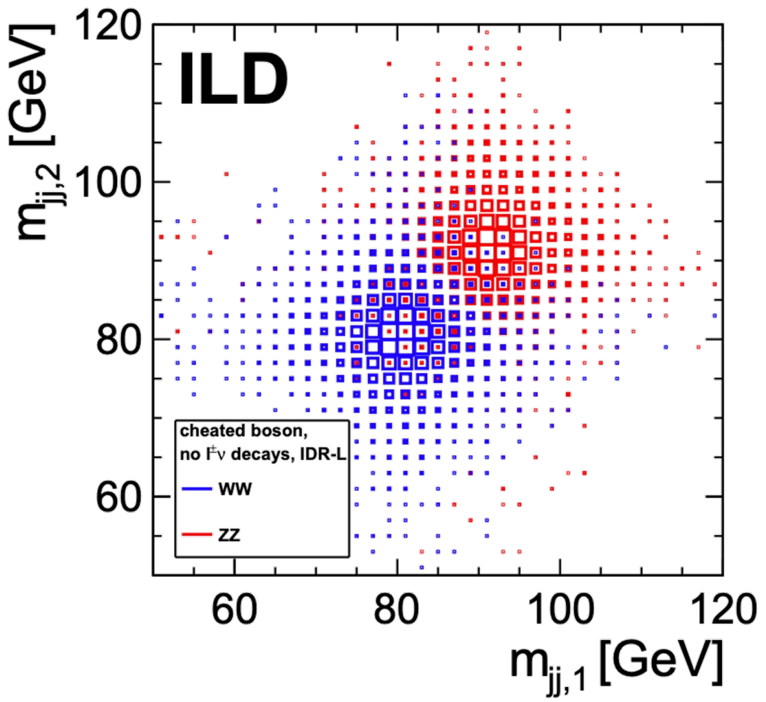
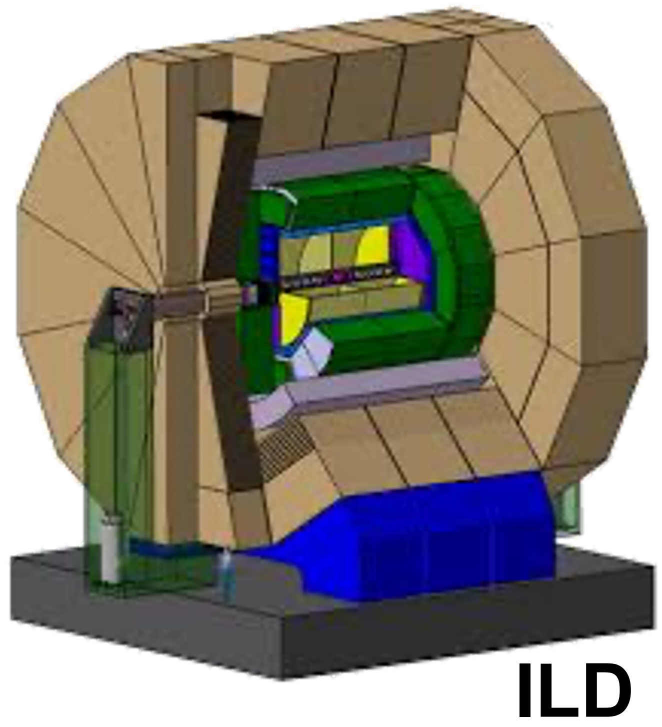
Software compensation in highly granular calorimeter system with dual-branch CNN

Xin Xia
on behalf of the CALO5D team
2025/9/19

Motivation

- High-precision energy measurement is essential for $W/H/Z$ studies on future colliders
- Target: Jet energy resolution < $30\%/\sqrt{E}$
- Approach: Particle Flow Algorithm (PFA) + Highly granular 5D calorimeter + Machine Learning

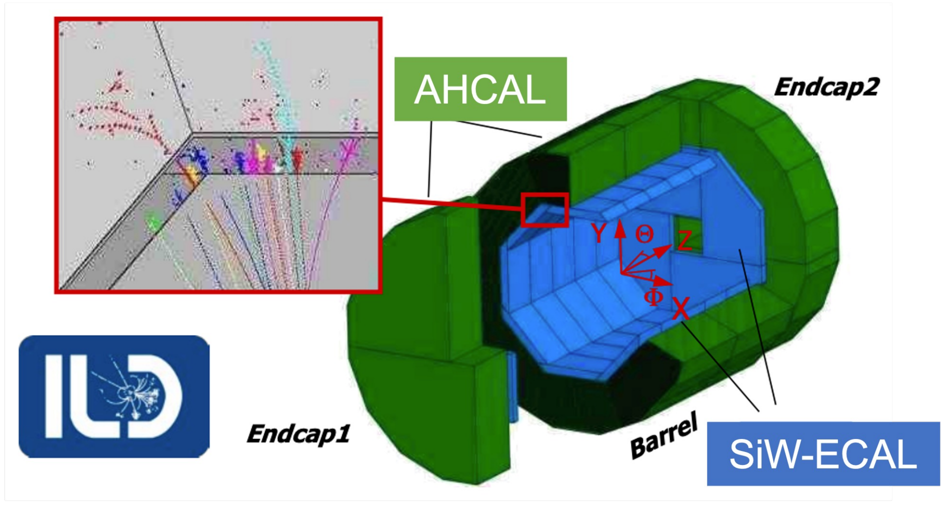


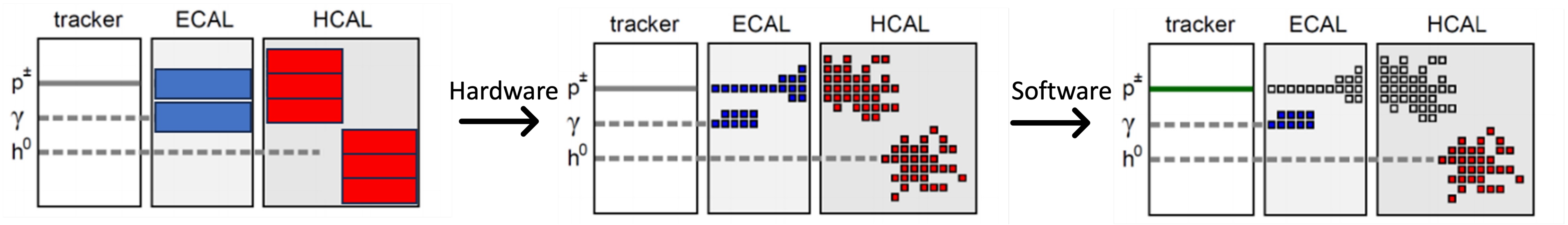


5D calorimeter
spatial (x, y, z),
energy (E),
timing (t)

Motivation

- What is PFA?
  - PFA: $E_{jet} = E_{tracker} + E_{ECAL} + E_{HCAL}$
    - 60% charged particles → Tracker
    - 30% photons → ECAL
    - 10% neutral hadrons → HCAL
  - High granularity calorimeter (imaging):
    - SiW-ECAL + AHCAL in ILD system
- $E_{ECAL} + E_{HCAL}$ as part of PFA & intrinsic performance of calorimeter   Ref: arXiv:2107.10207v3
- In my case: software compensation in the ECAL-HCAL calorimeter system with dual-branch CNN

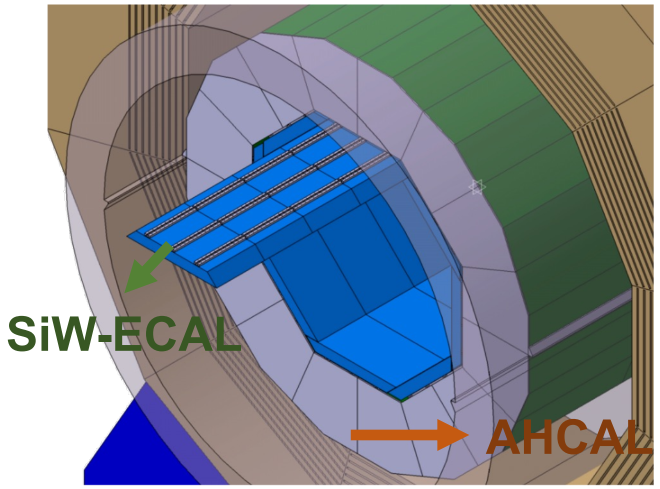


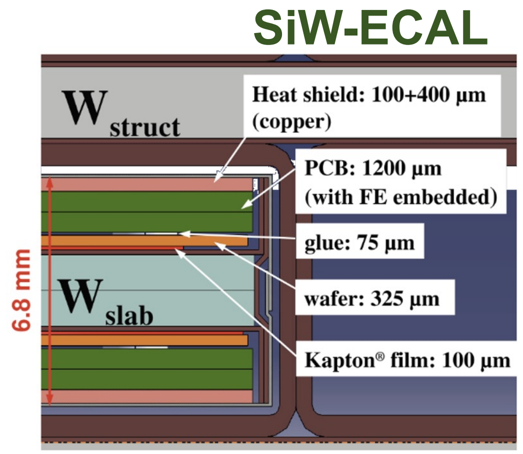


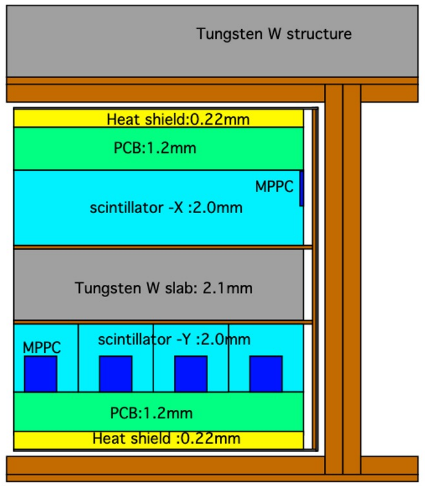
Detector introduction

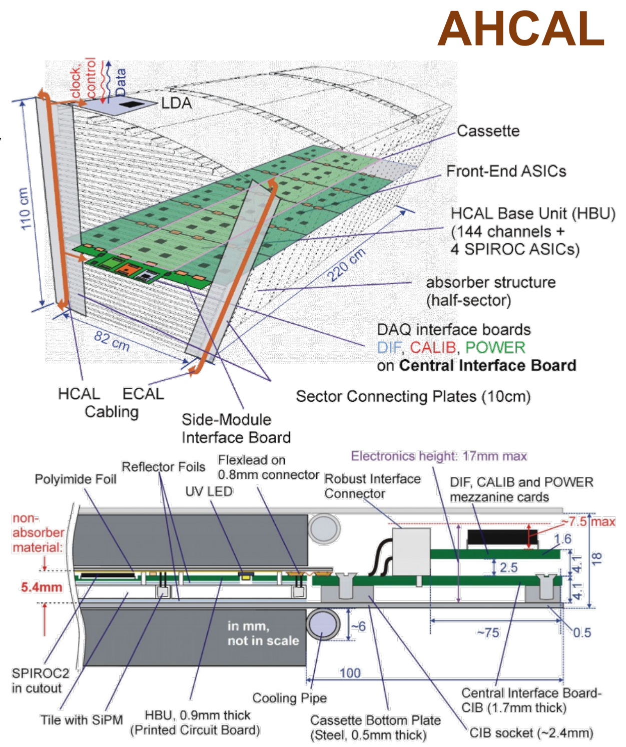
Si-W ECAL + AHCAL:

- Configuration: shown in table; Energy resolution of hadron < 60%/ $\sqrt{E}$
- Different response for electro-magnetic/hadronic shower, degrade energy measurement precision









AHCAL
clock, control
Data
LDA
Cassette
Front-End ASICs
HCAL Base Unit (HBU) (144 channels + 4 SPIROC ASICs)
110 cm
220 cm
absorber structure (half-sector)
82 cm
DAQ interface boards
DIF, CALIB, POWER
on Central Interface Board
HCAL ECAL
Cabling
Side-Module Interface Board
Sector Connecting Plates (10cm)
Electronics height: 17mm max
Polyimide Foil
Reflector Foils
Flexlead on 0.8mm connector
UV LED
Robust Interface Connector
DIF, CALIB and POWER mezzanine cards
non-absorber material:
5.4mm
~7.5 max
1.6
2.5
4.1
18
4.1
in mm, not in scale
~6
~75
0.5
100
SPIROC2 in cutout
Cooling Pipe
Central Interface Board-CIB (1.7mm thick)
Tile with SiPM
HBU, 0.9mm thick (Printed Circuit Board)
Cassette Bottom Plate (Steel, 0.5mm thick)
CIB socket (~2.4mm)

