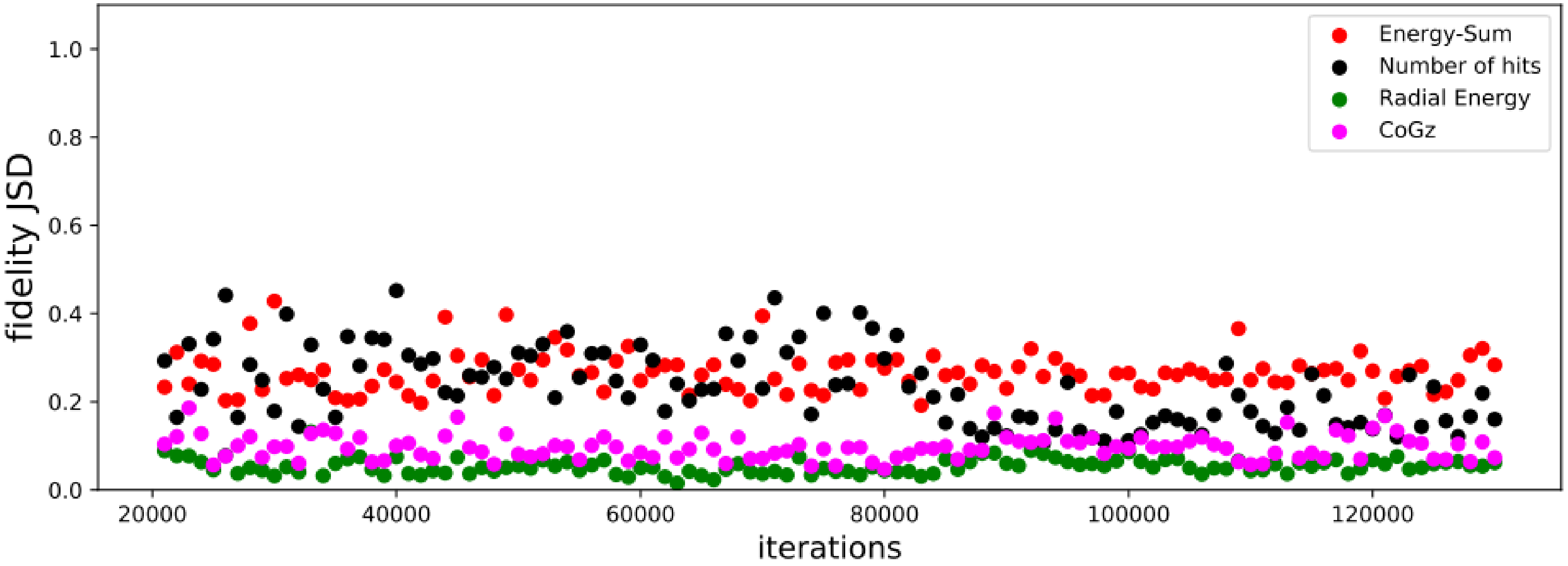
Fidelity Scan: 25x25 Pion showers



Save to a dictionary and read the minimum from there

In [1]:
```python
import json

with open('/beegfs/desy/user/eren/synthetic-data-generator/fidelity/recordJSD_25x25pion.txt', 'r') as file:
    jsdD = json.load(file)

esum_dct = {}
nhit_dct = {}
esum_sum_dct = {}
cog_dct = {}
cog_esum_dct = {}
for i in jsdD.keys():
    #print(i, jsdD[i]['Esum'])
    esum_dct[i] = jsdD[i]['Esum']
    nhit_dct[i] = jsdD[i]['Nhit']
    esum_sum_dct[i] = sum(jsdD[i]['Esum'])
    cog_dct[i] = jsdD[i]['CoG']
    cog_esum_dct[i] = sum(jsdD[i]['Esum']) + jsdD[i]['CoG']
```

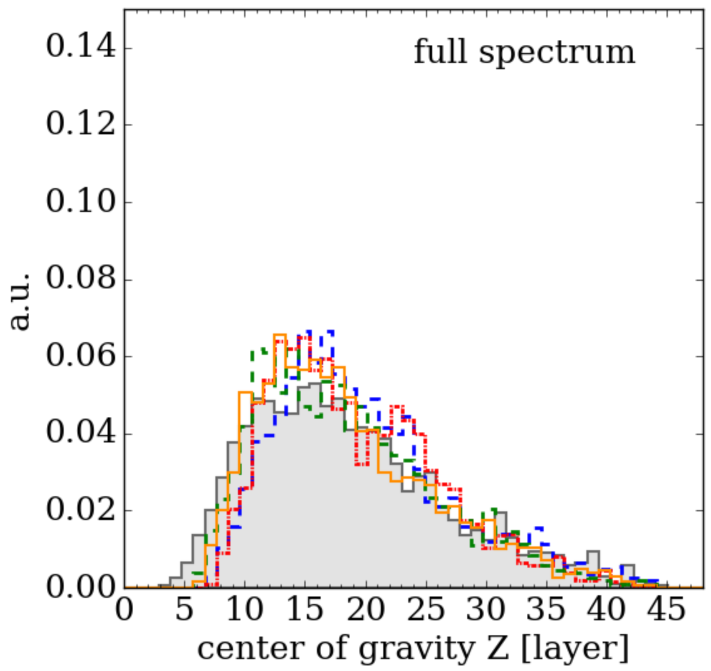
In [2]:
```python
#esum_dct
{k: v for k, v in sorted(nhit_dct.items(), key=lambda item: item[1])}
#{k: v for k, v in sorted(cog_dct.items(), key=lambda item: item[1])}
#{k: v for k, v in sorted(cog_esum_dct.items(), key=lambda item: item[1])}
#print (esum_dct['24000'])
#print (nhit_dct['60500'], esum_dct['60500'])
#cog_esum_dct
```

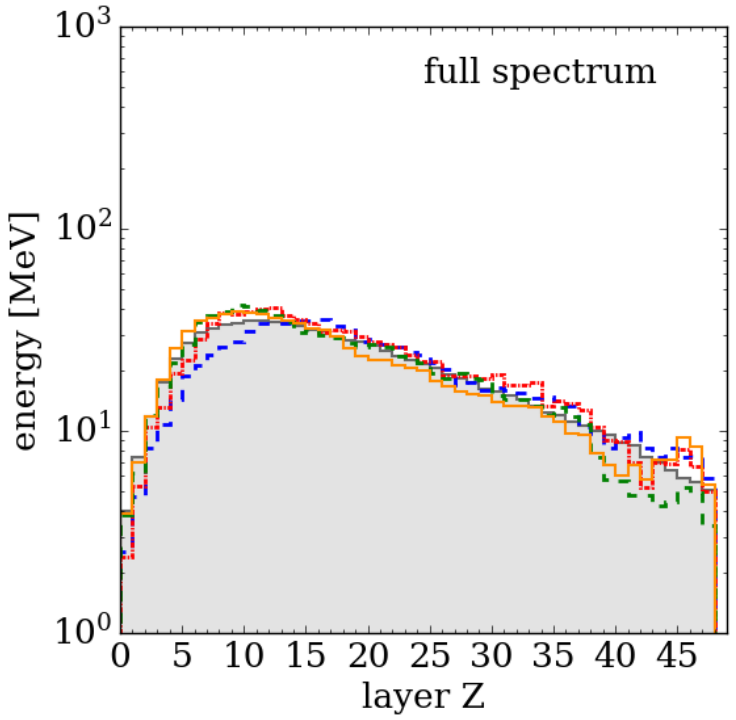
Out[2]:
```
{'93000': 0.10555265573123811,
 '98000': 0.11131531522650814,
 '100000': 0.1122426823368707,
 '97000': 0.11816997893768755,
 '122000': 0.11983364385448668,
```

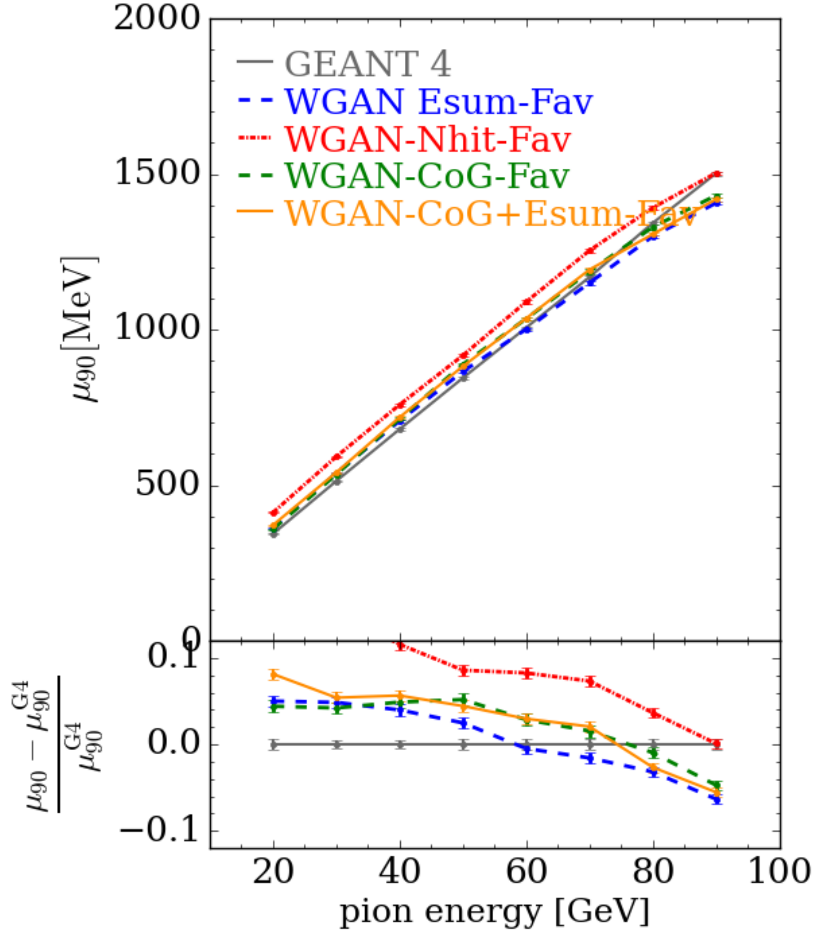
Fidelity Scan: 25x25 Pion showers

Preferences:

1) **Energy sum**
2) Occupancy
3) **Center of Gravity**
4) **Energy sum + Center of Gravity**



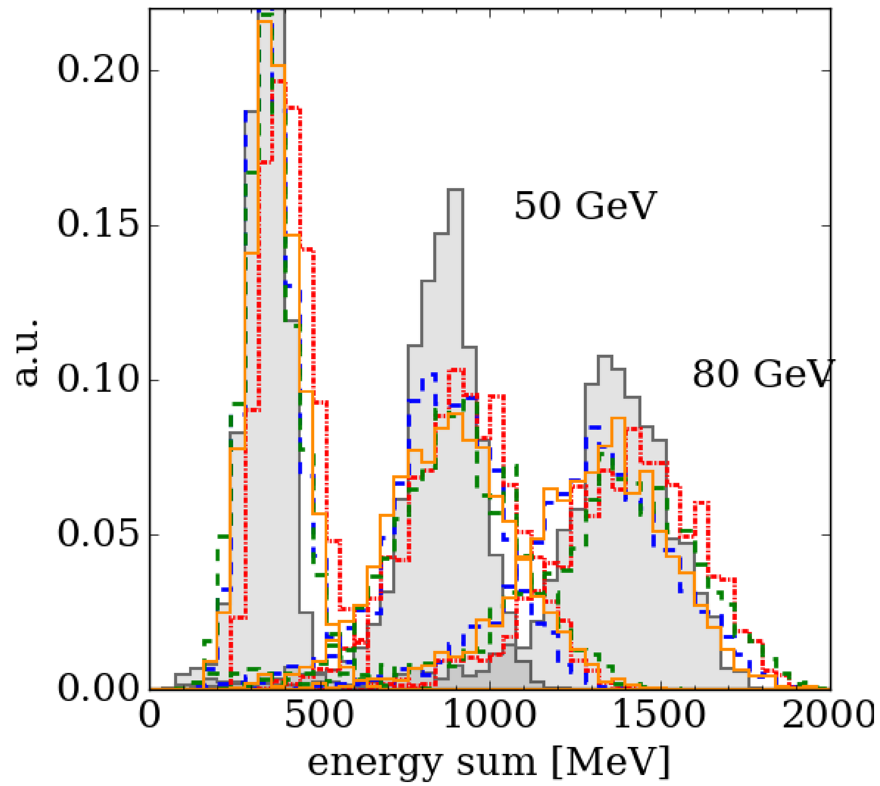


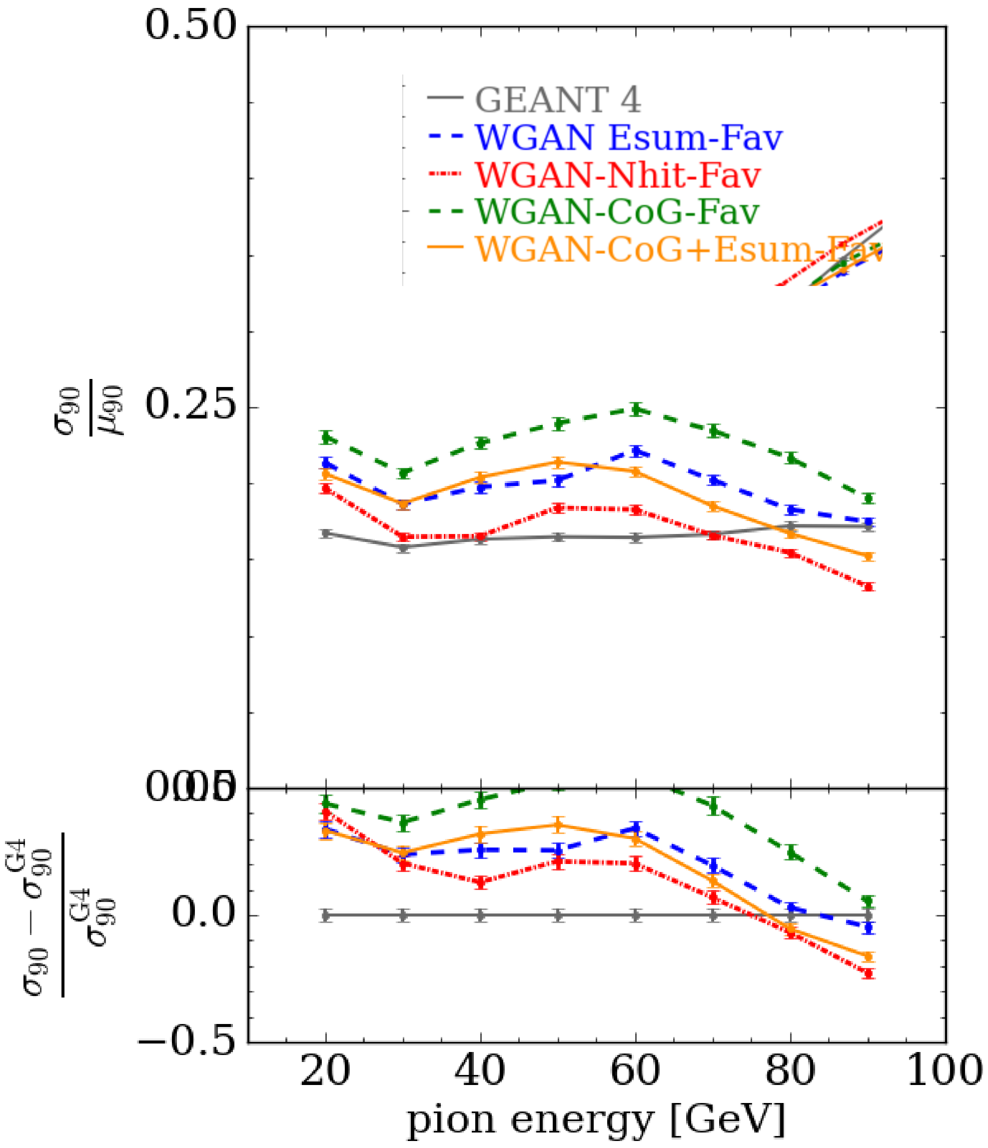


Fidelity Scan: 25x25 Pion showers

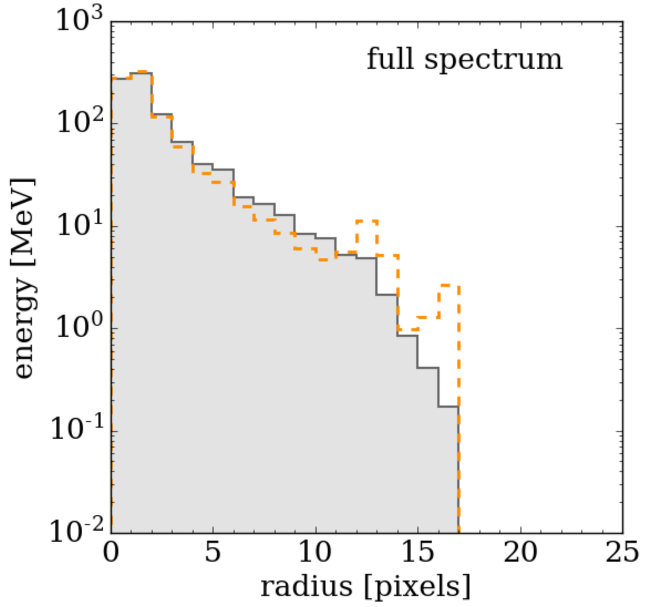
Preferences:

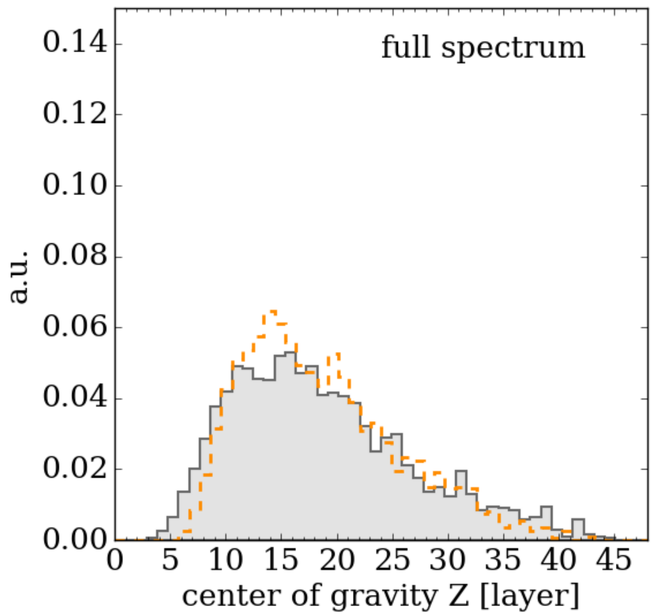
1) **Energy sum**
2) Occupancy
3) **Center of Gravity**
4) **Energy sum + Center of Gravity**

