Explore the opportunity of applying Generative AI in High Energy Physics

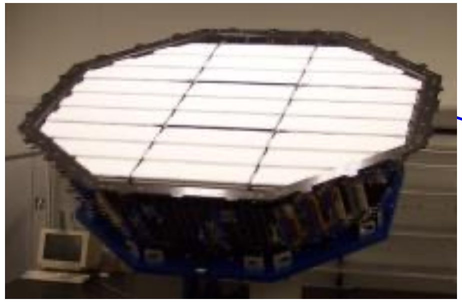
Hsin-Yi Chou

Institute of Physics, Academia Sinica

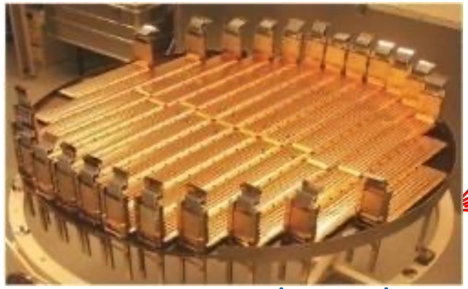
Oct 30, 2024

AMS (A TeV precision, multipurpose magnetic spectrometer)

Transition Radiation Detector (TRD)
Identify $p^{\pm}/e^{\pm}$

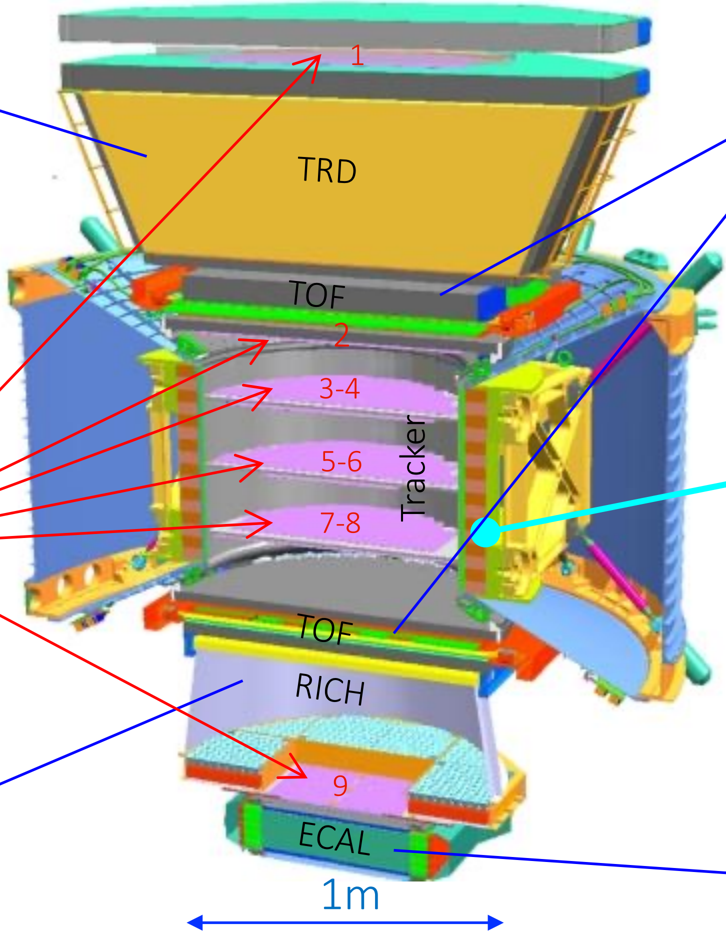


Silicon Tracker (Tracker)
Z, P



Ring Imaging Cherenkov (RICH)
Z, β

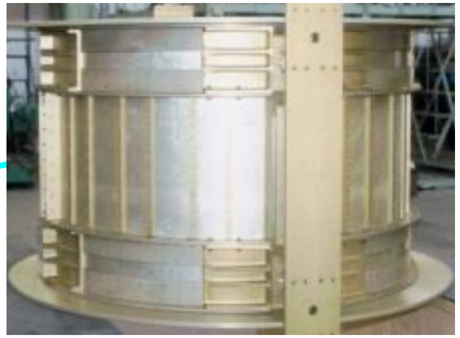
The charge (Z) and energy (E) or rigidity ($R \equiv P/Z$) are measurement independently by several detectors



