LAGAN and CaloGAN:

Generative Adversarial Networks for Jet Images and Calorimeter Simulation

Michela Paganini*, Luke de Oliveira, Ben Nachman

Yale

BERKELEY LAB

Layout

Yale

Main point: **we have a working solution!**

also see Sofia's talk

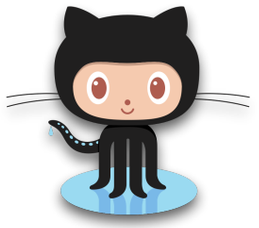
- Problem we are trying to tackle = calo simulation

also see Josh's and Amir's talks

- LAGAN `arXiv:1701.05927`
- CaloGAN `arXiv:1705.02355`

Emphasis on **reproducibility**:

also see Victoria + Matt's talk

Data	Code
DOI 10.17632/pvn3xc3wy5.1 zenodo MENDELEY DATA	github.com/hep-lbdl/CaloGAN  

Simulation

Yale

Theory

$$\mathcal{L} = -\frac{1}{4} F_{\mu\nu} F^{\mu\nu}$$
$$+ i \bar{\Psi} \not{D} \Psi + h.c.$$
$$+ \bar{\Psi}_i y_{ij} \Psi_j \phi + h.c.$$
$$+ |D_\mu \phi|^2 - V(\phi)$$

Hard Interactions (ME Calculations)

Parton Showering & Hadronization

Detector Sim. & material Interactions

Geant 4

Digitization

...

Simulation

Yale

Theory

Hard Interactions (ME Calculations)

LAGAN

Parton Showering & Hadronization

Detector Sim. & material Interactions

Digitization

...

Simulation

Yale

Theory

Hard Interactions (ME Calculations)

Parton Showering & Hadronization

Detector Sim. & material Interactions

Digitization

...

CaloGAN

Geant 4

Fast Simulation

Yale

1. “**Fast Simulation**”:

    - Parametrized showers, frozen showers, …
    - Not accurate enough

2. **Deep Learning** approaches:

    - Variational Auto-Encoders, Autoregressive Models, Generative Adversarial Networks, …

LAGAN

LAGAN for Jet Images

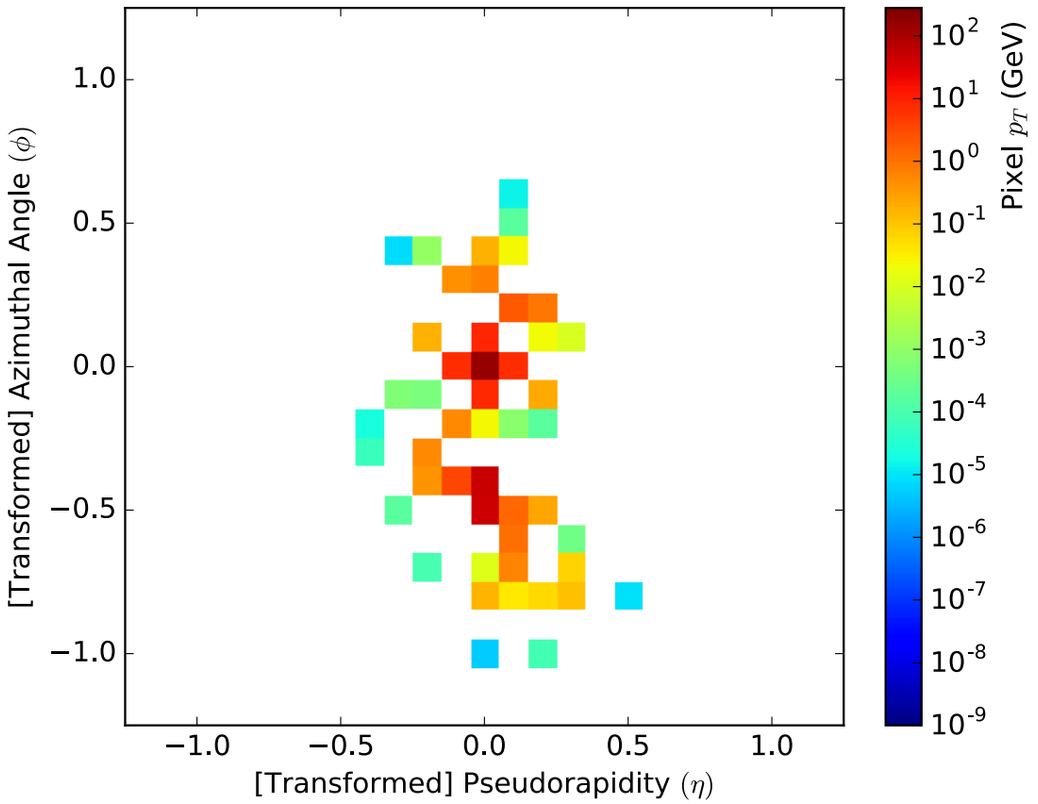
Yale

see my IML talk

see Ben's talk

arXiv:1701.05927

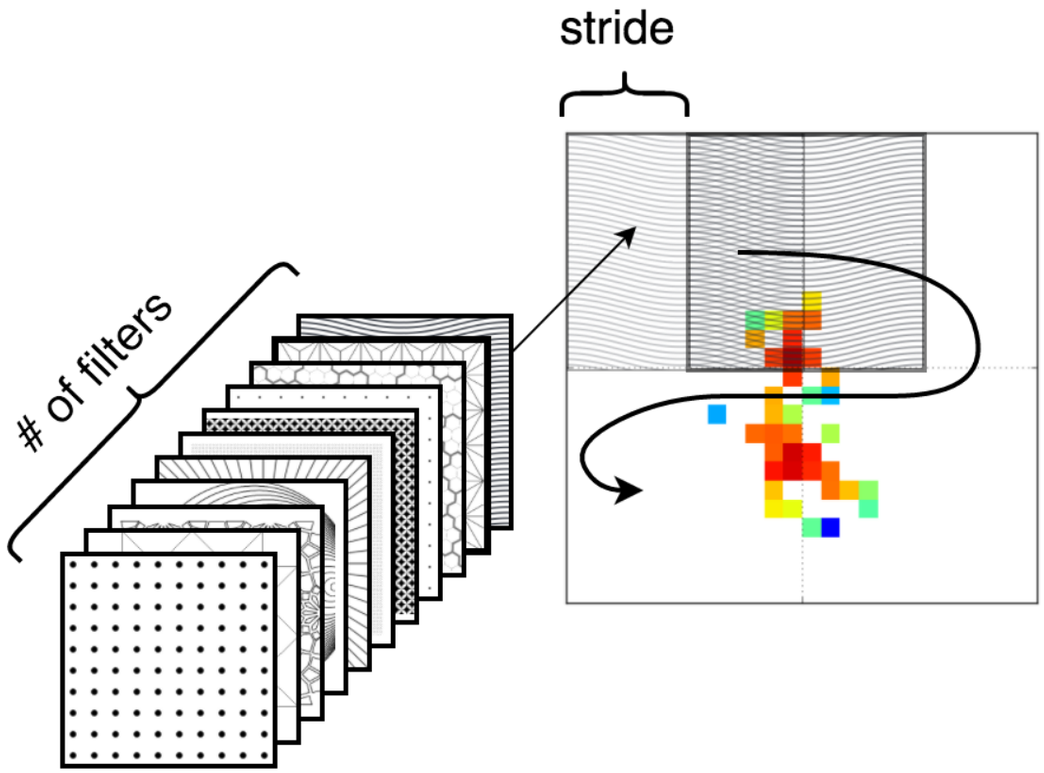
- Modification of DCGAN and ACGAN



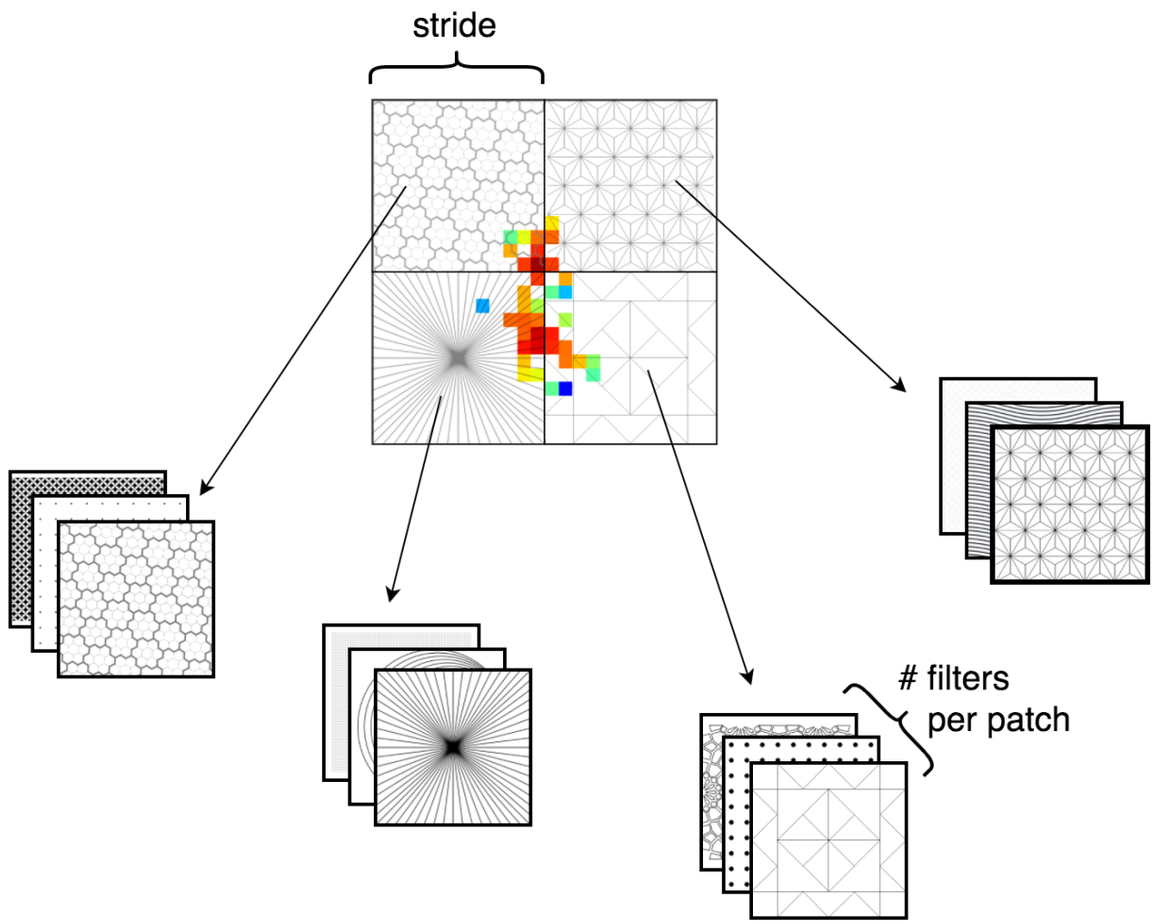
- Ad-hoc design to fit Physics data:
  - sparsity
  - high dynamic range
  - highly location-dependent features