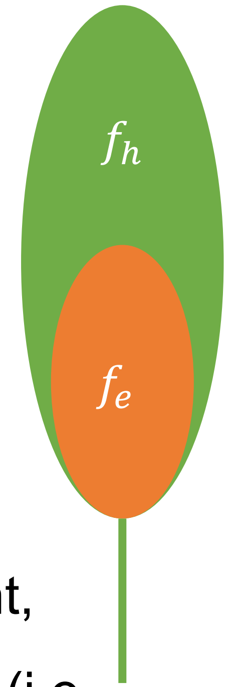
	#. Layers	Length	Cell size	Active material	Absorber	Type
SiW-ECAL	30 in 20 cm	~ $1\lambda_I$	0.5×0.5 cm^2	Silicon	Tungsten	Non-Compensating
AHCAL	48 in 1 m	~ $5\lambda_I$	3×3 cm^2	Scintillator	Steel	Non-Compensating

SC Algorithm

Software Compensation Algorithm:

- For a hadron (π^+) in a non-compensating calorimeter
  - Have electromagnetic components (e) and hadronic components (h), e/h >1
    - $\pi = f_{em} \cdot e + (1 - f_{em}) \cdot h$
    - $E_{rec} = \frac{e}{\pi} \cdot E_{dep} = \frac{e}{f_{em} \cdot e + (1 - f_{em}) \cdot h} \cdot E_{dep} = \frac{e/h}{1 + f_{em}(e/h - 1)} \cdot E_{dep}$
      - e/h: a constant value which depends on calorimeter
      - f_{em}: generated by π^0 in hadronic shower, fluctuates strongly from event to event, measured as the ratio of the energy deposited by electromagnetic components (i.e., photons and $e^{\pm}$ from $\pi^0 \to \gamma\gamma$ decays) to the total deposited energy.

$$f_{em} = \frac{\sum_i^{em} E_i}{\sum_i^{all} E_i} \ (em\ is\ \gamma, e^{\pm})$$



SC Algorithm

Software Compensation Algorithm :

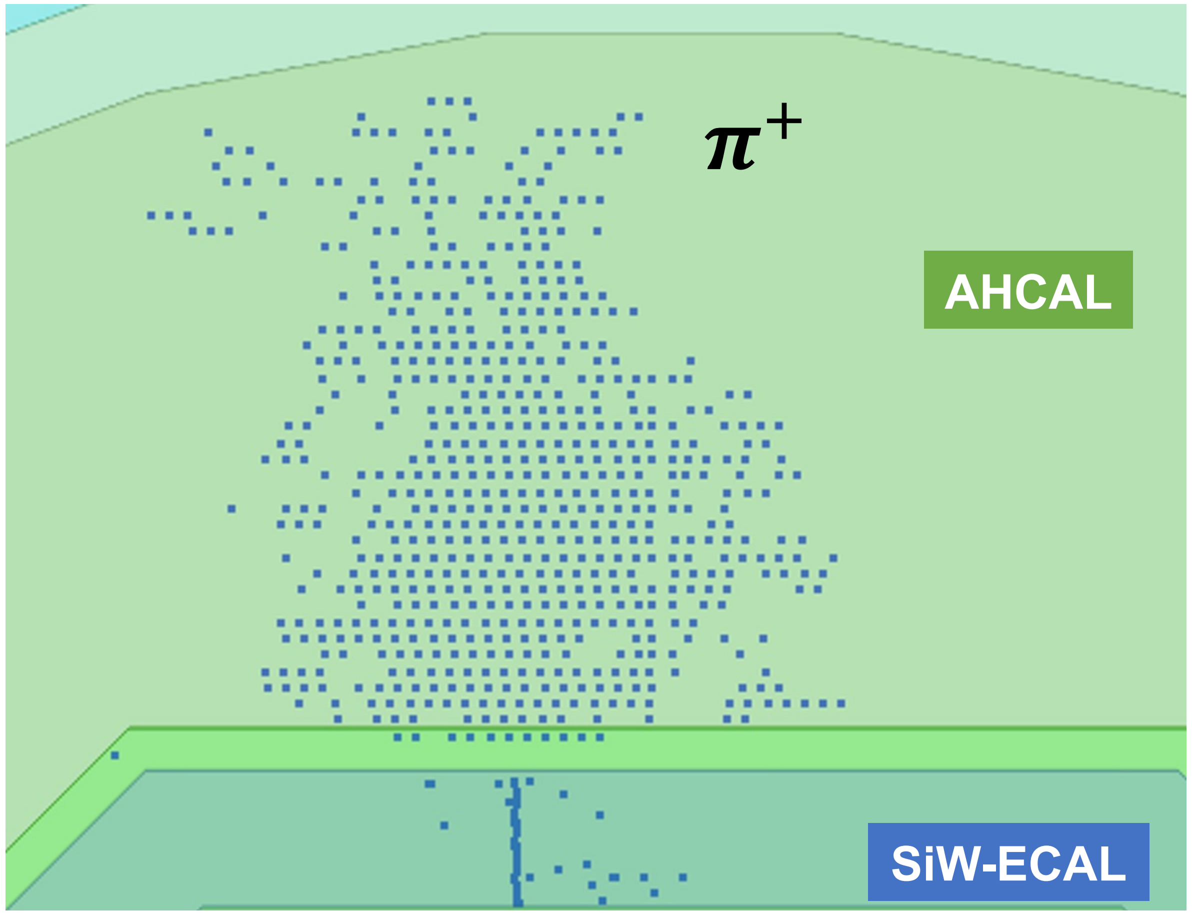
- In my case, a hadron in a system (SiW-ECAL + AHCAL):

  - $$E_{rec} = \frac{\left(\frac{e}{h}\right)^{ECAL}}{1 + f_{em}^{ECAL} \cdot \left(\left(\frac{e}{h}\right)^{ECAL} - 1\right)} \cdot E_{dep}^{ECAL} +$$

    $$\frac{\left(\frac{e}{h}\right)^{HCAL}}{1 + f_{em}^{HCAL} \cdot \left(\left(\frac{e}{h}\right)^{HCAL} - 1\right)} \cdot E_{dep}^{HCAL}$$

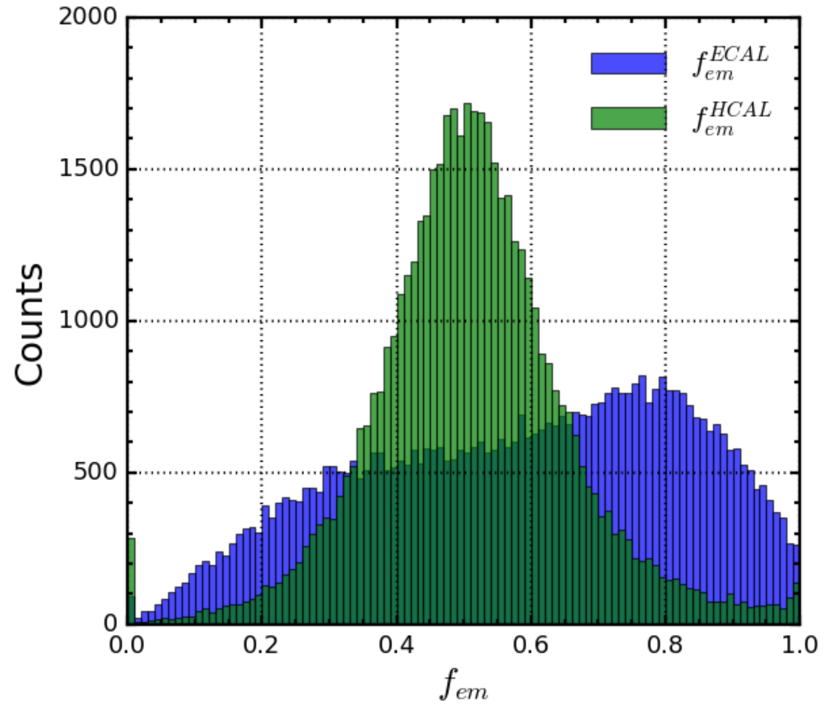
$$\downarrow$$

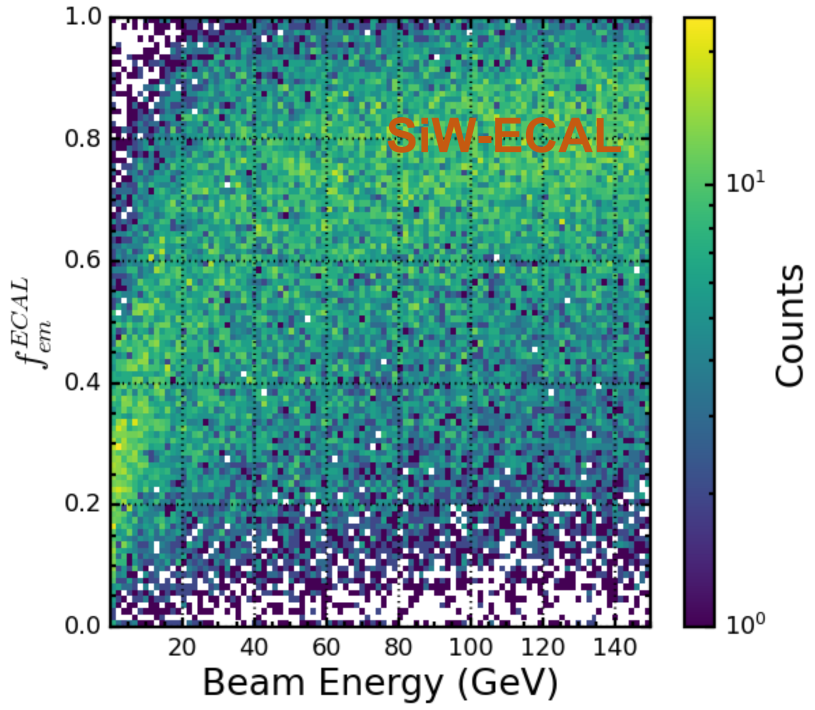
- f_{em}^{ECAL}
- f_{em}^{HCAL}

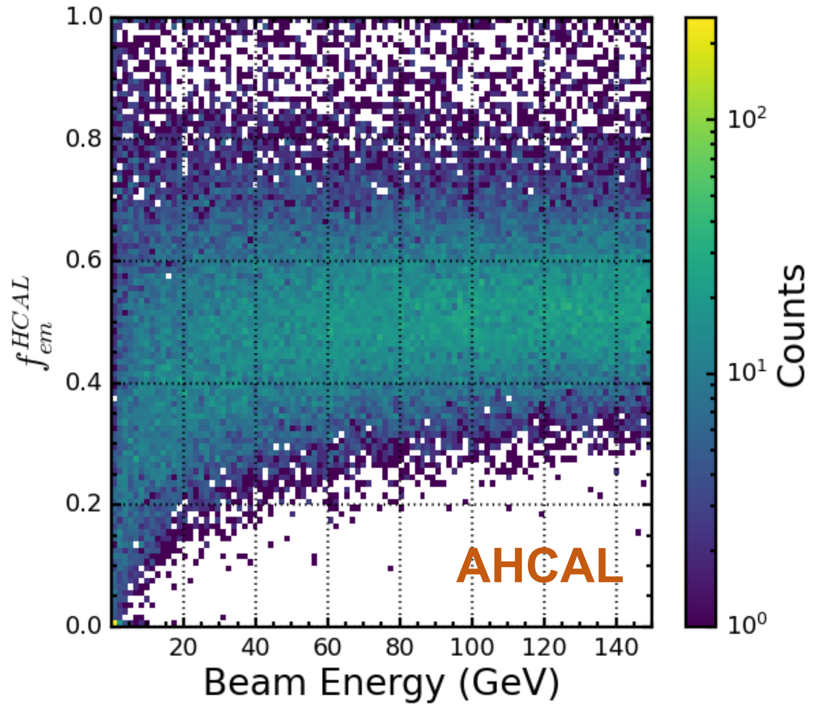


f_{em} behaviour

- f_{em} fluctuates strongly from event to event
- How to determine f_{em} event-by-event basis?
  - Dual-readout techniques
  - Rich input from imaging calorimeter + Marchine Learning (Pytorch + CNN)