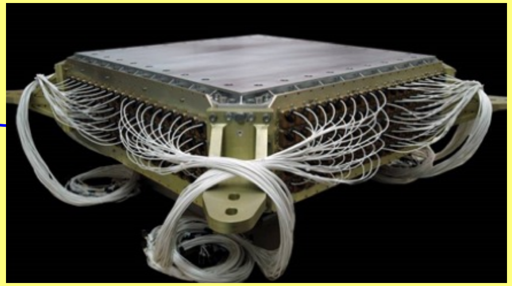
Volume: 5m x 4m x 3m
Weight: 7.5 tons

Time of Flight (TOF)
Z, β

Magnet (~0.14 Tesla)
$\pm Z$



Electromagnetic Calorimeter (ECAL)
E of $e^{\pm}$ and Identify $p^{\pm}/e^{\pm}$



Absolute Charge $|Z|$
Tracker & TOF

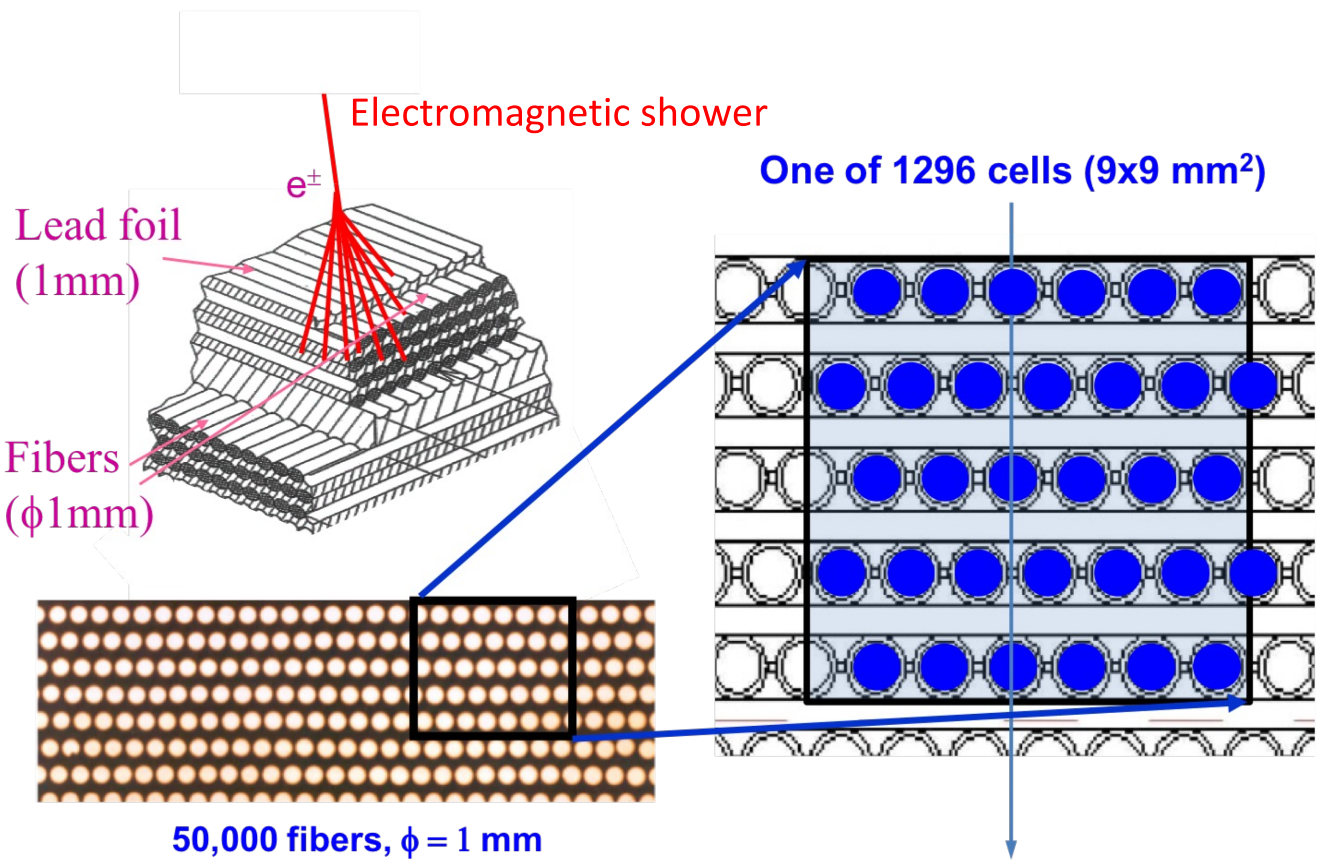
Energy of $e^{\pm}$
ECAL

Charge-Sign
Rigidity R
Tracker with Magnet

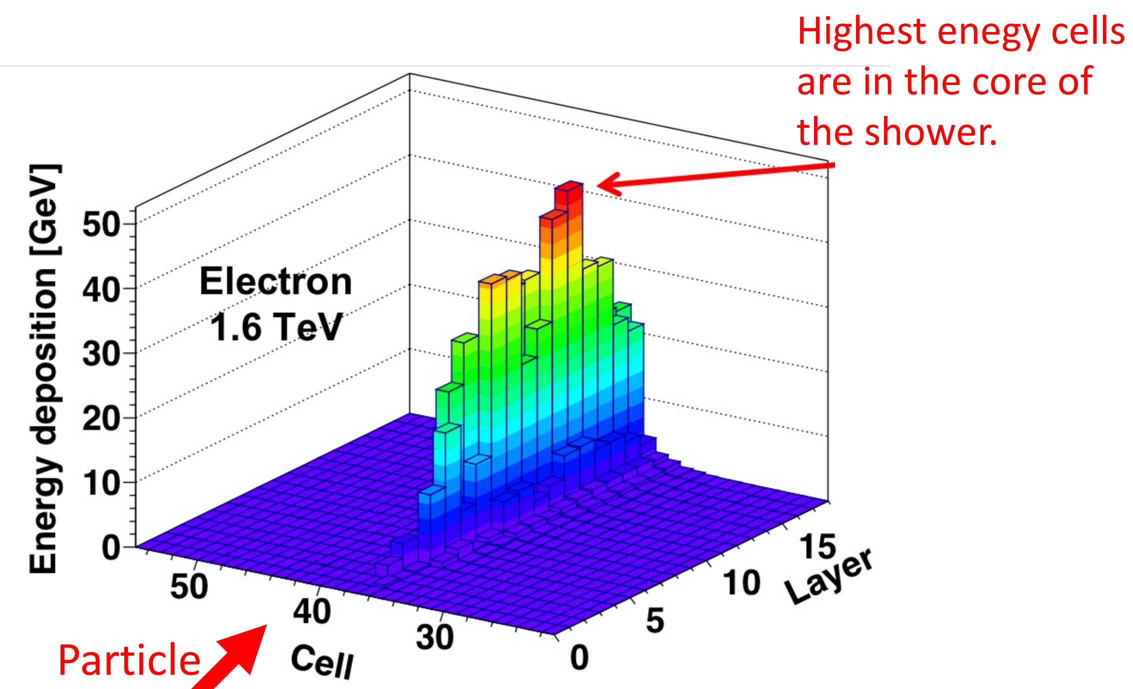
Velocity $\beta = v/c$
TOF & RICH

$p^{\pm}/e^{\pm}$ identification
TRD & ECAL

AMS Electromagnetic Calorimeter (ECAL)



50,000 fibers, ϕ = 1 mm distributed uniformly inside 1,200 lb of lead

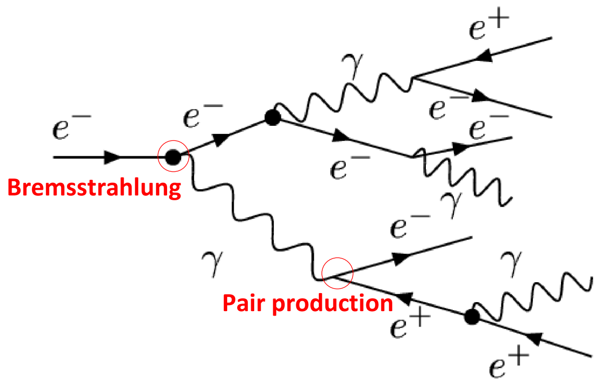


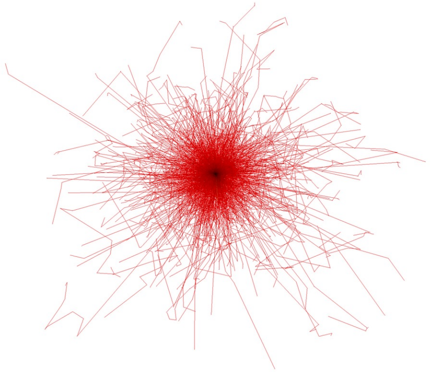
1296 cells = 72 cells/layer x 18 layers

Direction Projection of Layers (10 in Z-Y, 8 in Z-X)

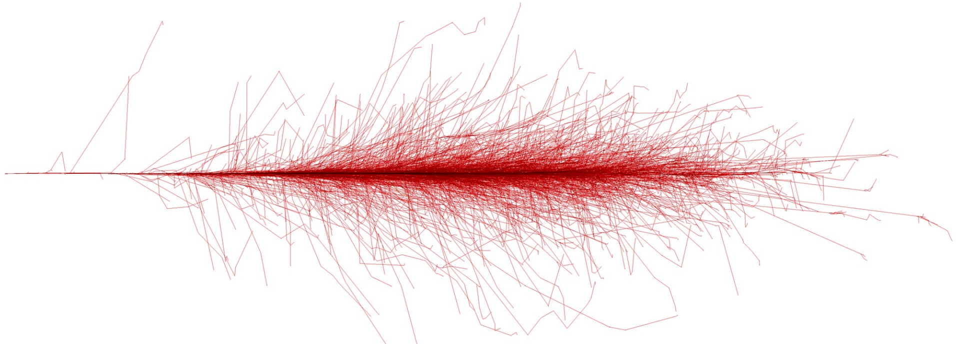
Y	Y	X	X	Y	Y	X	X	Y	Y	X	X	Y	Y	X	X	Y	Y

Electromagnetic (EM) Shower ($e^{\pm}$)



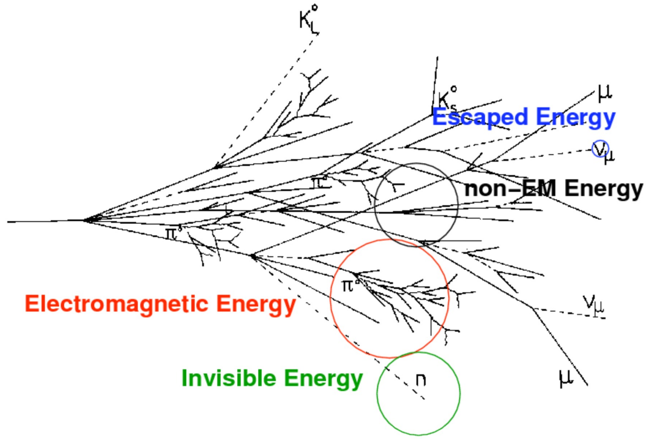


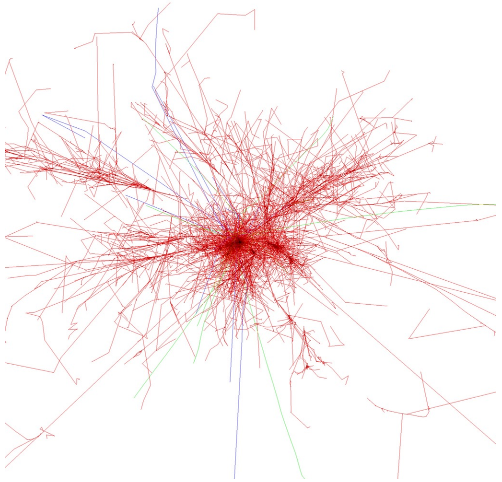
X-Y Projection



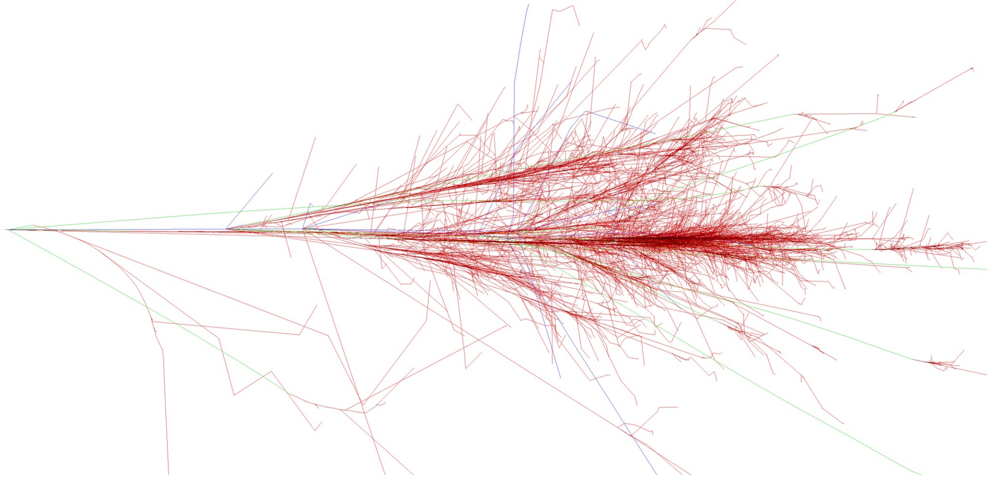
Z-X Projection

Hadron Shower ($p^{\pm}$)



