, Ariel Schwartzman
https://arxiv.org/abs/1511.05190

Parton Shower Uncertainties in Jet Substructure Analyses with Deep Neural Networks
James Barnard, Edmund Noel Dawe, Matthew J. Dolan, Nina Rajcic
https://arxiv.org/abs/1609.00607

Jet Substructure Classification in High-Energy Physics with Deep Neural Networks
Pierre Baldi, Kevin Bauer, Clara Eng, Peter Sadowski, Daniel Whiteson
https://arxiv.org/abs/1603.09349

Jet Flavor Classification in High-Energy Physics with Deep Neural Networks
Daniel Guest, Julian Collado, Pierre Baldi, Shih-Chieh Hsu, Gregor Urban, Daniel Whiteson
https://arxiv.org/abs/1607.08633

Deep learning in color: towards automated quark/gluon jet discrimination
Patrick T. Komiske, Eric M. Metodiev, Matthew D. Schwartz
https://arxiv.org/abs/1612.01551

Jet Substructure Studies with CMS Open Data
Aashish Tripathee, Wei Xue, Andrew Larkoski, Simone Marzani, Jesse Thaler
https://arxiv.org/abs/1704.05842

QCD-Aware Recursive Neural Networks for Jet Physics
Gilles Louppe, Kyunghyun Cho, Cyril Becot, Kyle Cranmer
https://arxiv.org/abs/1702.00748