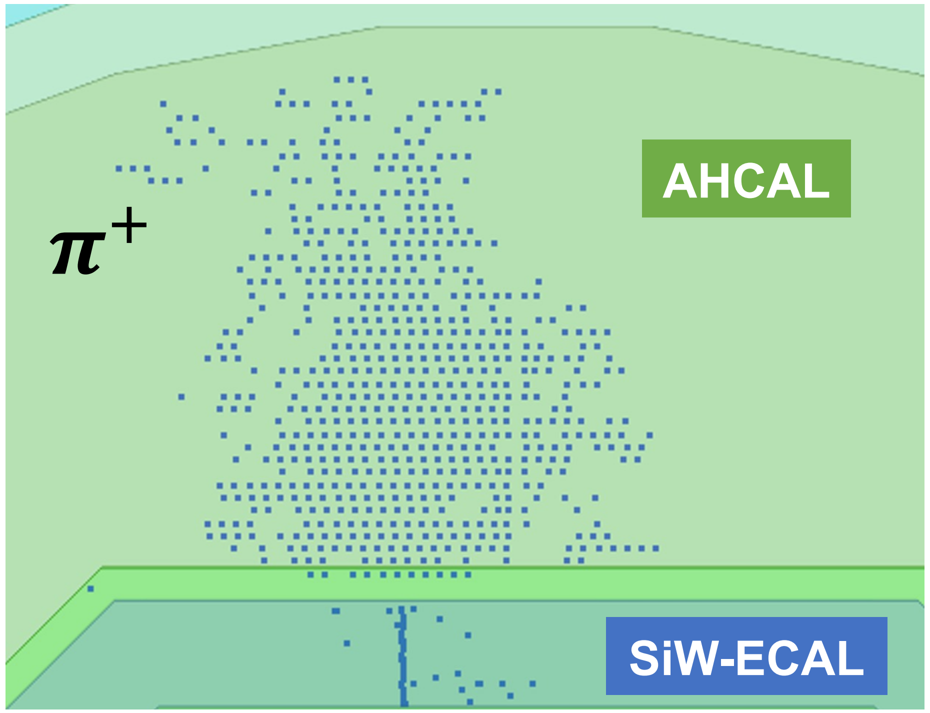






f_{em} behaviour

- f_{em} fluctuates strongly from event to event
- How to determine f_{em} event-by-event basis?
  - Dual-readout techniques
  - Rich input from imaging calorimeter + Marchine Learning (Pytorch + CNN)



	(x, y, z)	E	t
EM component	Narrow, compact, shorter	Higher density	Prompt
Hadronic component	Broader, longer	Lower, spread out	Delayed components

Work flow

Identify targets: f_{em} → Simulation & Reconstruction → Data preparation & pre-analysis → Training → Results: f_{em}

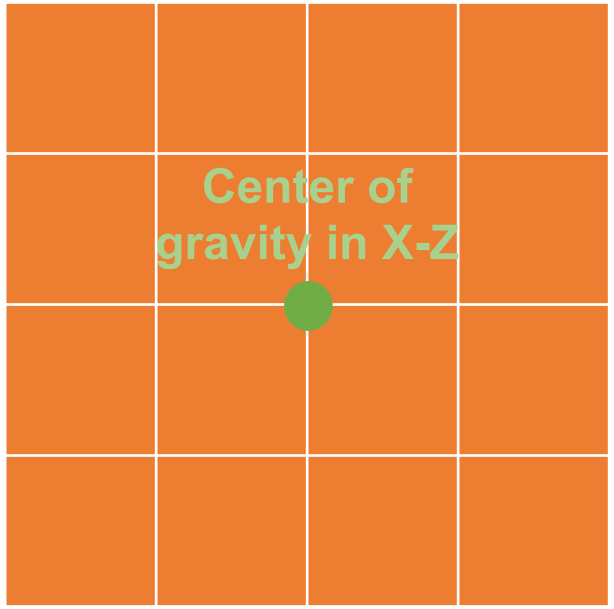
Platform	:	NAF Desy
Tool	:	dd4hep
Geometry	:	ILD
Particle	:	π^+ (QGSP_BERT)
Energy range	:	0 – 150 GeV
Data set	:	1 million
Output	:	.slcio

Platform	:	Openstack (Cloud@VirtualData)
Output	:	.root

Platform	:	Ruche (UPSaclay)
Frame	:	Pytorch
Model	:	Dual-branch 3D CNN
Output	:	.pth

Input

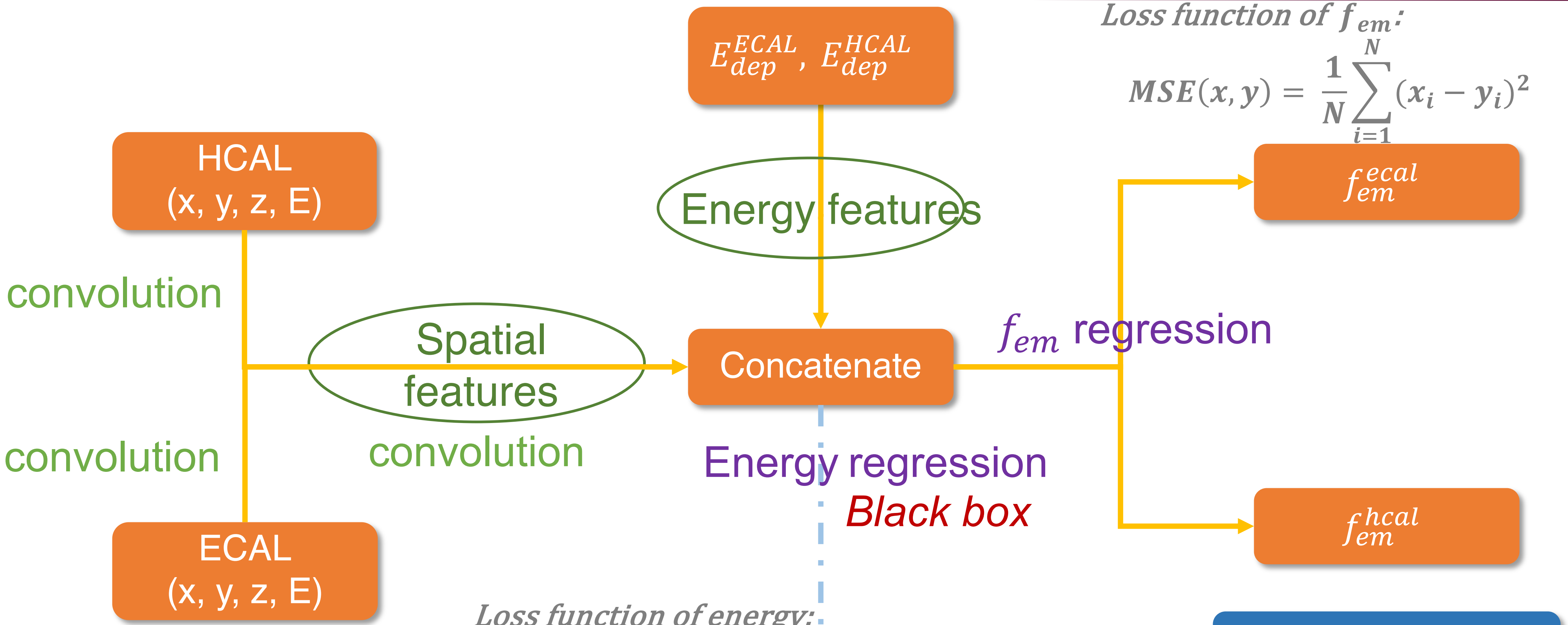
- **Input Shape**
  - Ideal way: feed all calorimeter hits from both ECAL and HCAL into the network
  - Problems:
    - Excessive input size and memory consumption
    - Many empty cells lead to sparse input tensors and inefficient computation
    - Difficult for CNN to effectively exploit local correlations
  - Solution:
    - Construct a fixed-size voxel grid centered on the shower center-of-gravity (COG) in the transverse plane, ECAL=(40,30,40),HCAL=(40,48,40)



Shuffling

- **Data Handling & Shuffling**
  - Issue:
    - 1M events (1k energies × 1k events); train/val/test split should reflect full spectrum, not block by energy
    - random access in .h5 is slow
    - Full shuffling causes OOM
  - Solution:
    - Use pre-chunked .h5 shards, mimic shuffle while avoiding OOM

Model



Loss function of f_{em}:

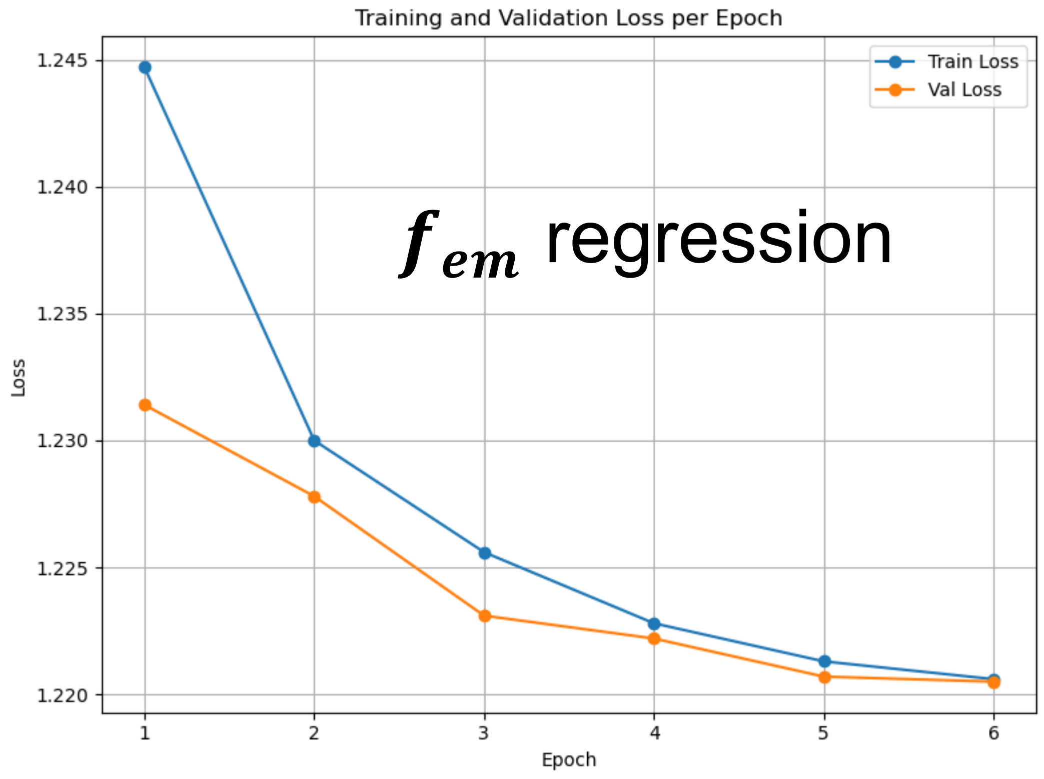
$$MSE(x, y) = \frac{1}{N}\sum_{i=1}^{N}(x_i - y_i)^2$$

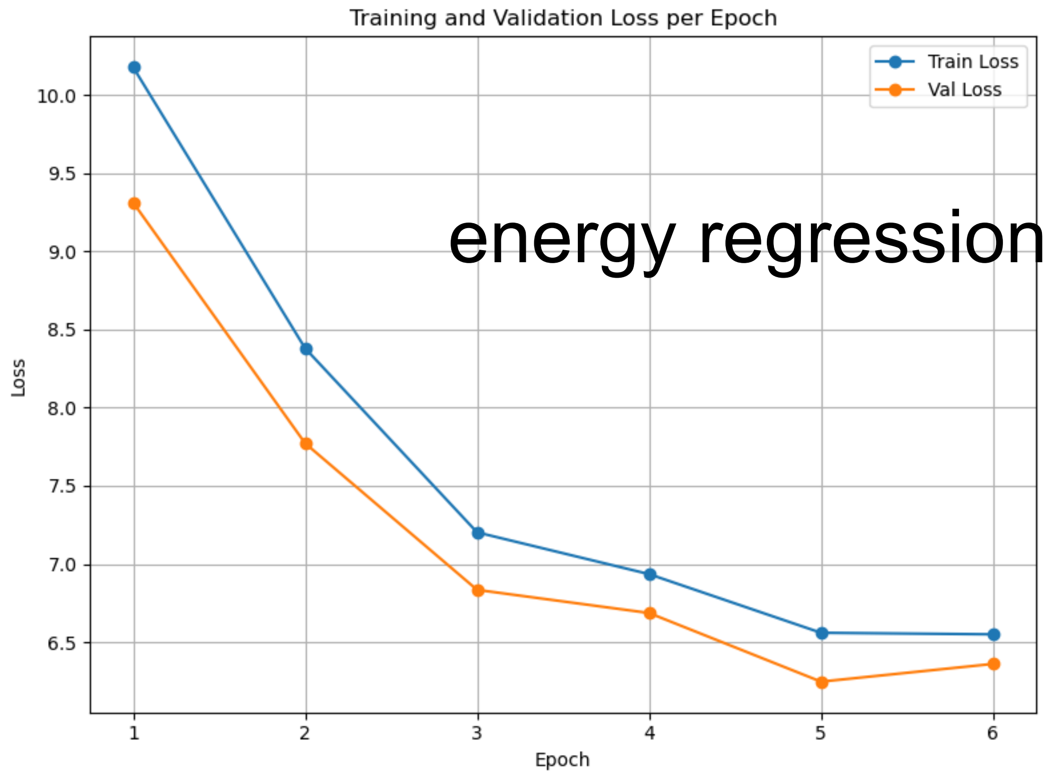
Loss function of energy:

$$SmoothL1(x, y) = \begin{cases} \frac{(x_i - y_i)^2}{2}, if|x - y| < 1 \\ |x - y| - 0.5, otherwise \end{cases}$$