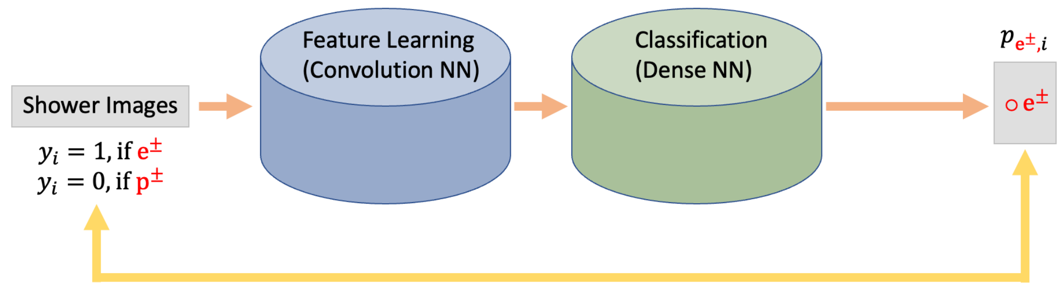


X-Y Projection



Z-X Projection

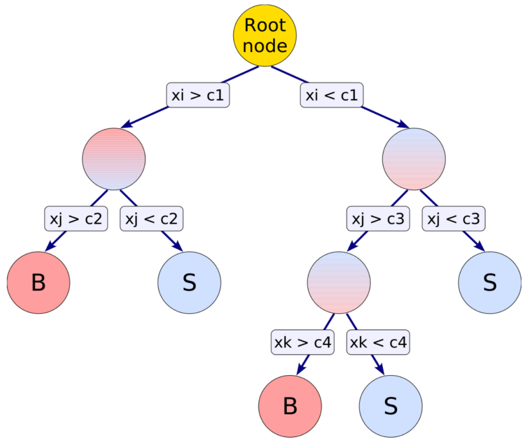
Classification



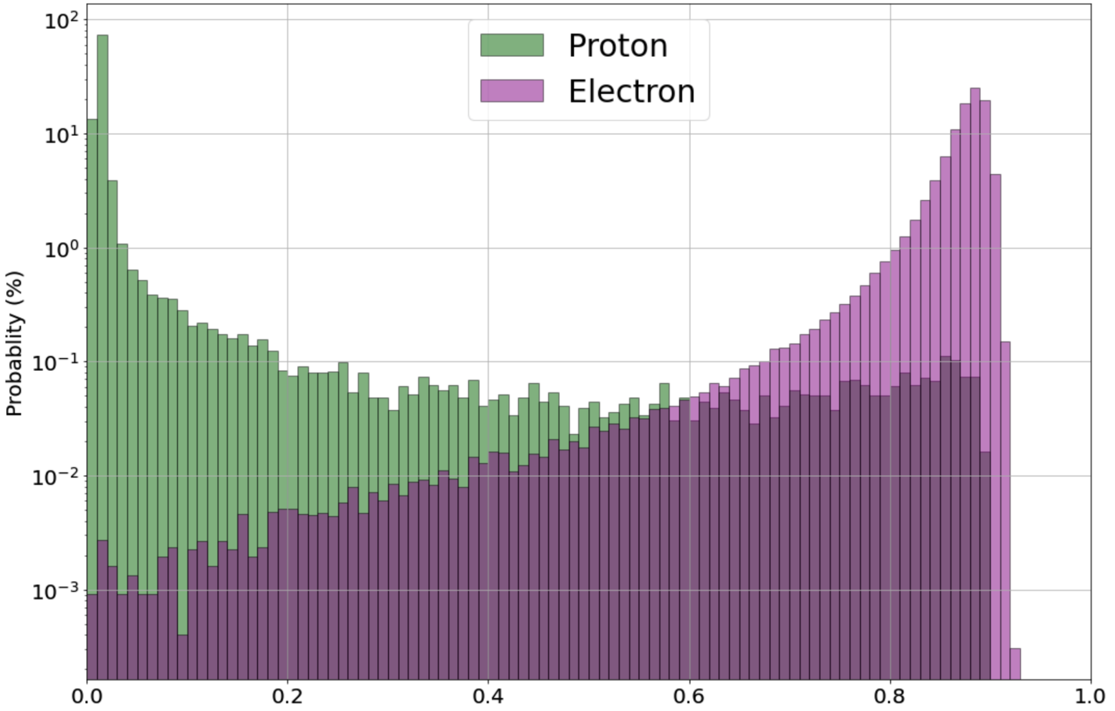
Loss function: Binary Cross Entropy

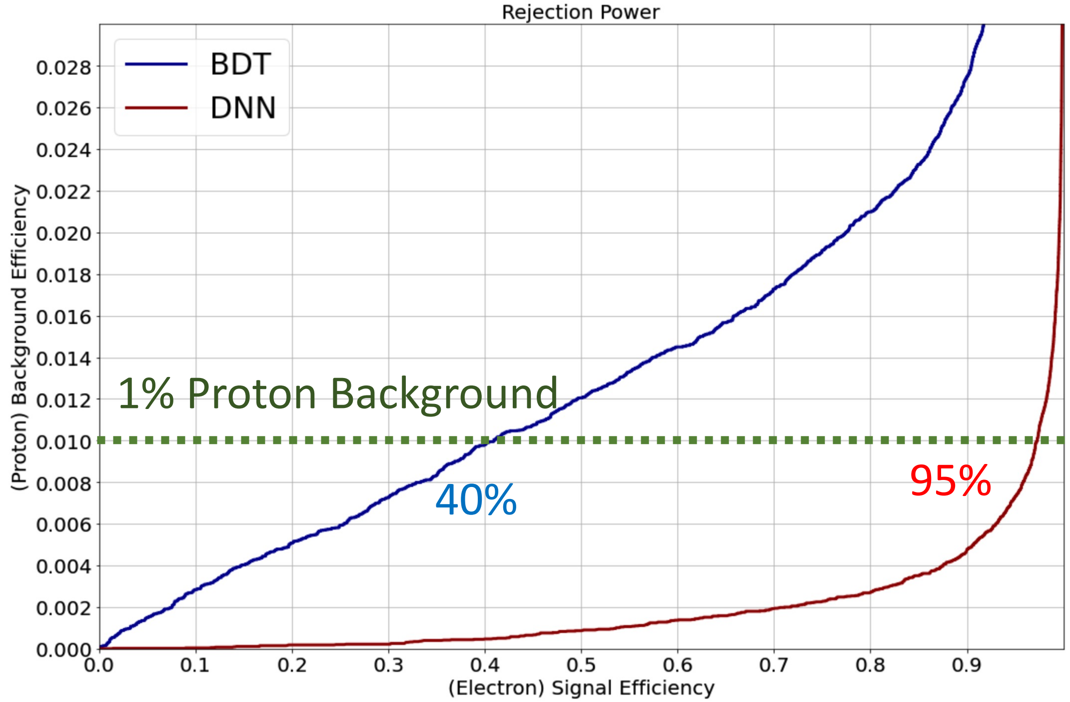
$$\mathcal{L} = -\frac{1}{N}\sum_{i=1}^{N}\left[y_i \cdot \ln\left(p_{e^\pm,i}\right) + (1 - y_i) \cdot \ln\left(1 - p_{e^\pm,i}\right)\right]$$

Boost Decision Tree (BDT)



Deep Neural Network (DNN)