LAGAN

Yale



Convolutional Layer
(weight sharing)

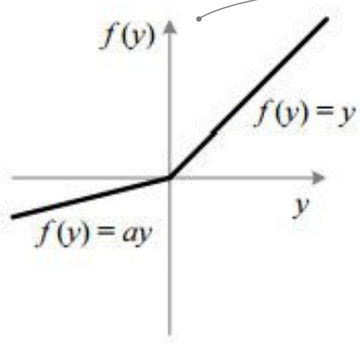


Locally-Connected Layer

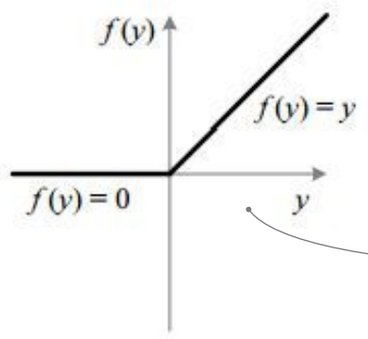
LAGAN

Yale

- Activation:



Leaky ReLU

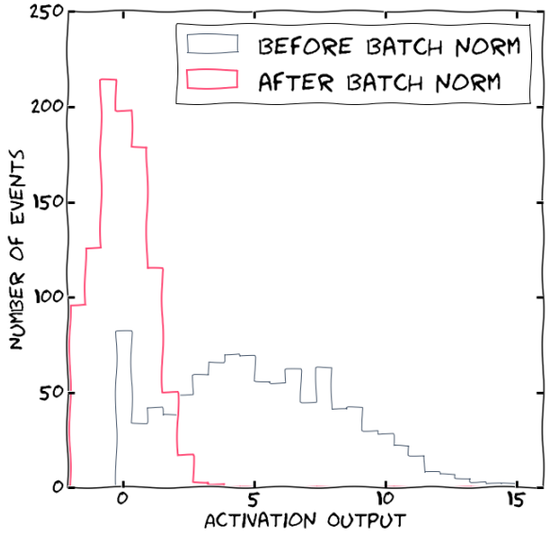


ReLU

recommended in GAN literature

to induce sparsity

- Batch Normalization:

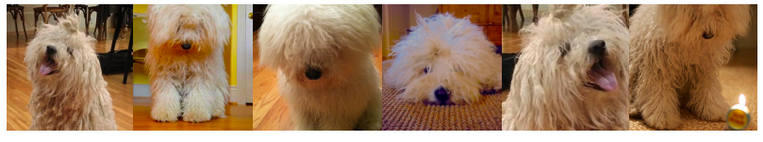


helps with high dynamic range

- Mini-batch Discrimination:

helps reduce mode collapse

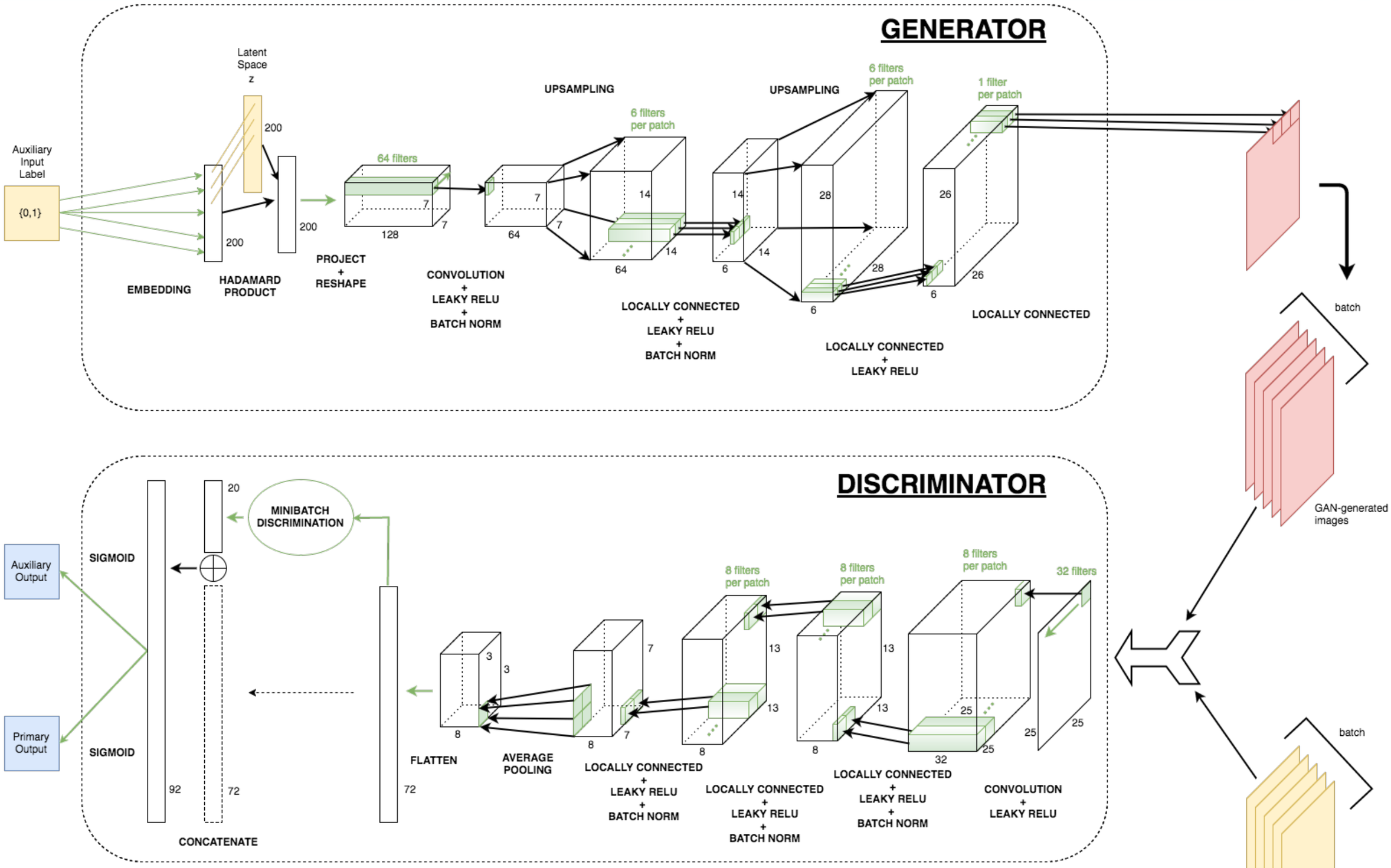
→ High Entropy



→ Low Entropy

LAGAN

Yale


GENERATOR
Auxiliary Input Label
{0,1}
Latent Space z
200
200
200
EMBEDDING
HADAMARD PRODUCT
PROJECT + RESHAPE
64 filters
128
7
7
CONVOLUTION + LEAKY RELU + BATCH NORM
64
UPSAMPLING
6 filters per patch
LOCALLY CONNECTED + LEAKY RELU + BATCH NORM
UPSAMPLING
6 filters per patch
LOCALLY CONNECTED + LEAKY RELU
1 filter per patch
LOCALLY CONNECTED
batch
GAN-generated images
DISCRIMINATOR
Auxiliary Output
Primary Output
SIGMOID
SIGMOID
20
MINIBATCH DISCRIMINATION
92
72
CONCATENATE
72
FLATTEN
AVERAGE POOLING
LOCALLY CONNECTED + LEAKY RELU + BATCH NORM
8 filters per patch
LOCALLY CONNECTED + LEAKY RELU + BATCH NORM
8 filters per patch
LOCALLY CONNECTED + LEAKY RELU + BATCH NORM
8 filters per patch
CONVOLUTION + LEAKY RELU
32 filters
batch
Pythia images

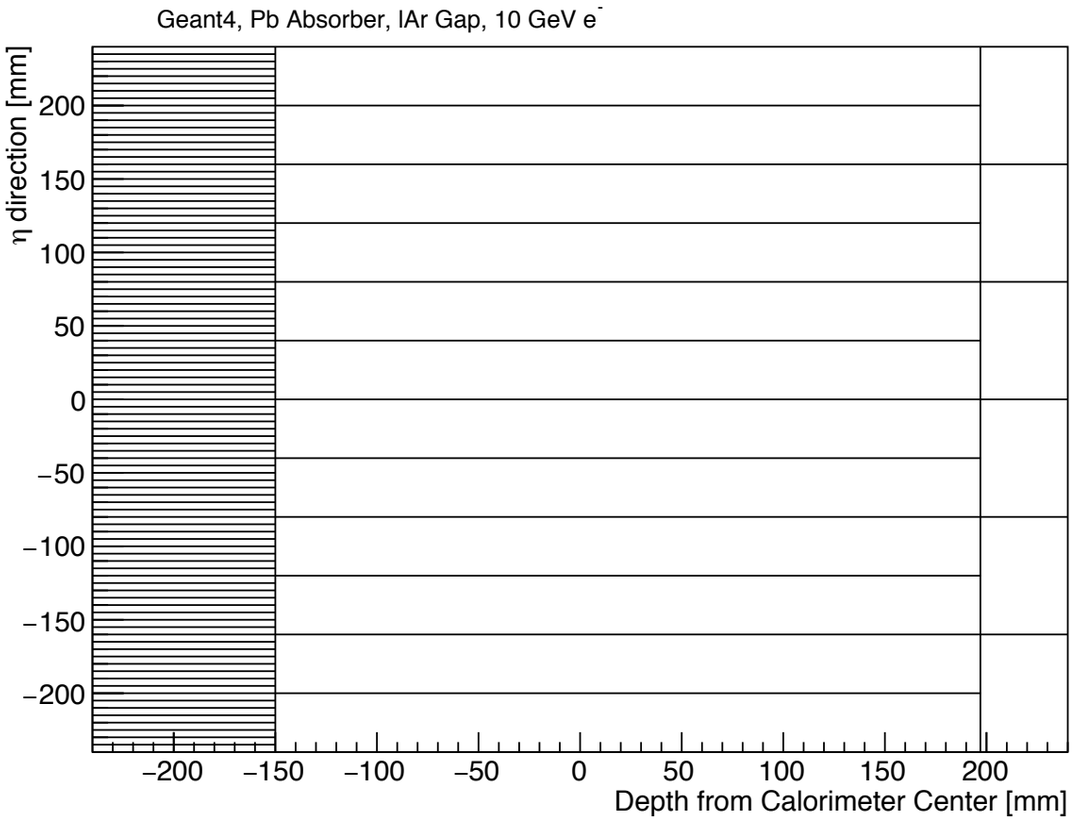
CaloGAN

Shower Images

Yale

- More realistic scenario.
- EM calorimeter drawing inspiration from the ATLAS geometry.
- Built with GEANT4.