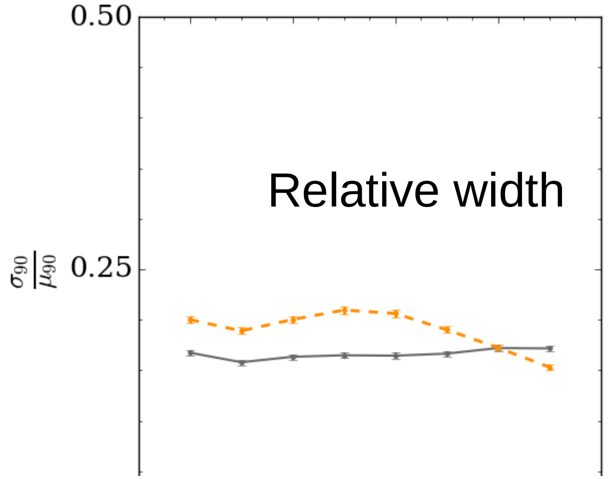


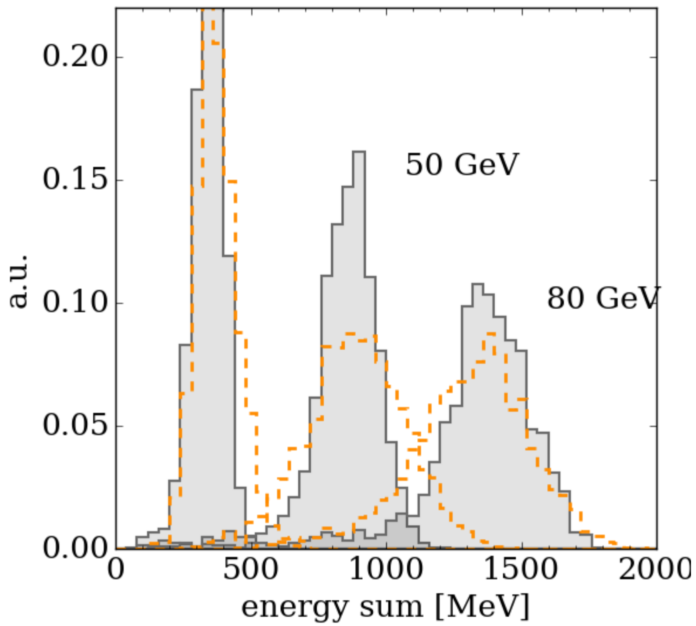


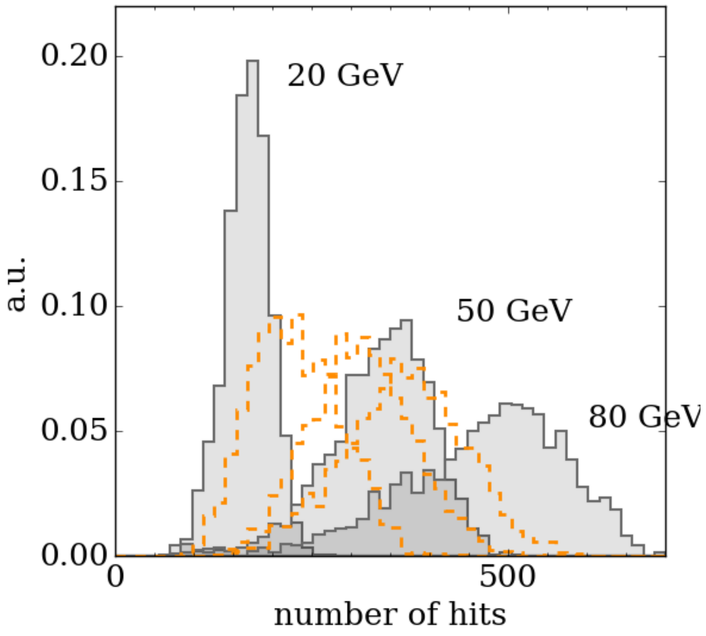
Summary: 25x25 Pion showers

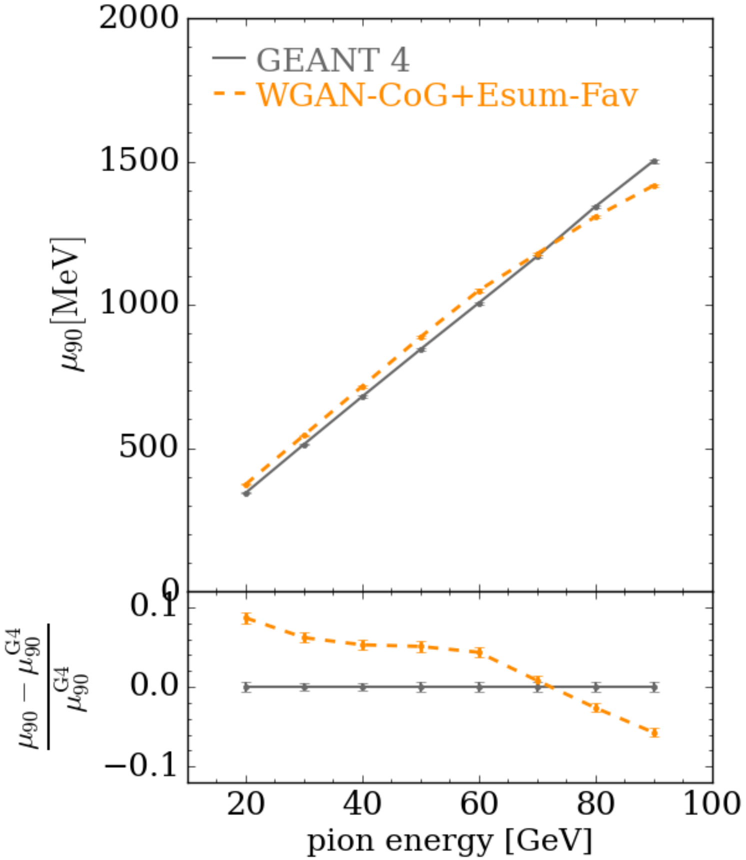






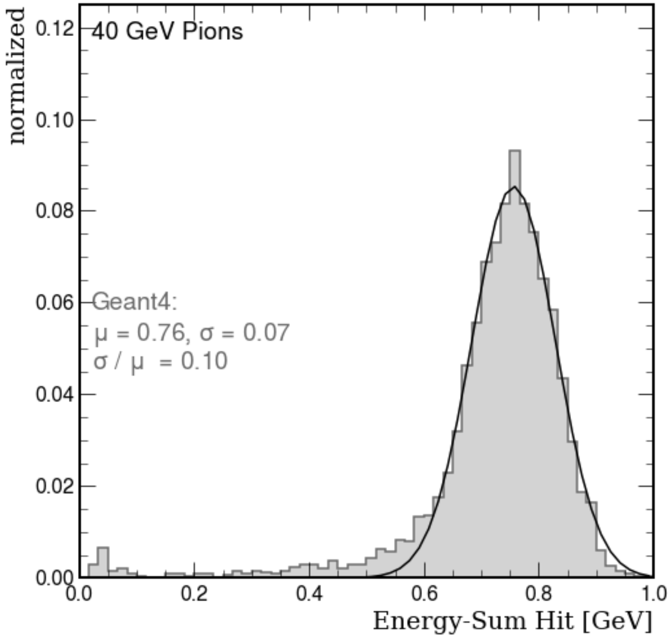


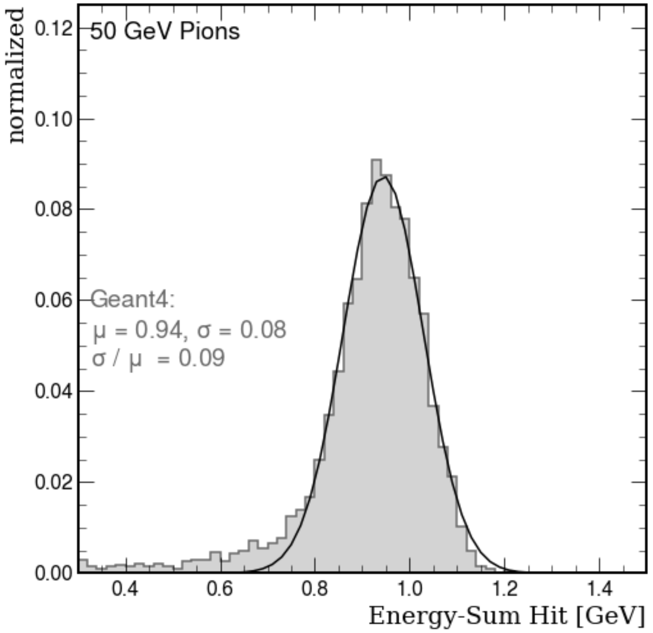


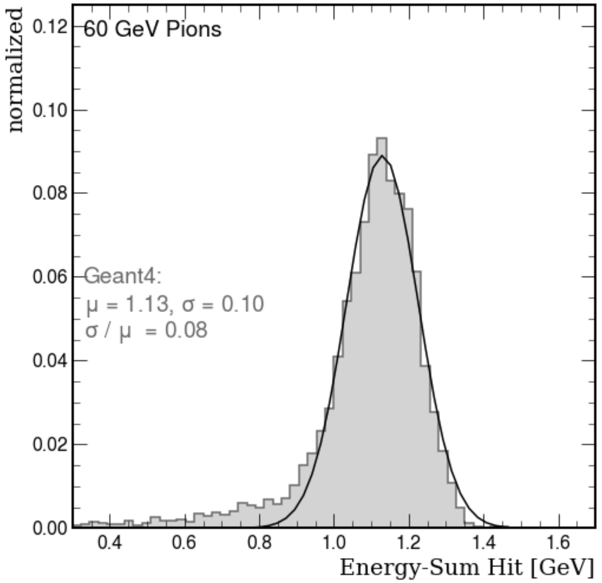


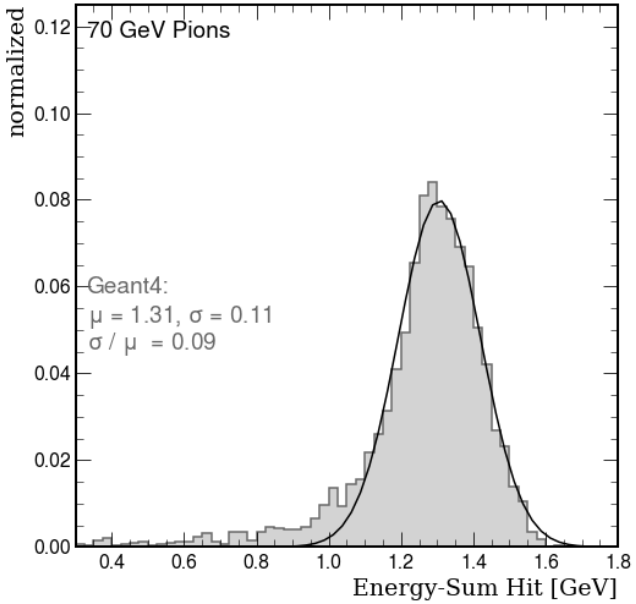
Thank you

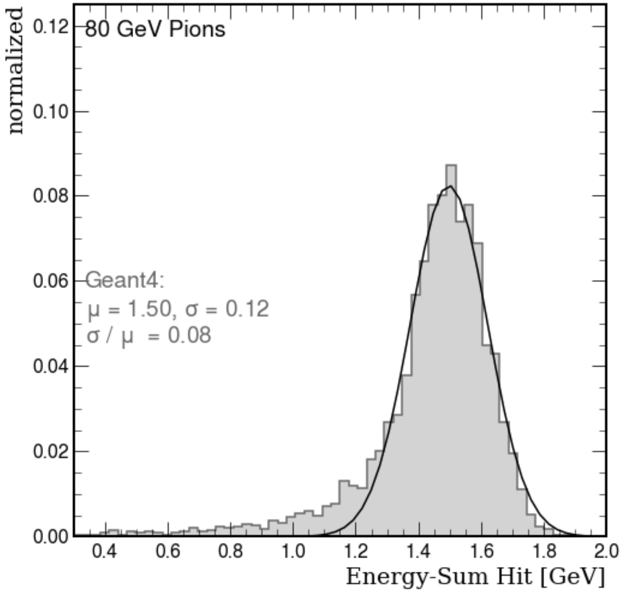
Energy Resolution (Pions, SIM Level)

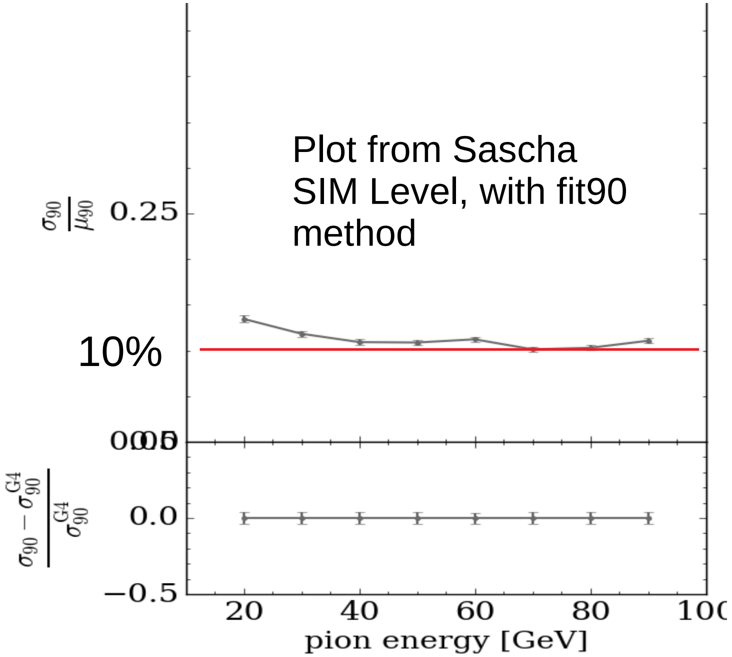




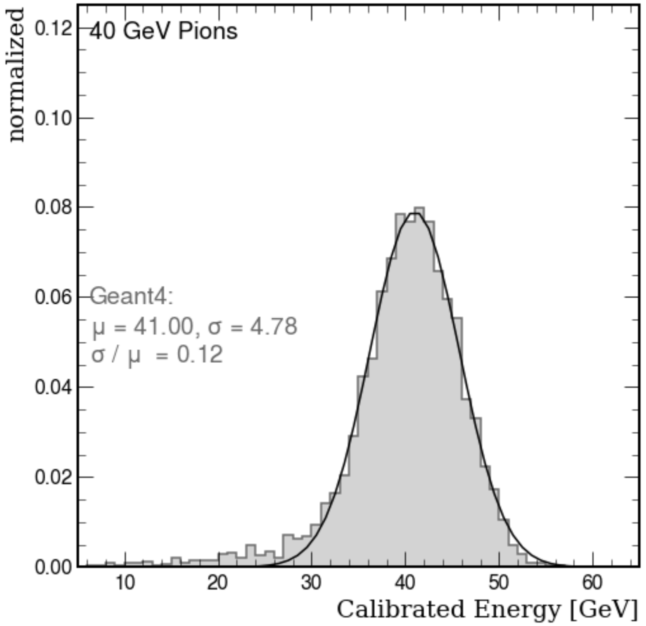


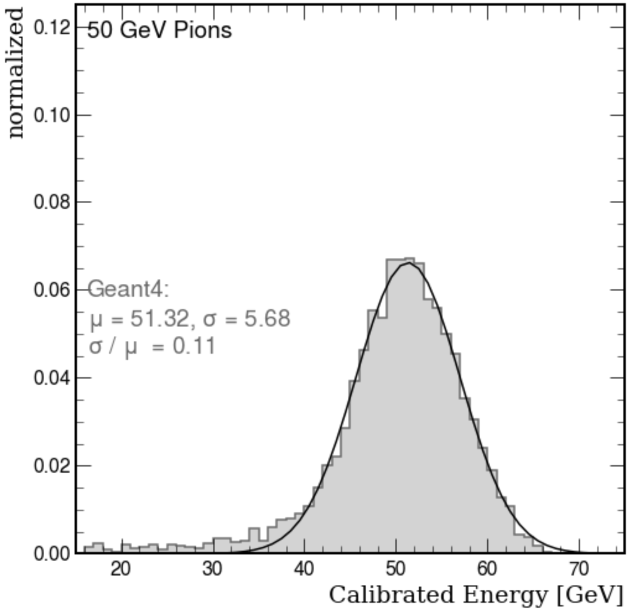


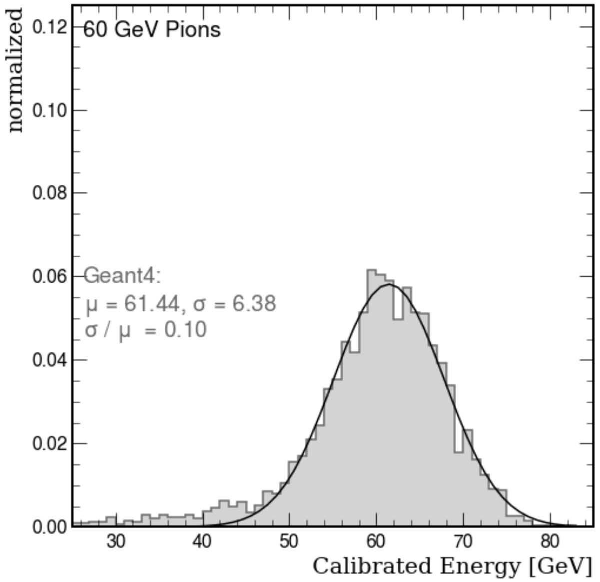


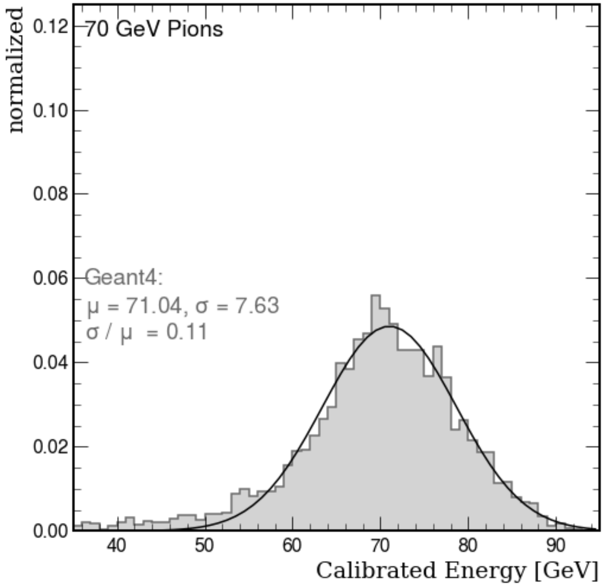


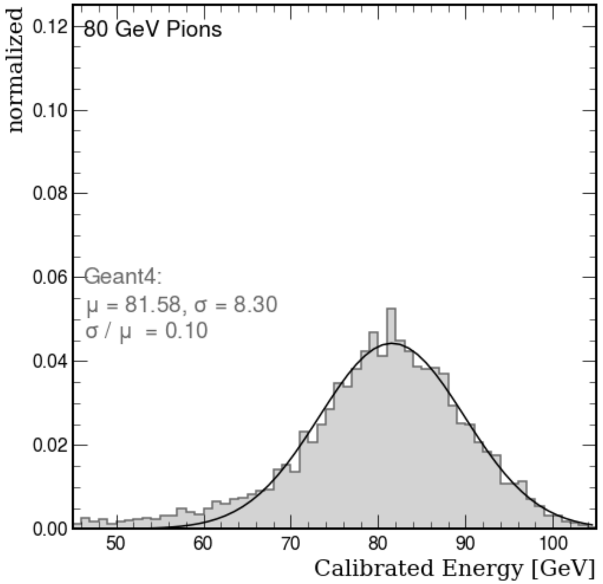
Energy Resolution (Pions, Digi+Calibration Level)

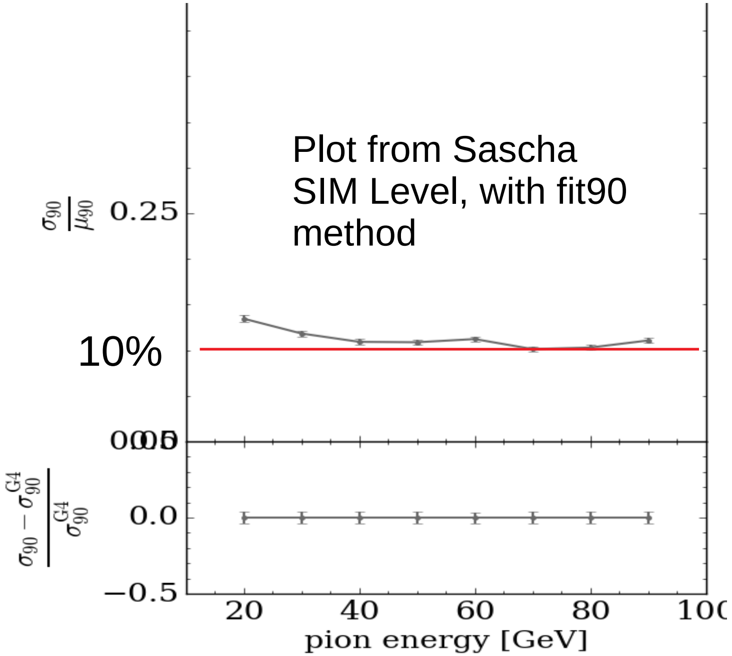






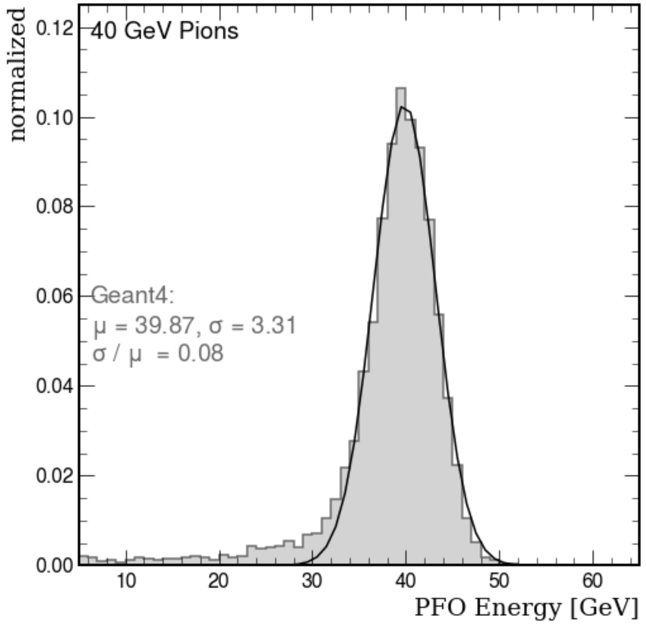


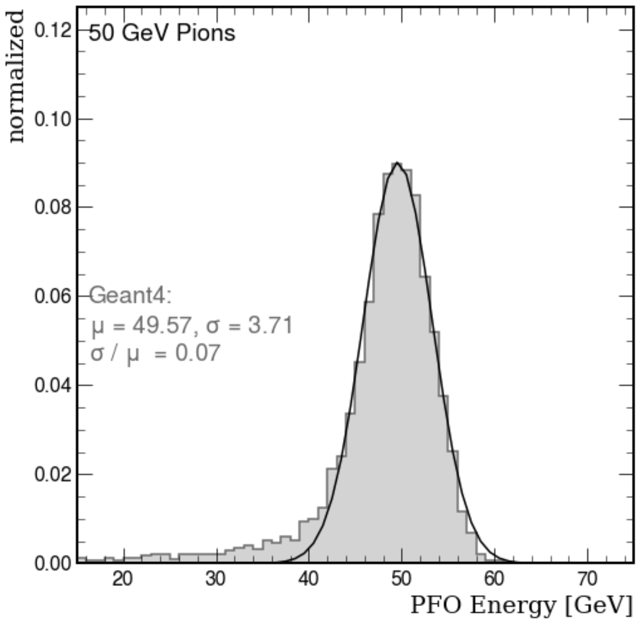


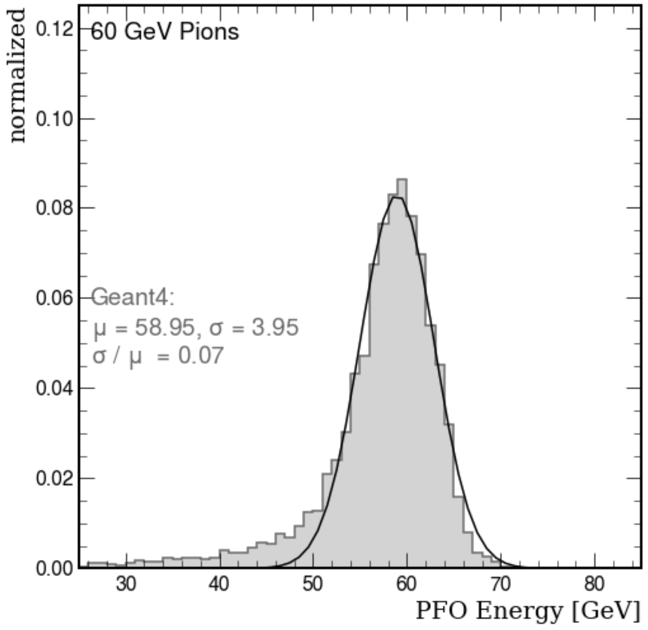


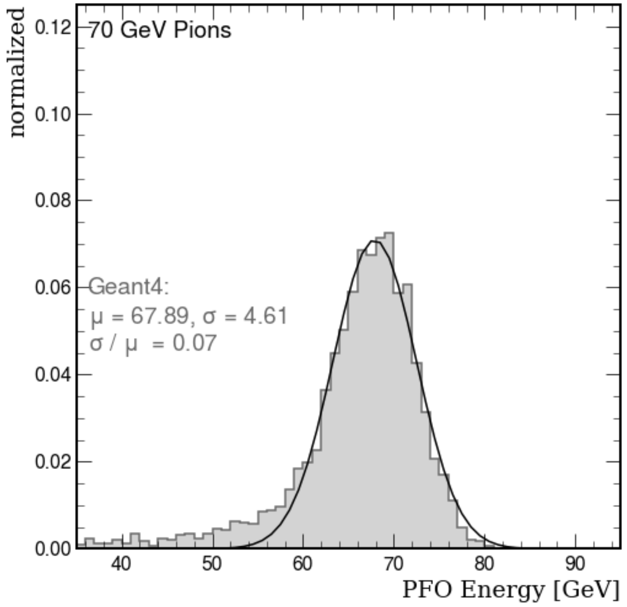
Plot from Sascha
SIM Level, with fit90
method

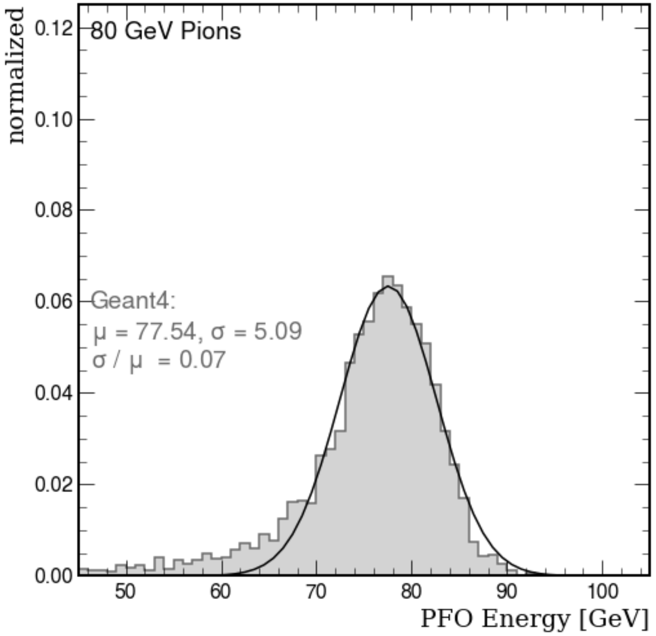
Energy Resolution (Pions, PFO Level)