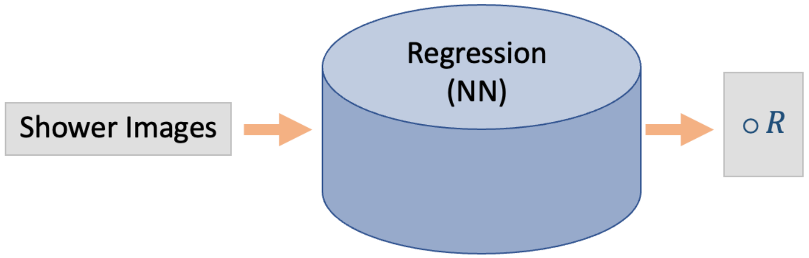




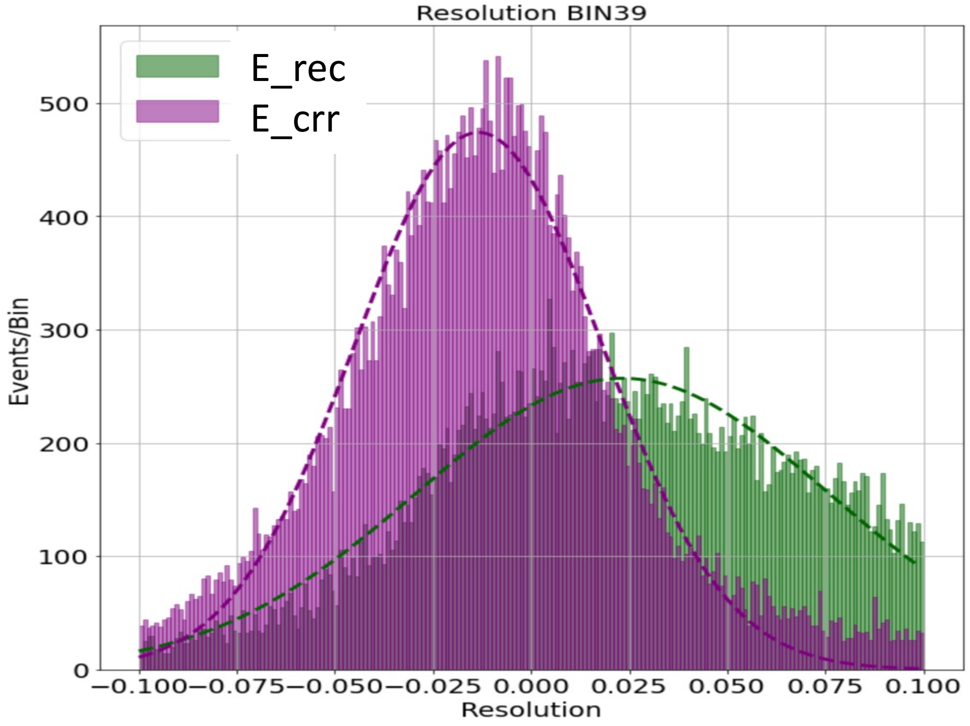
Electron Efficiency

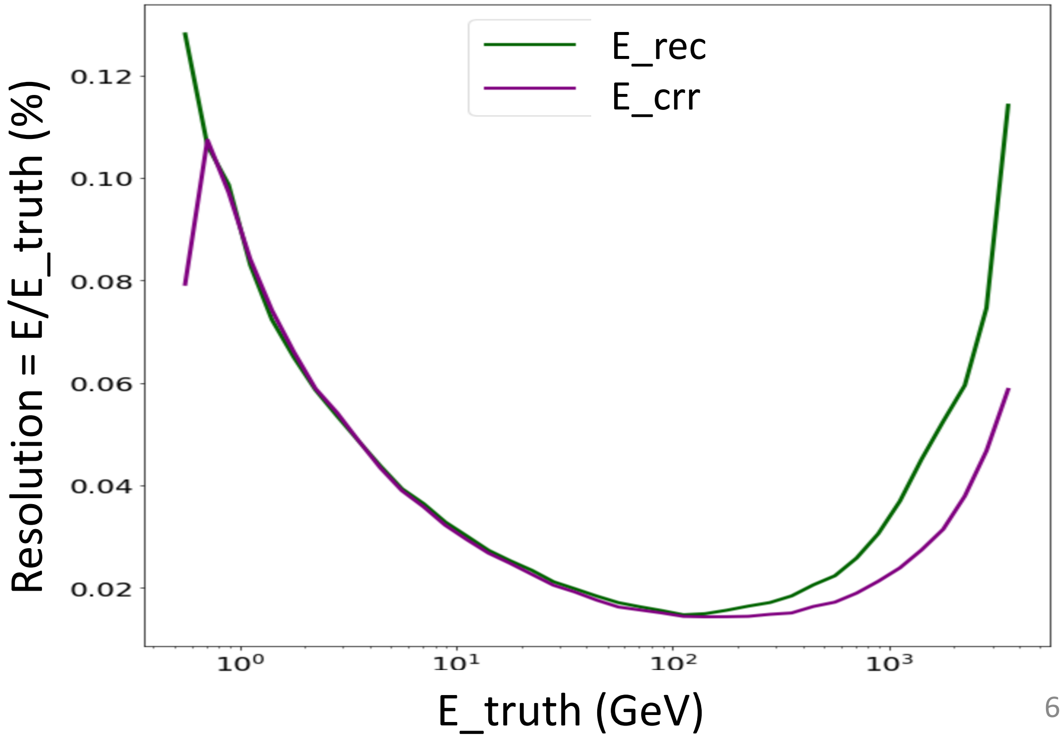
Electron Energy Correction (Regression)

- Input Variables: 18x72 ECAL cell energies
- Output Variable: R = ((E_rec / E_truth) - 1)
- Corrected Energy: E_crr = E_rec / (1 + R)



Preliminary Results (1584-2000 GeV)





Shower Simulation (Generator)

Physical Simulation
• Simulate the passage of particles through materials
▪ Monte Carlo Method: Geant4 toolkit
➢Hardware Resources: CPUs

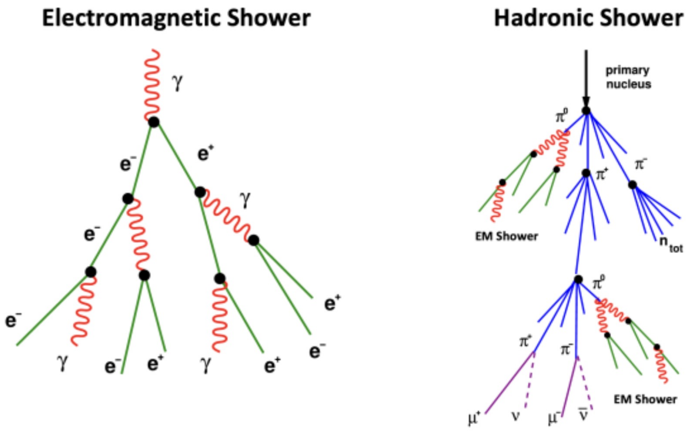
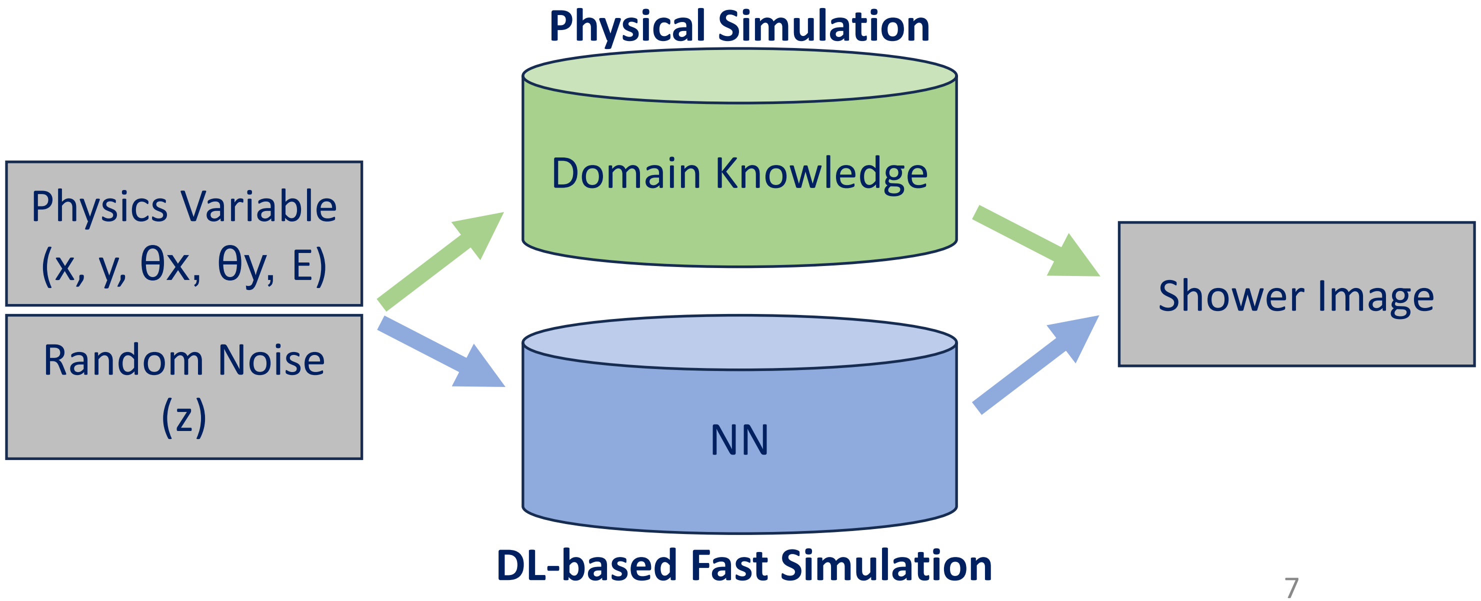