Loss function

1batch: ~0.1s
1epoch: ~4h
Limit: < 24h

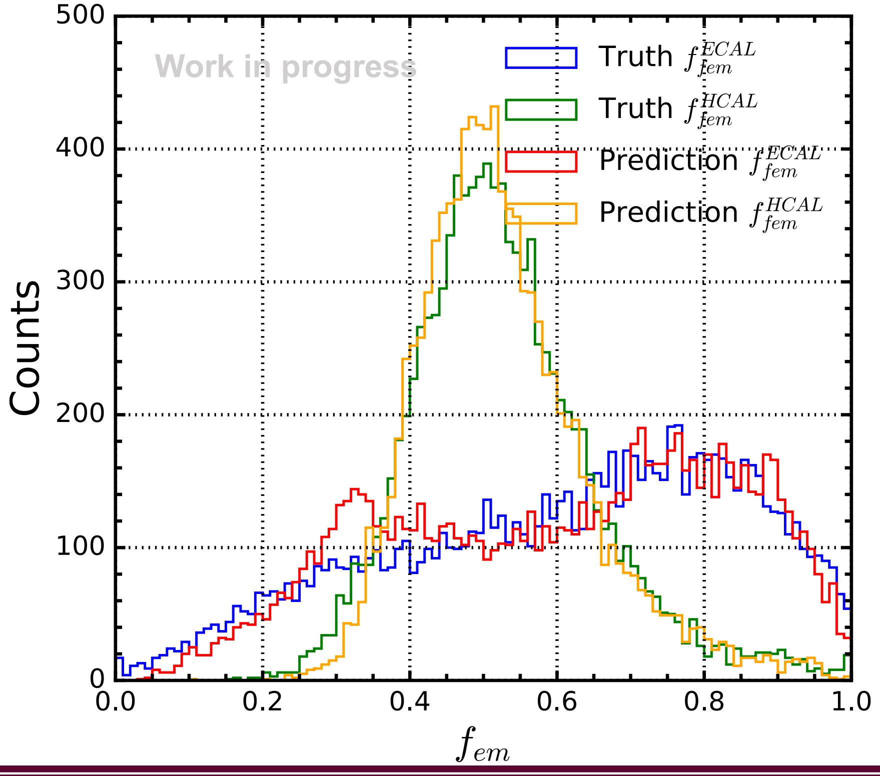


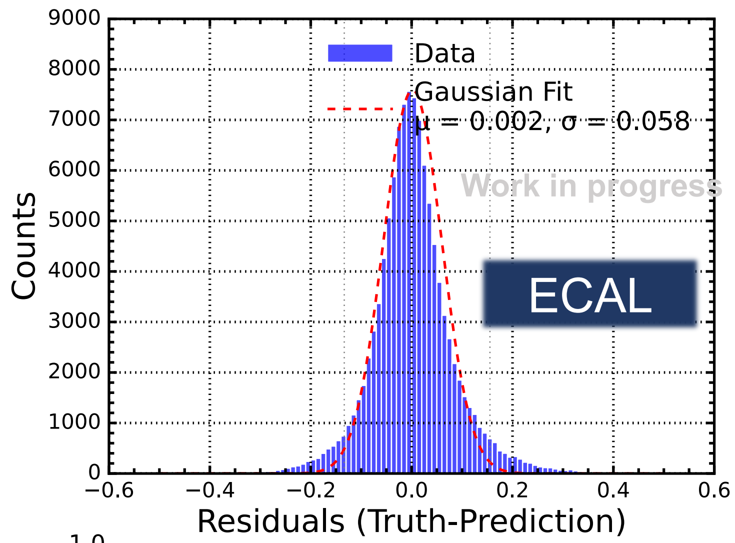


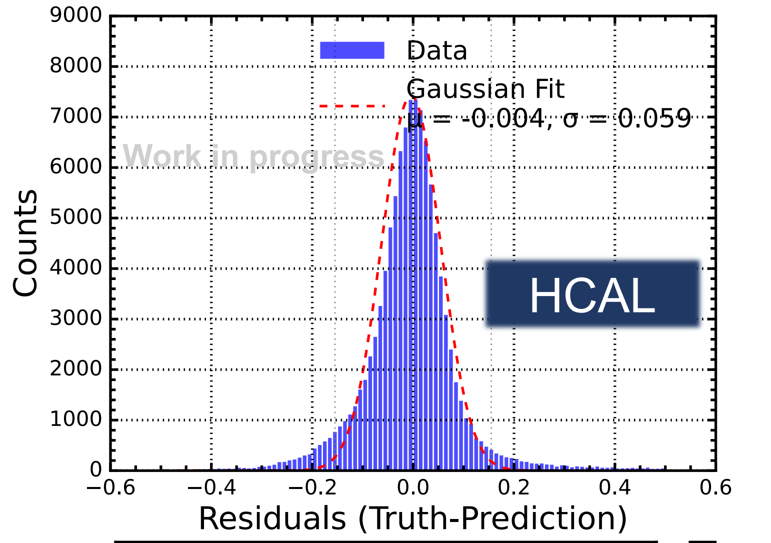
Results - f_{em} regression

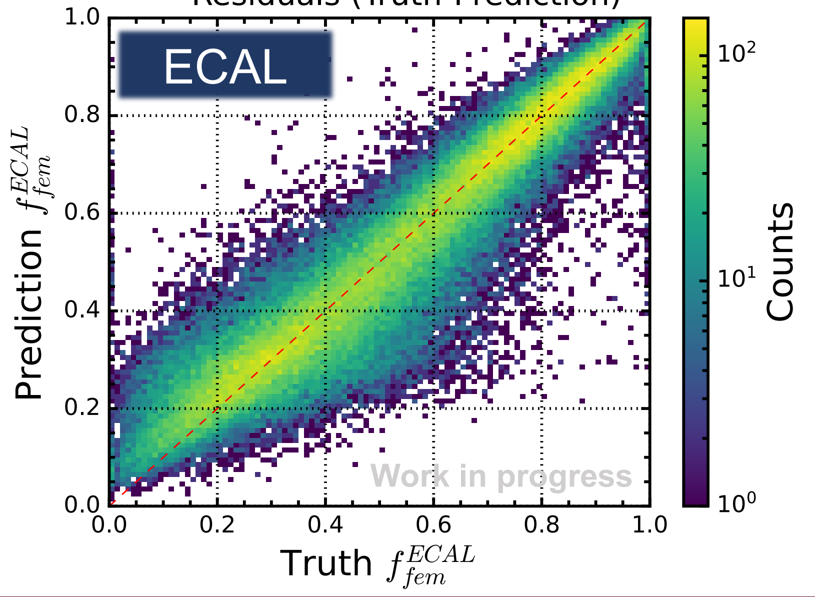
Results of f_{em}^{ECAL} and f_{em}^{HCAL} :

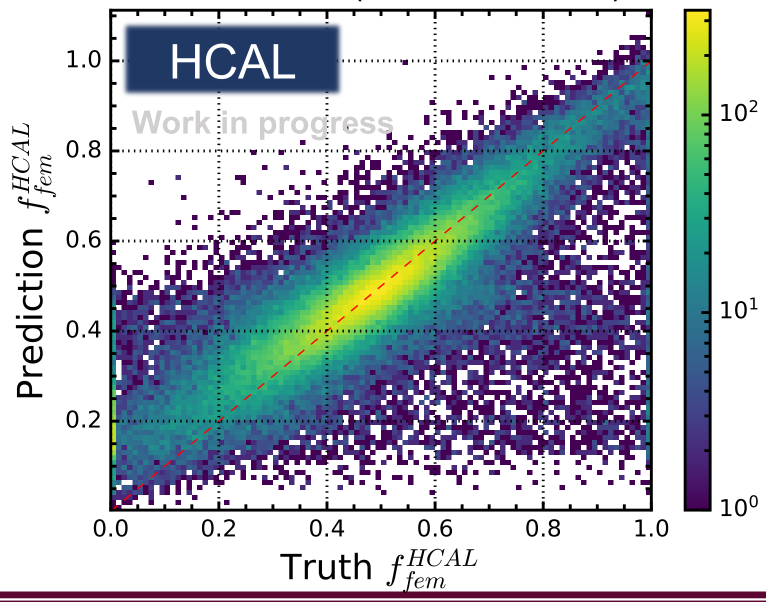
- Overall, the model shows good consistency between truth and prediction







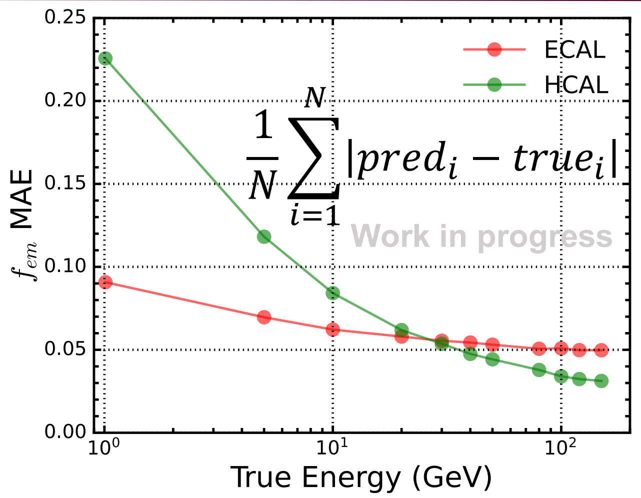


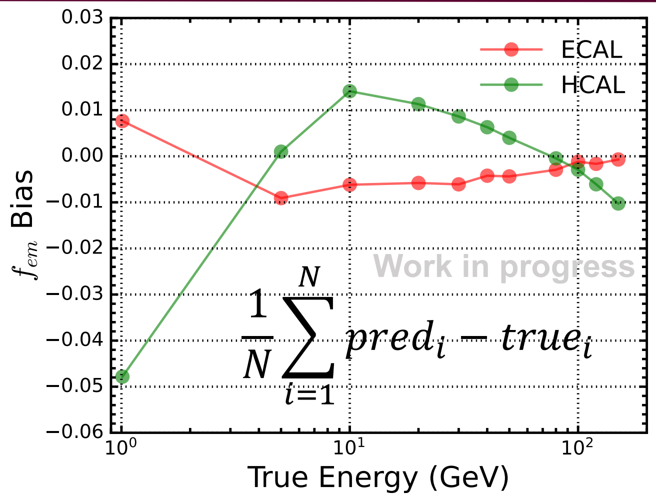


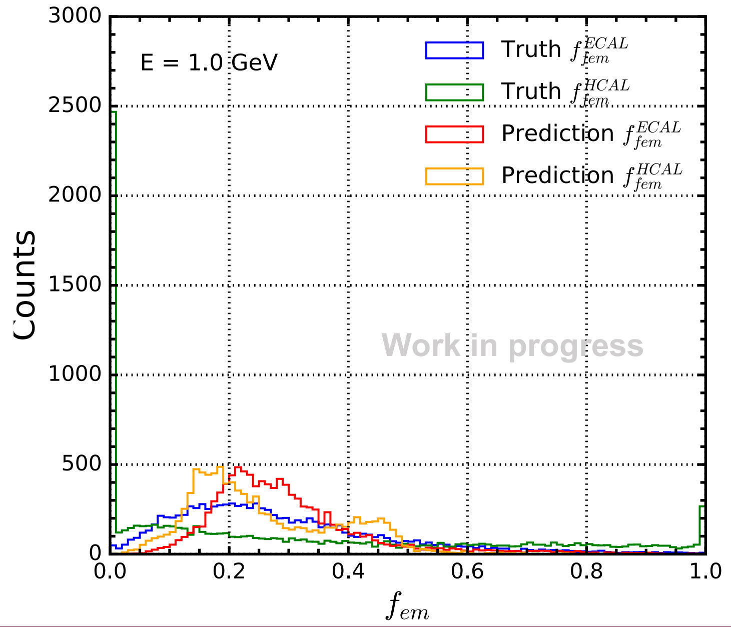
Results - f_{em} regression

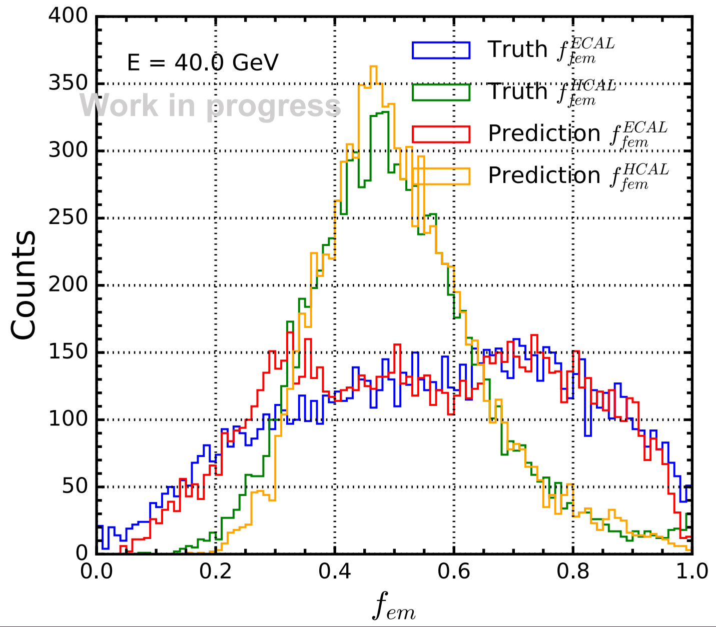
Results of f_{em}^{ECAL} and f_{em}^{HCAL} :