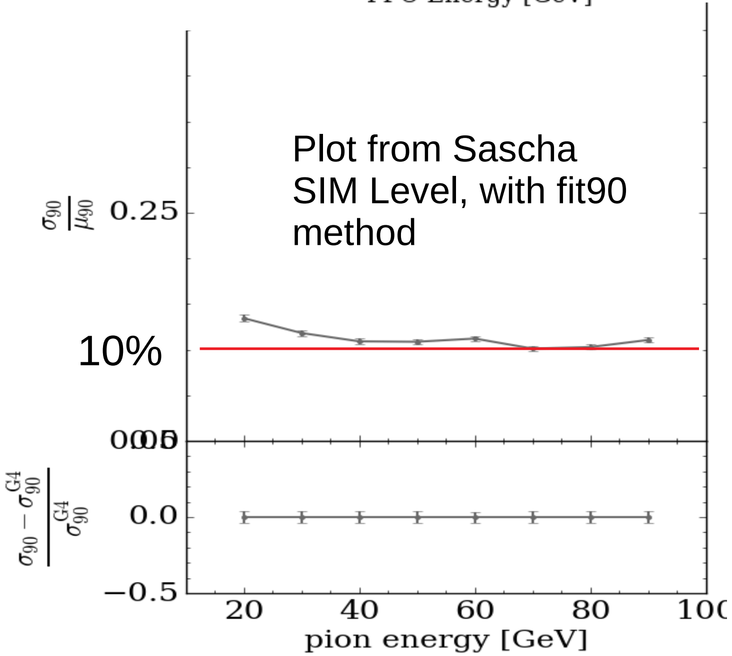






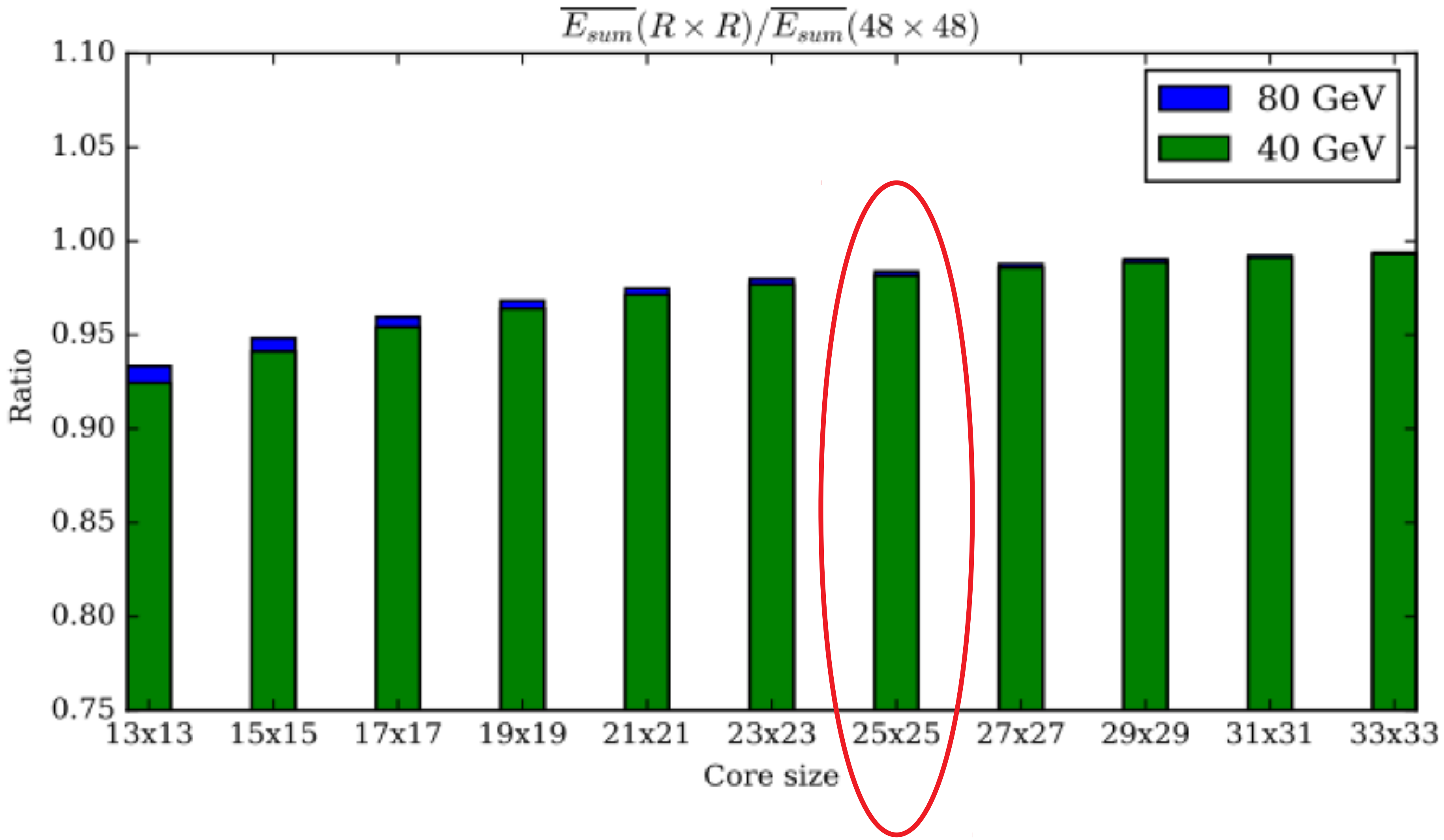






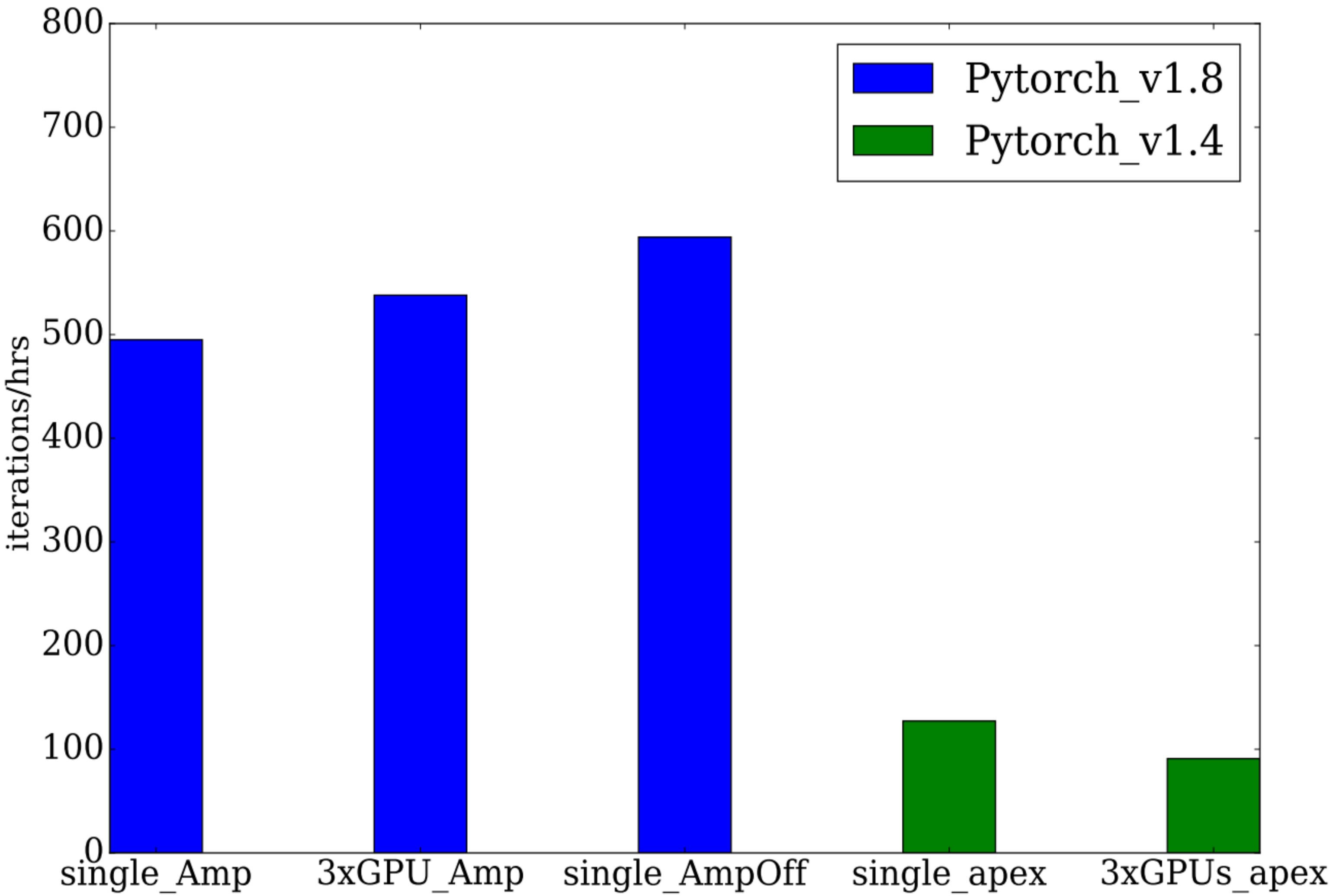
Core size scan (Energy Sum)

- Reminder: We have used 13x13 core size for pion showers up to now.



Pytorch 1.8 training time comparisons

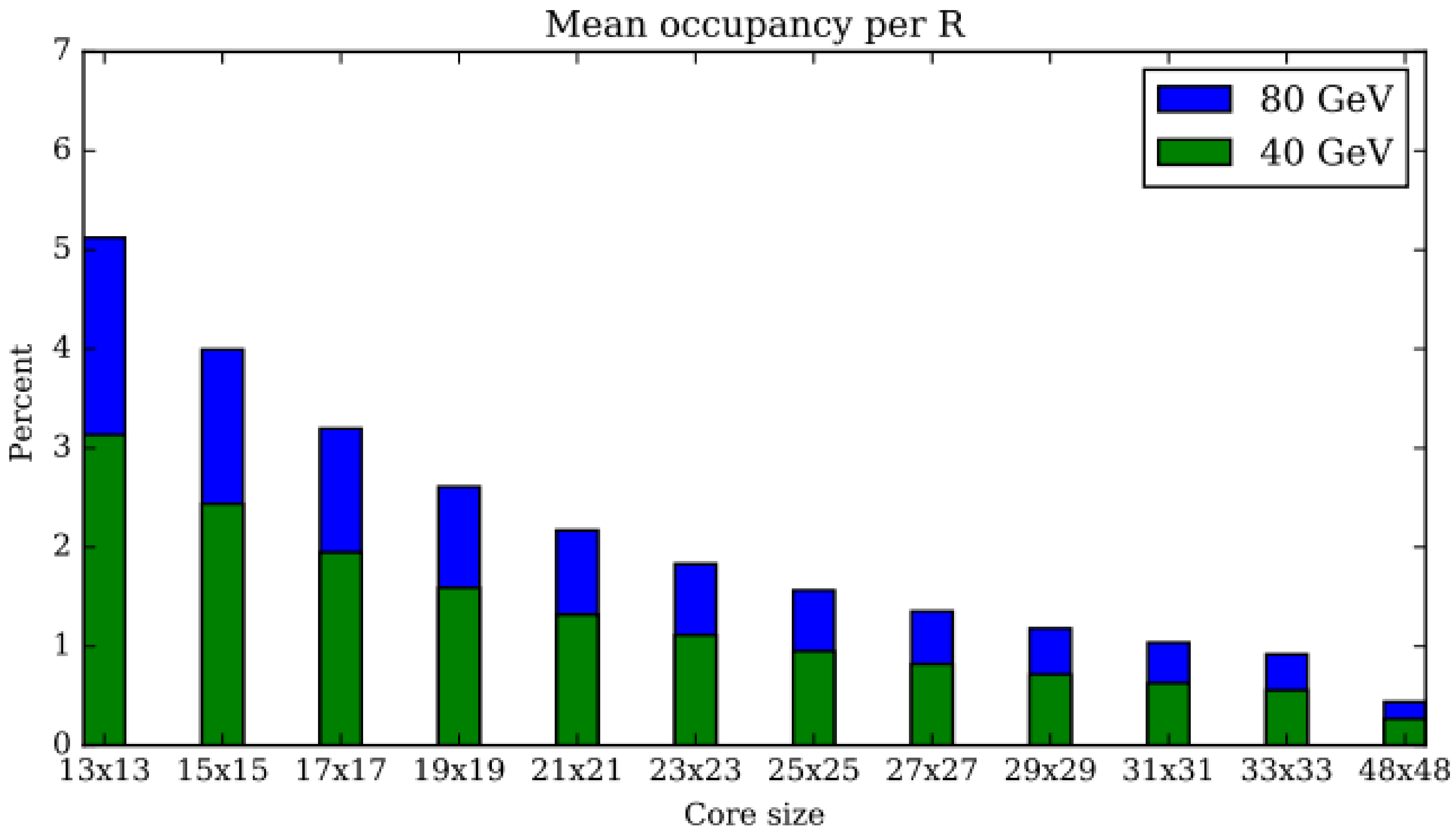
- **Exactly same** network architecture (WGAN-LO) and data (pions 13x13)
- Amp: Automatic-mixed-precision (native in Pytorch **since** v1.6)
- Amp in Apex: NVIDIA-maintained utility for mixed precision training



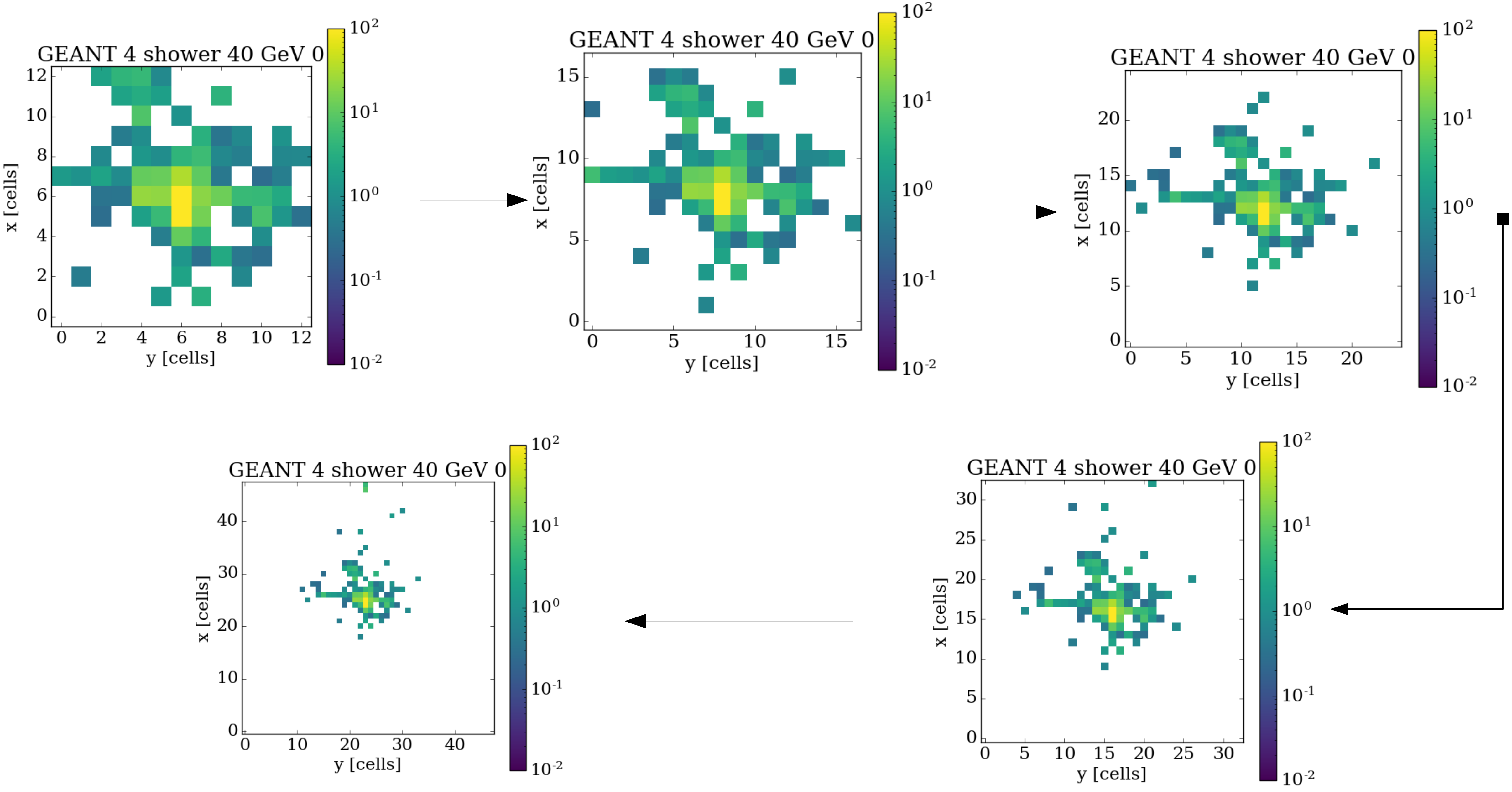
3xGPU means:
Distributed-Data-Parallel
across 3 GPU nodes!!

Core size scan (Mean occupancy)

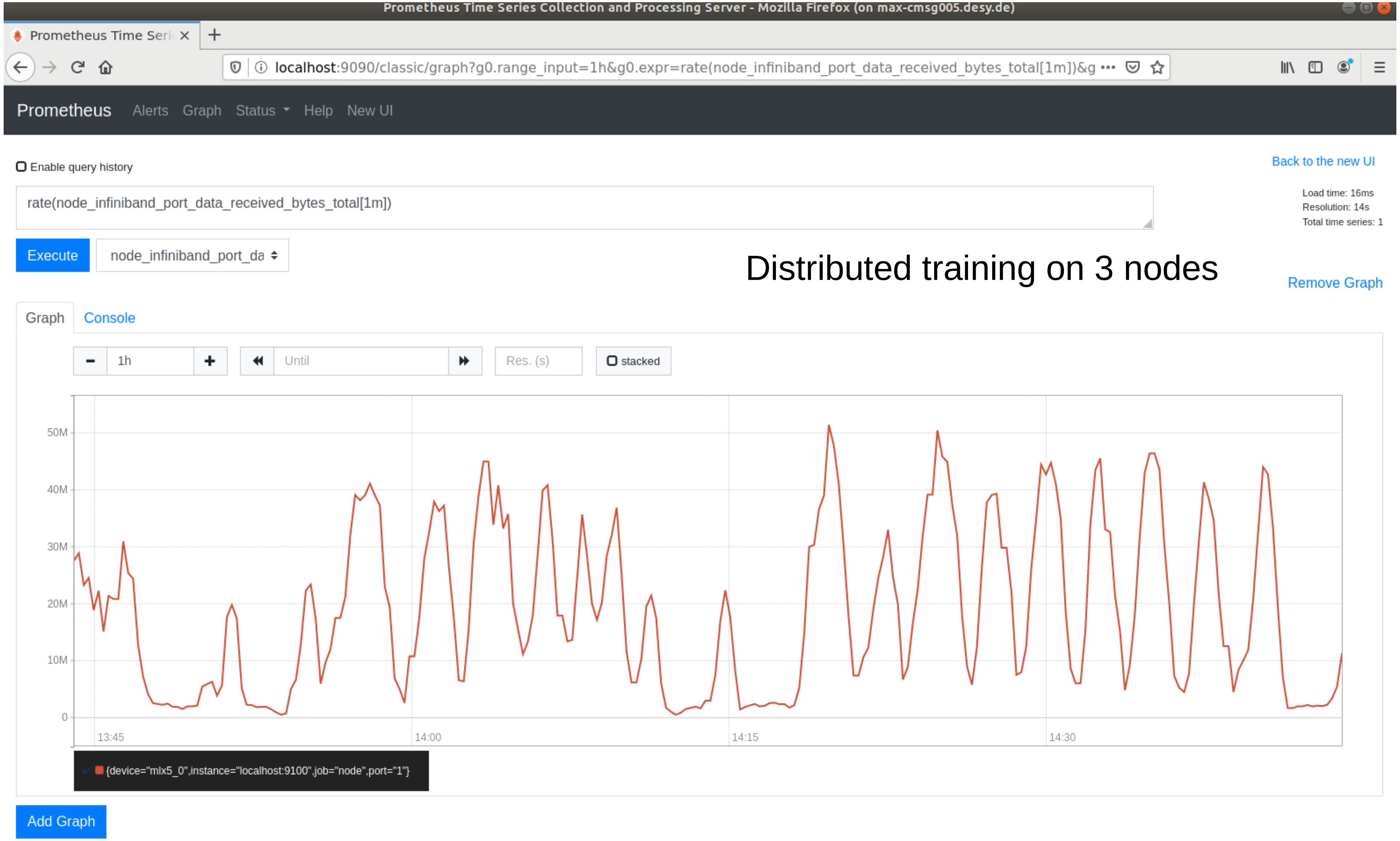
- How about active cells ? (after MIP cut)



Some examples



Bonus: Infiniband (IB) throughput in maxwell during training

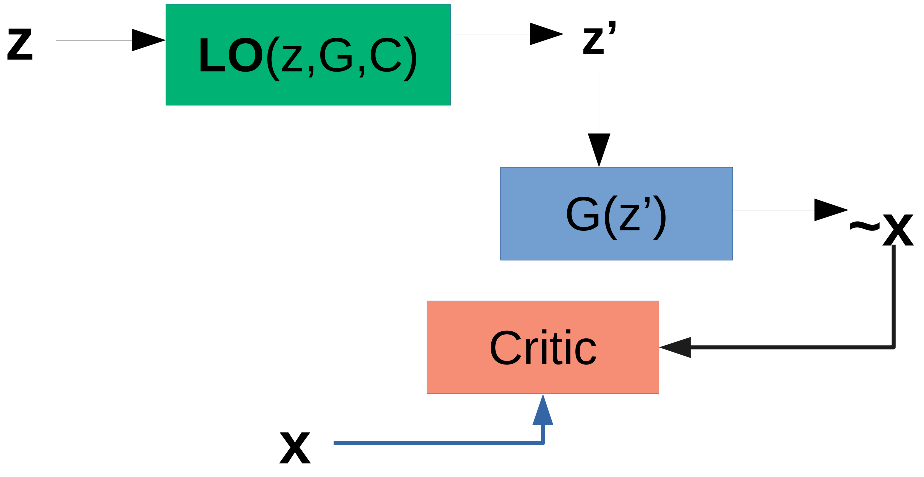


Distributed training on 3 nodes

At the peak of 50Mb / min → 6.64 Mbit / sec

Fidelity of Photon Showers (Getting High) with WGAN-LO

- Fidelity scan based on 6 distributions with JSD



scipy.spatial.distance.jensenshannon¶

scipy.spatial.distance.jensenshannon(*p*, *q*, *base=None*) [source]

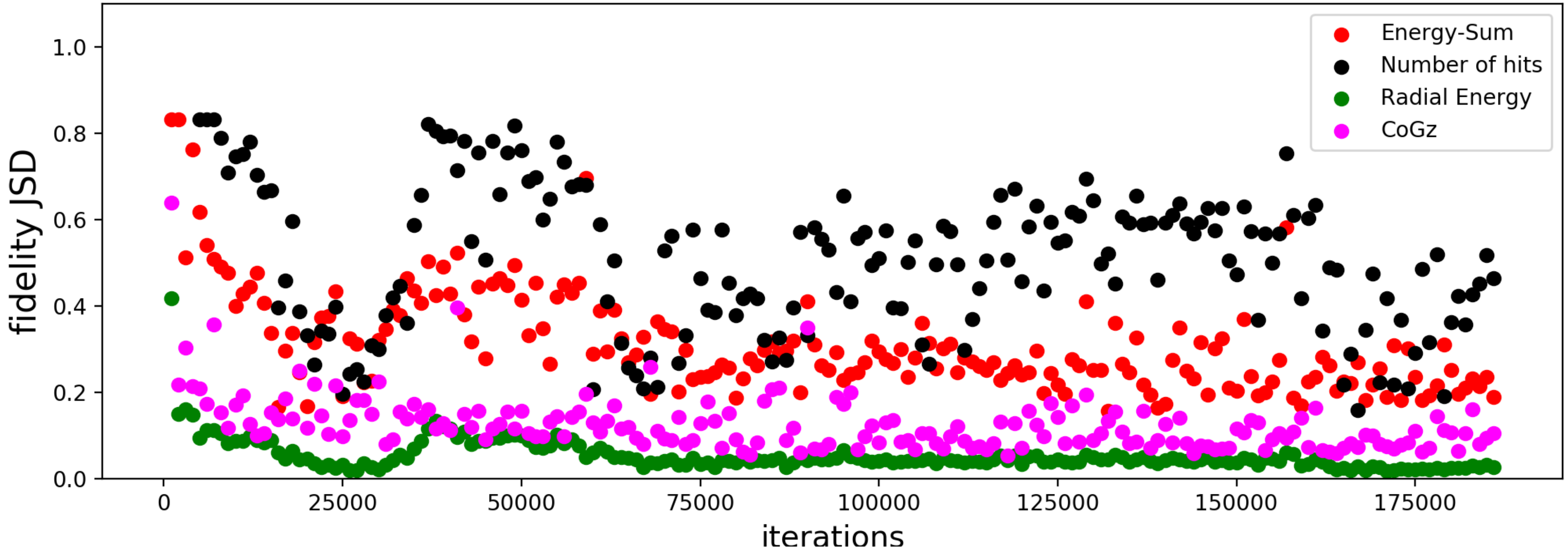
Compute the Jensen-Shannon distance (metric) between two 1-D probability arrays. This is the square root of the Jensen-Shannon divergence.

The Jensen-Shannon distance between two probability vectors p and q is defined as,

$$\sqrt{\frac{D(p \parallel m) + D(q \parallel m)}{2}}$$

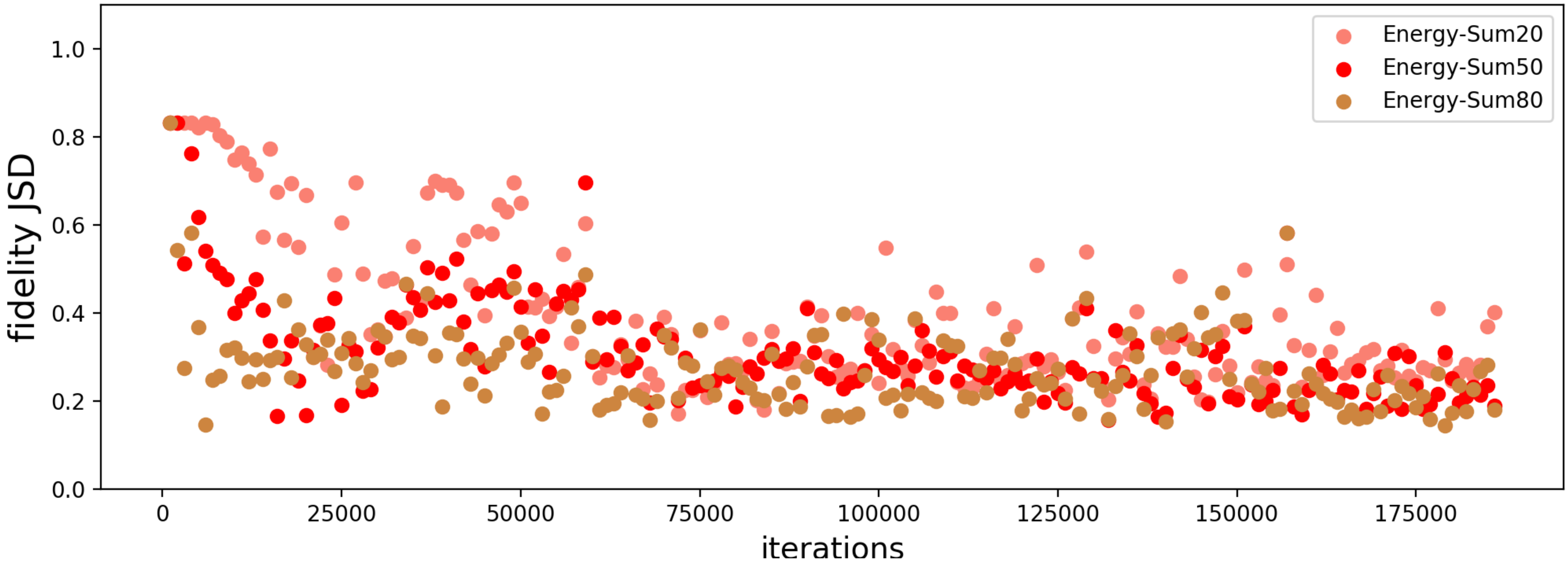
where m is the pointwise mean of p and q and D is the Kullback-Leibler divergence.

This routine will normalize p and q if they don't sum to 1.0.