Figure 1. Electromagnetic shower and hadronic shower.

DL-based Fast Simulation
• 100 to 1000 faster than the physical simulation
▪ DL Algorithms: Generative Adversarial Networks (GANs) Diffusion Model …
➢Hardware Resources: GPUs



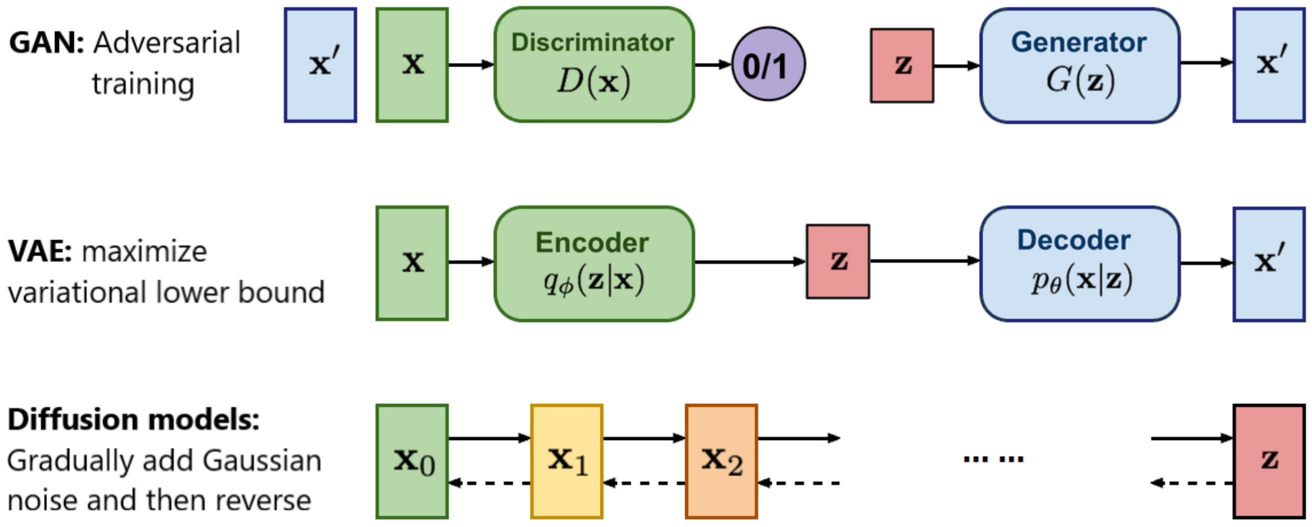
Generator

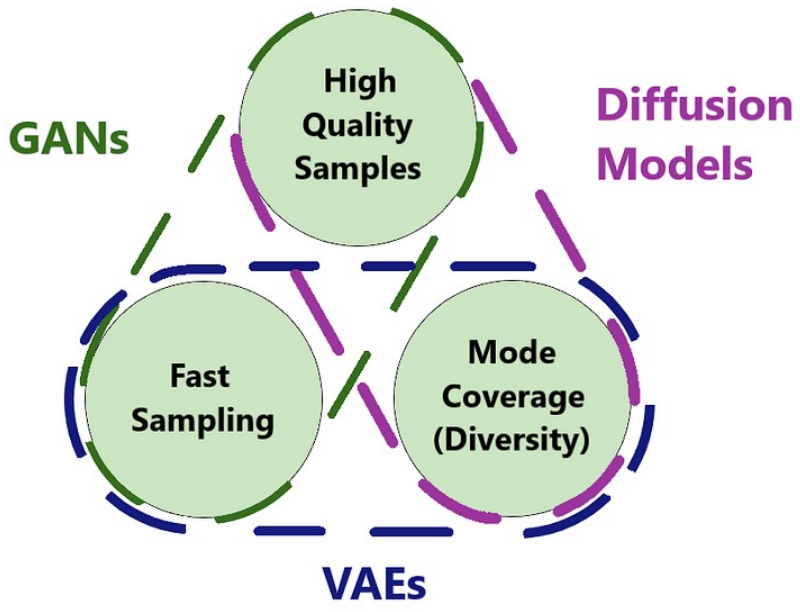
https://pub.towardsai.net/diffusion-models-vs-gans-vs-vaes-comparison-of-deep-generative-models-67ab93e0d9ae

GANs [1, 2] learn to generate new data similar to a training dataset. It consists of two neural networks, a generator, and a discriminator, that play a two-player game. The generator takes in random values sampled from a normal distribution and produces a synthetic sample, while the discriminator tries to distinguish between the real and generated sample. The generator is trained to produce realistic output that can fool the discriminator, while the discriminator is trained to correctly distinguish between the real and generated data. The top row of Figure 1 shows the scheme of its work.

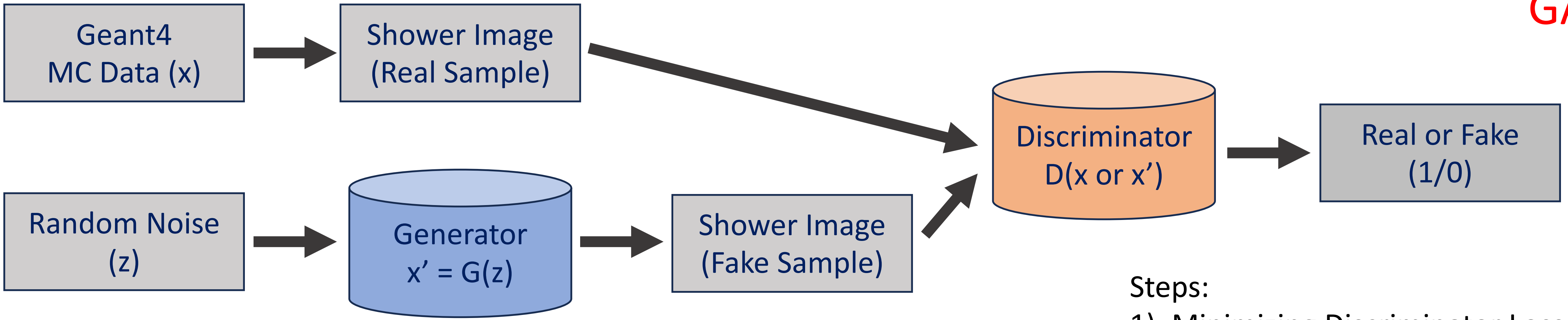
VAEs [3, 4] consist of an encoder and a decoder. The encoder maps high-dimensional input data into a low-dimensional representation, while the decoder attempts to reconstruct the original high-dimensional input data by mapping this representation back to its original form. The encoder outputs the normal distribution of the latent code as a low-dimensional representation by predicting the mean and standard deviation vectors. The middle row of Figure 1 demonstrates its work.

Diffusion models [5, 6] consist of forward diffusion and reverse diffusion processes. Forward diffusion is a Markov chain that gradually adds noise to input data until white noise is obtained. It is not a learnable process and typically takes 1000 steps. The reverse diffusion process aims to reverse the forward process step by step removing the noise to recover the original data. The reverse diffusion process is implemented using a trainable neural network. The bottom row of Figure 1 shows that.