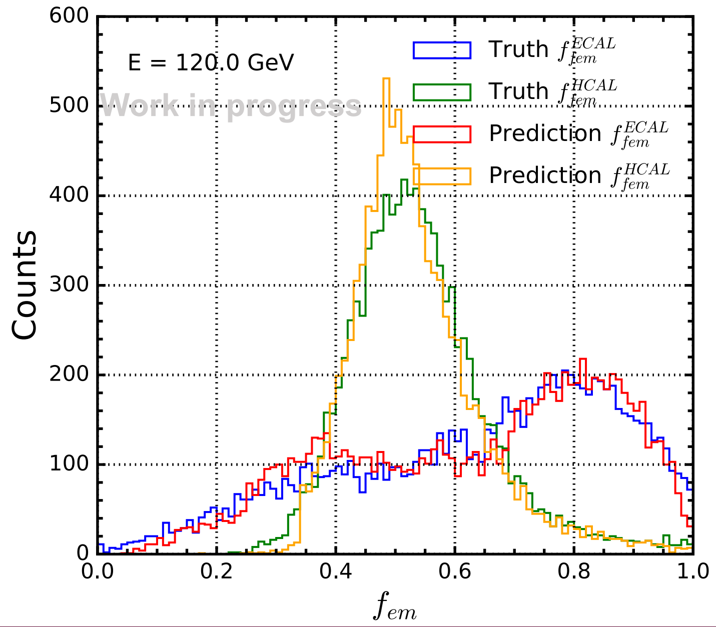
- Performance degrades at lower energies → requires refinement
- Futher step: reconstruct energy by $\frac{e}{h}$ and f_{em}







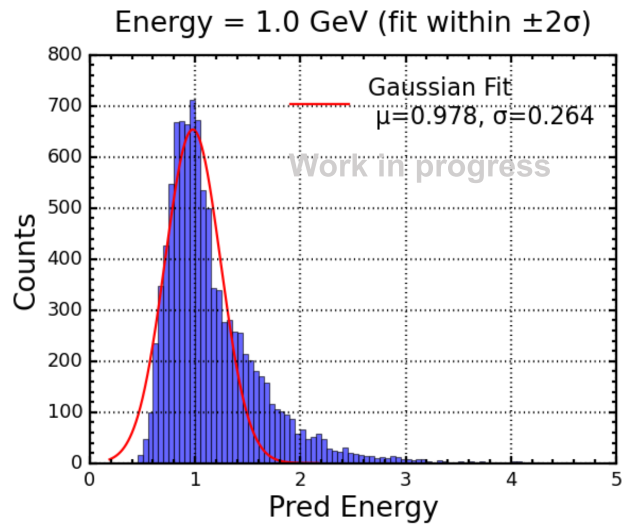


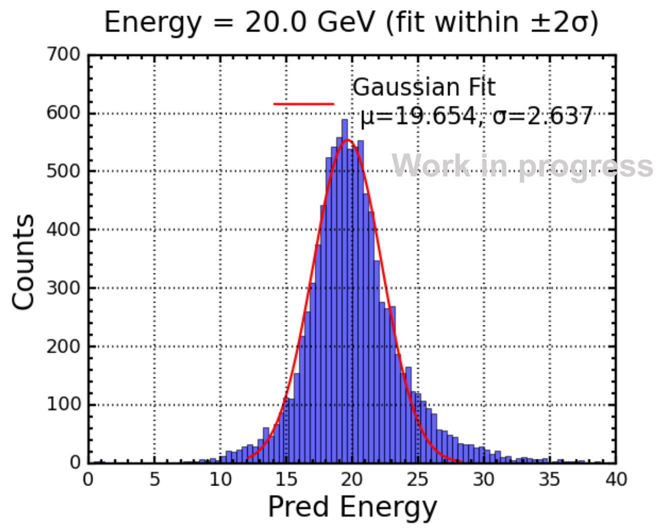


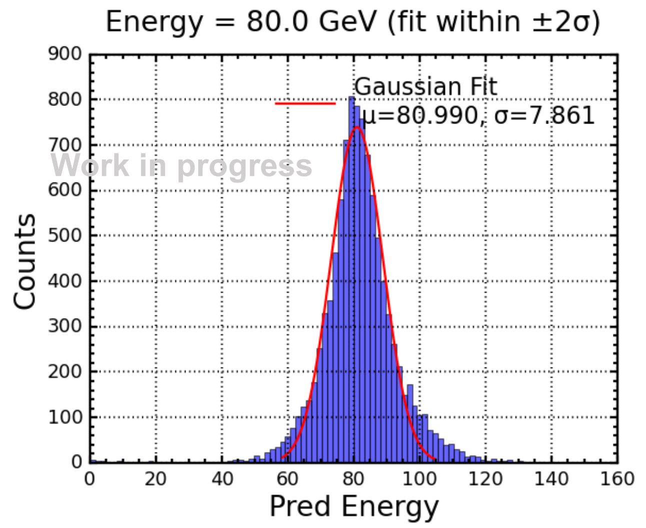
Results – energy regression

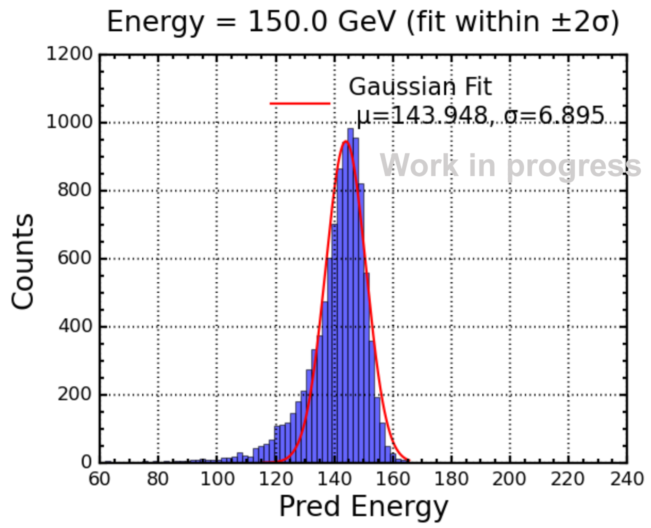
Results of energy prediction :

- Performance degradation at low and high energies
  - Low energies
    - *limited statistics*
  - High energies
    - energy leakage
  - Need future study





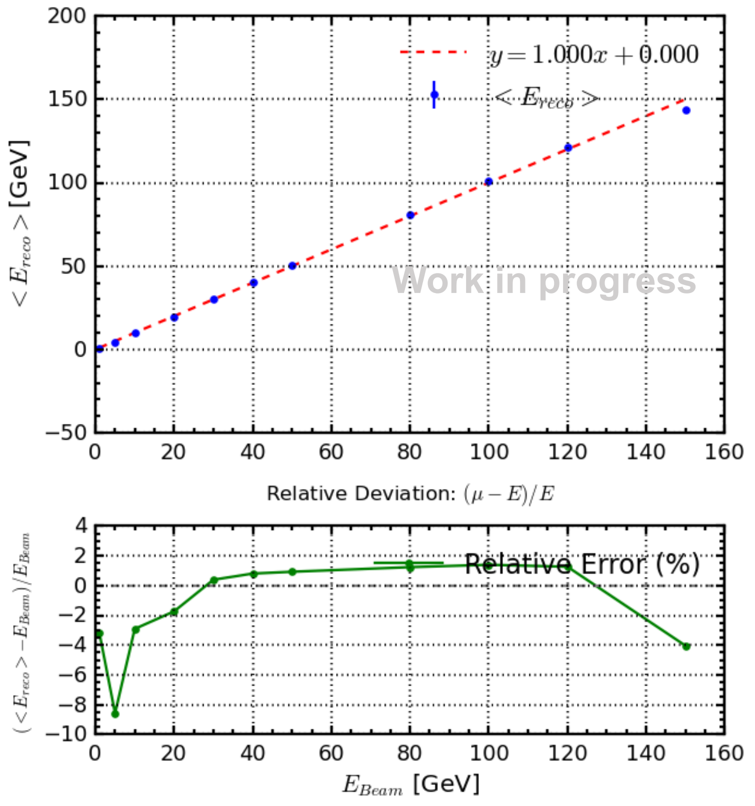


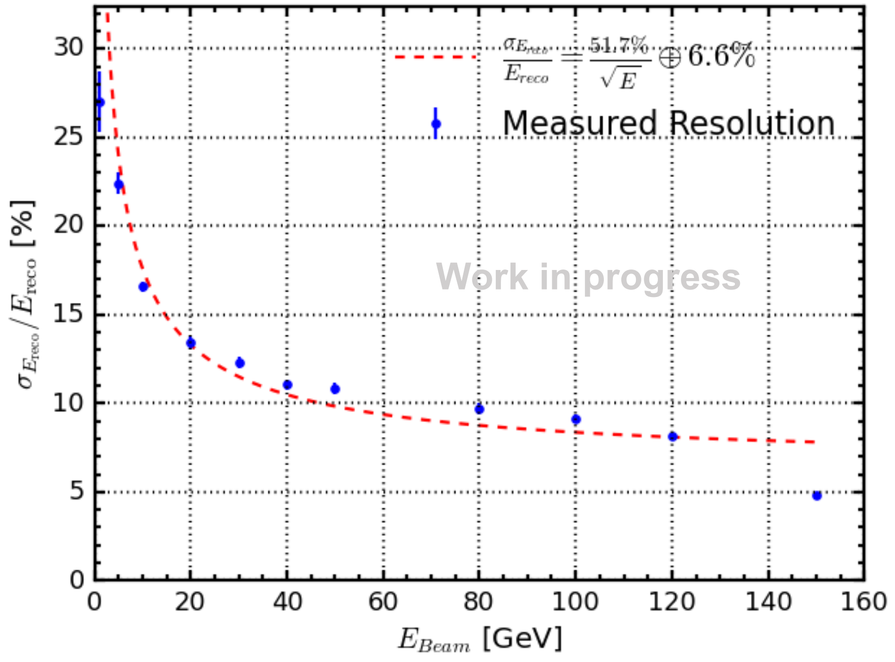


Results – energy regression

Preliminary results of energy reconstruction:

- Very preliminary → strong limitations at low & high energies
- Further refinement and validation required





Summary & Plans

Summary:

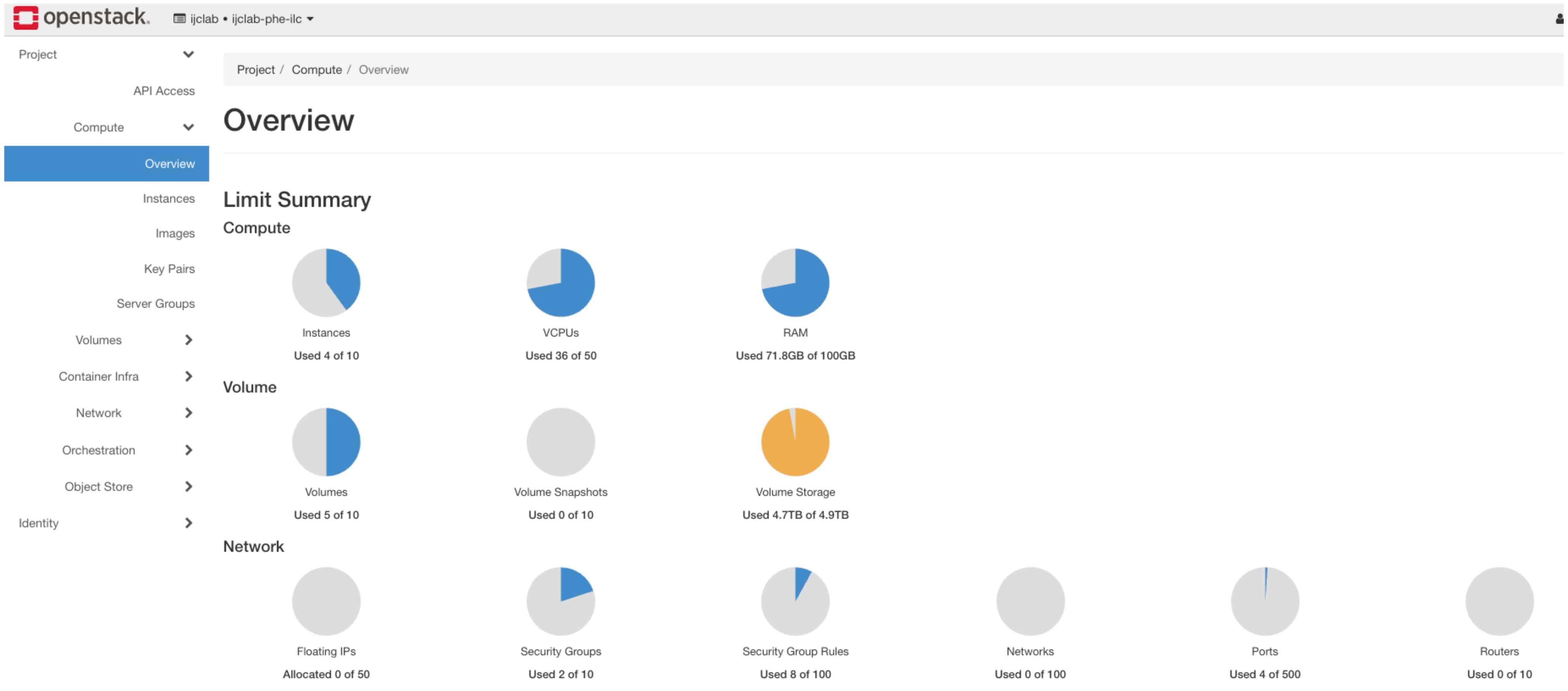
- Established a full software compensation chain using a dual-branch CNN with Si-W ECAL + AHCAL system
- f_{em} regression model shows accurate prediction, showing strong potential for energy reconstruction
- Energy regression: get very preliminary results, showing potential; low and high energy regions require further improvement