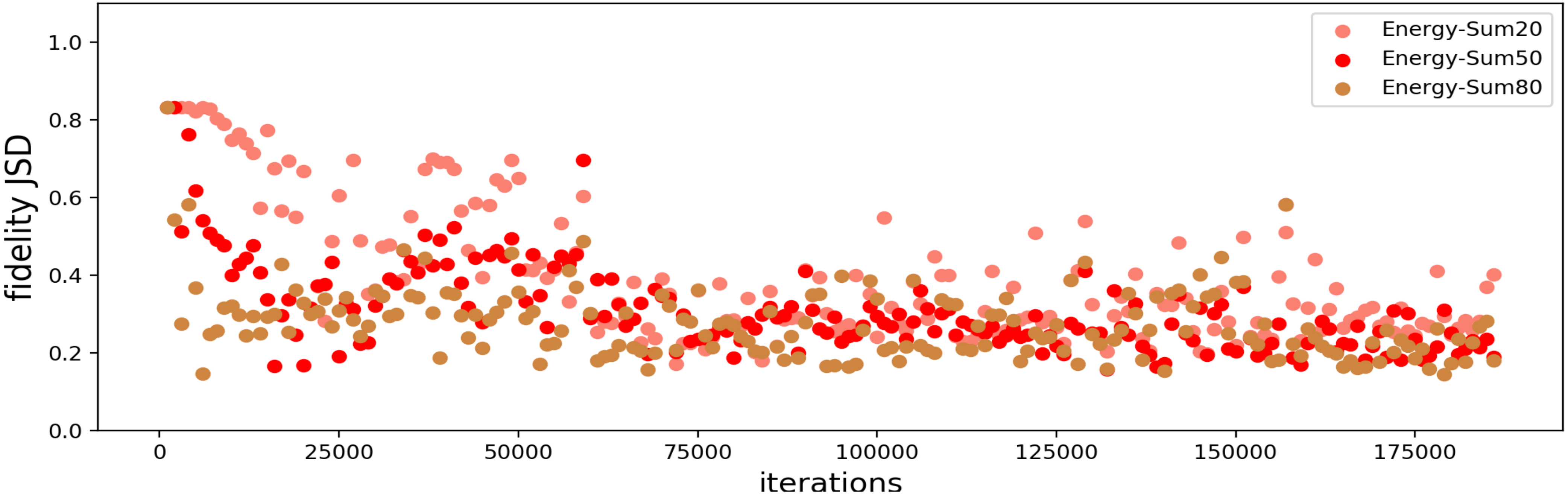
Fidelity of Photon Showers (Getting High) with WGAN-LO

- Fidelity scan for Esum → 20, 50 and 80 GeV



Fidelity of Photon Showers (Getting High) with WGAN-LO

- Fidelity scan for Esum → 20, 50 and 80 GeV

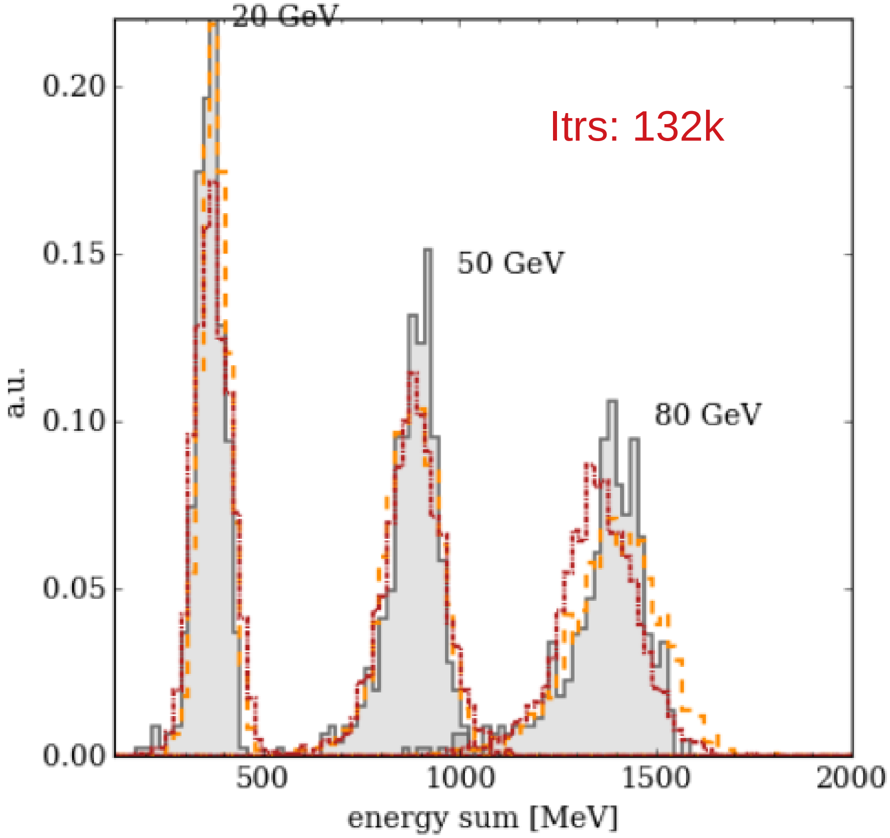


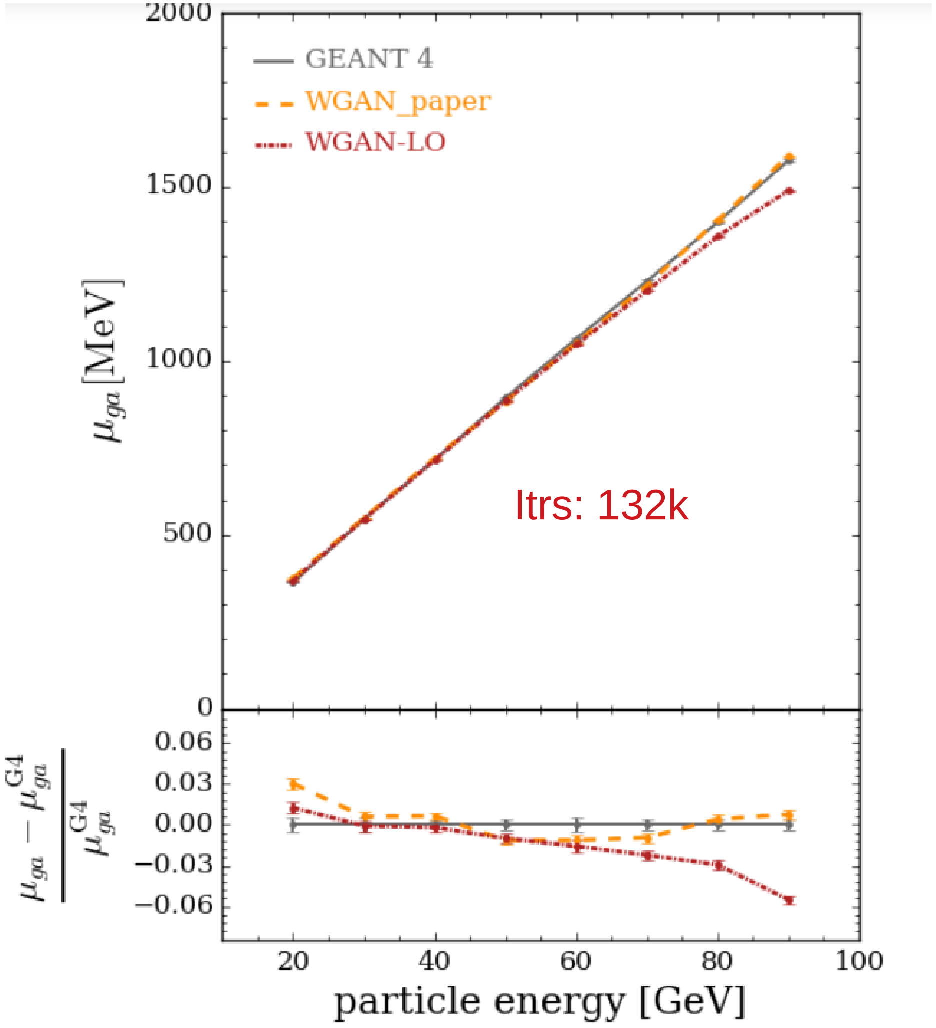

```
In [15]: {k: v for k, v in sorted(esum_sum_dct.items(), key=lambda item: item[1])}

Out[15]: {'132000': 0.5202102358010448,
          '72000': 0.5813686076064758,
          '159000': 0.5949365376640033,
          '68000': 0.6171981438028666,
          '177000': 0.620575063039553,
          '126000': 0.6280052494607875,
          '137000': 0.6372517409402512,
          '155000': 0.6400065488422447,
          '140000': 0.6488753373121894,
          '165000': 0.6510598361638933,
```

Fidelity of Photon Showers (Getting High) with WGAN-LO

- Fidelity scan for Esum → 20, 50 and 80 GeV

```
In [15]: {k: v for k, v in sorted(esum_sum_dct.items(), key=lambda item: item[1])}

Out[15]: {'132000': 0.5202102358010448,
          '72000': 0.5813686076064758,
          '159000': 0.5949365376640033,
          '68000': 0.6171981438028666,
          '177000': 0.620575063039553,
          '126000': 0.6280052494607875,
          '137000': 0.6372517409402512,
          '155000': 0.6400065488422447,
          '140000': 0.6488753373121894,
          '165000': 0.6510598361638933,
```

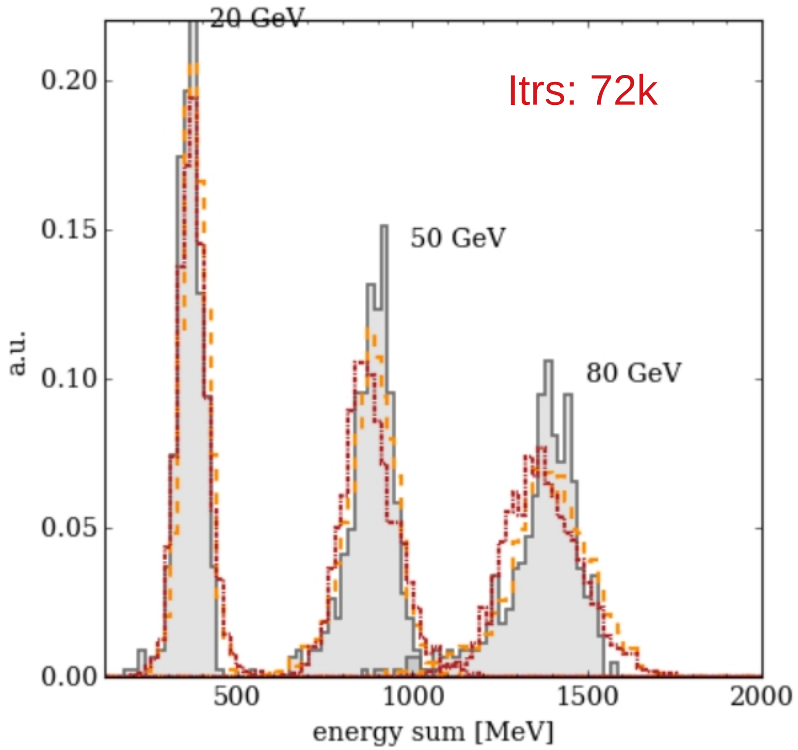


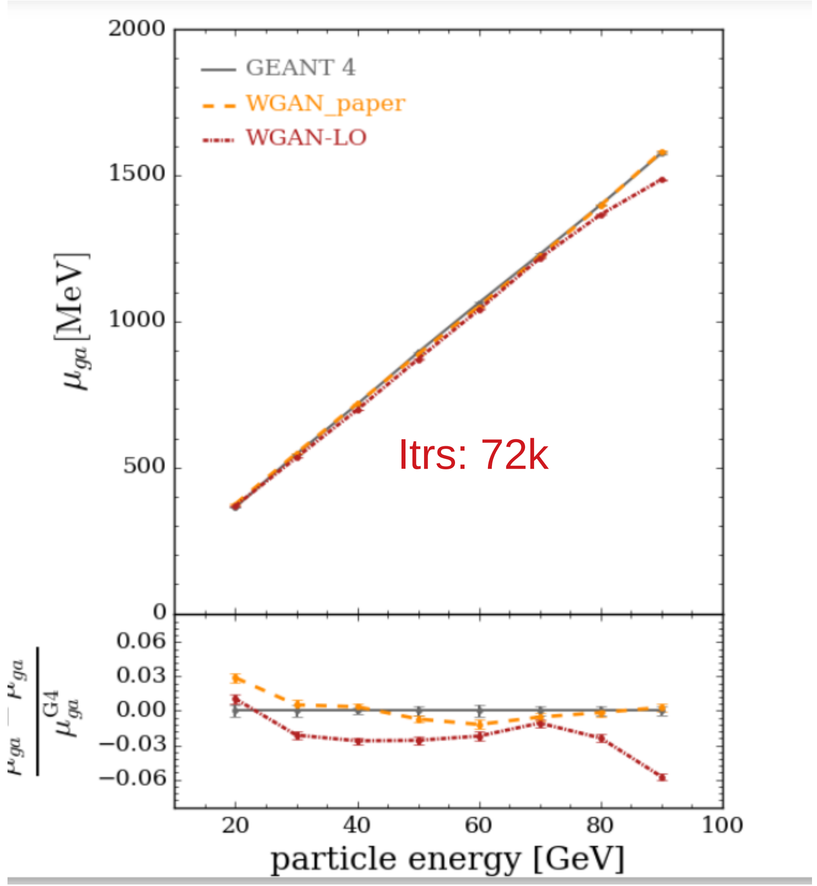


Fidelity of Photon Showers (Getting High) with WGAN-LO

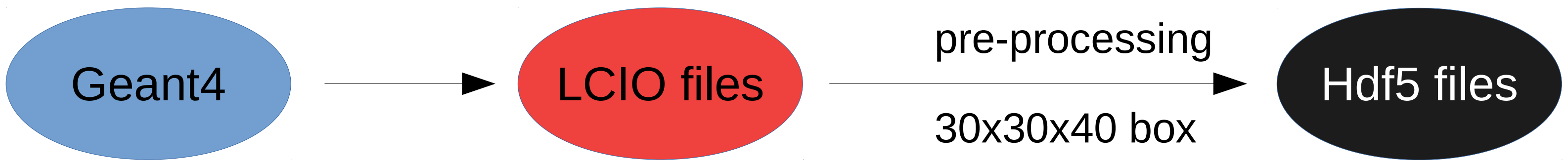
- Fidelity scan for Esum → 20, 50 and 80 GeV

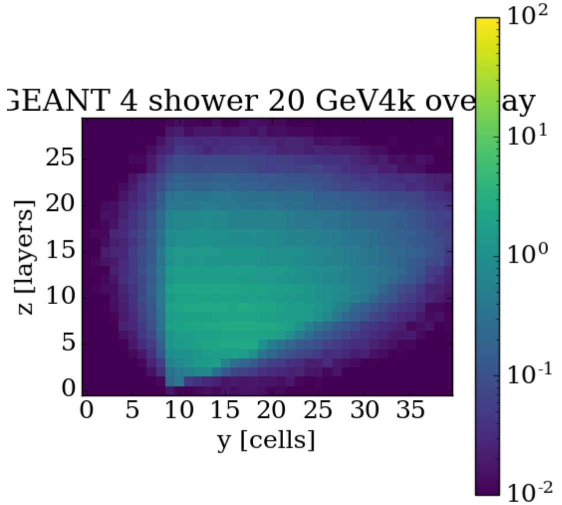
```
In [15]: {k: v for k, v in sorted(esum_sum_dct.items(), key=lambda item: item[1])}

Out[15]: {'132000': 0.5202102358010448,
          '72000': 0.5813686076064758,
          '159000': 0.5949365376640033,
          '68000': 0.6171981438028666,
          '177000': 0.620575063039553,
          '126000': 0.6280052494607875,
          '137000': 0.6372517409402512,
          '155000': 0.6400065488422447,
          '140000': 0.6488753373121894,
          '165000': 0.6510598361638933,
```

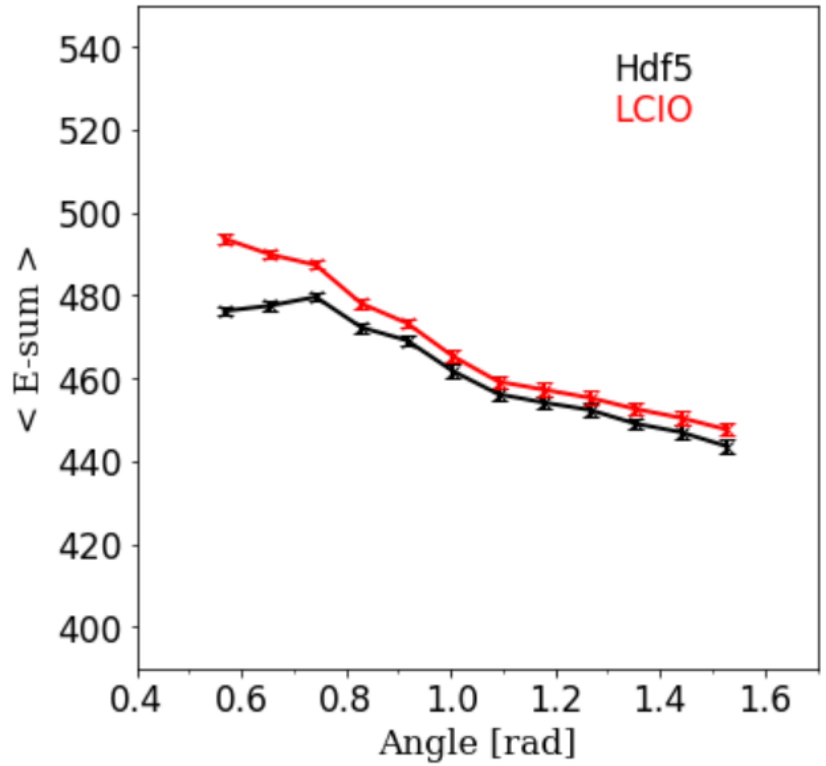


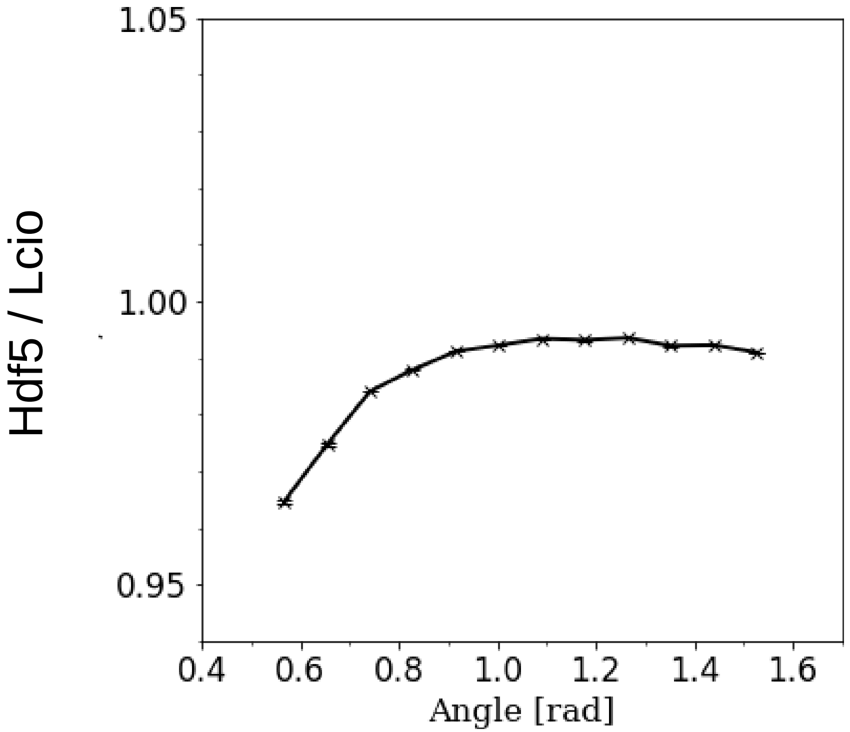


Photon Showers with Angle [solved]



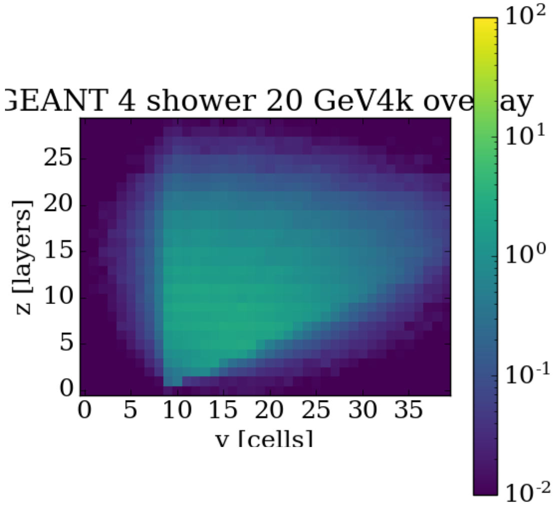


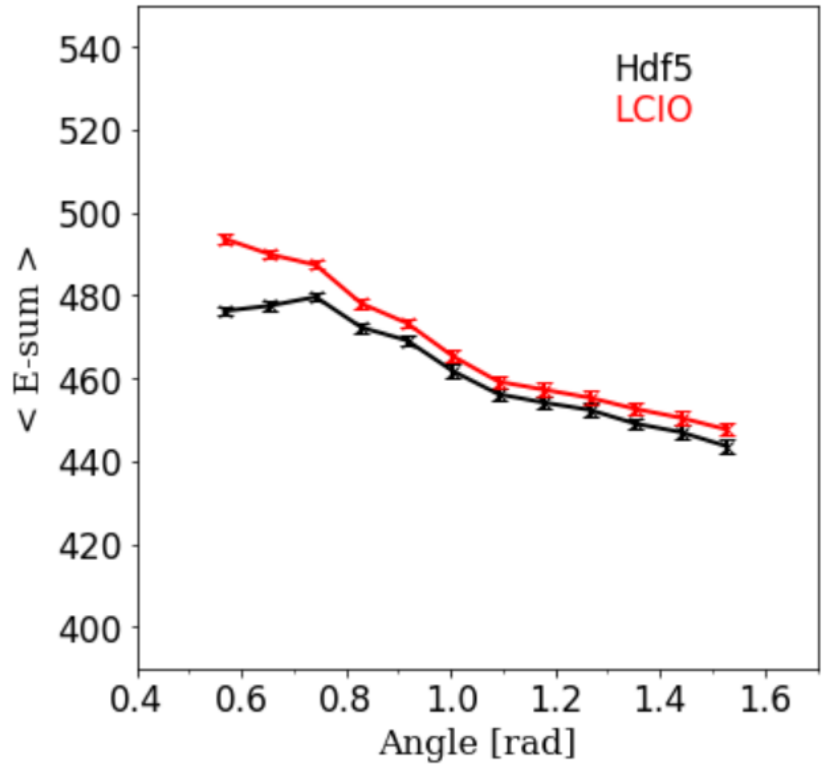




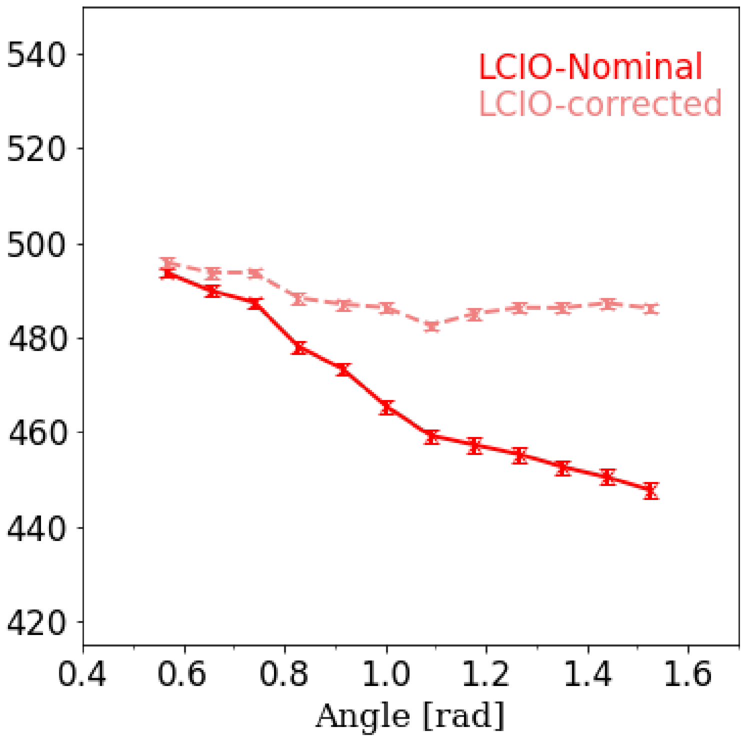
Photon Showers with Angle [Why slope?]