- Heterogeneous longitudinal segmentation into **3 layers**.
- **Irregular granularity** in eta and phi.
- Sequence of alternating lead and liquid argon sublayers.

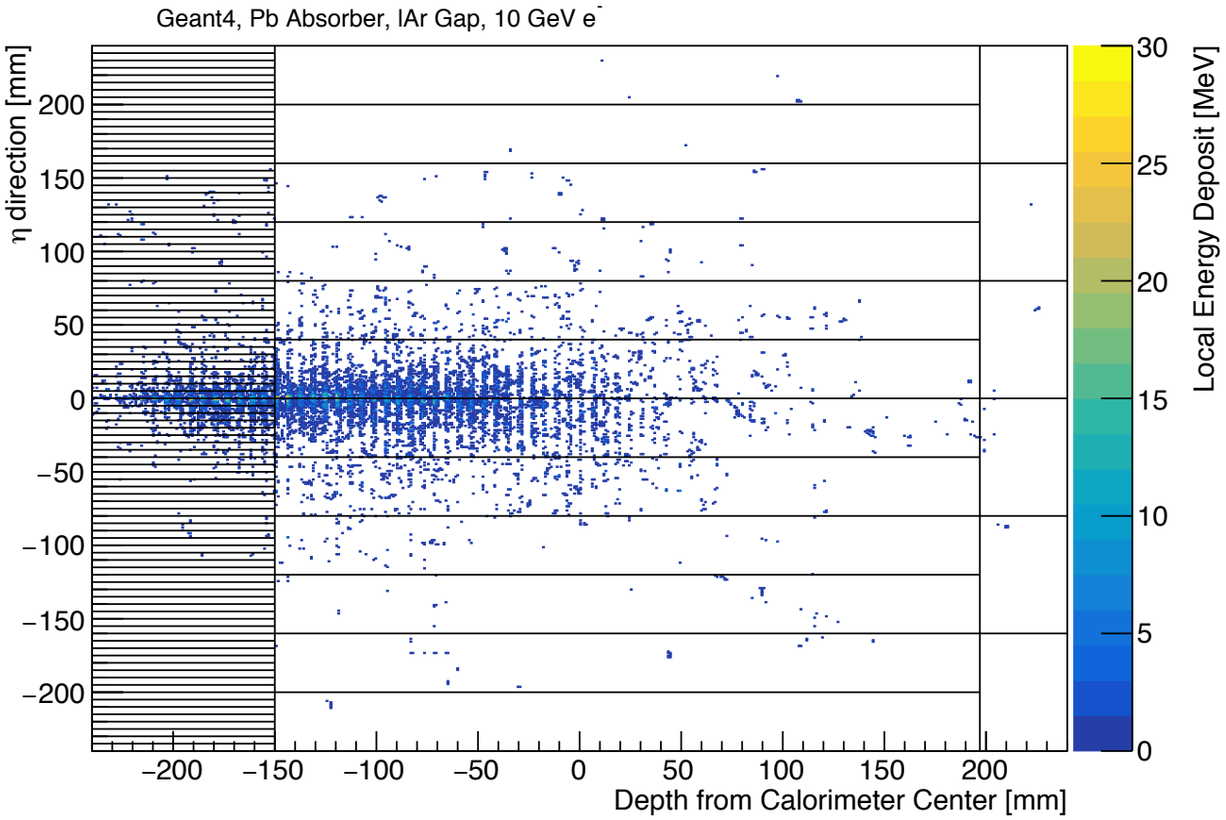


Shower Images

Yale

- More realistic scenario.
- EM calorimeter drawing inspiration from the ATLAS geometry.
- Built with GEANT4.

- Heterogeneous longitudinal segmentation into
- **Irregular granularity**
- Sequence of alternating lead and liquid argo



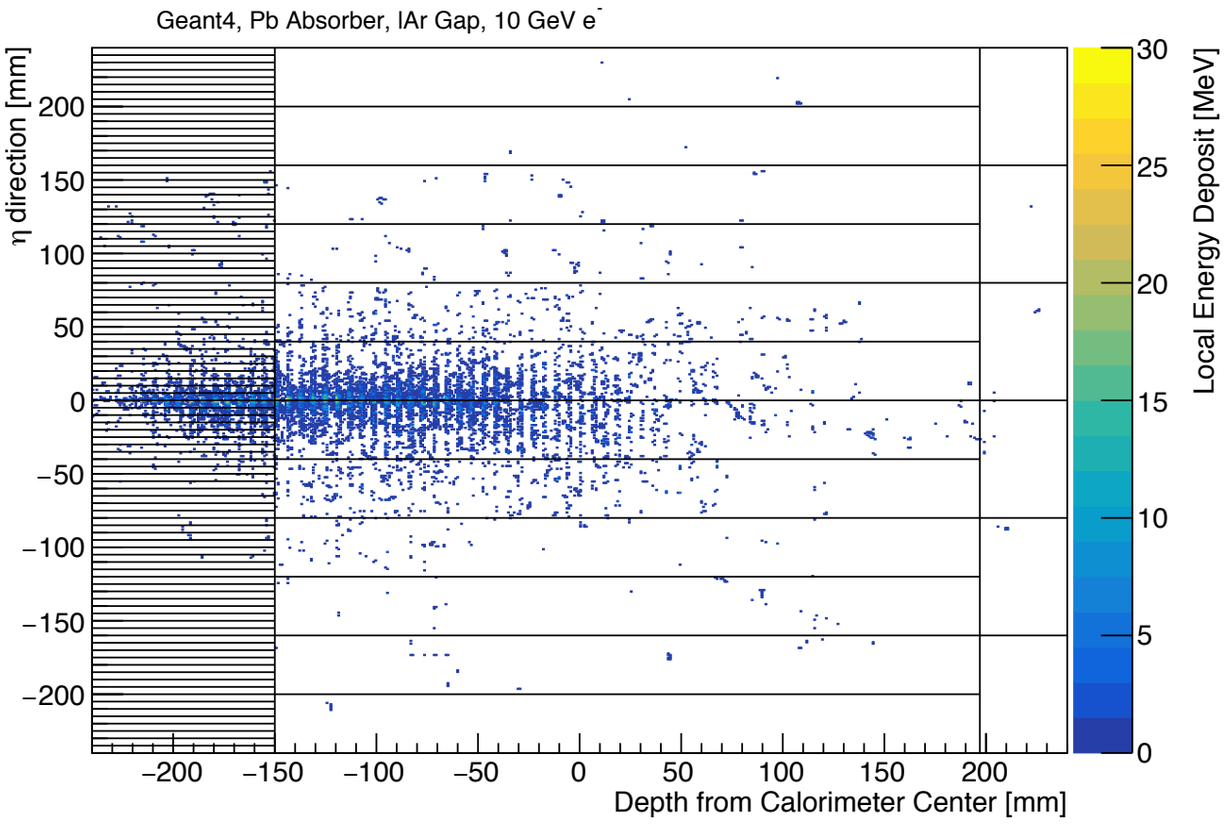
Simulated showers of e^+, pi^+ and y incident $\perp$ to the center of the detector with uniform energy in [1, 100] GeV

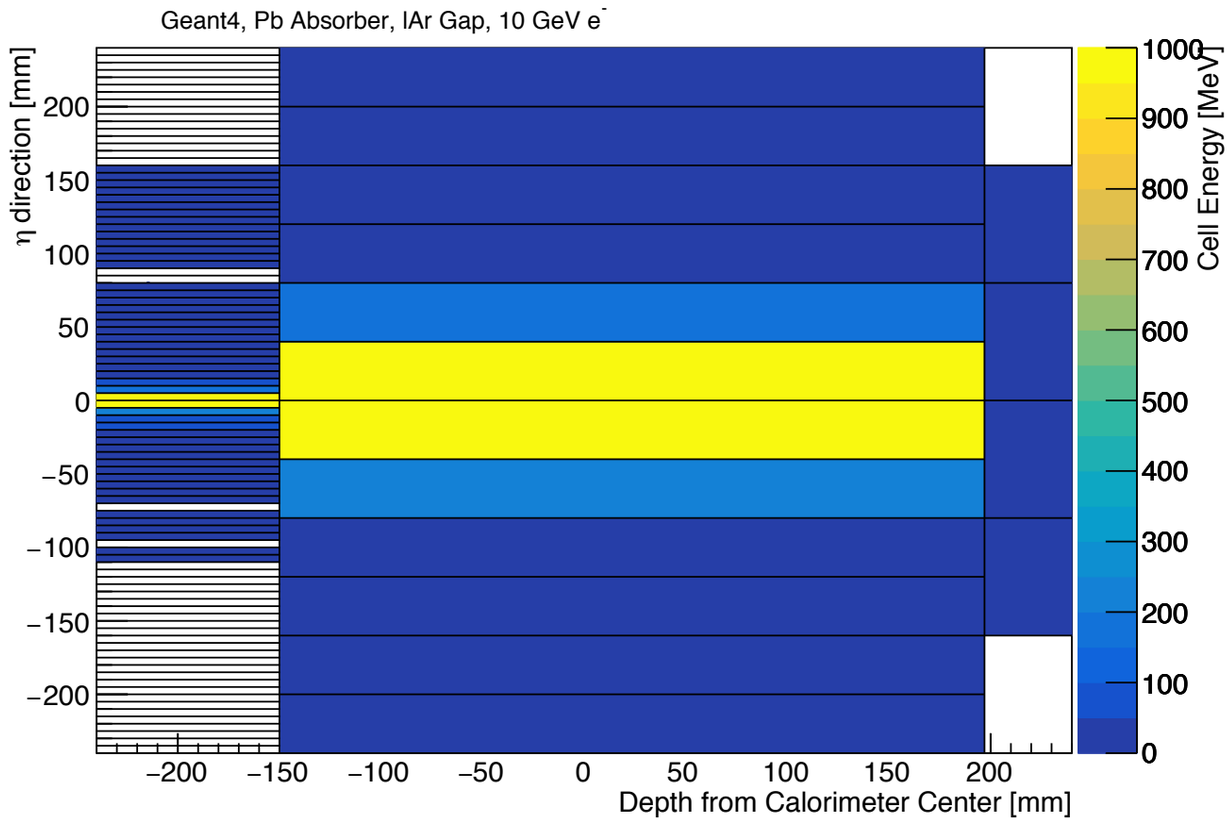
Shower Images

Yale

- More realistic scenario.
- EM calorimeter drawing inspiration from the ATLAS geometry.
- Built with GEANT4.