GAN



Steps:

1). Minimizing Discriminator Loss

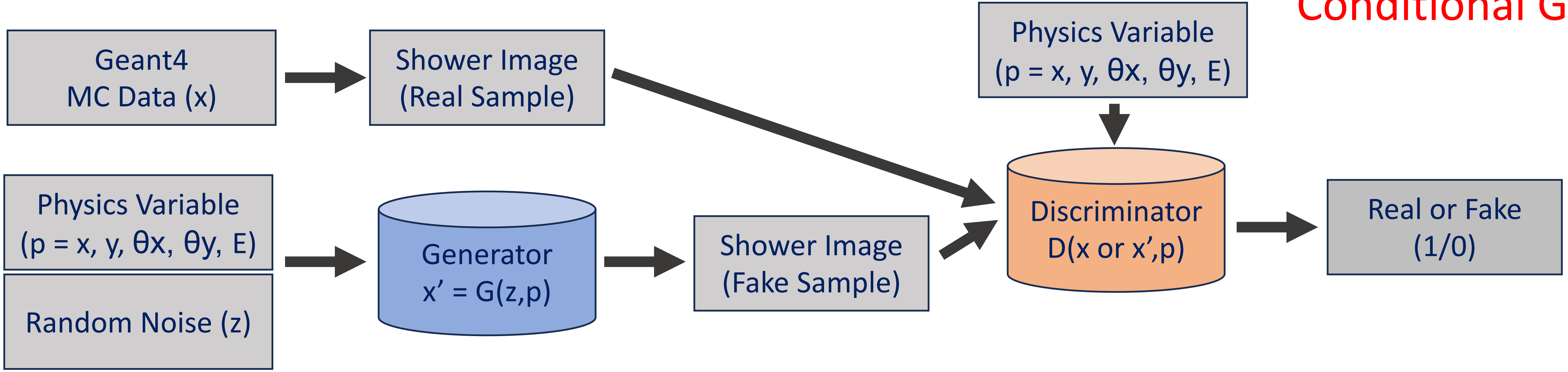
2). Minimizing Generator Loss

3). Repeat 1) and 2)

Discriminator Loss = -mean{ log[D(x)] + log[1 - D(x')] } = -{ log[D(x)] + log[1 - D(G(z))] }

Generator Loss = -mean{ log[D(G(z))] }

Conditional GAN



Discriminator Loss = -{ log[D(x,p)] + log[1 - D(G(z,p),p)] }

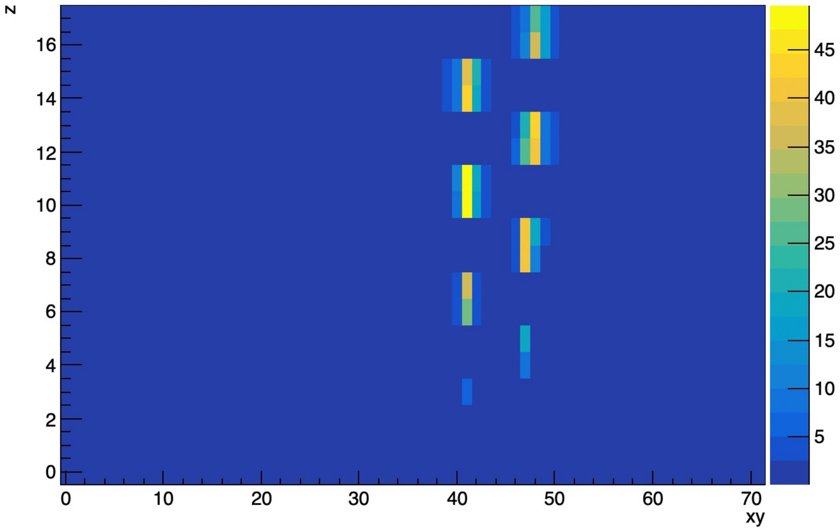
Generator Loss = -log[D(G(z,p))] + λ * MSR(G(z,p), x)

MSR(G(z,p), x) = mean {(G(z,p)-x)^2}

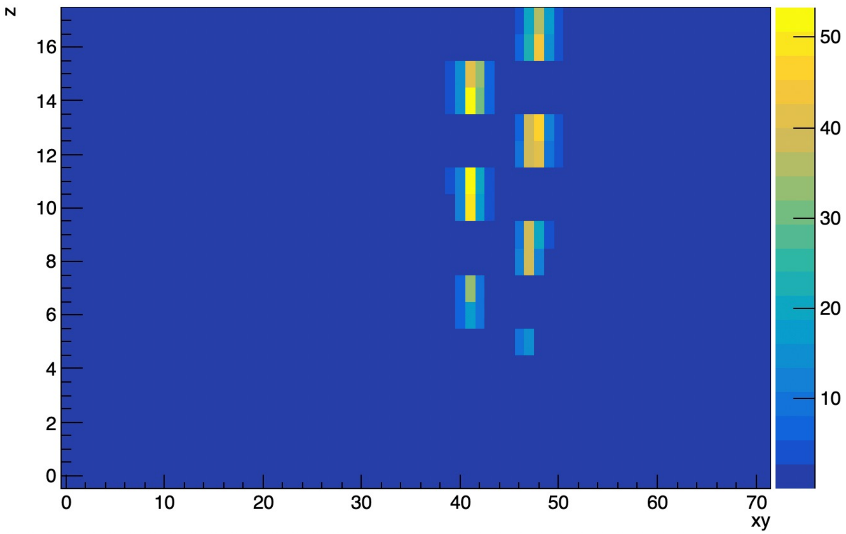
Step 1:

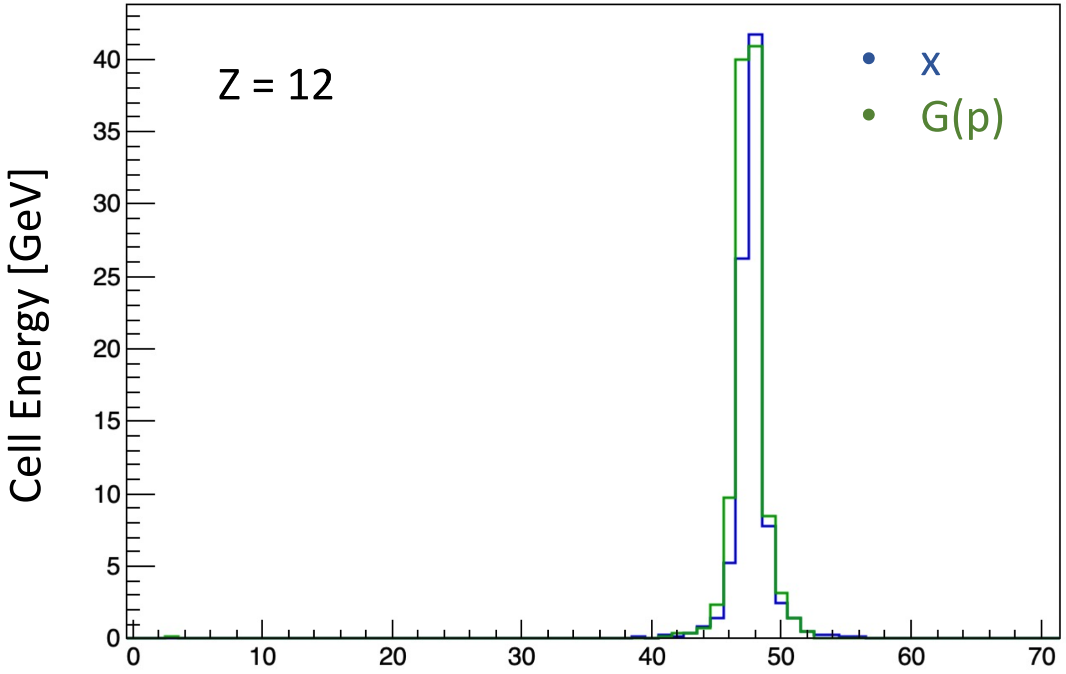
Generator Loss = MSR(G(p), x)