Plans:

- Determine the energy resolution based on f_{em}
- Explore the use of timing information to further enhance reconstruction performance

Thank you!

Backup


openstack. ijclab • ijclab-phe-ilc

Project
API Access
Compute
Overview
Instances
Images
Key Pairs
Server Groups
Volumes
Container Infra
Network
Orchestration
Object Store
Identity

Project / Compute / Overview

Overview

Limit Summary

Compute

Instances
Used 4 of 10

VCPUs
Used 36 of 50

RAM
Used 71.8GB of 100GB

Volume

Volumes
Used 5 of 10

Volume Snapshots
Used 0 of 10

Volume Storage
Used 4.7TB of 4.9TB

Network

Floating IPs
Allocated 0 of 50

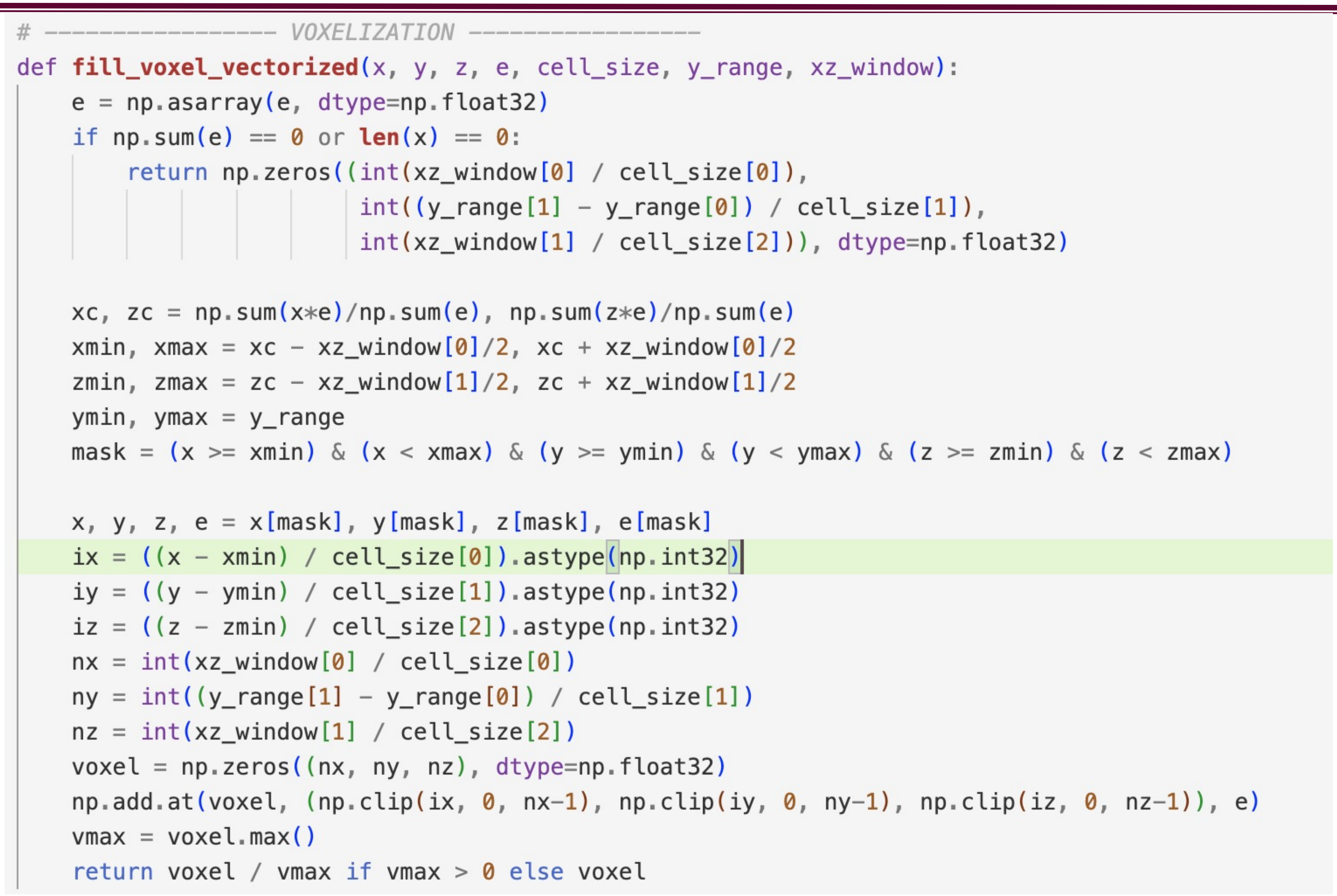
Security Groups
Used 2 of 10

Security Group Rules
Used 8 of 100

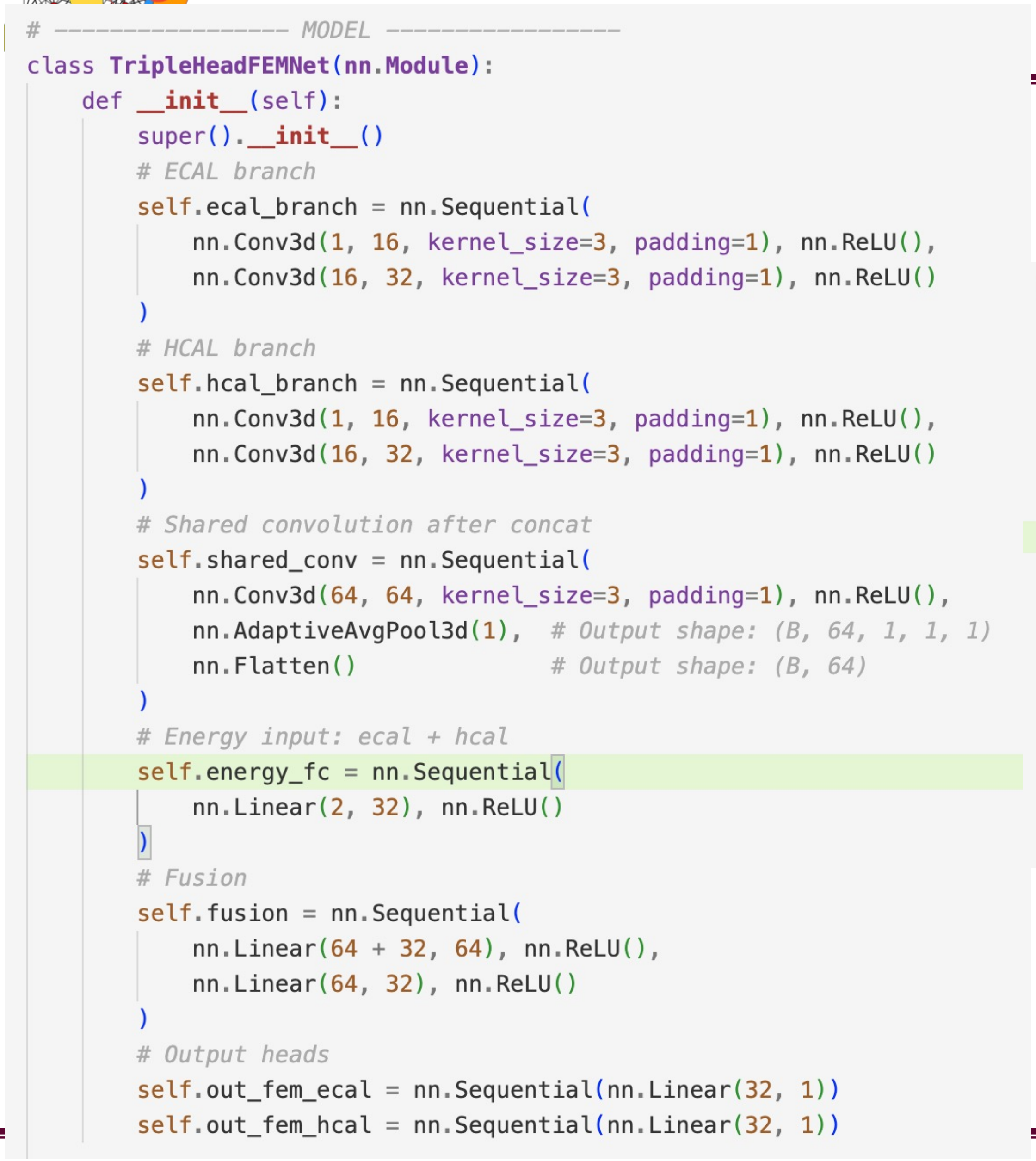
Networks
Used 0 of 100

Ports
Used 4 of 500

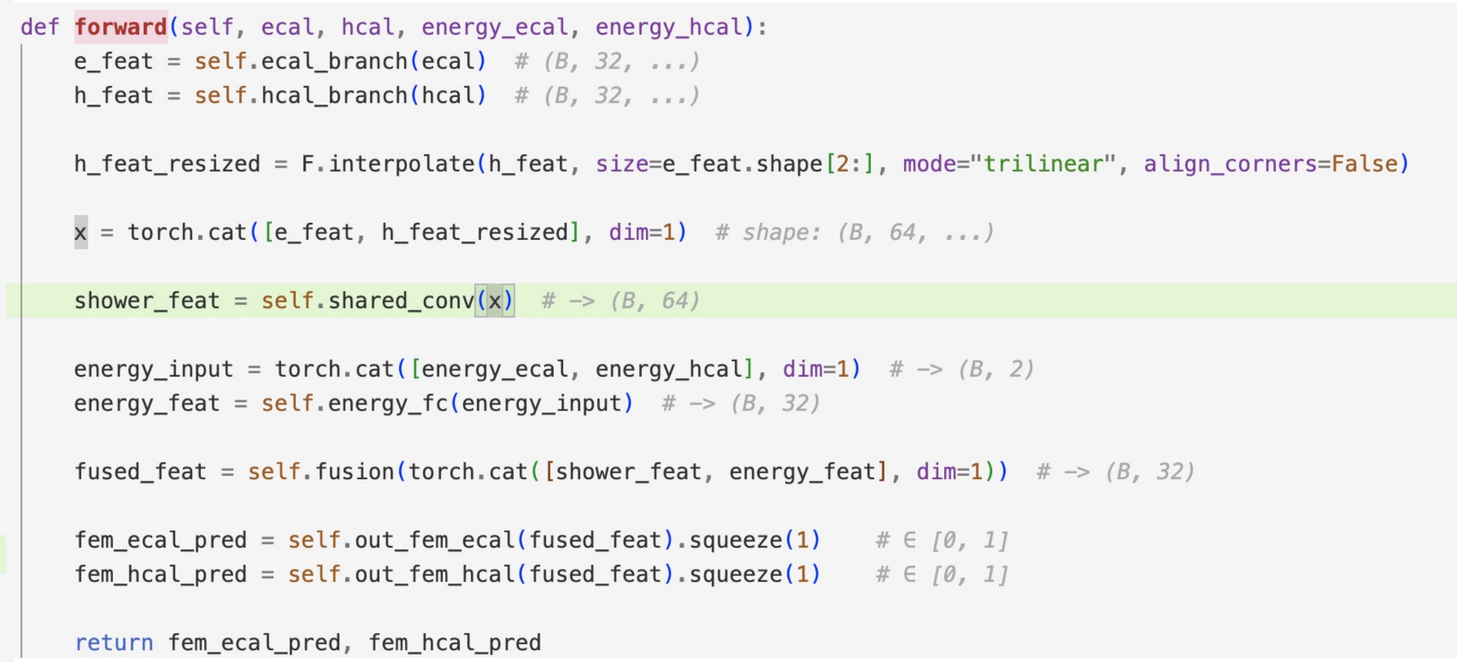
Routers
Used 0 of 10



```python
# ----------------- VOXELIZATION -----------------
def fill_voxel_vectorized(x, y, z, e, cell_size, y_range, xz_window):
    e = np.asarray(e, dtype=np.float32)
    if np.sum(e) == 0 or len(x) == 0:
        return np.zeros((int(xz_window[0] / cell_size[0]),
                         int((y_range[1] - y_range[0]) / cell_size[1]),
                         int(xz_window[1] / cell_size[2])), dtype=np.float32)

    xc, zc = np.sum(x*e)/np.sum(e), np.sum(z*e)/np.sum(e)
    xmin, xmax = xc - xz_window[0]/2, xc + xz_window[0]/2
    zmin, zmax = zc - xz_window[1]/2, zc + xz_window[1]/2
    ymin, ymax = y_range
    mask = (x >= xmin) & (x < xmax) & (y >= ymin) & (y < ymax) & (z >= zmin) & (z < zmax)

    x, y, z, e = x[mask], y[mask], z[mask], e[mask]
    ix = ((x - xmin) / cell_size[0]).astype(np.int32)
    iy = ((y - ymin) / cell_size[1]).astype(np.int32)
    iz = ((z - zmin) / cell_size[2]).astype(np.int32)
    nx = int(xz_window[0] / cell_size[0])
    ny = int((y_range[1] - y_range[0]) / cell_size[1])
    nz = int(xz_window[1] / cell_size[2])
    voxel = np.zeros((nx, ny, nz), dtype=np.float32)
    np.add.at(voxel, (np.clip(ix, 0, nx-1), np.clip(iy, 0, ny-1), np.clip(iz, 0, nz-1)), e)
    vmax = voxel.max()
    return voxel / vmax if vmax > 0 else voxel
```