- Heterogeneous longitudinal segmentation into
- **Irregular granularity**
- Sequence of alternating lead and liquid argo





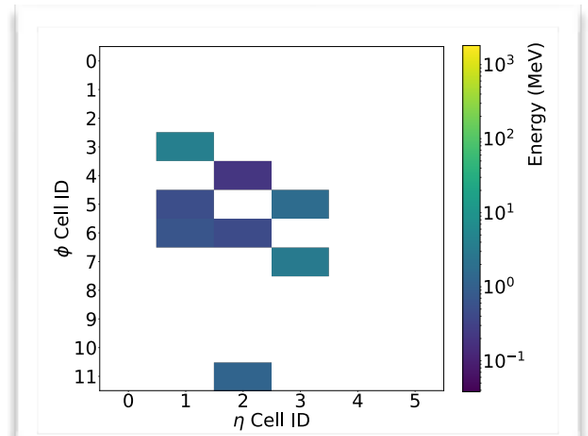
Simulated showers of e^+, pi^+ and y incident ⊥ to the center of the detector with a uniform energy in [1, 100] GeV

Shower Images

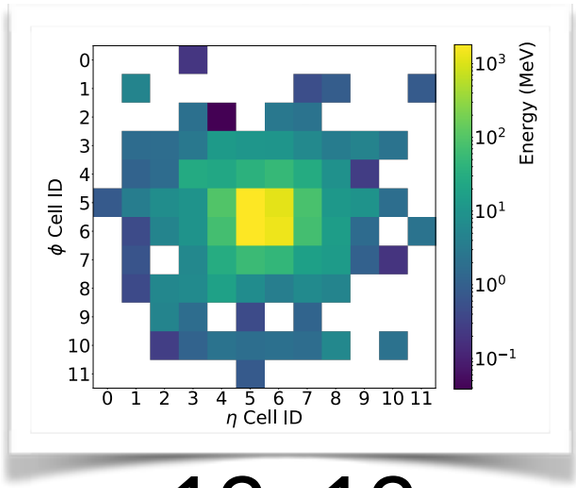
Yale

- Energy depositions in each layer as a **2D image**, similar to jet image

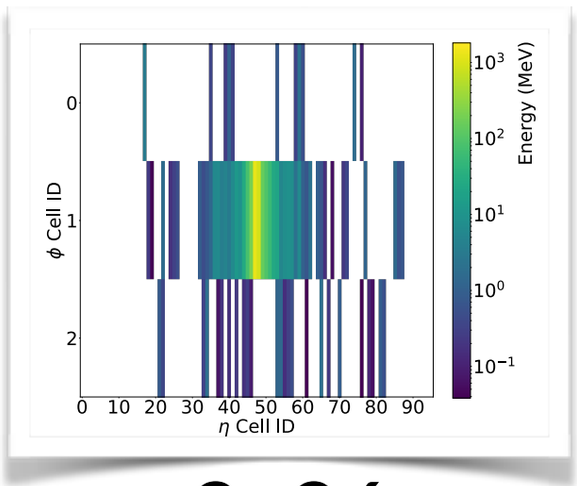
Layer	z segmentation [mm]	η segmentation [mm]	ϕ segmentation [mm]
0	90	5	160
1	347	40	40
2	43	80	40



12x6

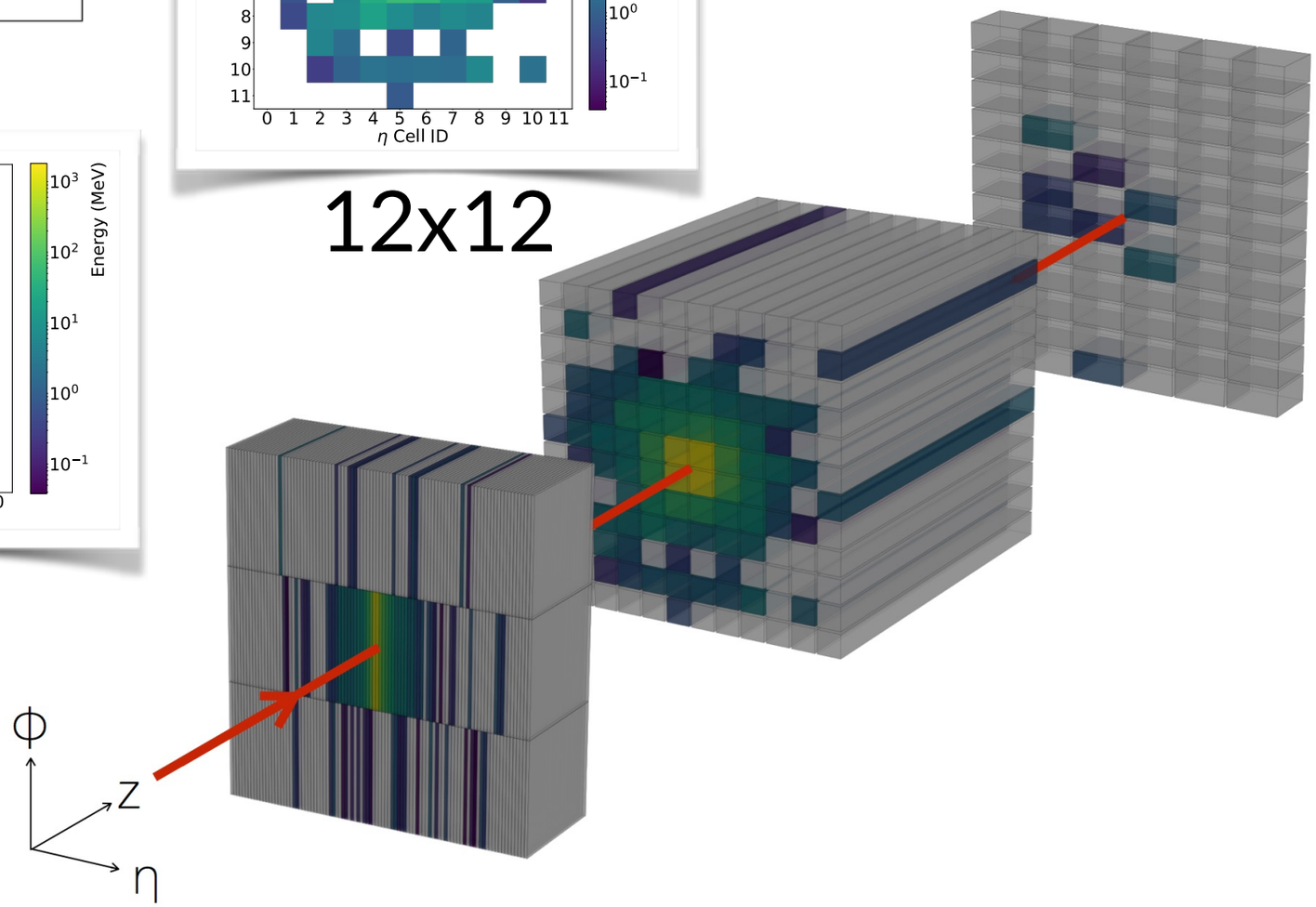


12x12



3x96

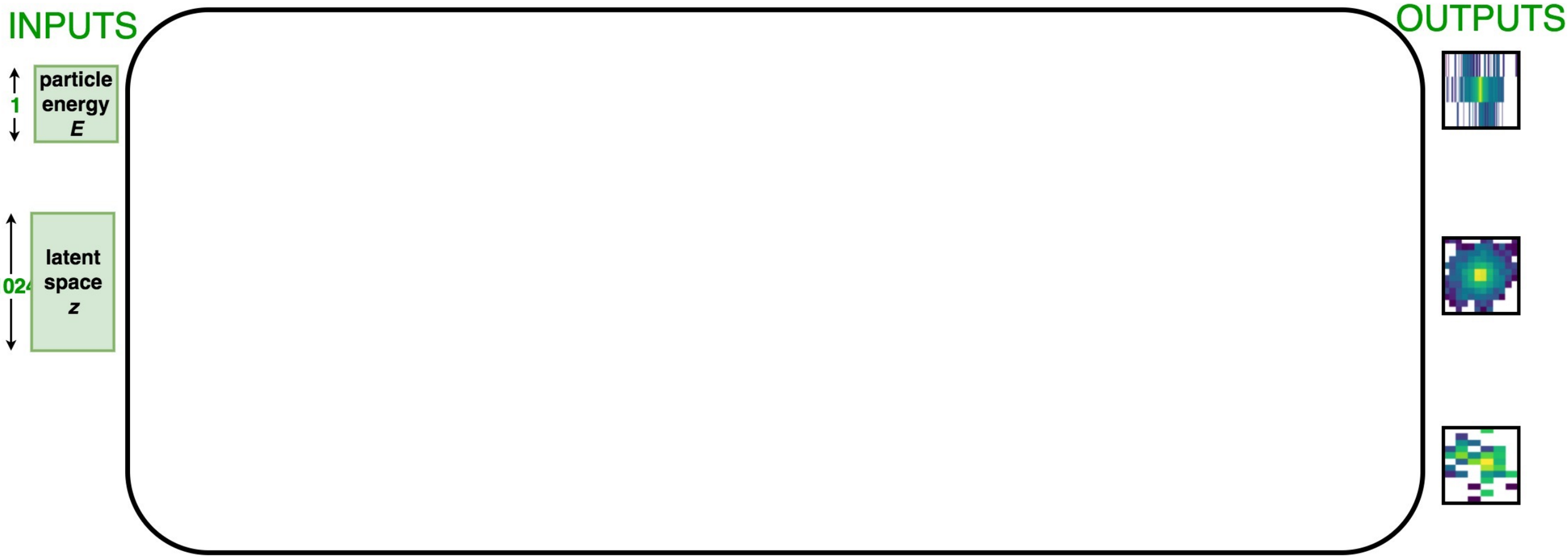
- <u>Goal</u>: generate this fixed representation