Geant4 → LCIO files → (pre-processing, 30x30x40 box) → Hdf5 files

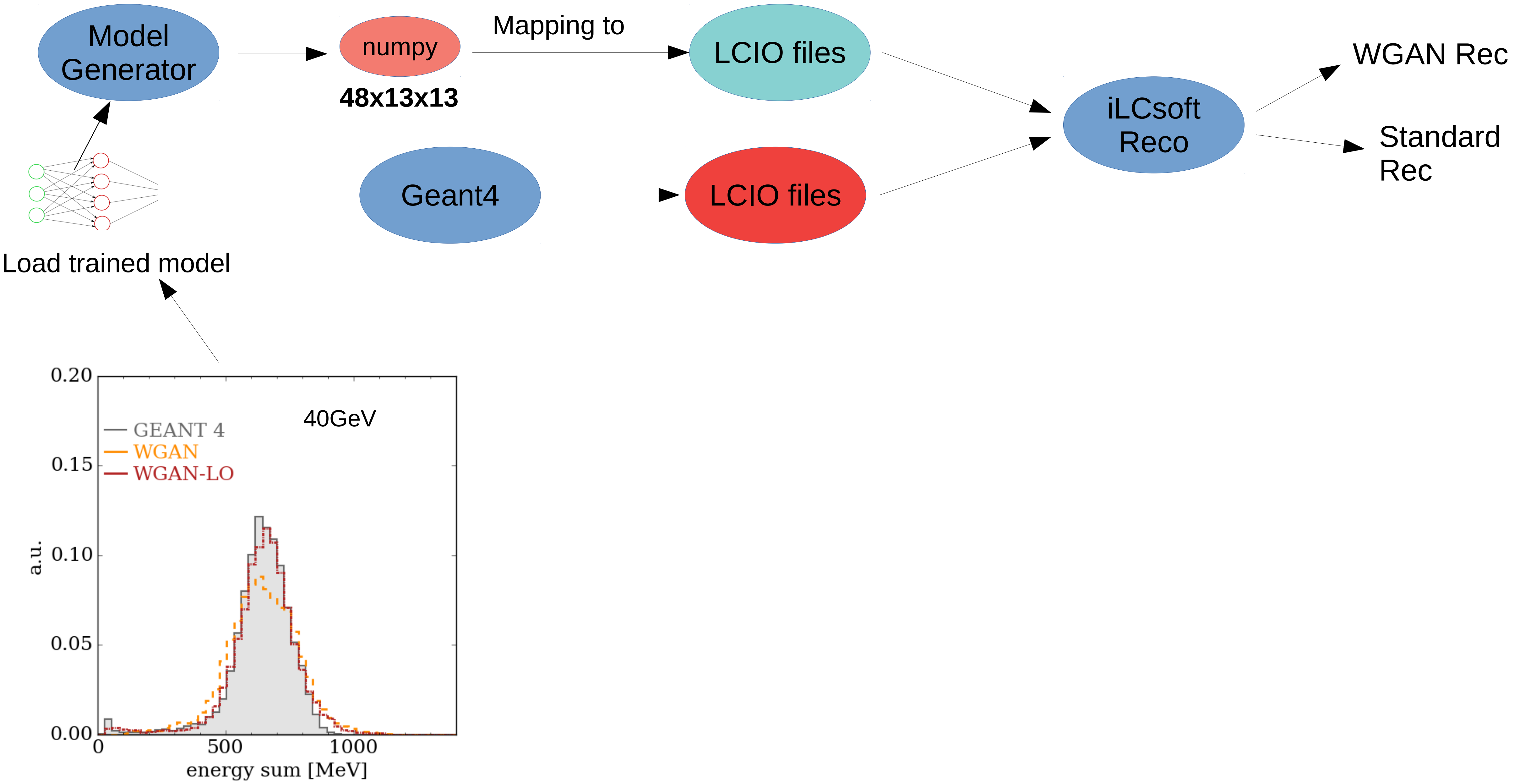




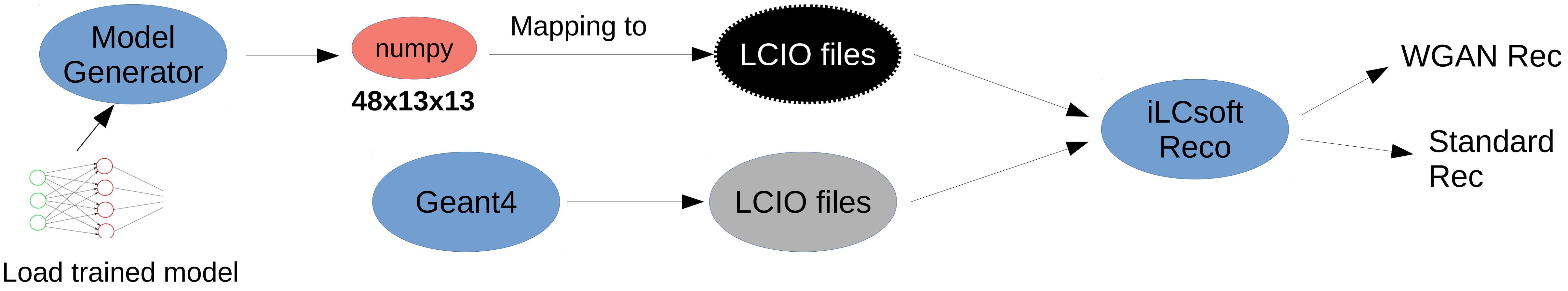
Last 10 layers are multiply by 2

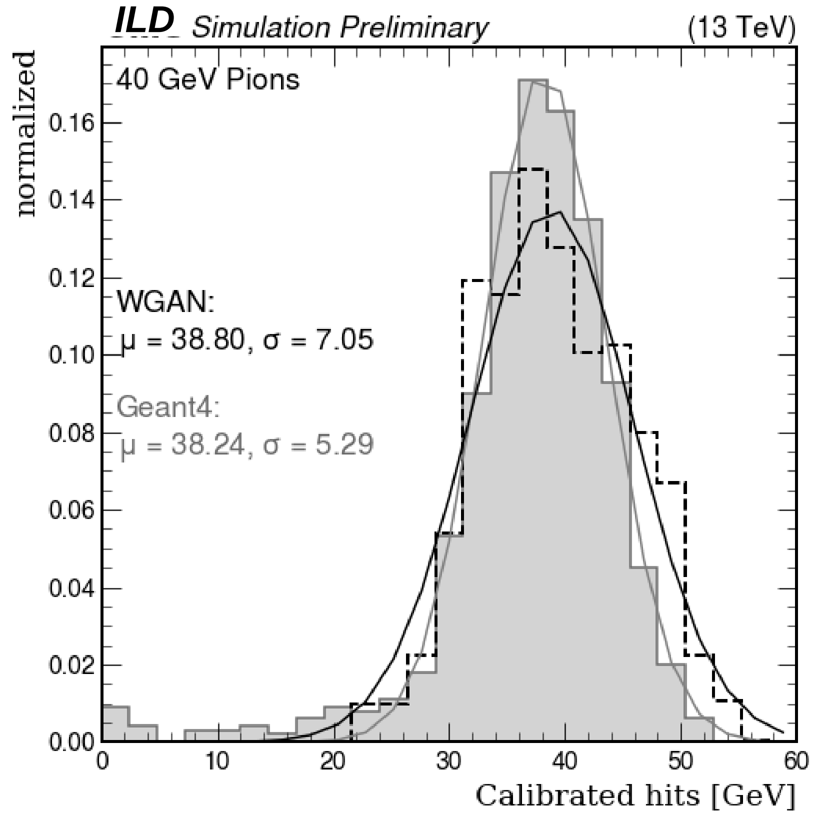


Pion Showers [Reconstruction, *in progress*]

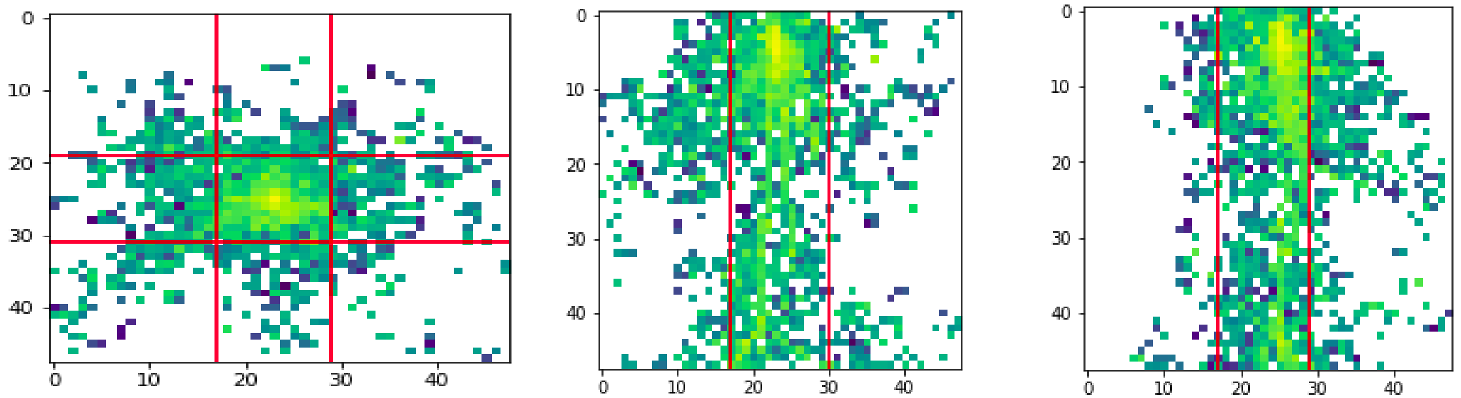


Pion Showers [Reconstruction, *in progress*]



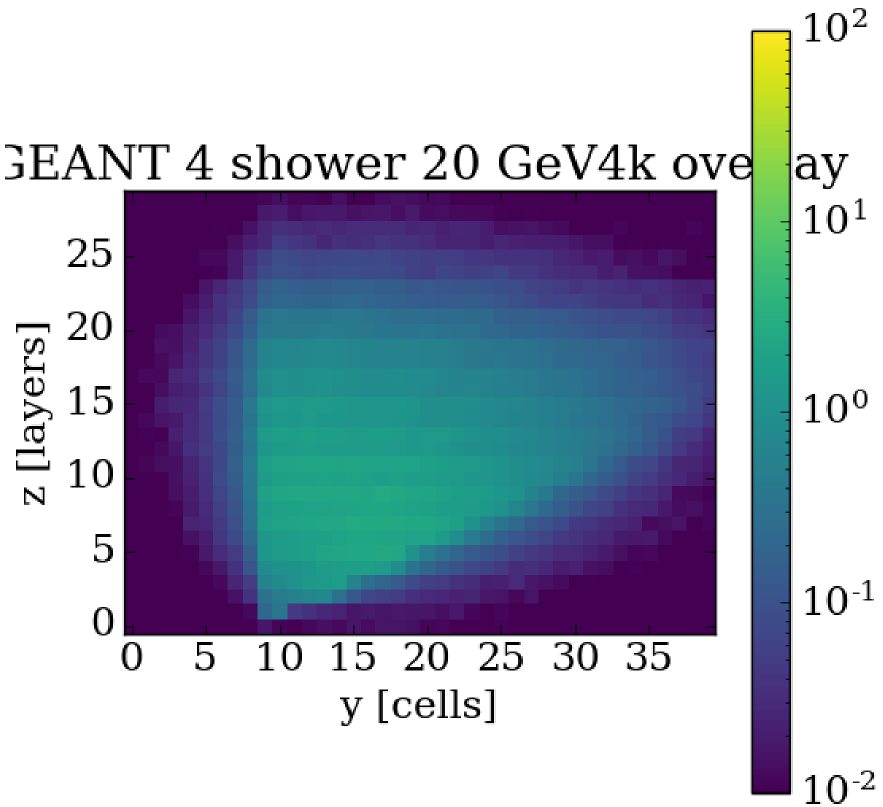


Reminder: These are core showers. Need to cut full LCIO for a fair comparison

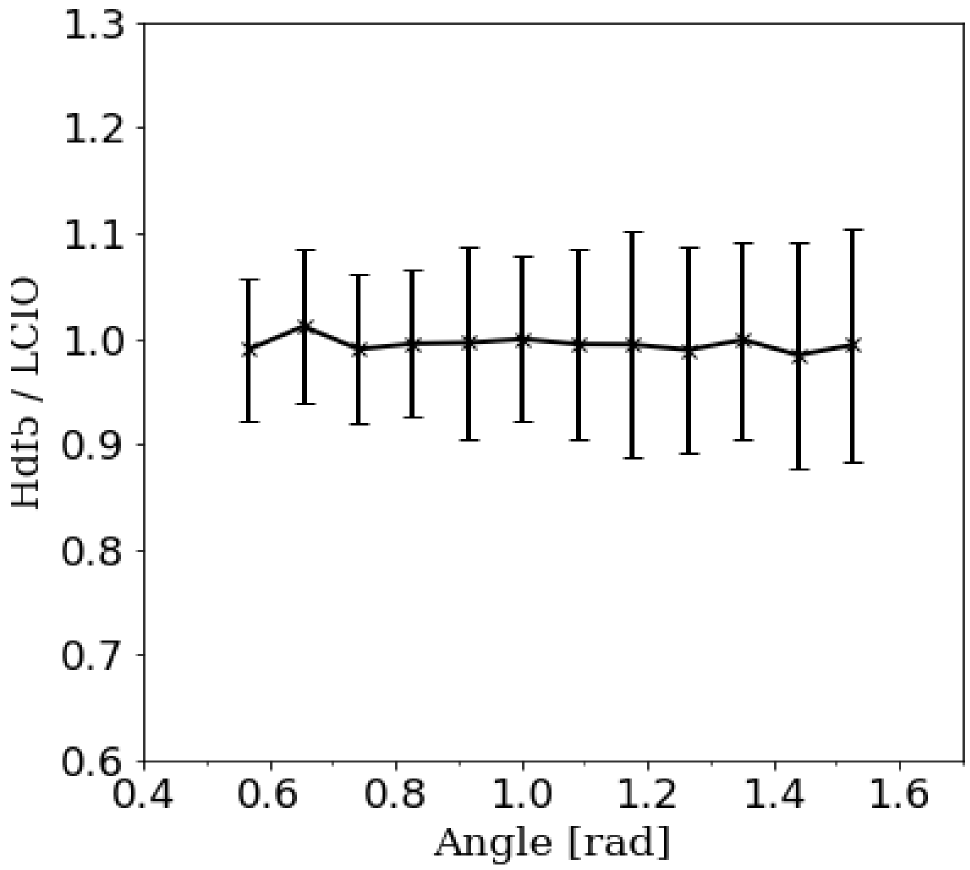


Can we contain full shower ?

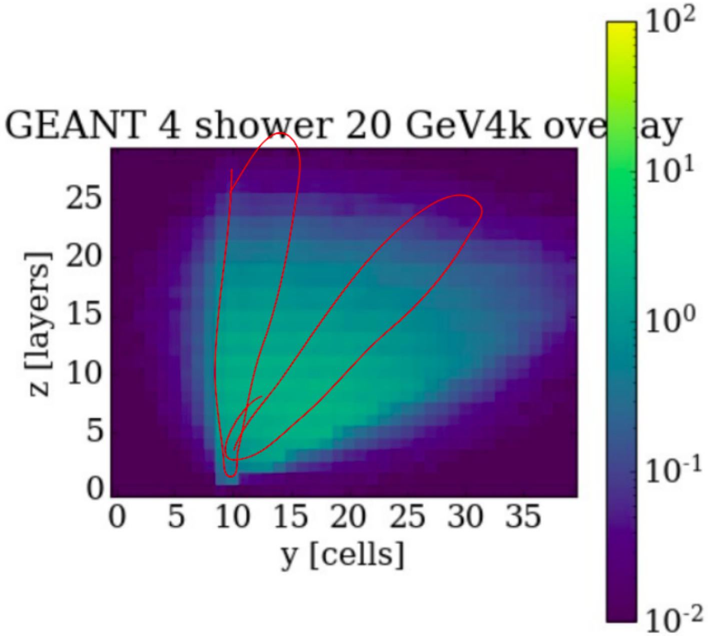
- **30x30x40** showers (layers, x, y) with extended y-coordinate
- Gun position is very close to ECAL: 1mm!
- Angle is from 90deg to 30deg



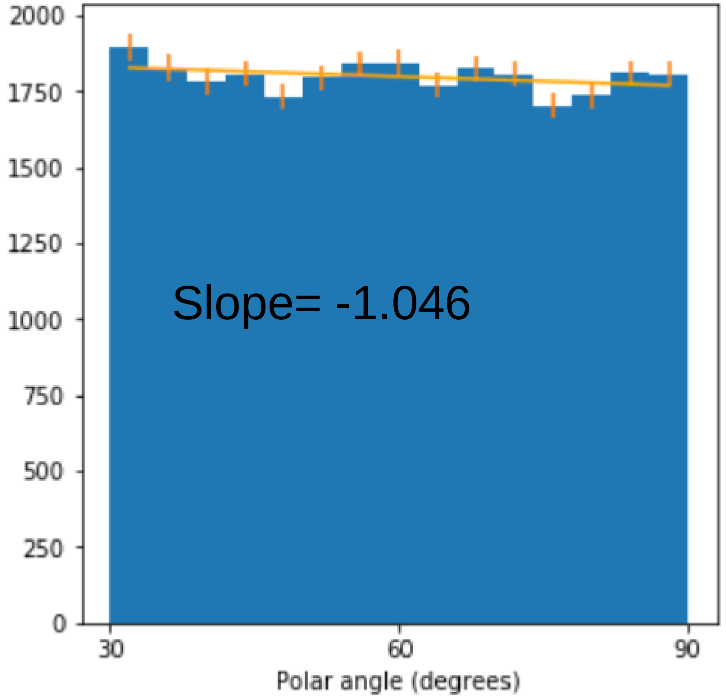
Sanity check



Issue: Why the angle is not *fully* uniform ?



Plot angle labels
from hdf5 file

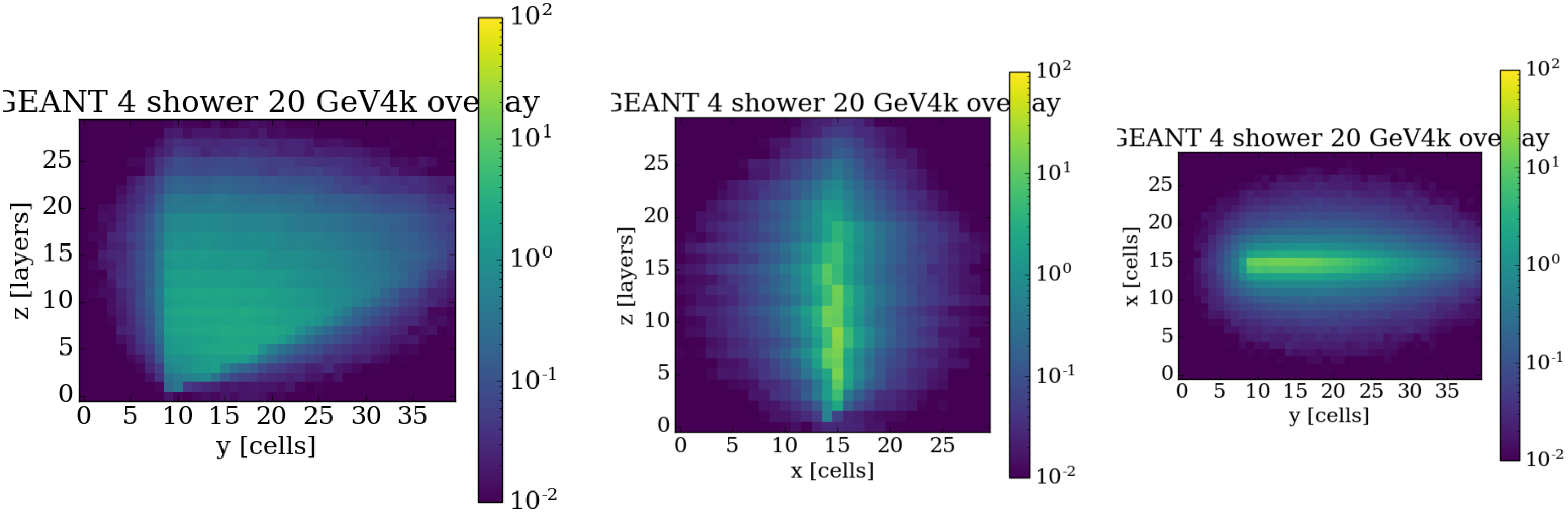



```
##
SIM.gun.distribution = 'uniform'
SIM.gun.energy = 20*GeV

##  isotropic distribution for the particle gun
##
##     use the options phiMin, phiMax, thetaMin, and thetaMax to limit the range of randomly distributed directions
##     if one of these options is not None the random distribution will be set to True and cannot be turned off!
##
SIM.gun.isotrop = False
SIM.gun.multiplicity = 1
SIM.gun.particle = "gamma"
SIM.gun.phiMax = 1.57079

## Minimal azimuthal angle for random distribution
SIM.gun.phiMin = 1.57079

##  position of the particle gun, 3 vector
SIM.gun.position = (0.0, 1810*mm, -5.0*cm)
SIM.gun.thetaMax = 1.57079
SIM.gun.thetaMin = 0.52360
```

ddsim config file in ILDConfig

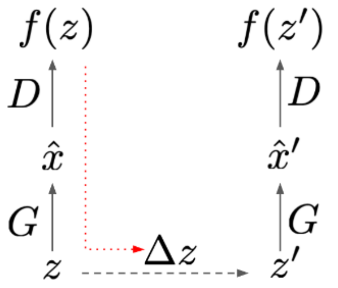
Photon Showers with angle

- **30x30x40** showers (layers, x, y) with extended y-coordinate
- Gun position is very close to ECAL: 1mm!
- Implemented corrections both x and y positions due to artifacts (due to irregularities)
- Angle is from 90deg to 30deg



A new WGAN

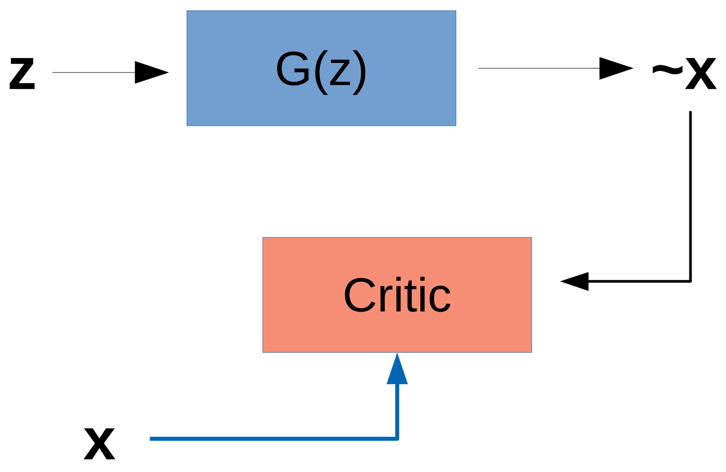
- Trained on pion showers. Approx half a million
- Shower is 48x13x13
- Architectures
  - very similar to WGAN in our “getting high paper”
  - Latent Optimized WGAN, inspired by DeepMind

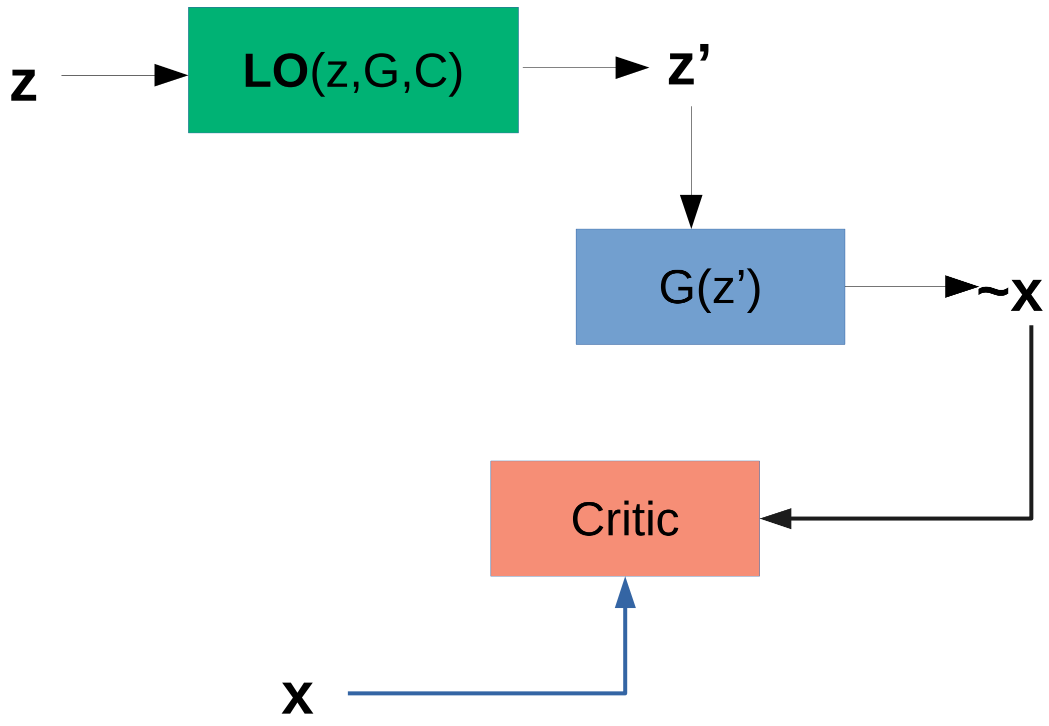


arXiv: 1912.00953

Figure 3: (**a**) Schematic of LOGAN. We first compute a forward pass through G and D with a sampled latent z. Then, we use gradients from the generator loss (dashed red arrow) to compute an improved latent, z'. After we use this optimised latent code in a second forward pass, we compute gradients of the discriminator back through the latent optimisation into the model parameters θ_D, θ_G. We use these gradients to update the model. (**b**) Truncation curves illustrate the FID/IS trade-off