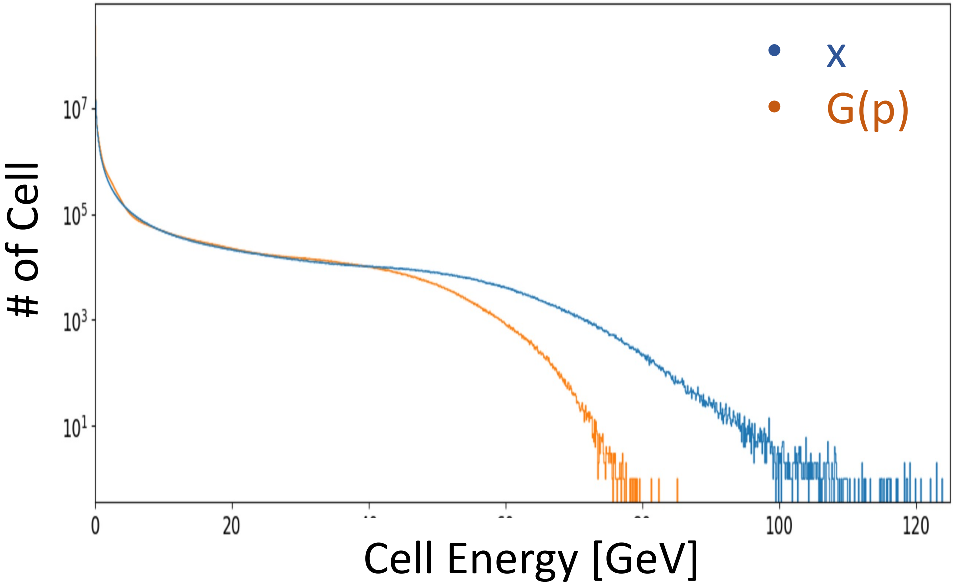
Real image (Geant4 MC) E = 1466 GeV



Reconstruction image G(p)



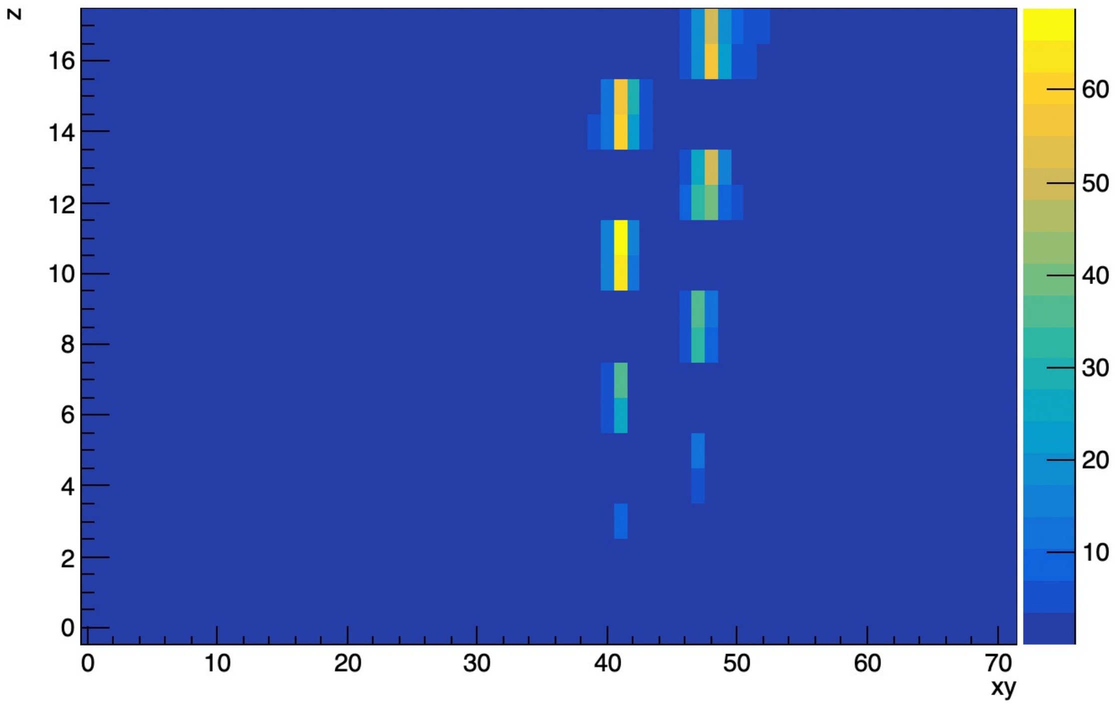


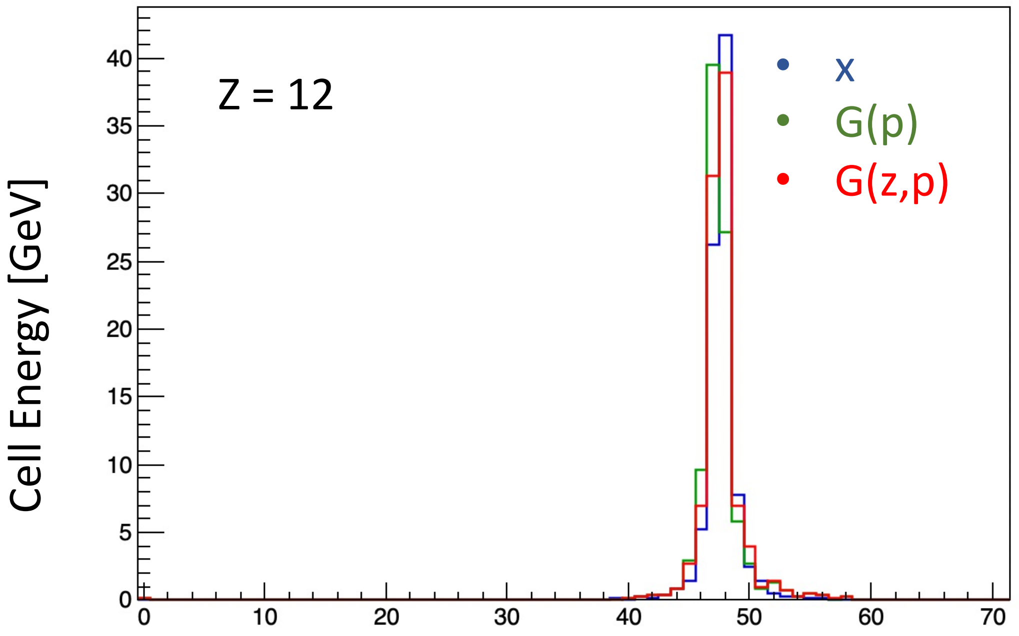


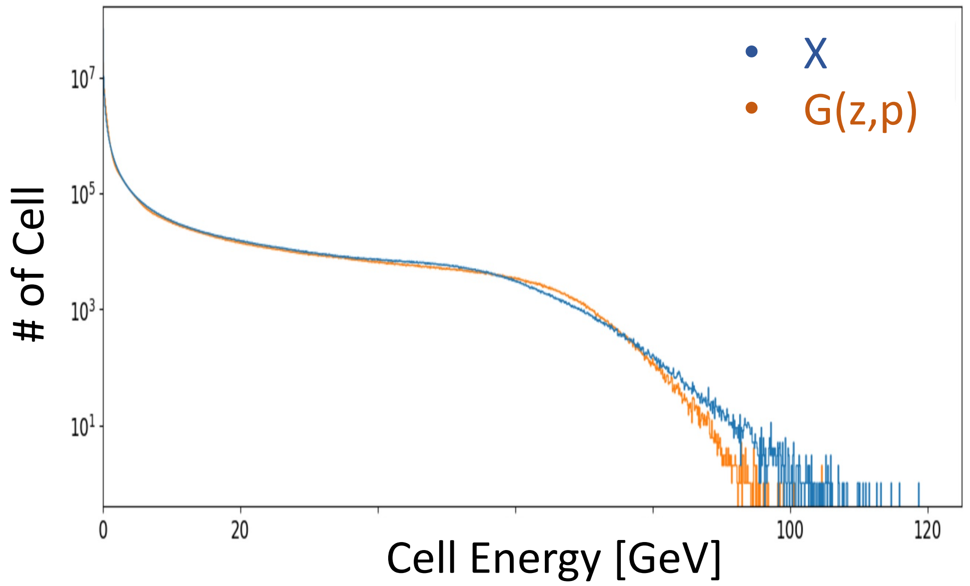
Step 2:

Discriminator Loss = -{ log[D(x,p)] + log[1 - D(G(z,p),p)] }
Generator Loss = -log[D(G(z,p))] + λ * MSR(G(z,p), x)