CaloGAN Generator

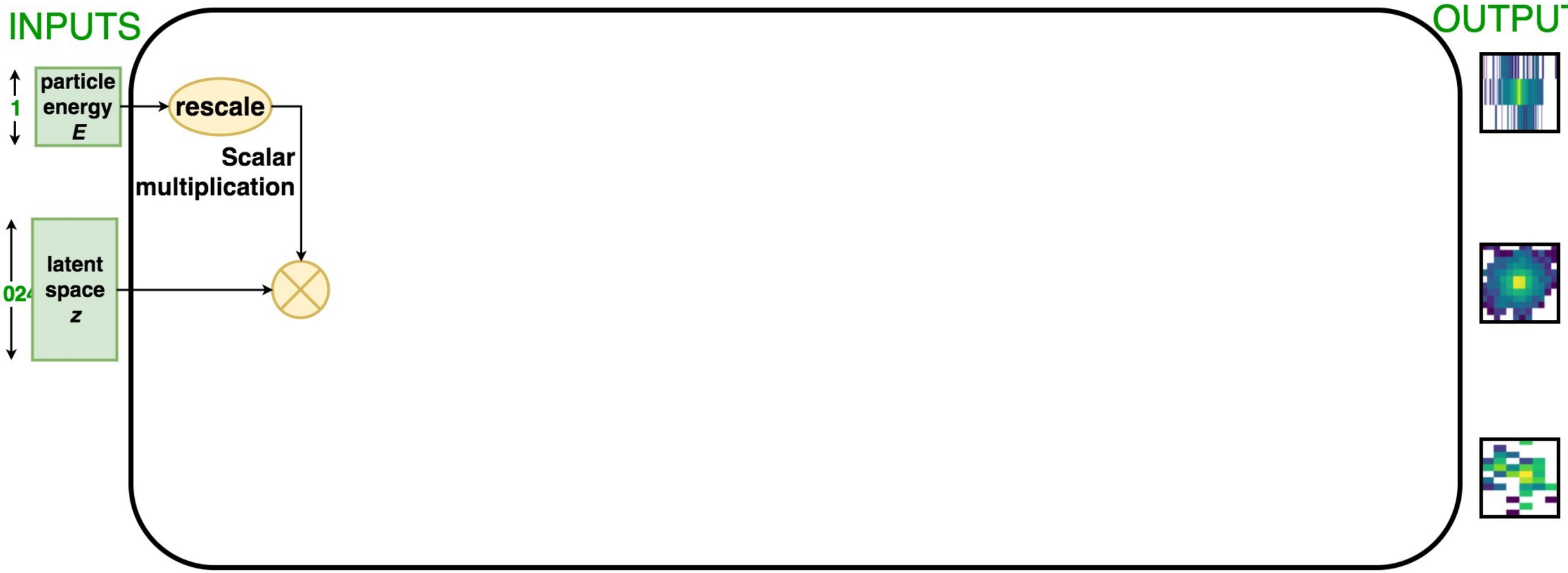
Yale

Tested ACGAN and C-GAN, but obtained best results with one net per particle type



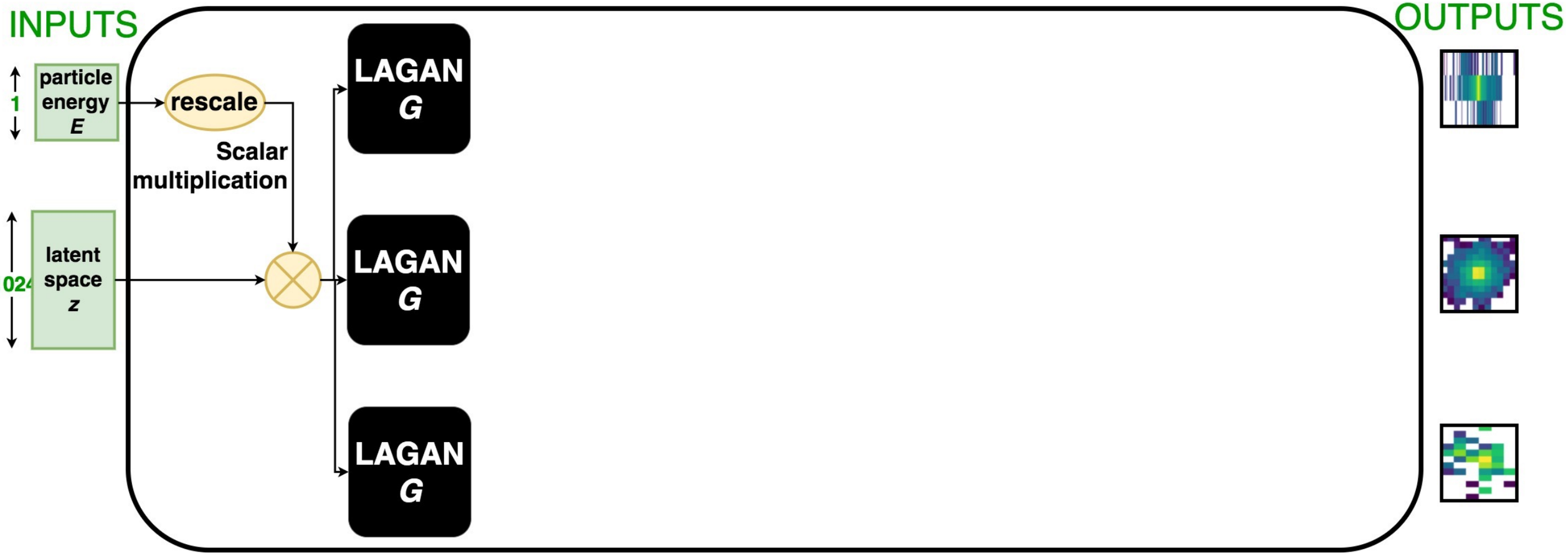
CaloGAN Generator

Yale



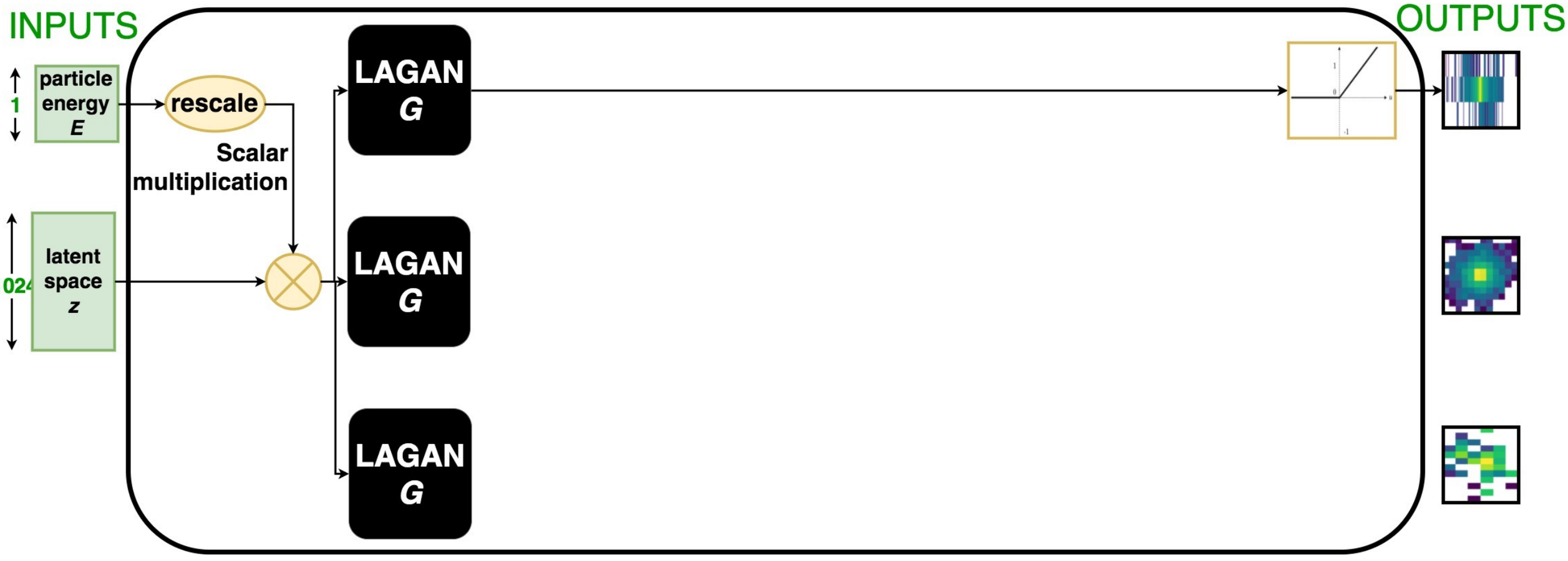
CaloGAN Generator

Yale



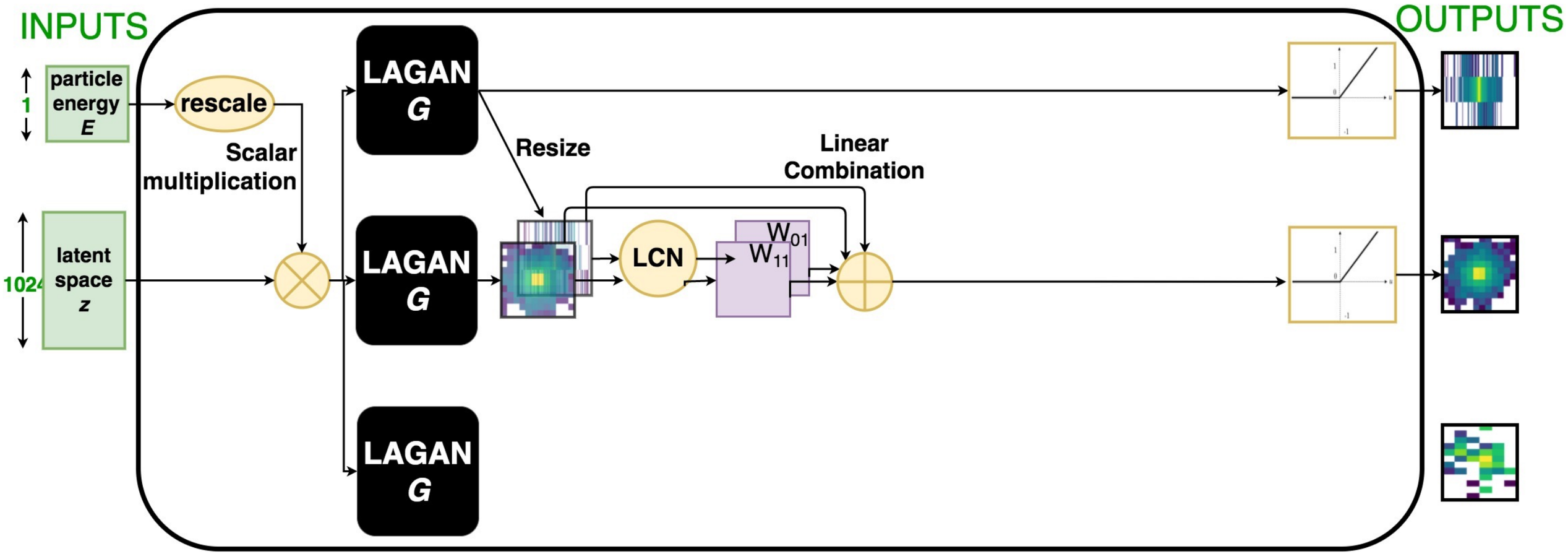
CaloGAN Generator

Yale



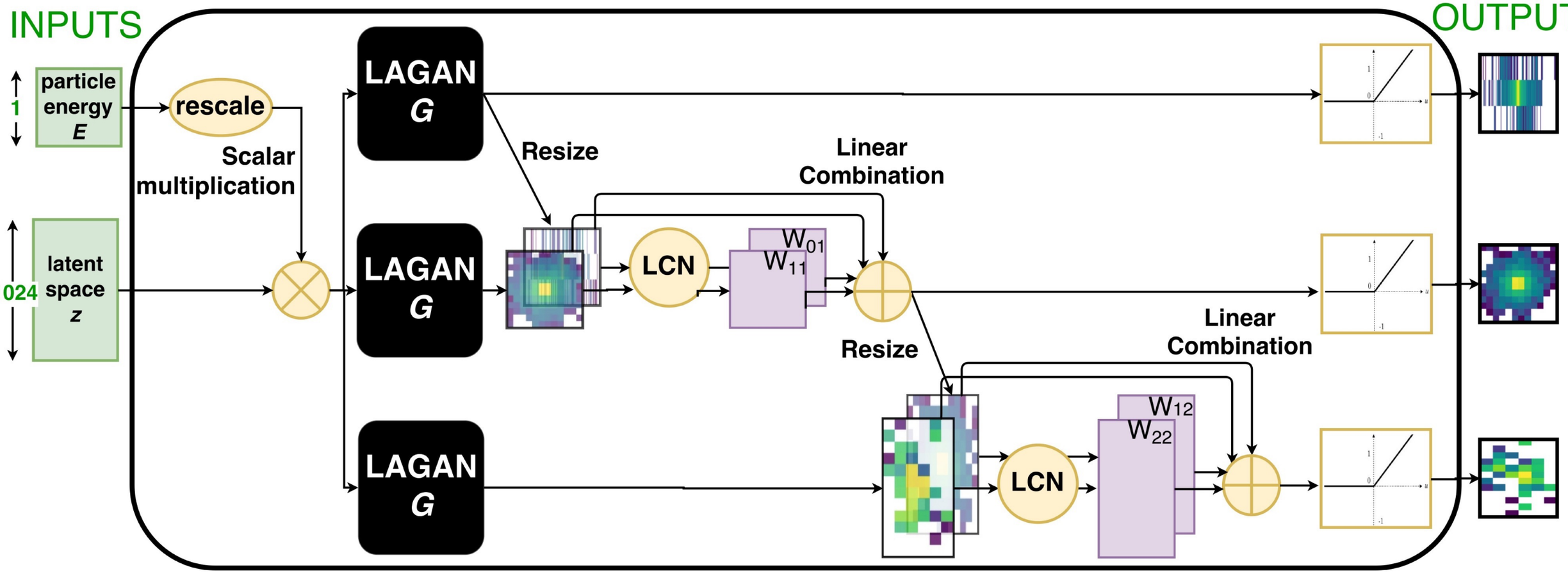
CaloGAN Generator

Yale



CaloGAN Generator

Yale



CaloGAN Discriminator

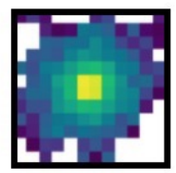
Yale

OUTPUTS

fake vs. real

reco. energy