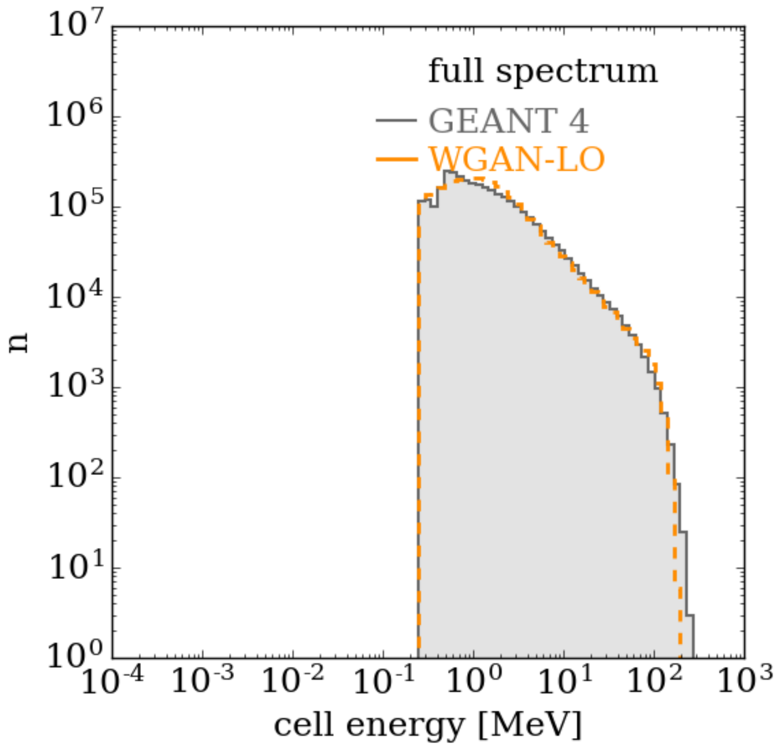
Our classical WGAN

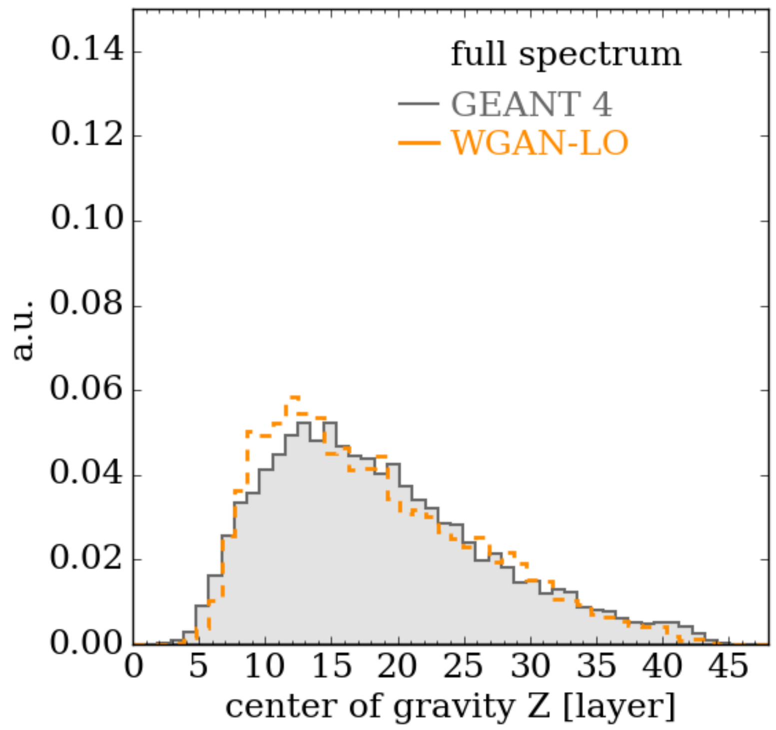


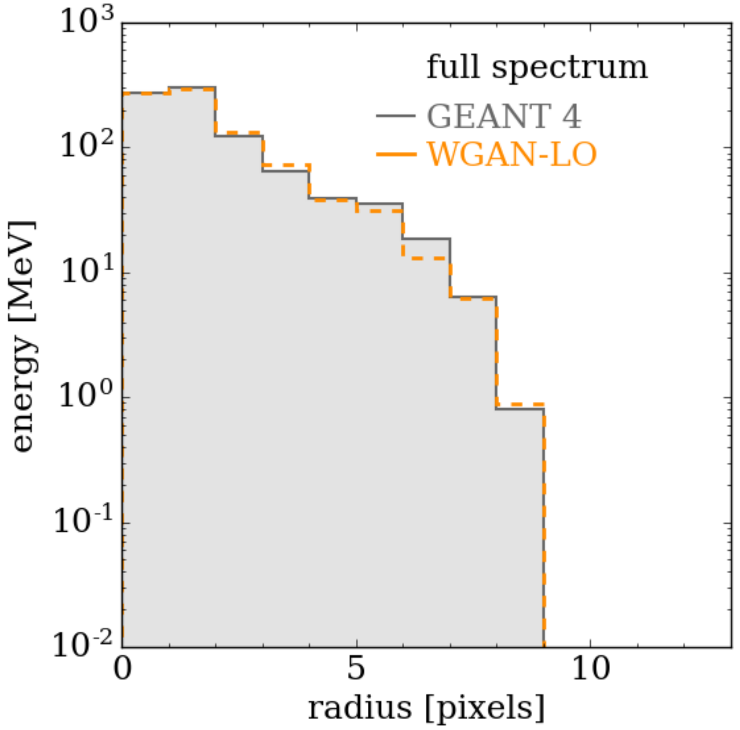


WGAN Latent Opt.

- Trained on **uniform energy showers 10-100 GeV**. Approx half a million
- Shower is 48x13x13

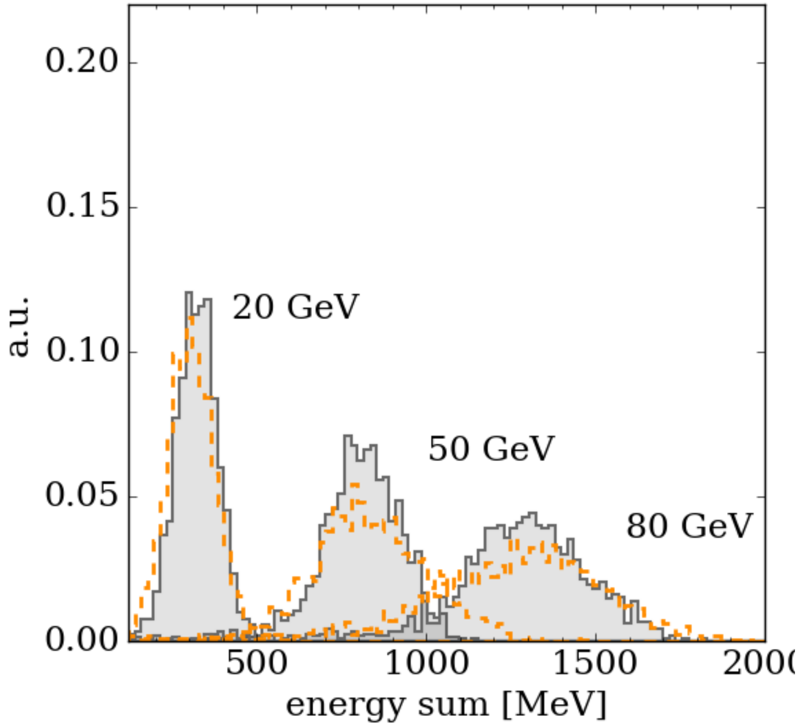


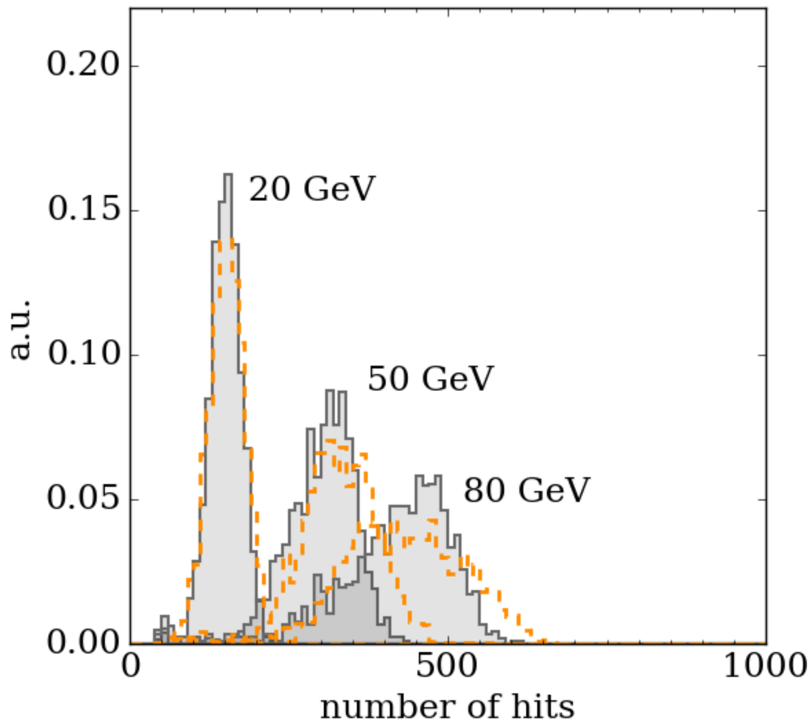


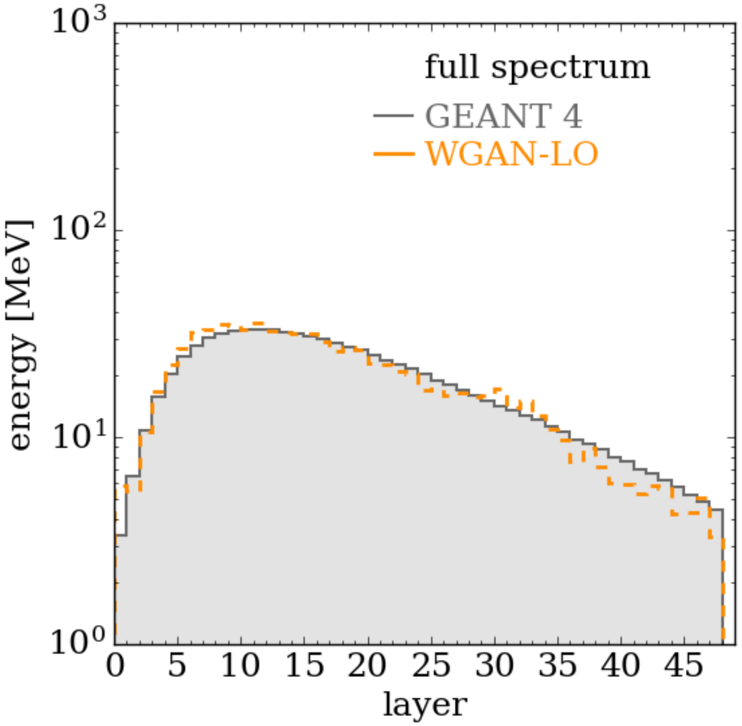


WGAN Latent Opt.

- Trained on **uniform energy showers 10-100 GeV**. Approx half a million
- Shower is 48x13x13

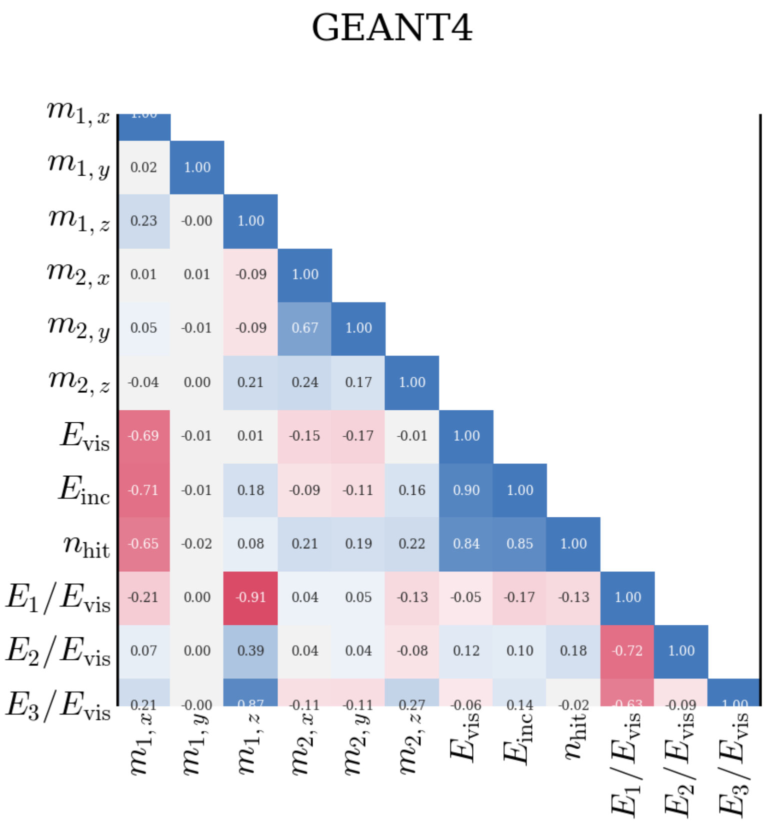


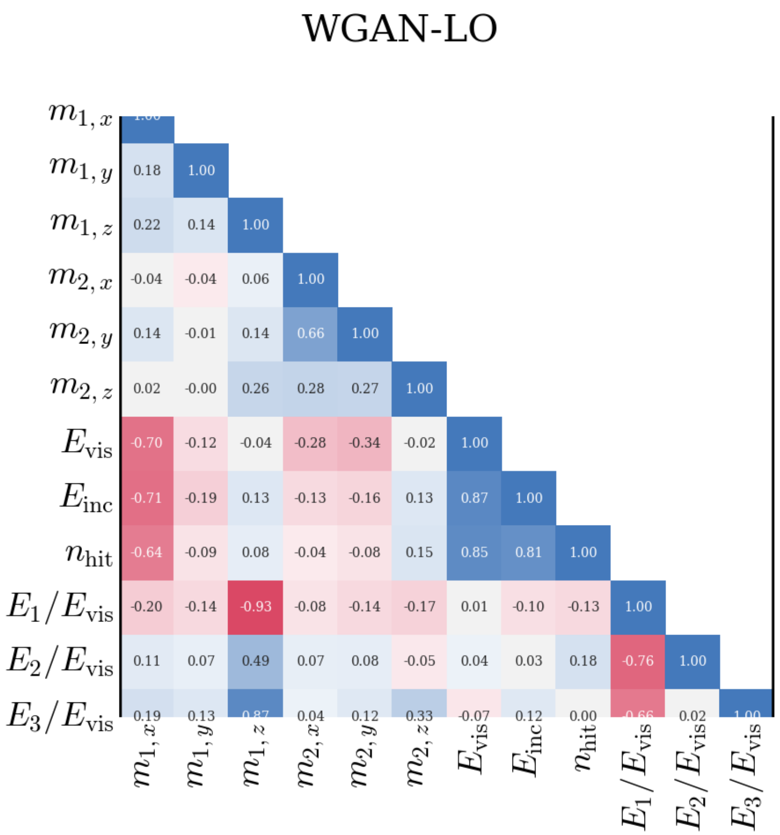


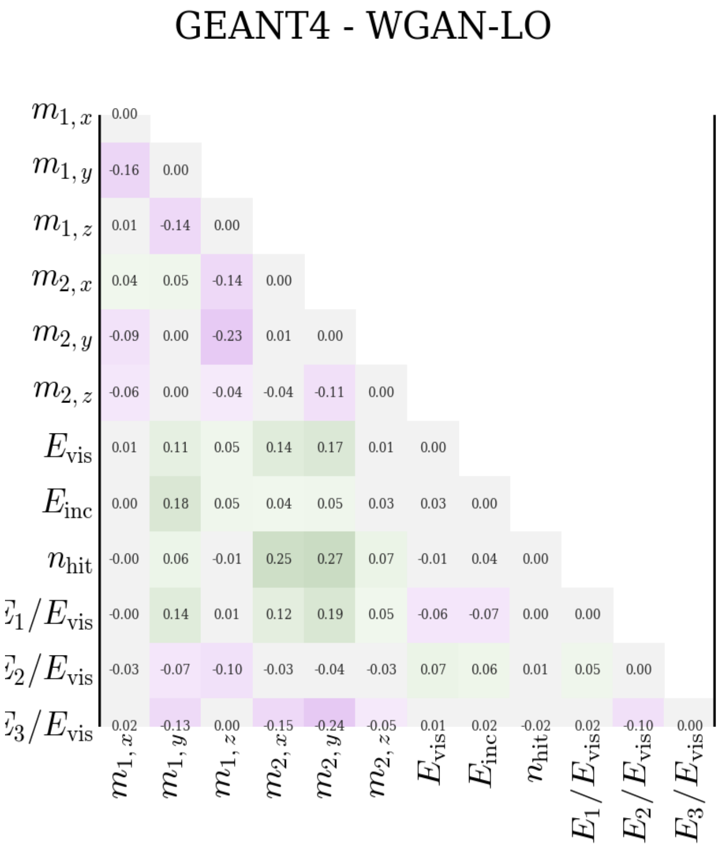


Fit to Gaussian for
linearity and width!!

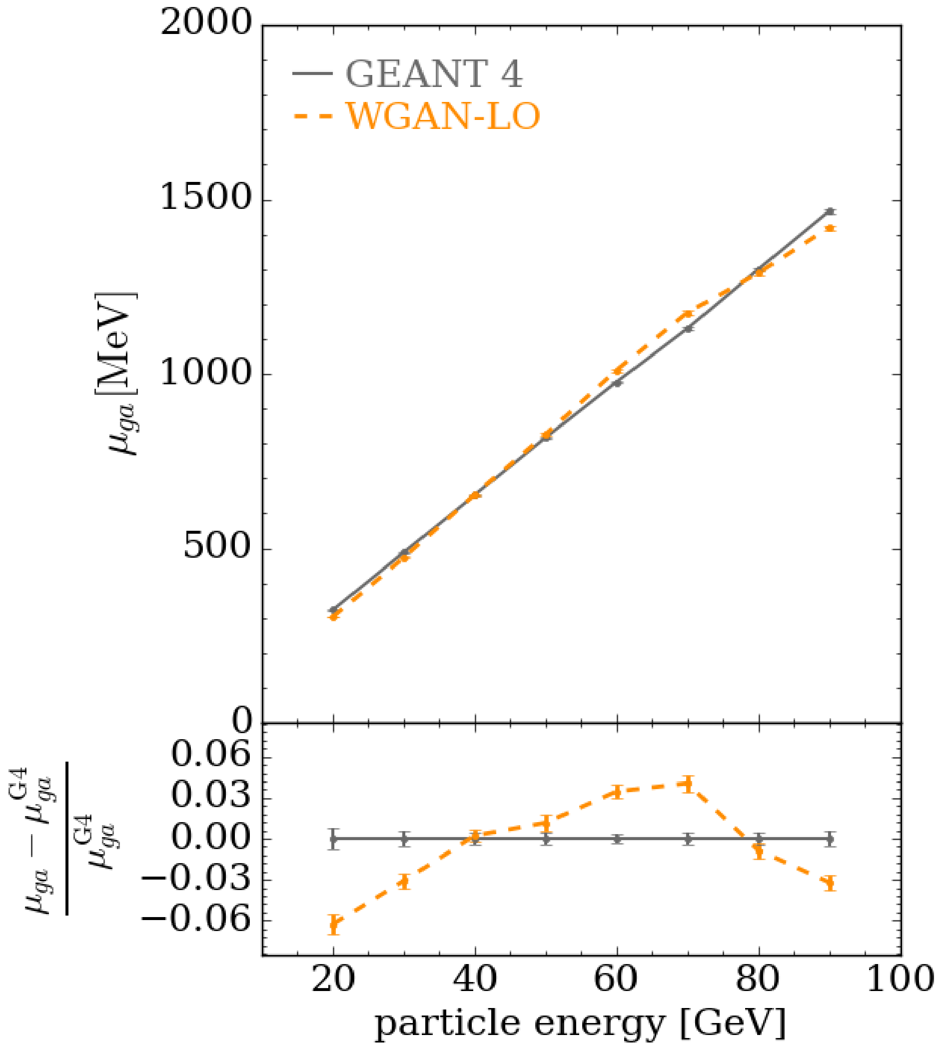
Linear Correlations

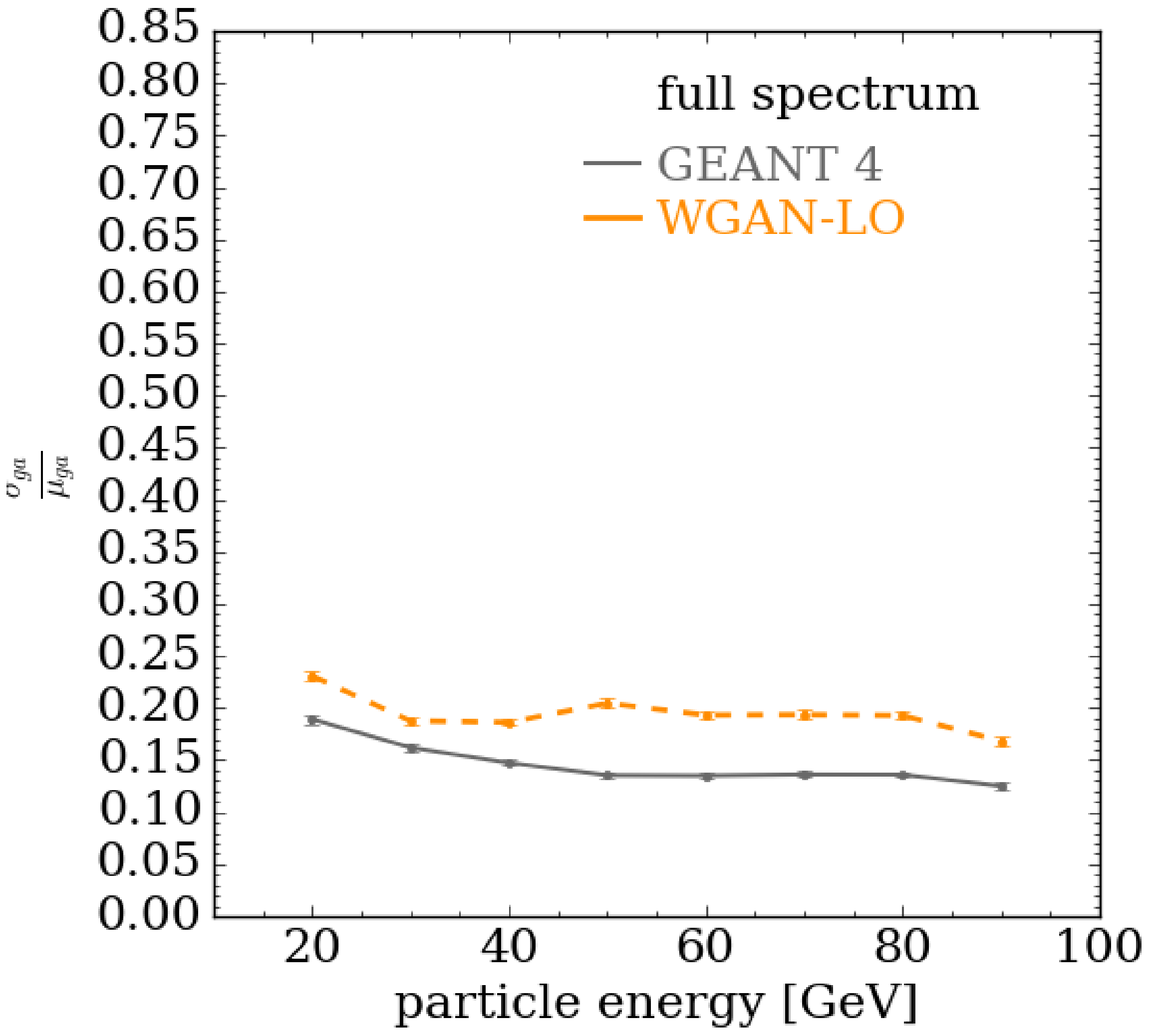




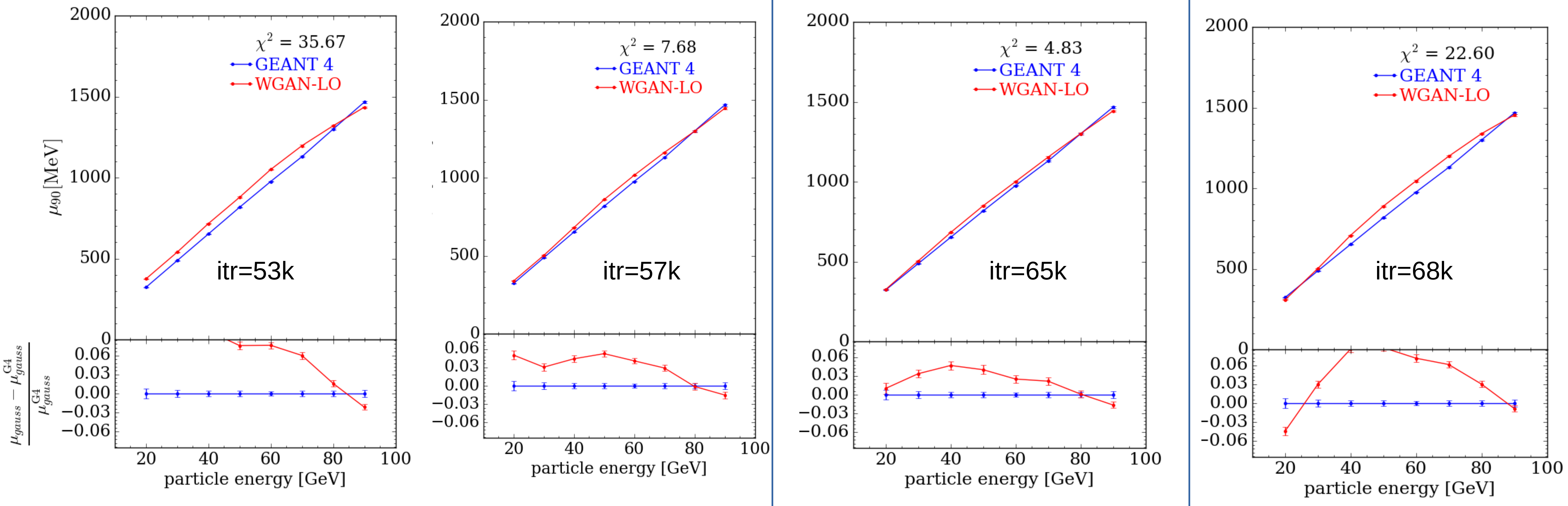


Linearity and Width





How to choose best iterations (i.e epoch)