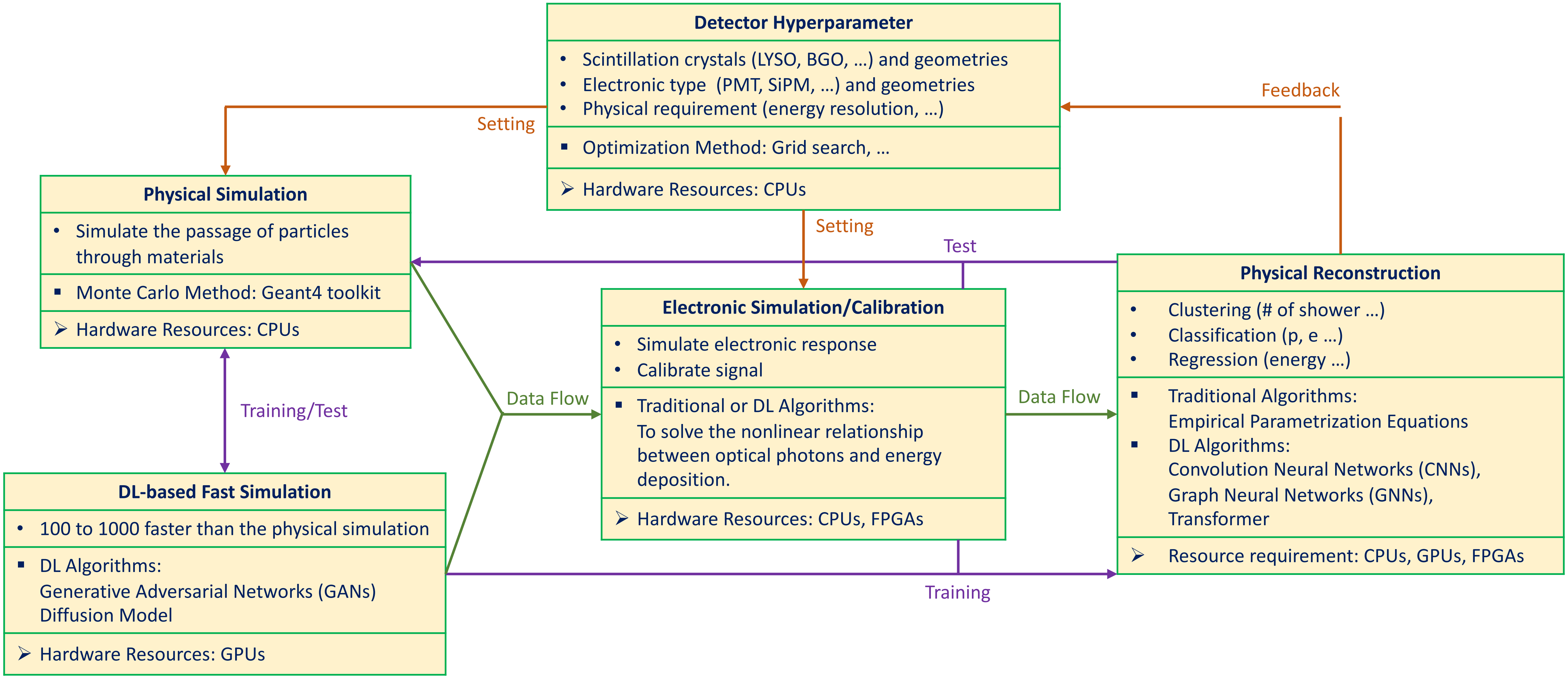
Simulation image G(z,p) E = 1466 GeV







Calorimeter Design Tool in High Energy Physics



Computing Resource

[Remote Terminal]

slurm-ui-asiop.twgrid.org

slurm-ui03.twgrid.org

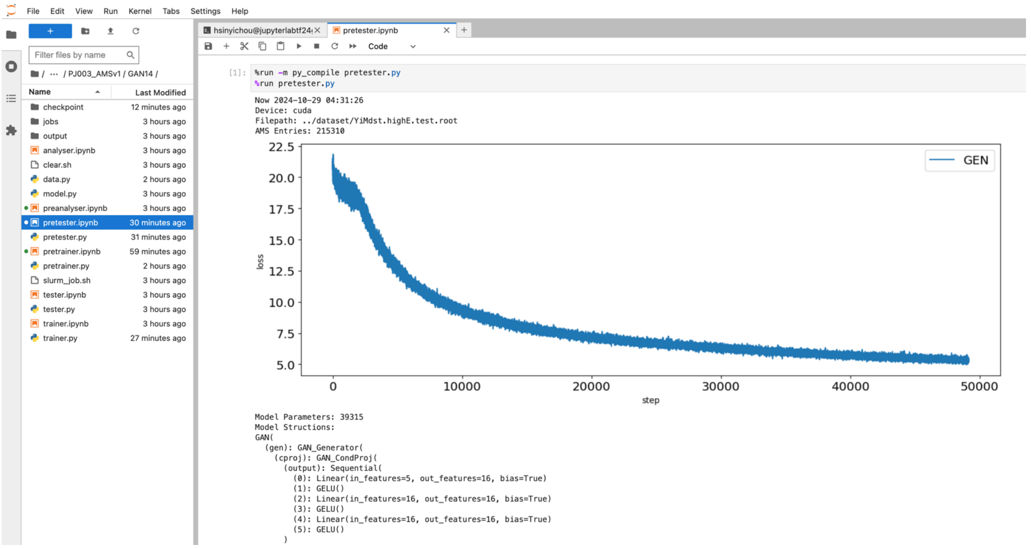
…

Vaults | SFTP | ASGC-Slurm-ASIOP | CERN-Lxplus9 | CERN-Lxplus9 (1)

```python
import torch
import torch.nn as nn
import torch.nn.functional as F

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

class PixelShuffle1D(torch.nn.Module):
    def __init__(self, upscale_factor):
        super(PixelShuffle1D, self).__init__()
        self.upscale_factor = upscale_factor

    def forward(self, x):
        batch_size, channels, length = x.size()

        # Check if channels can be evenly divided by upscale_factor
        if channels % self.upscale_factor != 0:
            raise ValueError(f"Channels must be divisible by {self.upscale_factor}")

        # Perform pixel shuffle
        channels //= self.upscale_factor
        x = x.view(batch_size, channels, self.upscale_factor, length)
        x = x.permute(0, 1, 3, 2)  # Rearrange dimensions
        x = x.contiguous().view(batch_size, channels, length * self.upscale_factor)
        return x

class PixelUnshuffle1D(torch.nn.Module):
    def __init__(self, downscale_factor):
        super(PixelUnshuffle1D, self).__init__()
        self.downscale_factor = downscale_factor

    def forward(self, x):
        batch_size, channels, length = x.size()

        # Check if the length is divisible by downscale_factor
        if length % self.downscale_factor != 0:
            raise ValueError(f"Length must be divisible by {self.downscale_factor}")

        # Perform pixel unshuffle
        out_channels = channels * self.downscale_factor
        out_length = length // self.downscale_factor

        # Reshape and permute the tensor
        x = x.view(batch_size, channels, out_length, self.downscale_factor)
        x = x.permute(0, 1, 3, 2)  # Rearrange dimensions
        x = x.contiguous().view(batch_size, out_channels, out_length)
        return x
```

[DiCOS Apps]

Jupyter Lab GPU



[Slurm]

CPUs, GPUs A100, V100, …

```
(ml) [hsinyichou@slurm-ui-asiop GAN14]$ squeue | grep a100
           230030      a100  HMC0034   twchiu PD       0:00      1 (Nodes required for
           231049      a100  HMC0036   twchiu PD       0:00      1 (Priority)
           232091      a100  HMC0035   twchiu PD       0:00      1 (Priority)
           232098      a100  HMC0033   twchiu PD       0:00      1 (Priority)
           233667      a100 handbook noke0081  R      18:00      1 hp-teslaa03
           232635      a100 handbook noke0081  R    1:01:53      1 hp-teslaa03
           194890 a100_long urg_seja       rc PD       0:00      1 (Dependency)
           194889 a100_long urg_seja       rc PD       0:00      1 (Dependency)
           194888 a100_long urg_seja       rc PD       0:00      1 (Dependency)
           194887 a100_long urg_seja       rc PD       0:00      1 (Dependency)
           194886 a100_long urg_seja       rc PD       0:00      1 (Dependency)
           176131 a100_long urg_seja       rc PD       0:00      1 (Dependency)
           176130 a100_long urg_seja       rc PD       0:00      1 (Dependency)
           176129 a100_long urg_seja       rc PD       0:00      1 (Dependency)
           176128 a100_long urg_seja       rc PD       0:00      1 (Dependency)
           176127 a100_long urg_seja       rc PD       0:00      1 (Dependency)
```