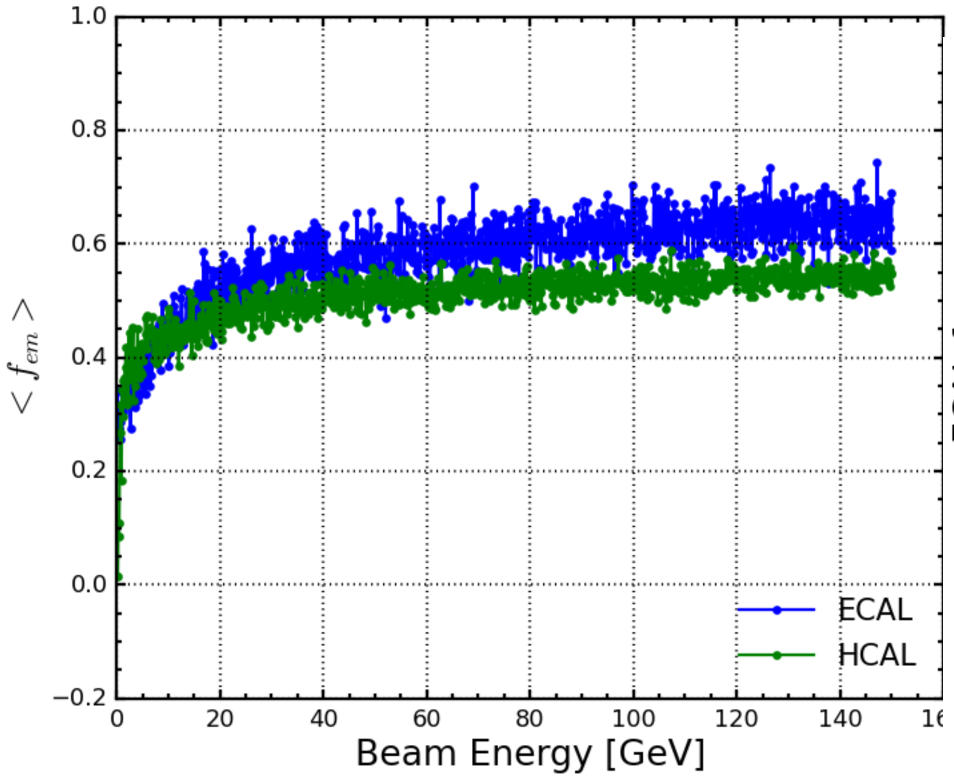


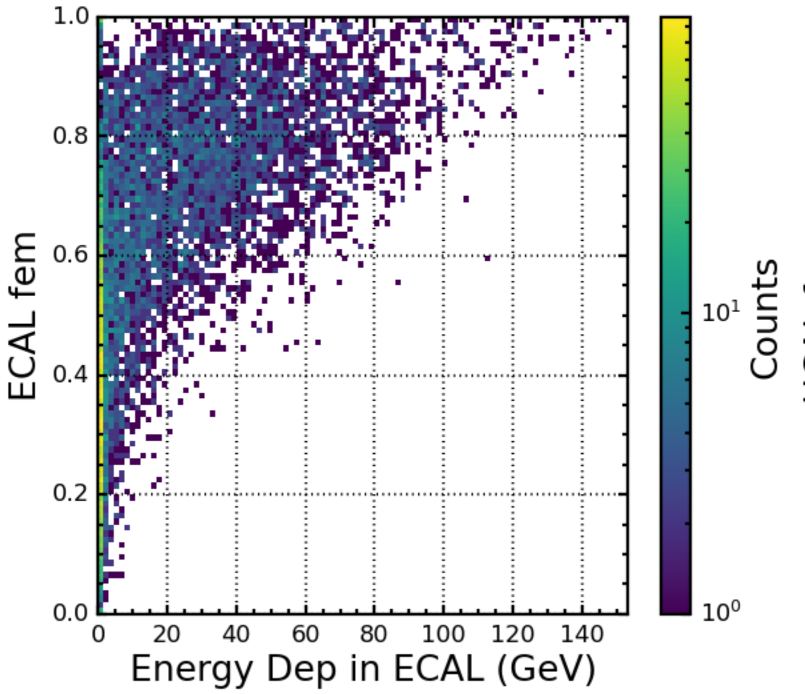
```python
# ------------------ MODEL -----------------
class TripleHeadFEMNet(nn.Module):
    def __init__(self):
        super().__init__()
        # ECAL branch
        self.ecal_branch = nn.Sequential(
            nn.Conv3d(1, 16, kernel_size=3, padding=1), nn.ReLU(),
            nn.Conv3d(16, 32, kernel_size=3, padding=1), nn.ReLU()
        )
        # HCAL branch
        self.hcal_branch = nn.Sequential(
            nn.Conv3d(1, 16, kernel_size=3, padding=1), nn.ReLU(),
            nn.Conv3d(16, 32, kernel_size=3, padding=1), nn.ReLU()
        )
        # Shared convolution after concat
        self.shared_conv = nn.Sequential(
            nn.Conv3d(64, 64, kernel_size=3, padding=1), nn.ReLU(),
            nn.AdaptiveAvgPool3d(1),  # Output shape: (B, 64, 1, 1, 1)
            nn.Flatten()              # Output shape: (B, 64)
        )
        # Energy input: ecal + hcal
        self.energy_fc = nn.Sequential(
            nn.Linear(2, 32), nn.ReLU()
        )
        # Fusion
        self.fusion = nn.Sequential(
            nn.Linear(64 + 32, 64), nn.ReLU(),
            nn.Linear(64, 32), nn.ReLU()
        )
        # Output heads
        self.out_fem_ecal = nn.Sequential(nn.Linear(32, 1))
        self.out_fem_hcal = nn.Sequential(nn.Linear(32, 1))
```

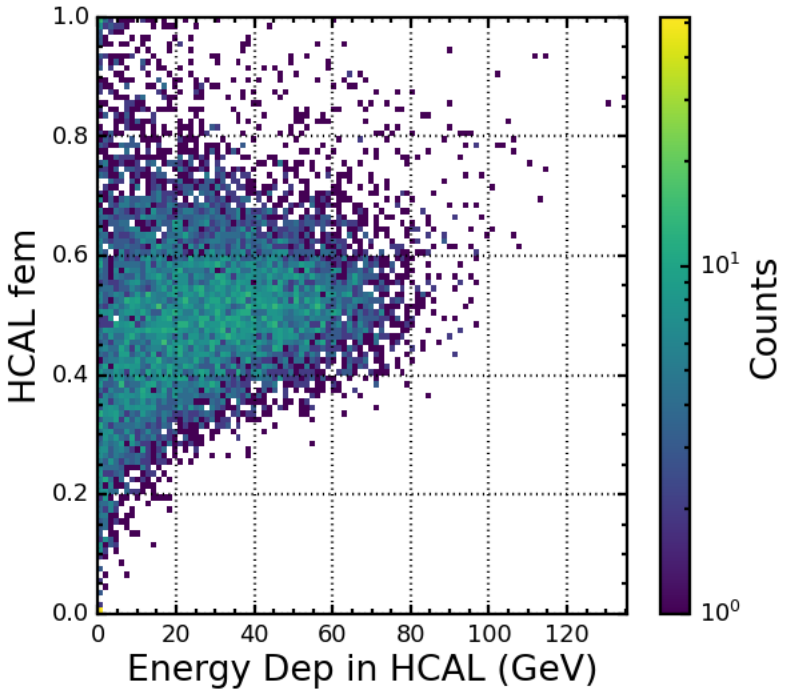


```python
def forward(self, ecal, hcal, energy_ecal, energy_hcal):
    e_feat = self.ecal_branch(ecal)  # (B, 32, ...)
    h_feat = self.hcal_branch(hcal)  # (B, 32, ...)

    h_feat_resized = F.interpolate(h_feat, size=e_feat.shape[2:], mode="trilinear", align_corners=False)

    x = torch.cat([e_feat, h_feat_resized], dim=1)  # shape: (B, 64, ...)

    shower_feat = self.shared_conv(x)  # -> (B, 64)

    energy_input = torch.cat([energy_ecal, energy_hcal], dim=1)  # -> (B, 2)
    energy_feat = self.energy_fc(energy_input)  # -> (B, 32)

    fused_feat = self.fusion(torch.cat([shower_feat, energy_feat], dim=1))  # -> (B, 32)

    fem_ecal_pred = self.out_fem_ecal(fused_feat).squeeze(1)    # ∈ [0, 1]
    fem_hcal_pred = self.out_fem_hcal(fused_feat).squeeze(1)    # ∈ [0, 1]

    return fem_ecal_pred, fem_hcal_pred
```