The chosen

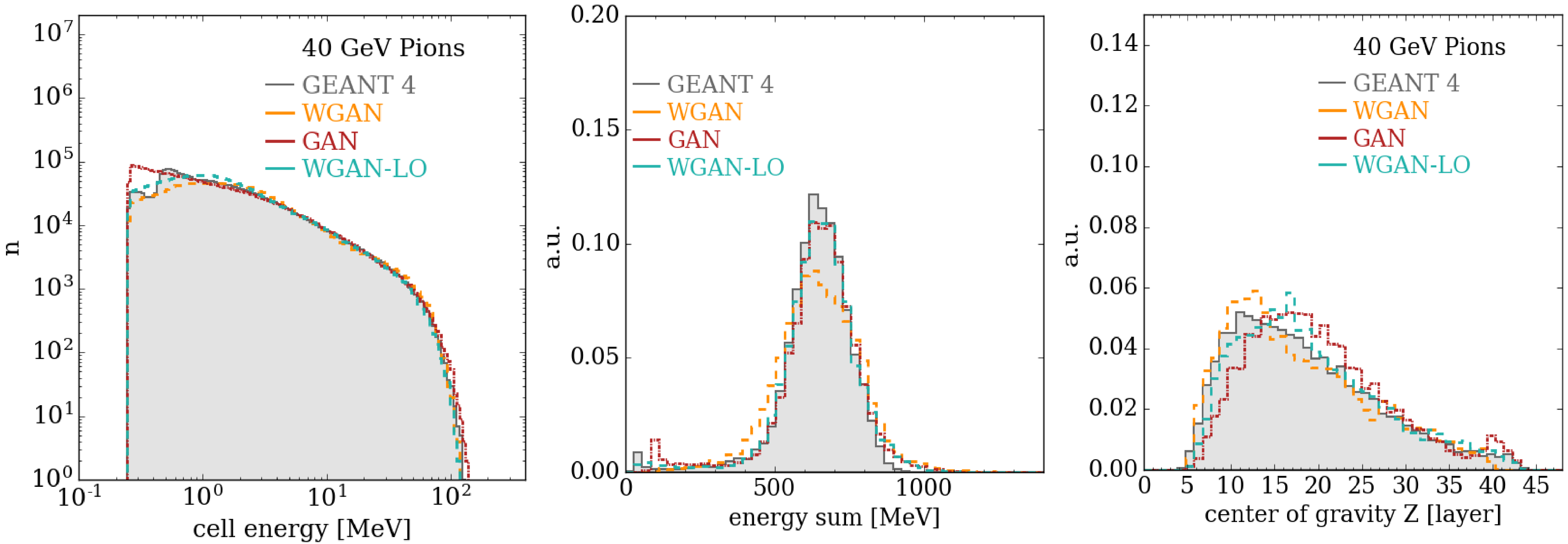
Example:

```
>>> chisquare([16, 18, 16, 14, 12, 12], f_exp=[16, 16, 16, 16, 16, 8])
(3.5, 0.62338762774958223)
```

Training

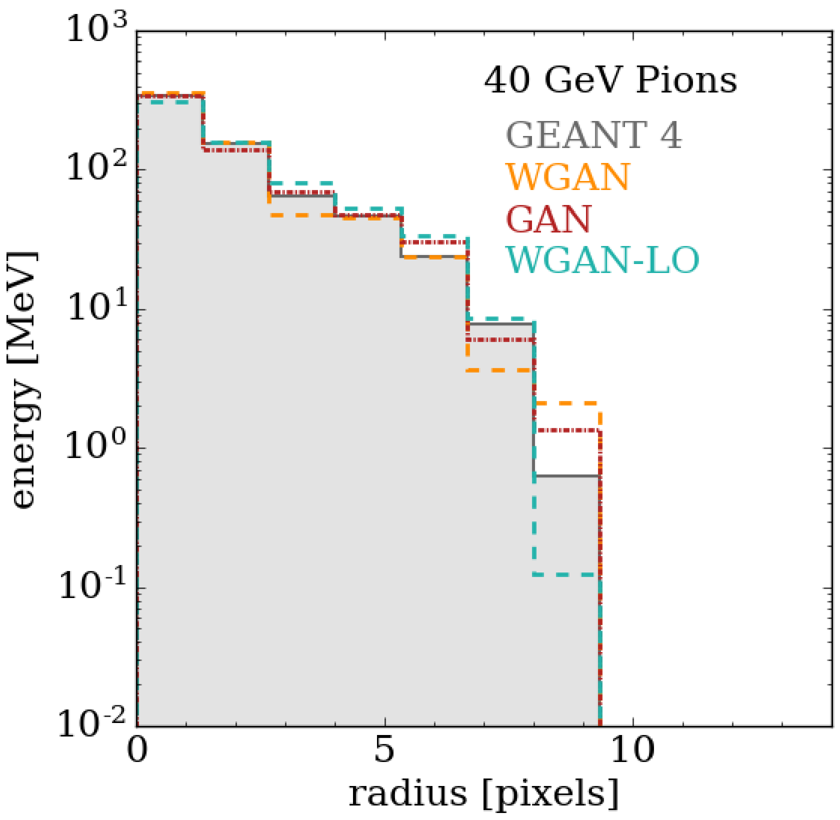
WGAN update

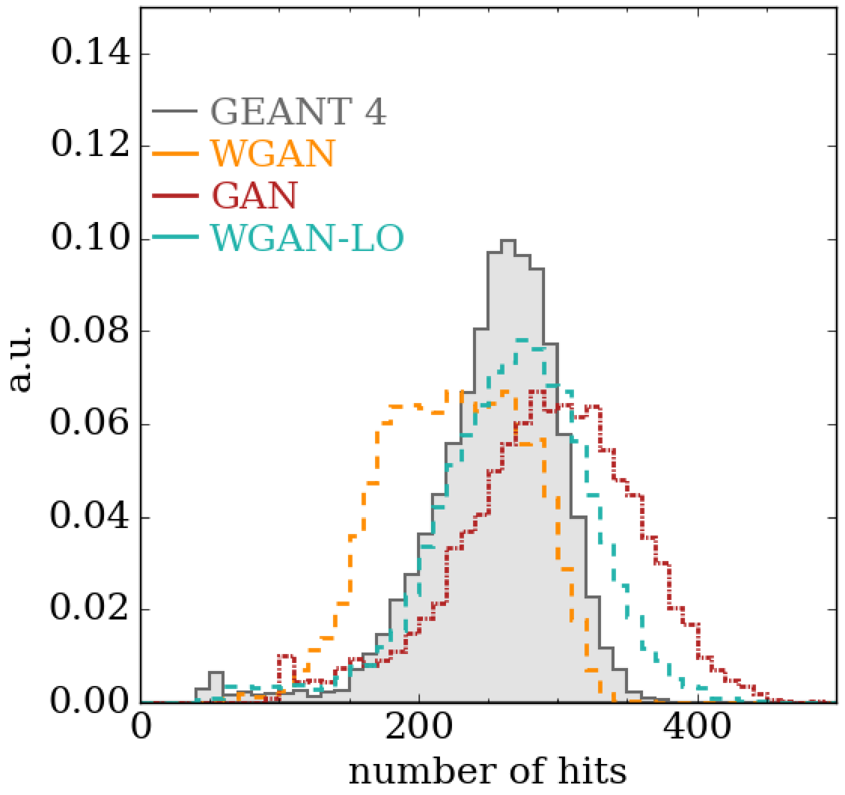
- Trained on 40 GeV showers. Approx half a million
- Shower is 48x13x13

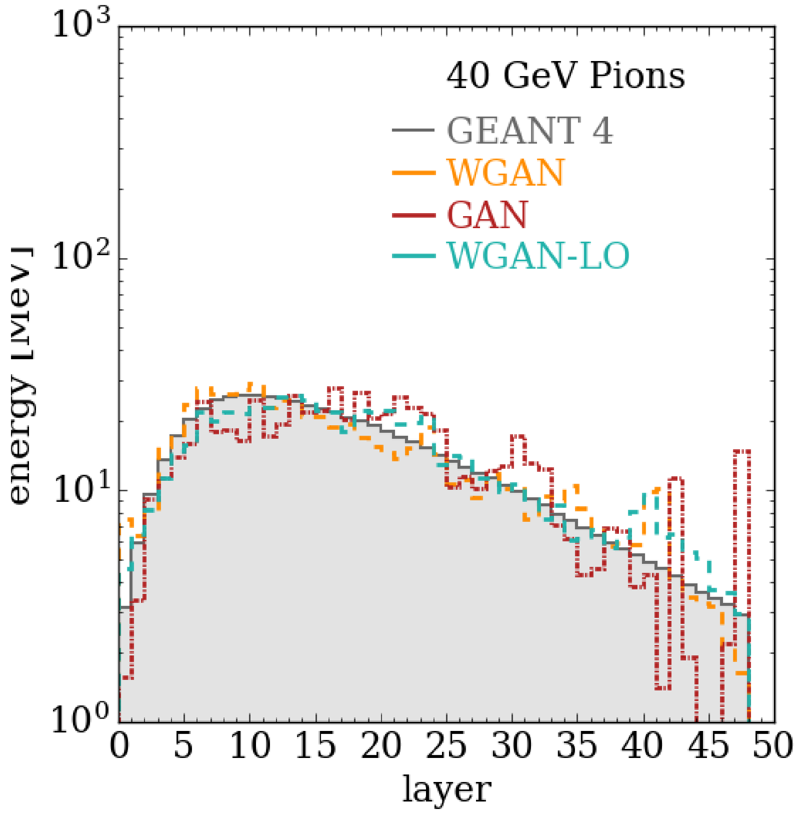


WGAN update

- Trained on 40 GeV showers. Approx half a million
- Shower is 48x13x13

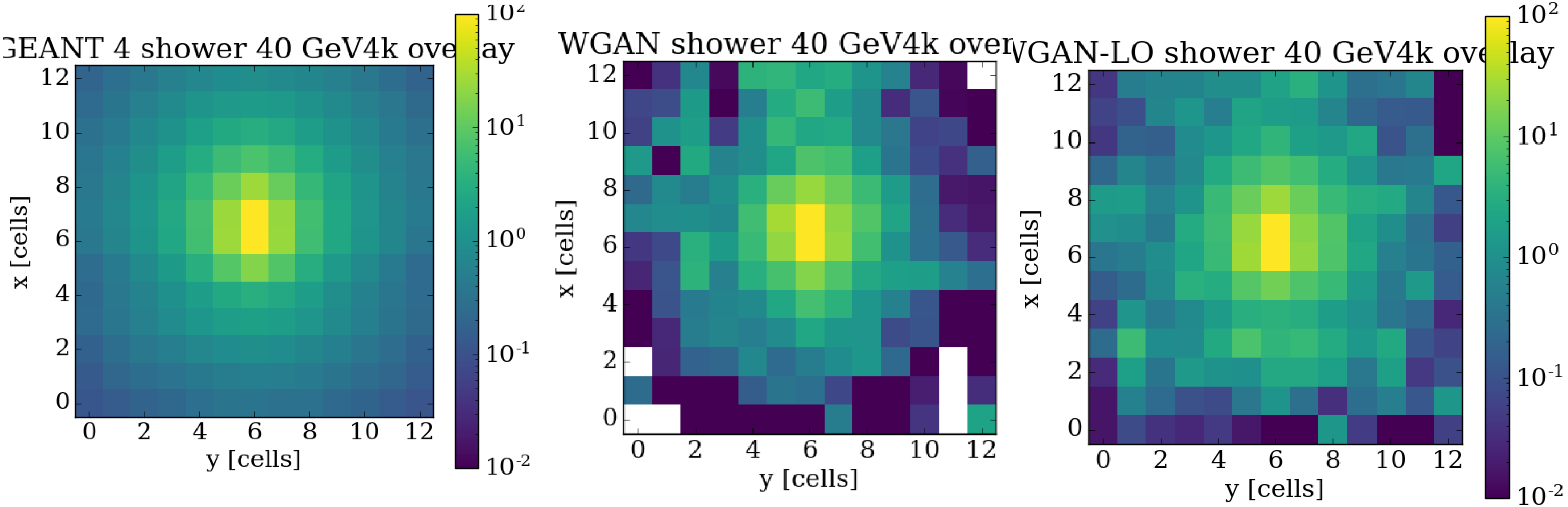






WGAN update

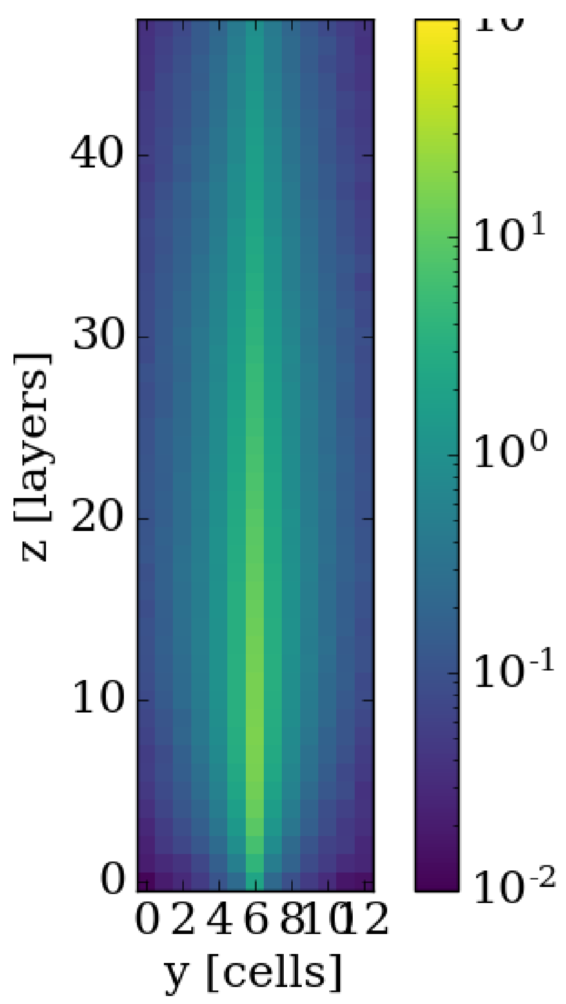
Overlay in X-Y



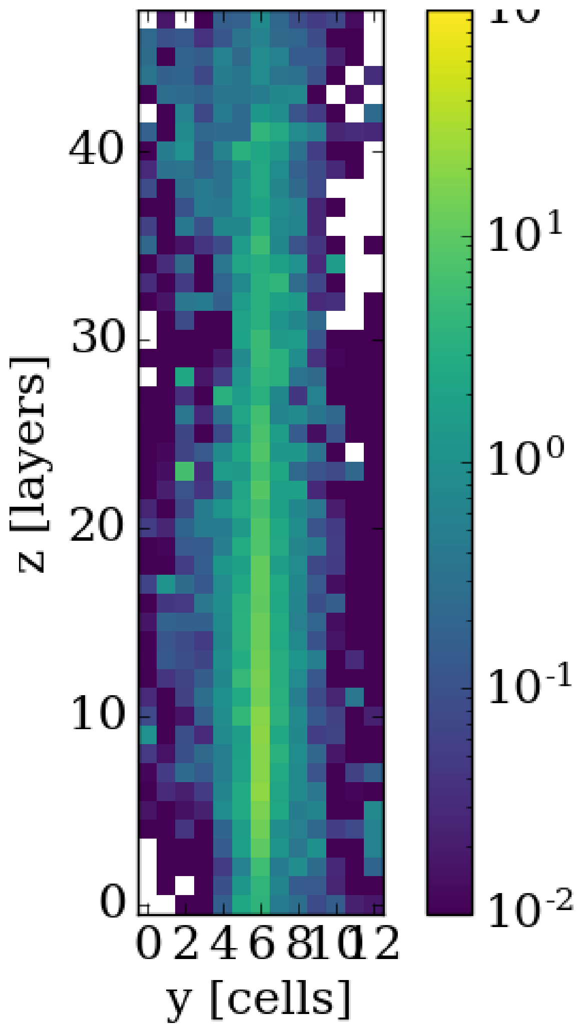
WGAN update

Overlay in Y-Z

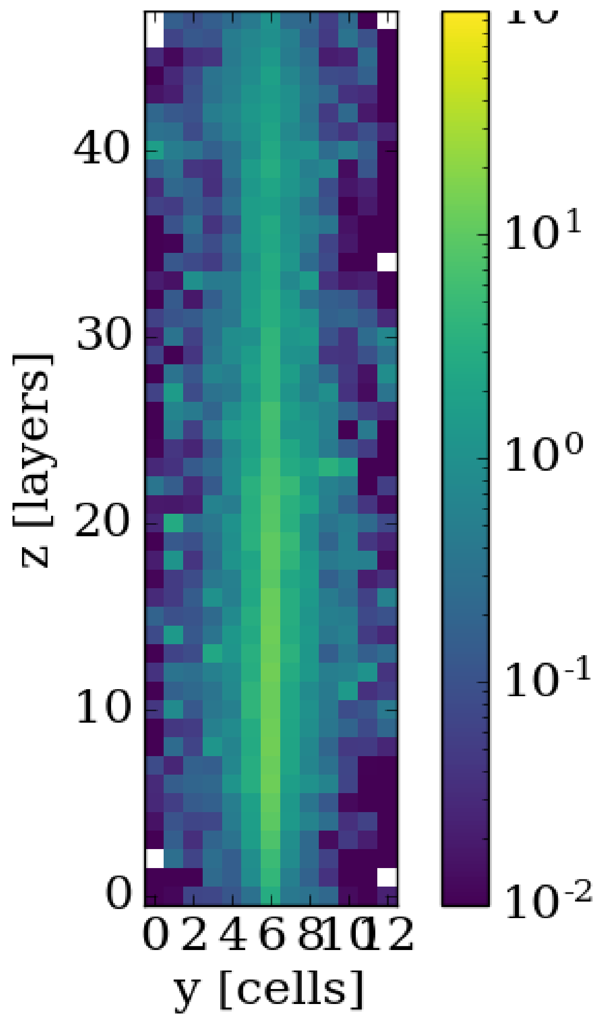
Geant4



WGAN



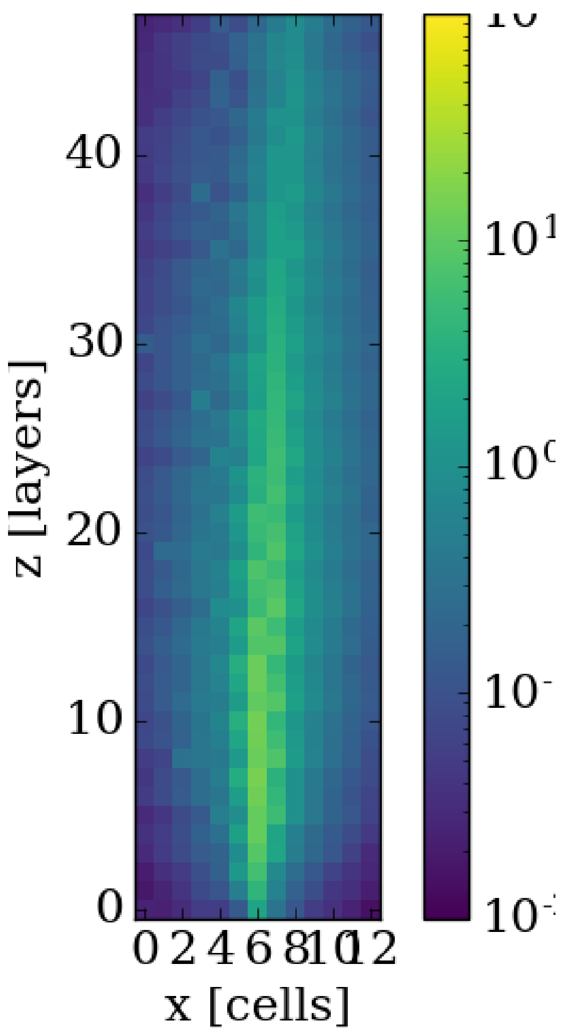
WGAN-LO



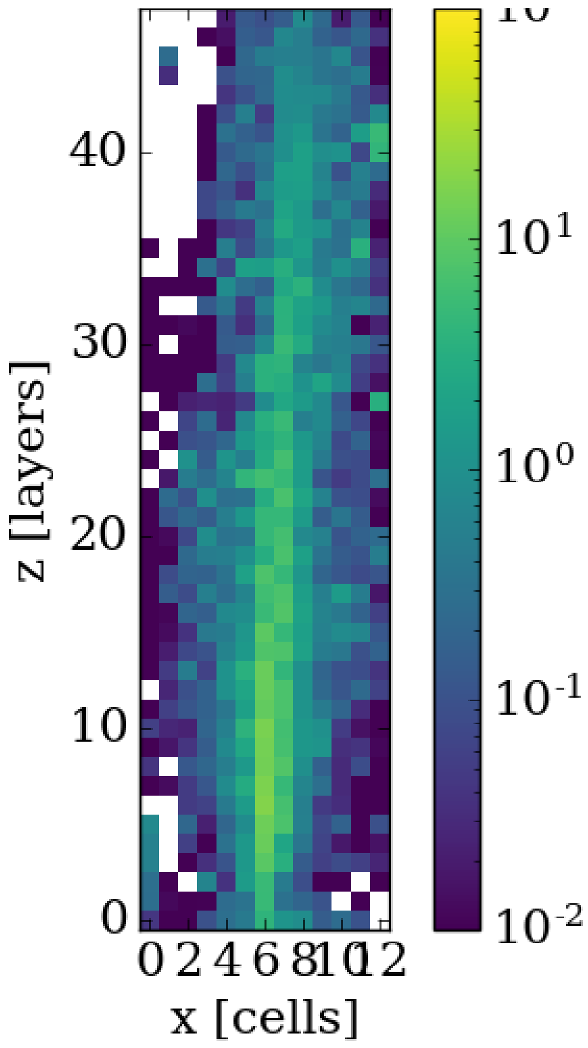
WGAN update

Overlay in Z-X

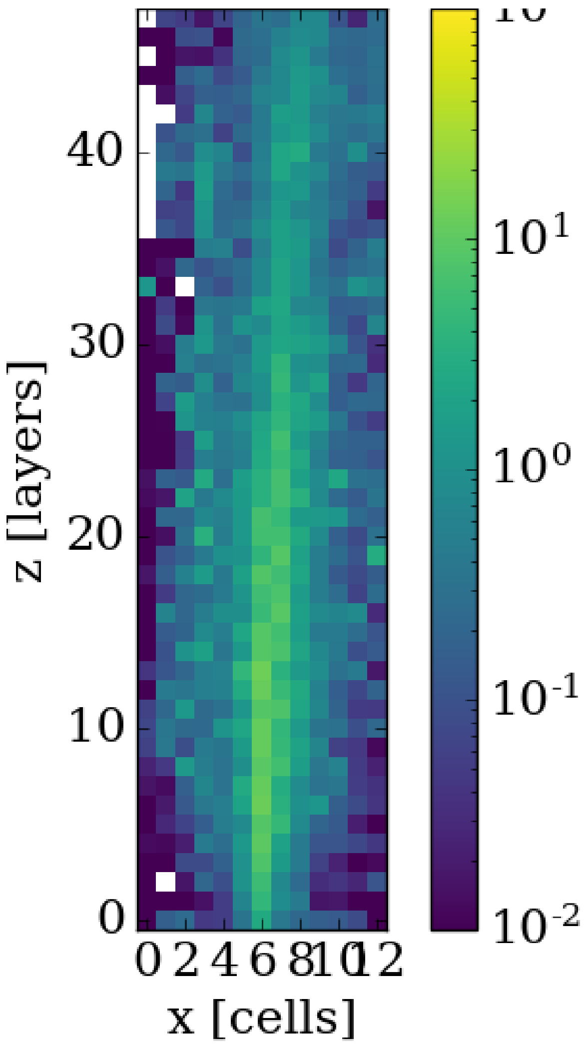
Geant4



WGAN

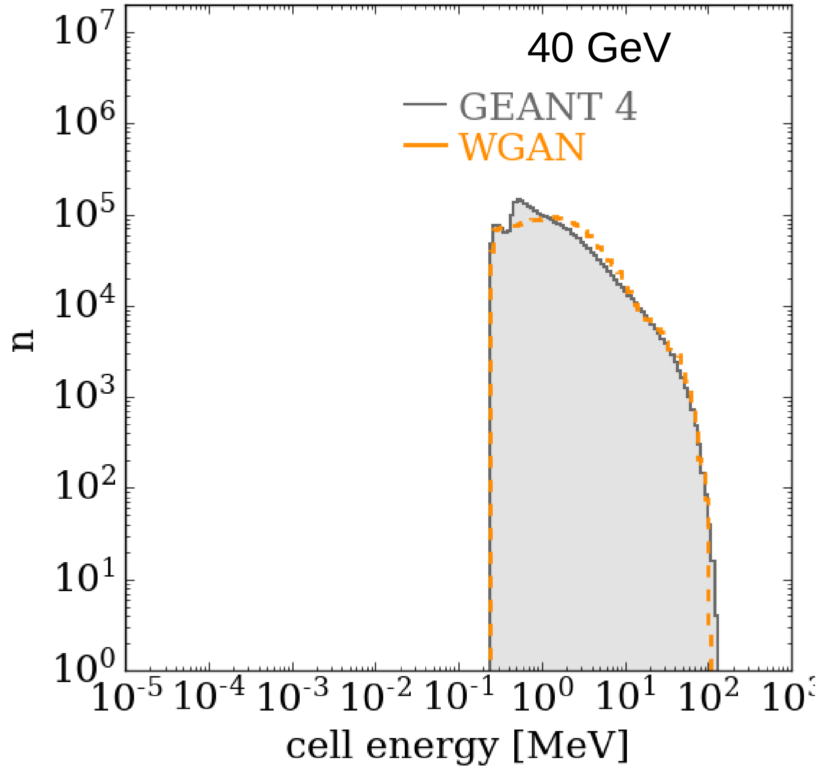


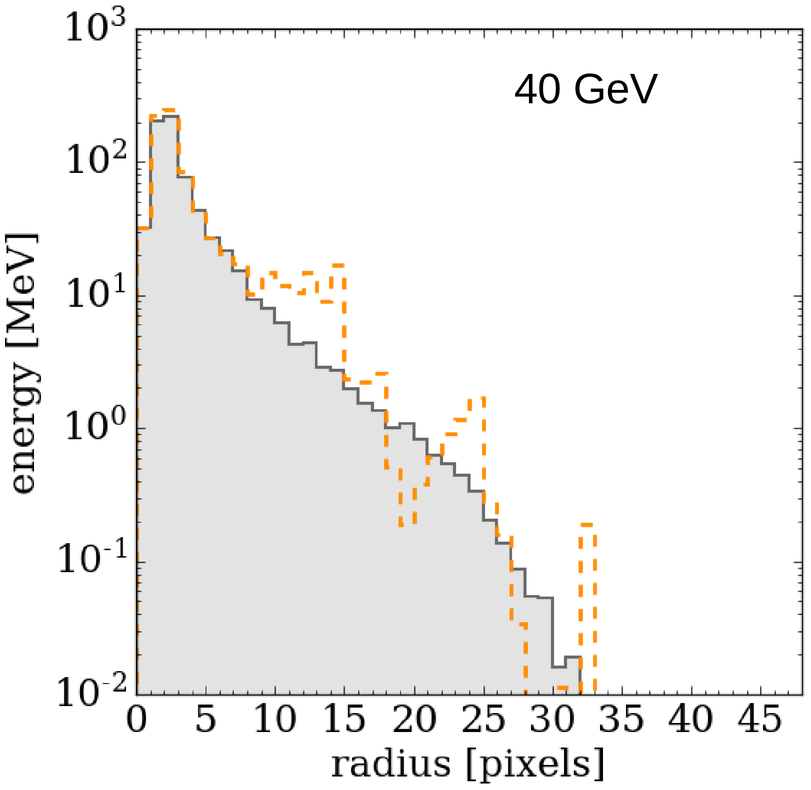
WGAN-LO

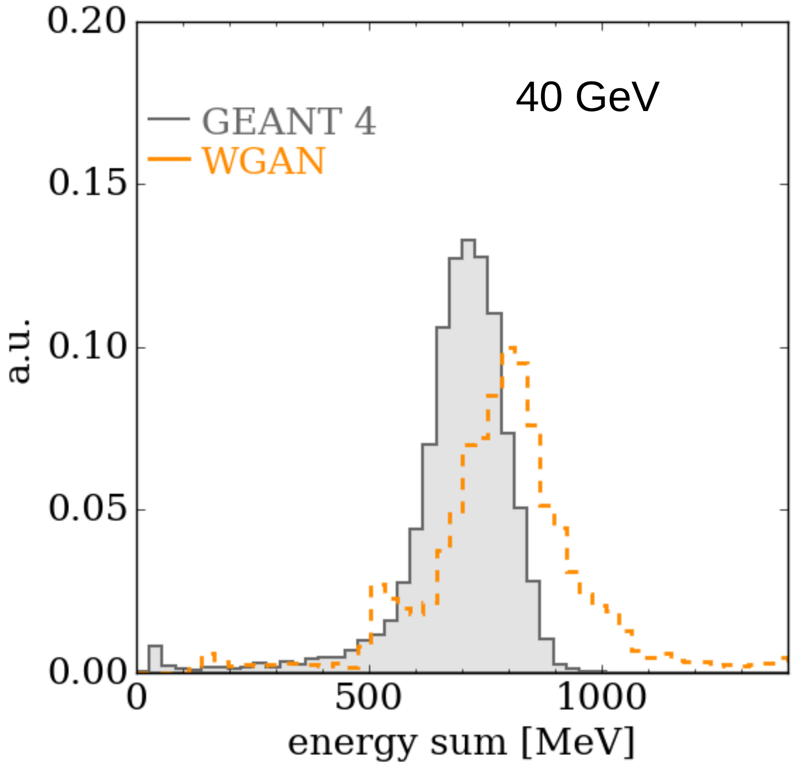


WGAN update

- Trained on 40 GeV showers. Approx half a million
- Shower is 48x48x48
- Architecture is very similar to WGAN in our “getting high paper”

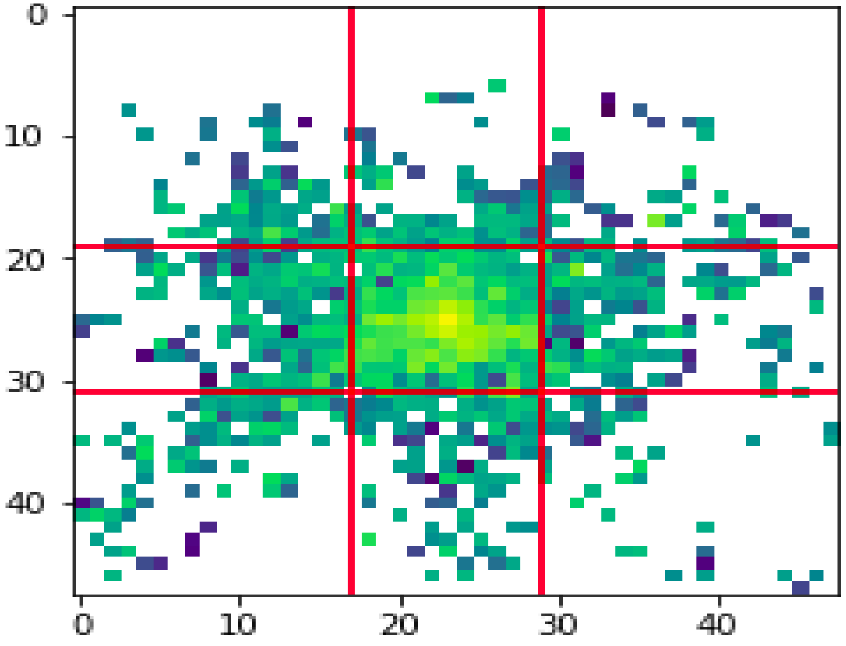




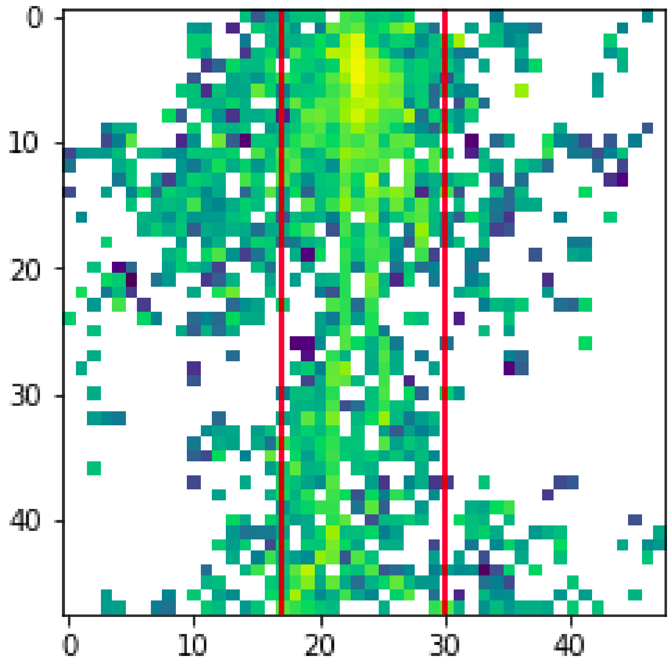


WGAN update: core

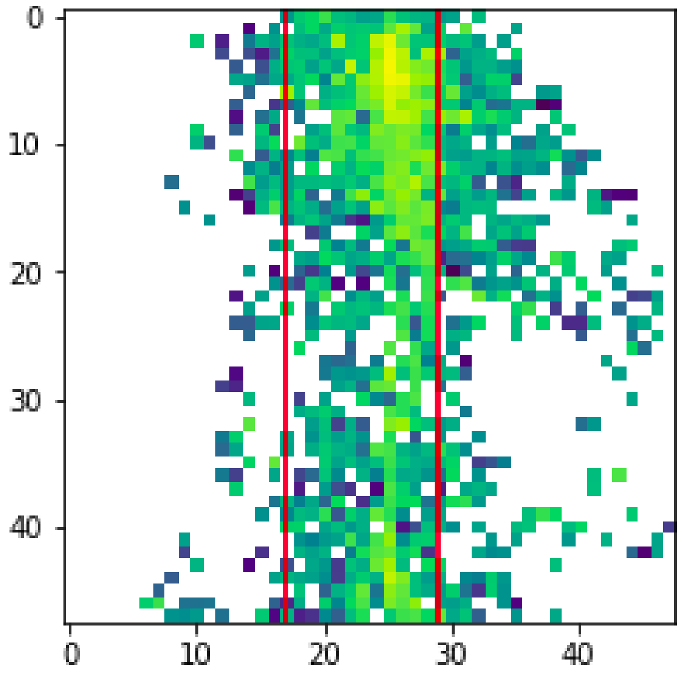
x-y



z-y

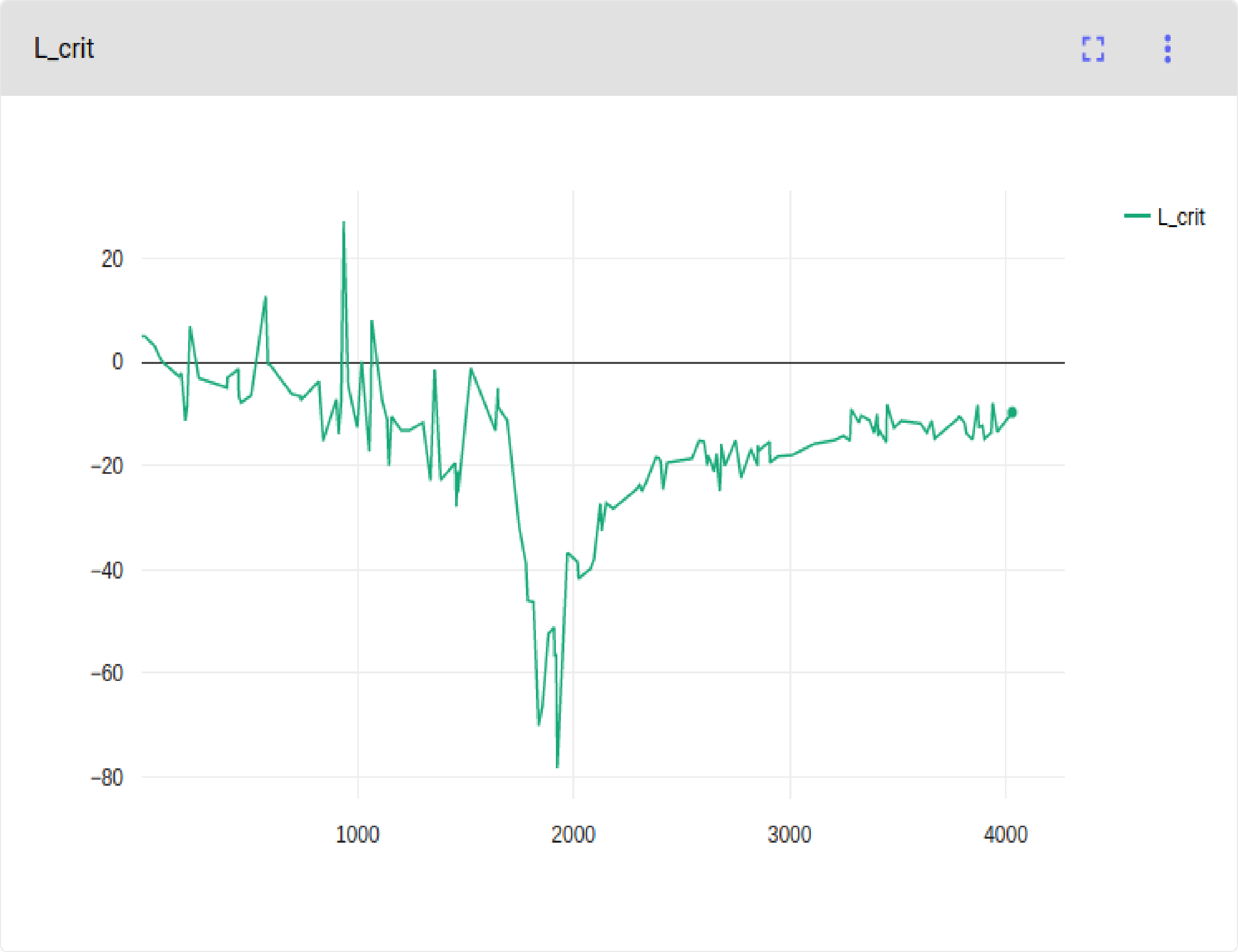


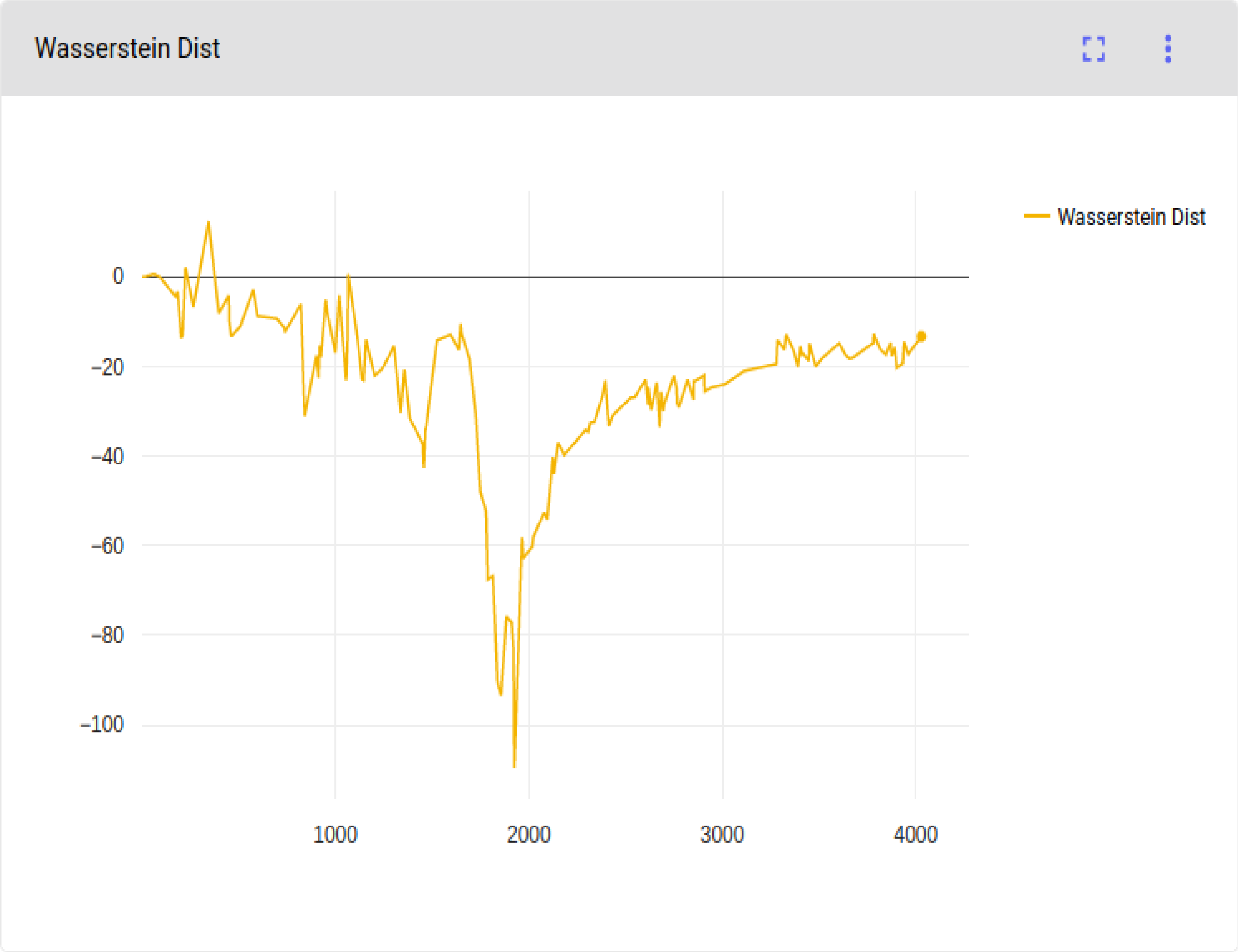
z-x



WGAN update: core

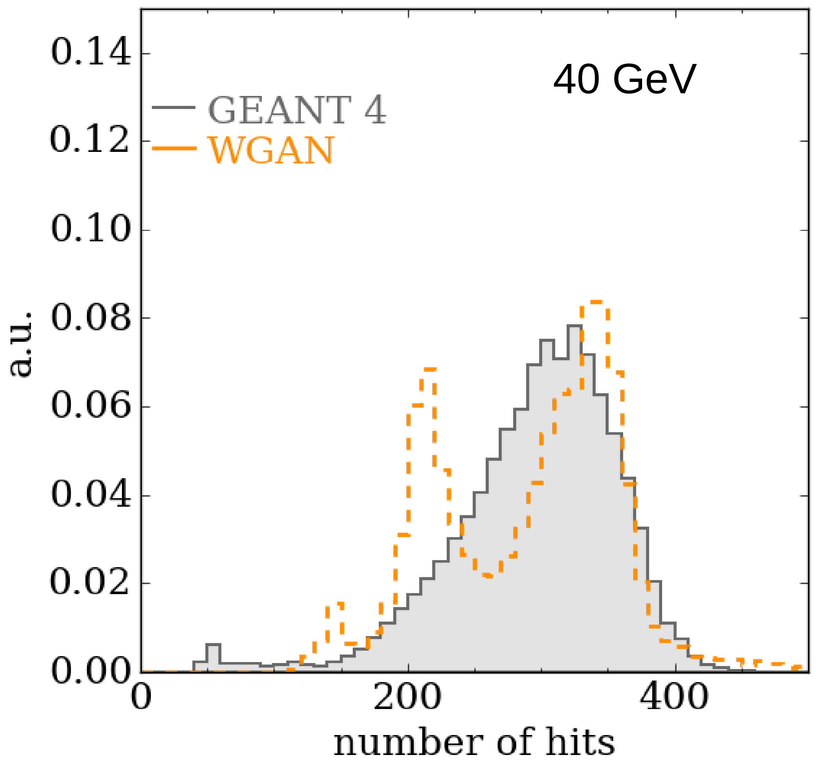
- Training started yesterday on 3 P100s

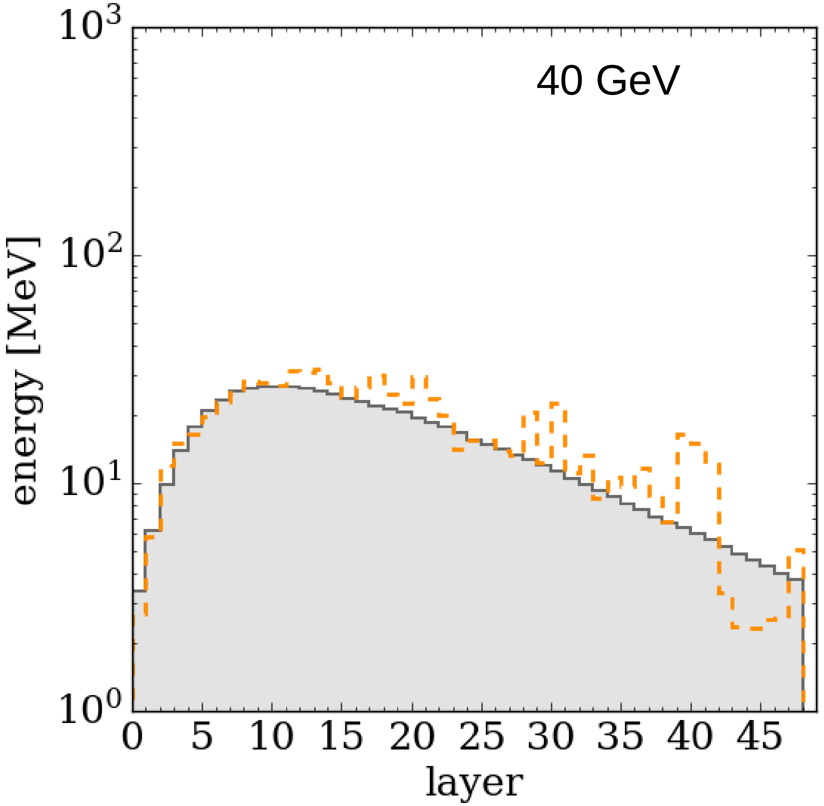


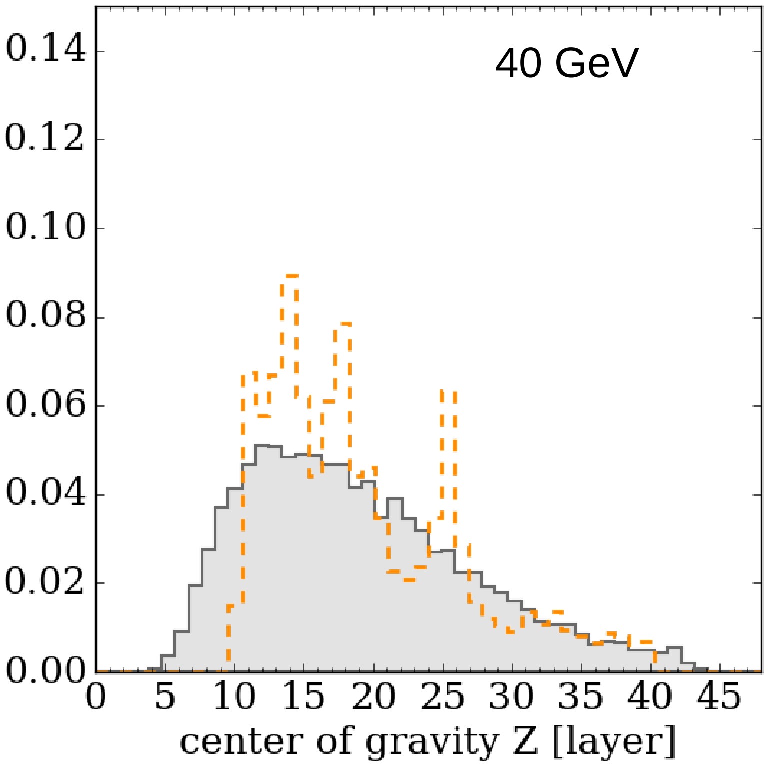


WGAN update

- Trained on 40 GeV showers. Approx half a million
- Shower is 48x48x48
- Architecture is very similar to WGAN in our “getting high paper”







WGAN update

Some examples