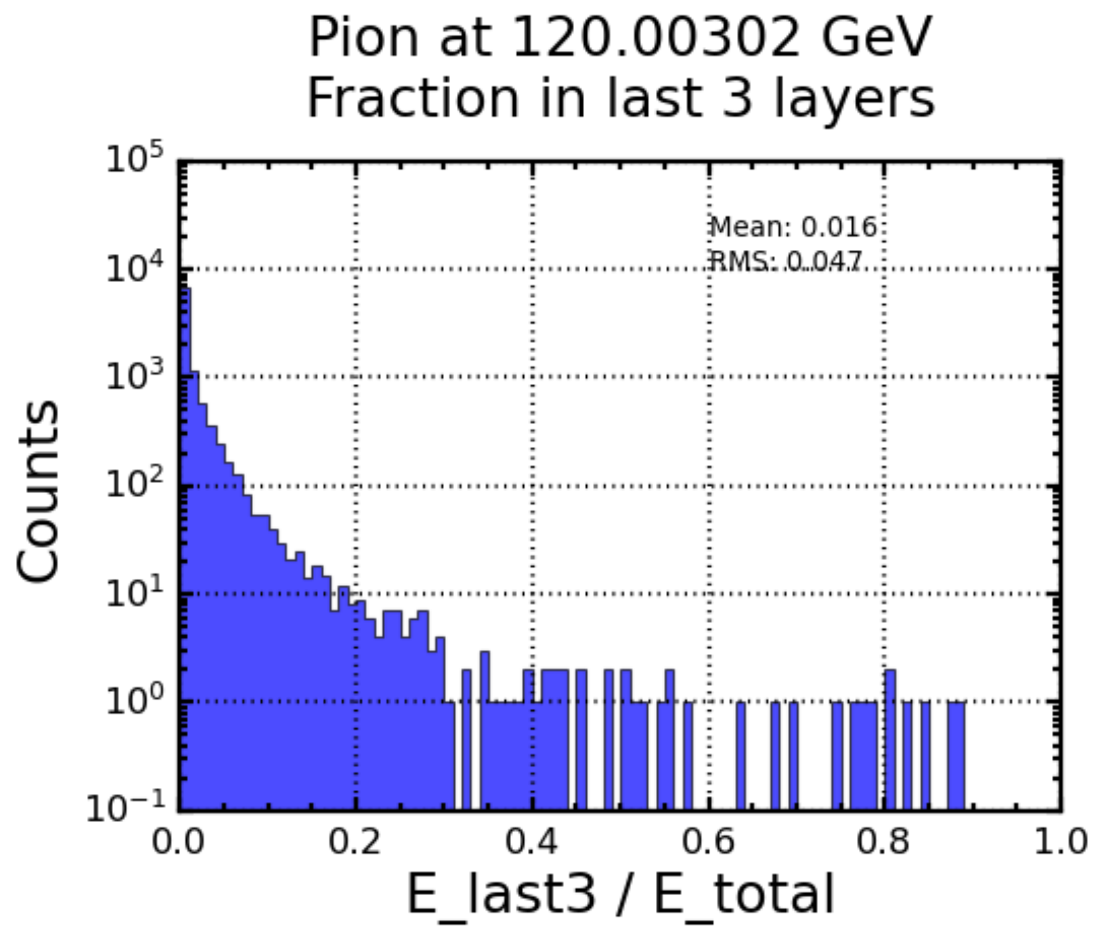
f_{em} behaviour

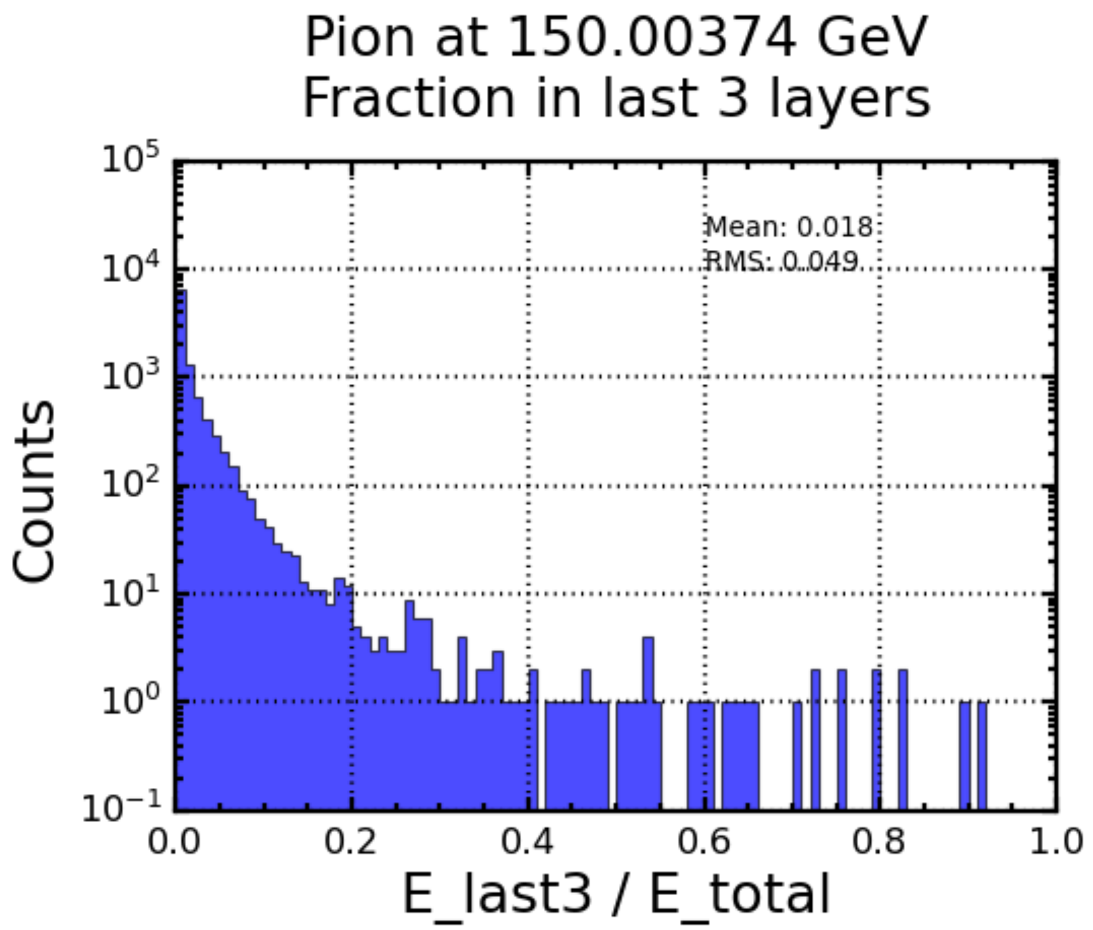




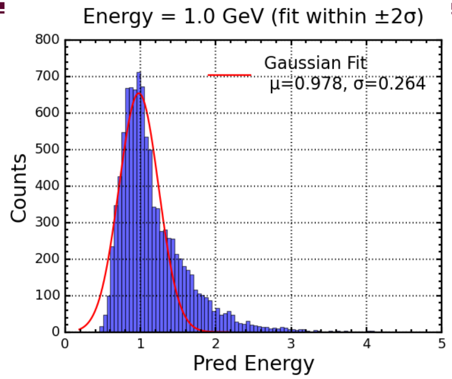


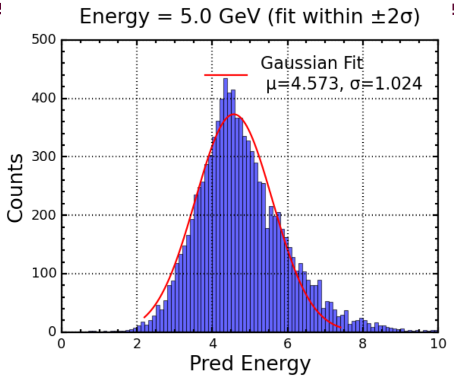
Energy leakage

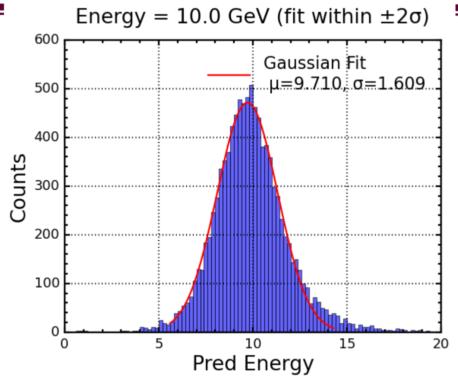


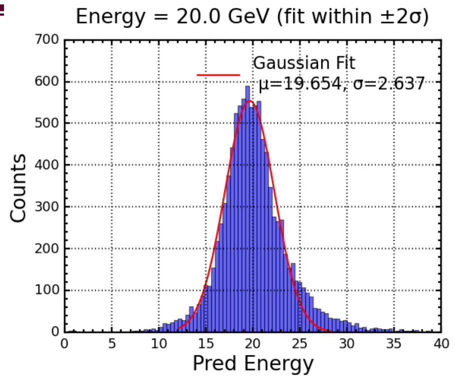


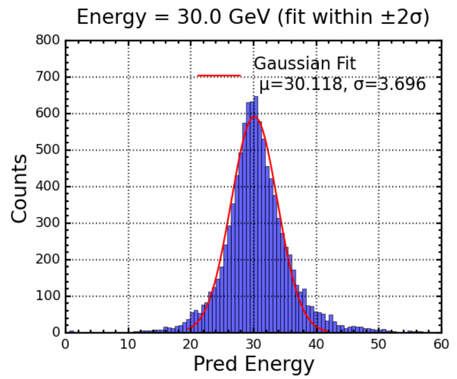
All predicted energy distribution

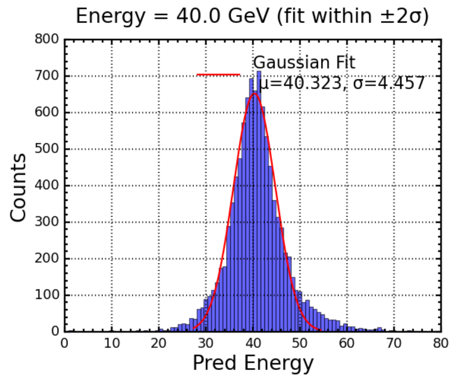

Energy = 1.0 GeV (fit within ±2σ)
Gaussian Fit
μ=0.978, σ=0.264
Counts
Pred Energy

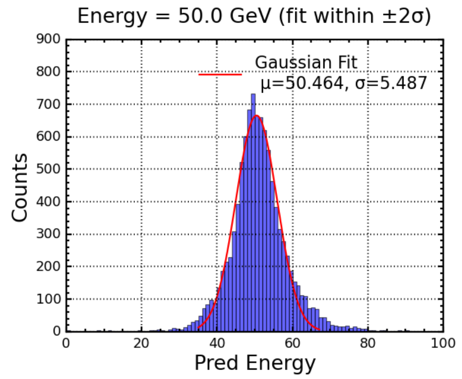

Energy = 5.0 GeV (fit within ±2σ)
Gaussian Fit
μ=4.573, σ=1.024
Counts
Pred Energy

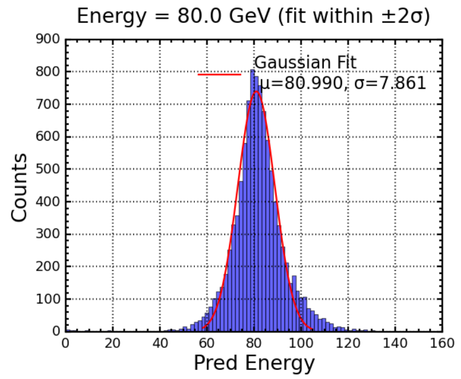

Energy = 10.0 GeV (fit within ±2σ)
Gaussian Fit
μ=9.710, σ=1.609
Counts
Pred Energy

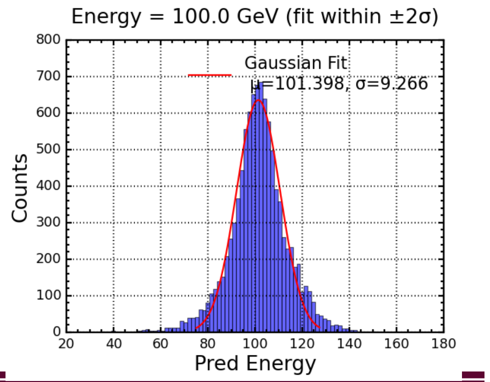

Energy = 20.0 GeV (fit within ±2σ)
Gaussian Fit
μ=19.654, σ=2.637
Counts
Pred Energy

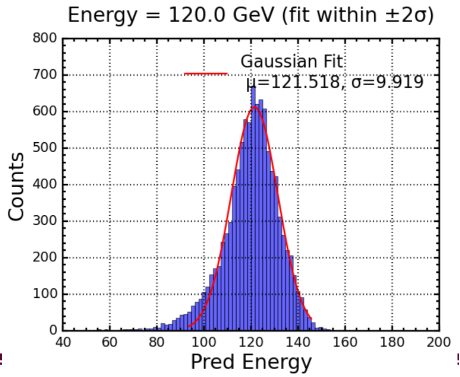

Energy = 30.0 GeV (fit within ±2σ)
Gaussian Fit
μ=30.118, σ=3.696
Counts
Pred Energy

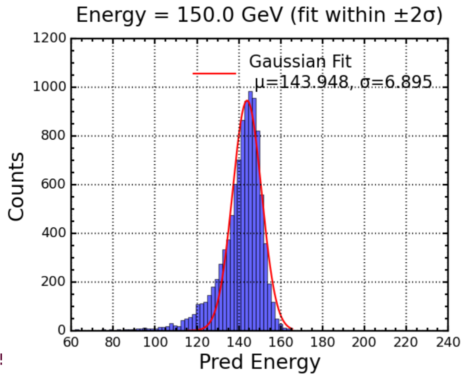

Energy = 40.0 GeV (fit within ±2σ)
Gaussian Fit
μ=40.323, σ=4.457
Counts
Pred Energy


Energy = 50.0 GeV (fit within ±2σ)
Gaussian Fit
μ=50.464, σ=5.487
Counts
Pred Energy


Energy = 80.0 GeV (fit within ±2σ)
Gaussian Fit
μ=80.990, σ=7.861
Counts
Pred Energy


Energy = 100.0 GeV (fit within ±2σ)
Gaussian Fit
μ=101.398, σ=9.266
Counts
Pred Energy


Energy = 120.0 GeV (fit within ±2σ)
Gaussian Fit
μ=121.518, σ=9.919
Counts
Pred Energy


Energy = 150.0 GeV (fit within ±2σ)
Gaussian Fit
μ=143.948, σ=6.895
Counts
Pred Energy