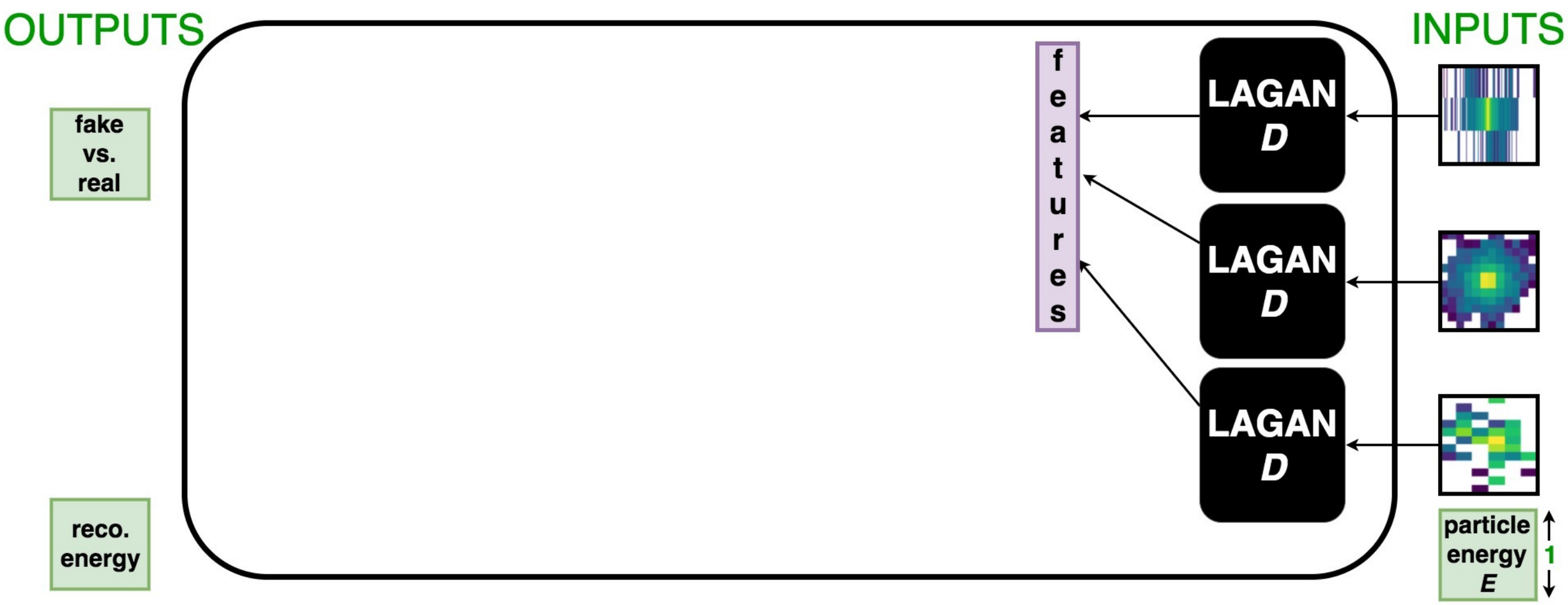
INPUTS



particle energy *E* ↑ 1 ↓

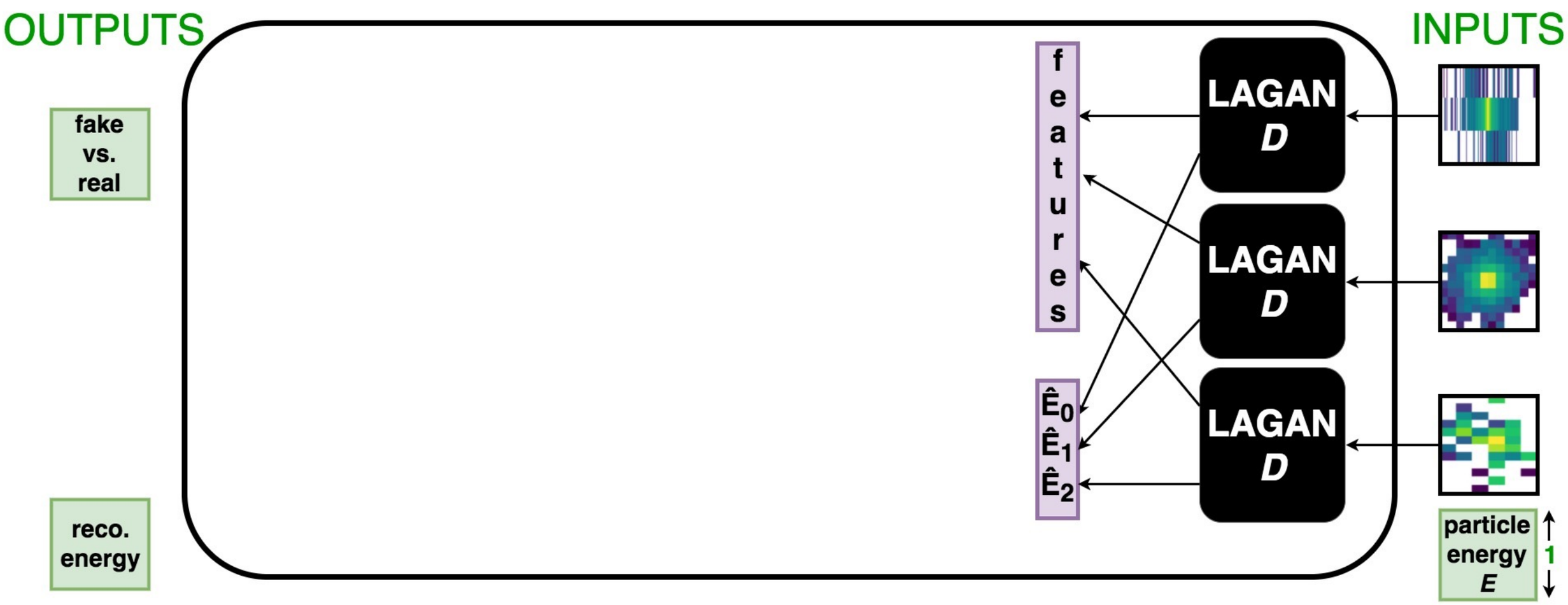
CaloGAN Discriminator

Yale



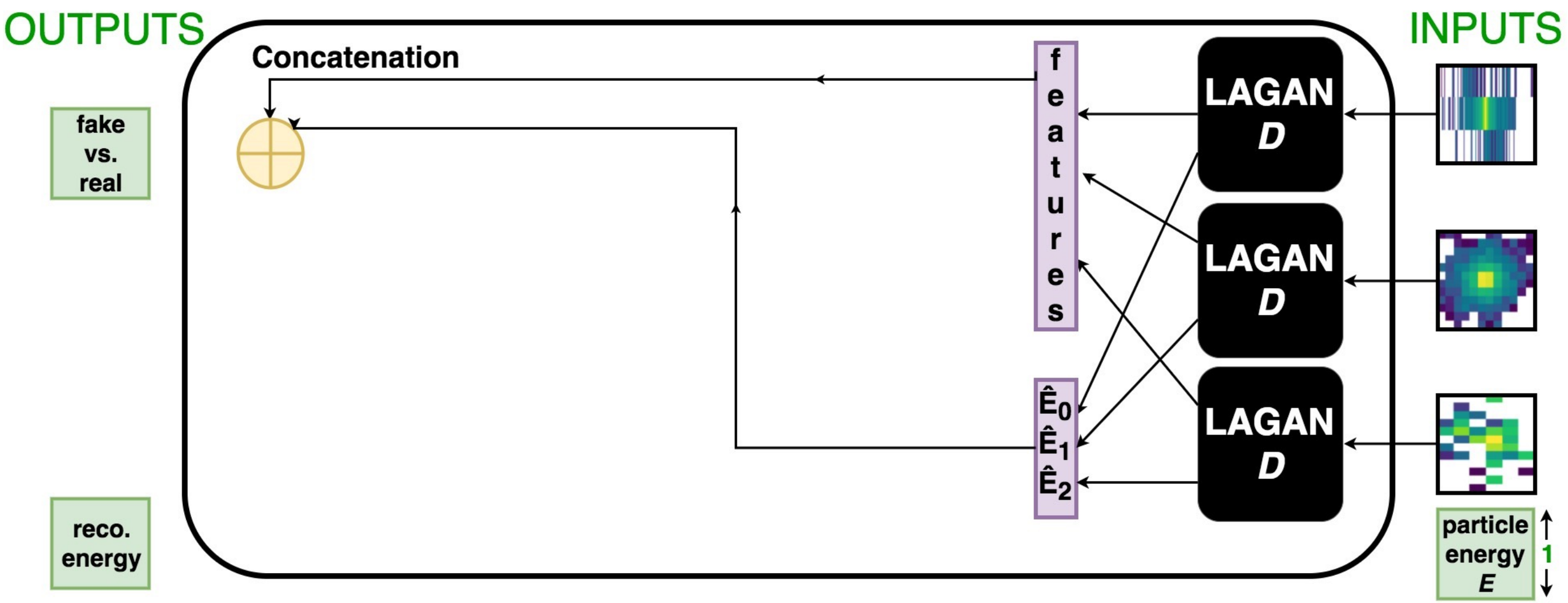
CaloGAN Discriminator

Yale



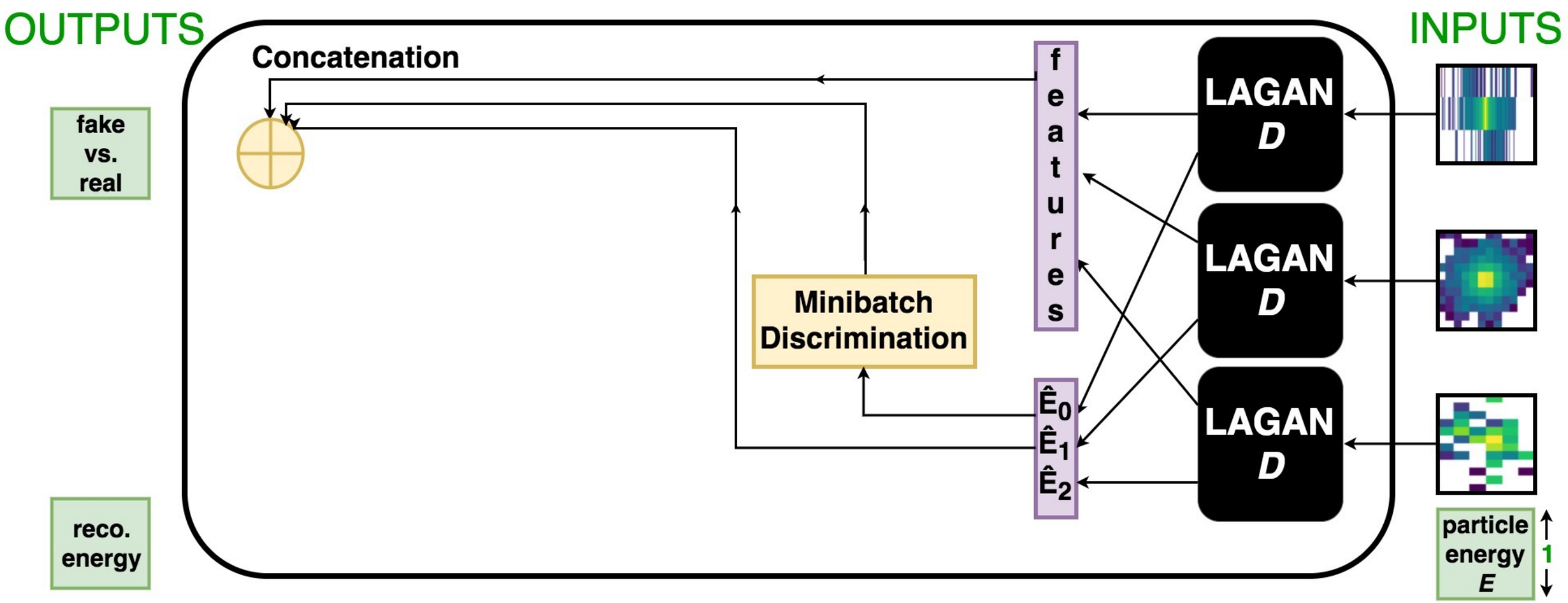
CaloGAN Discriminator

Yale



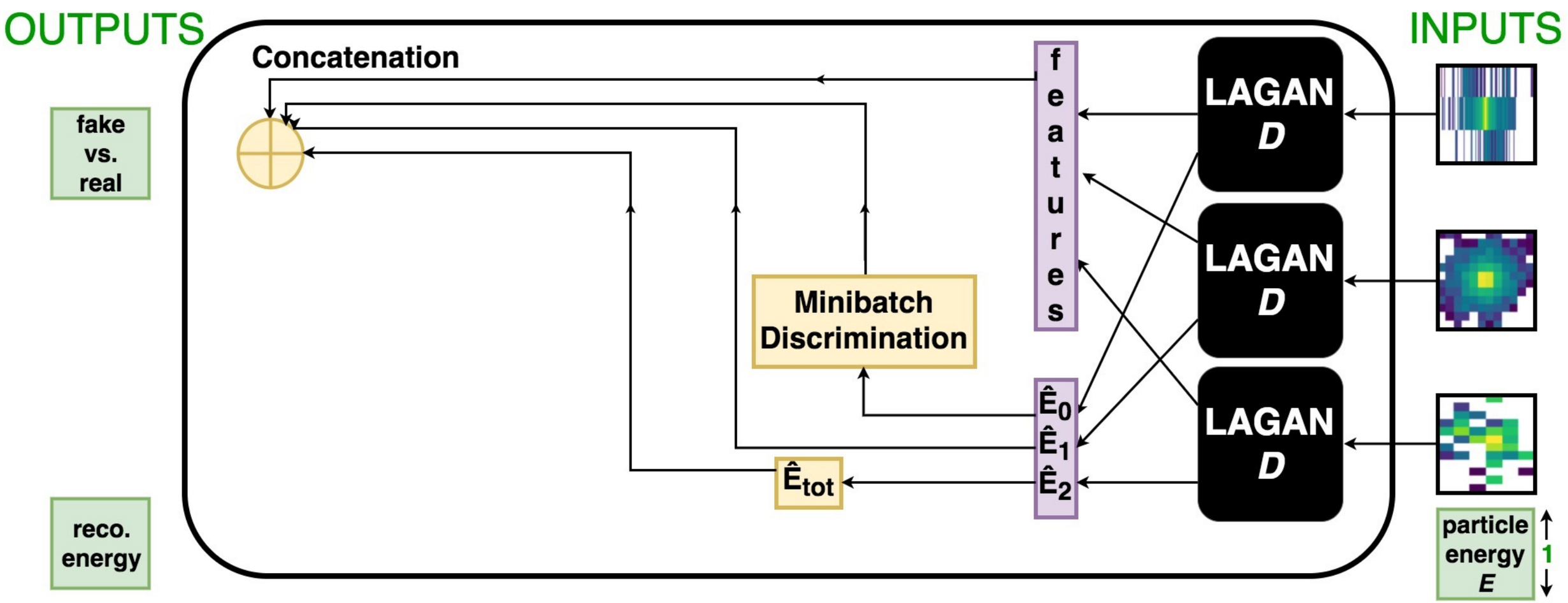
CaloGAN Discriminator

Yale



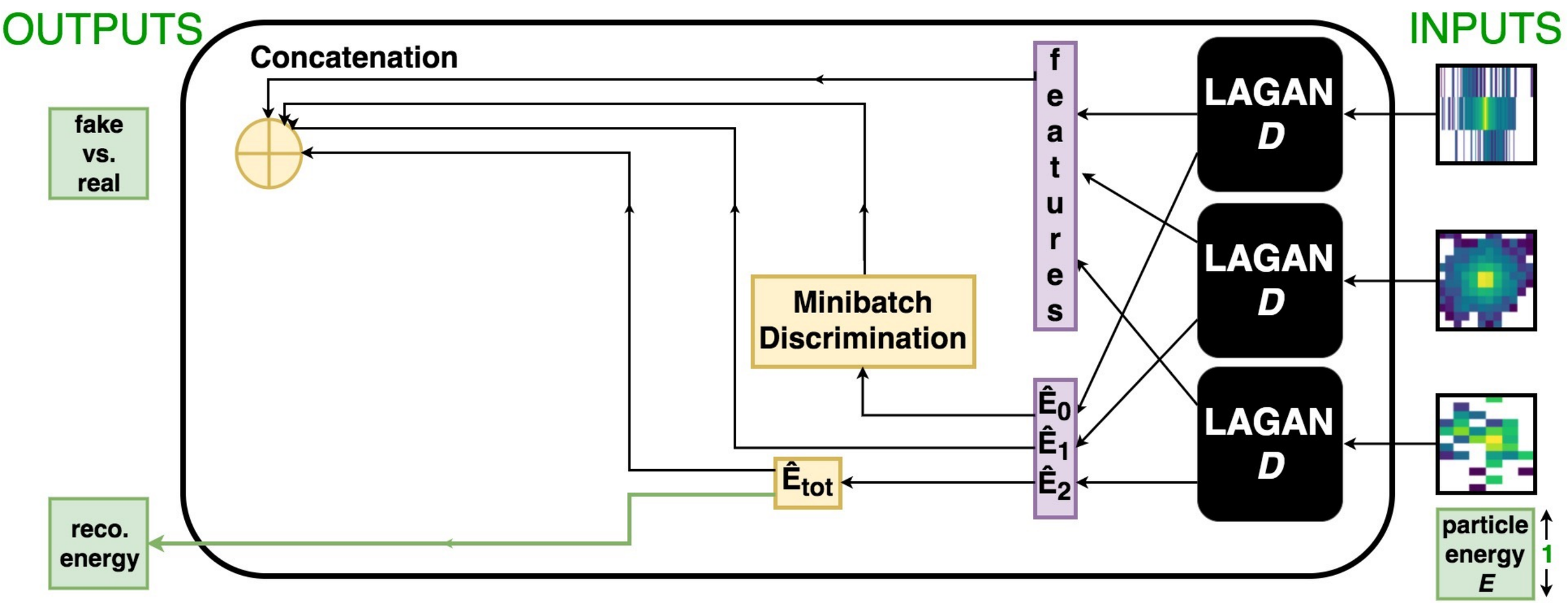
CaloGAN Discriminator

Yale



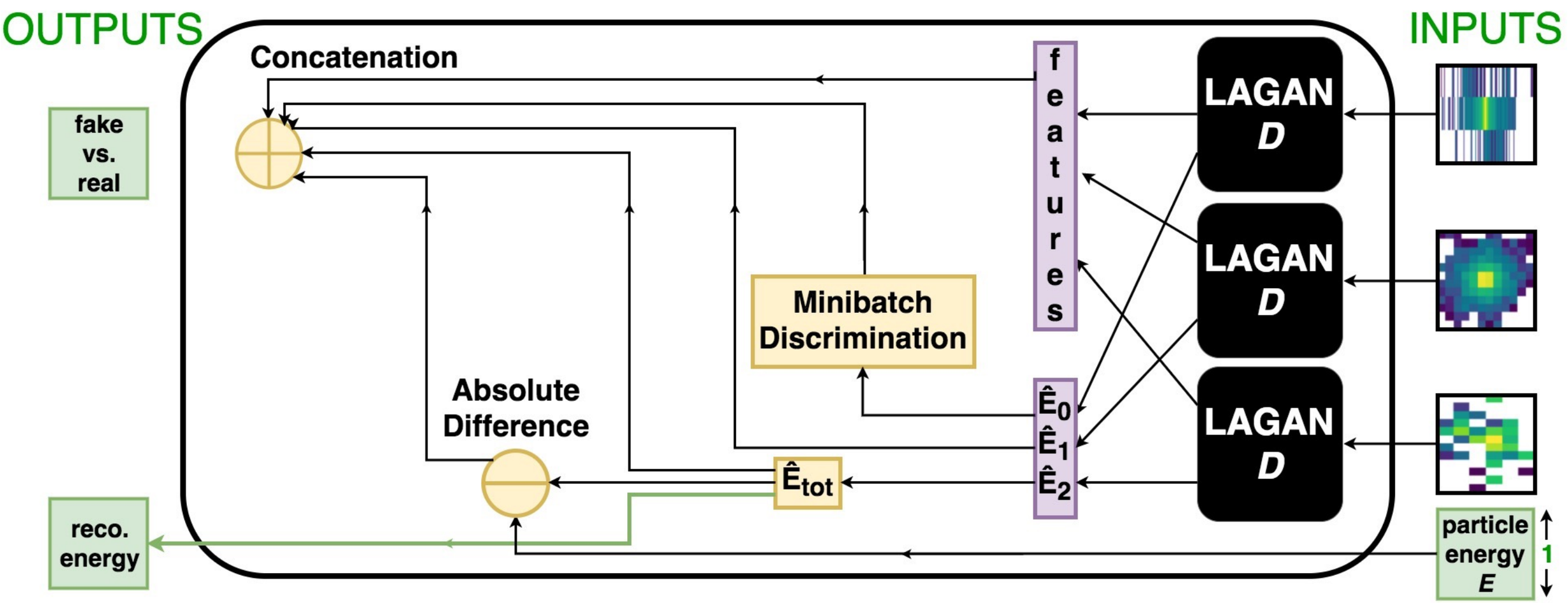
CaloGAN Discriminator

Yale



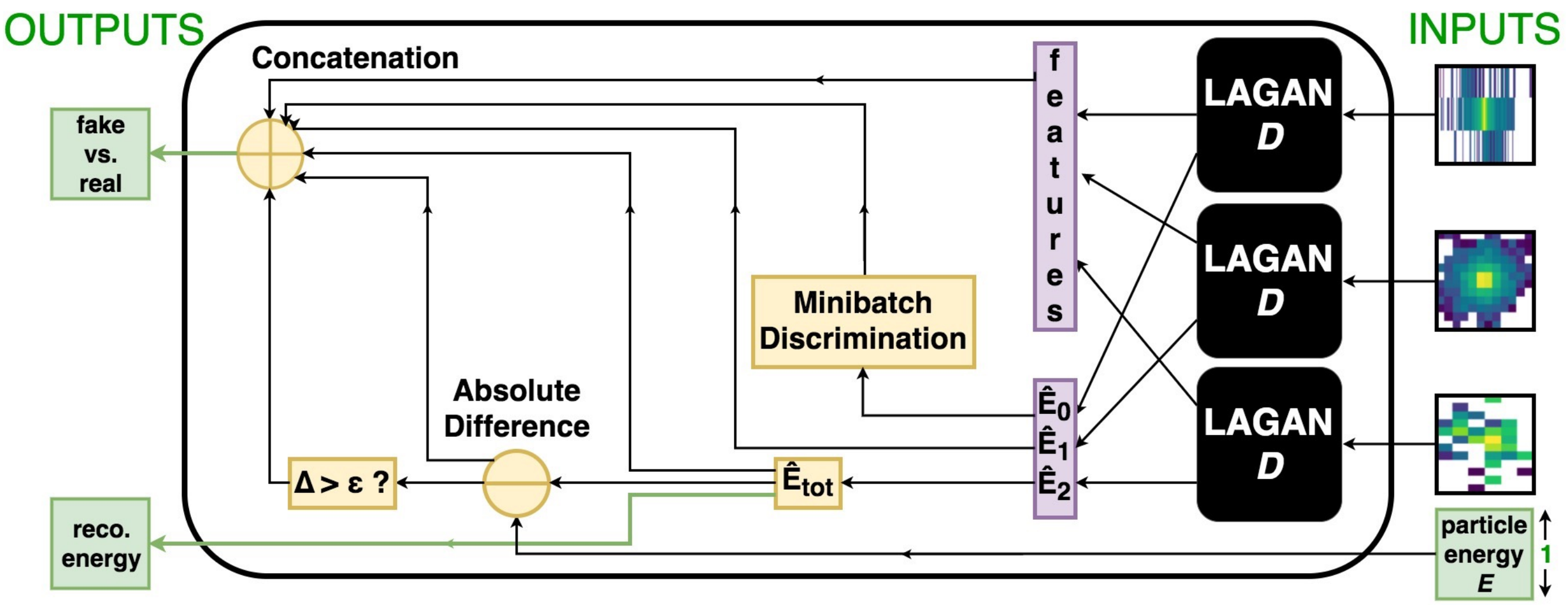
CaloGAN Discriminator

Yale



CaloGAN Discriminator

Yale



Encoding Domain Knowledge

Yale

- Deep Learning is flexible enough to let us write in physics constraints
- These can be used as **features,** or optimized for in a **loss**

```python
def single_layer_energy(x):
    shape = K.get_variable_shape(x)
    # this one is easy - since we define E to be the sum of everything in a
    # layer-image, and since our batches are, for example, of shape (batch_size, 3, 96, 1)
    # we can take a sum (K.sum) over the last three axes. The reshape (-1, 1) is just to
    # turn it into a column vector
    return K.reshape(K.sum(x, axis=range(1, len(shape))), (-1, 1))

def sparsity_level(x):
    # this is the *actual* shape (like, a tuple of integers)
    _shape = K.get_variable_shape(x)
    # this is the *symbolic* shape (like, a tuple of tf.Variables of type int32)
    shape = K.shape(x)

    # this gets us a floating point of the number of pixels in a given layer
    # (since its a prod over everything but the batch dim)
    # For example, for layer 0, it would be 288.0 (3 x 96)
    total = K.cast(K.prod(shape[1:]), K.floatx())
    return K.reshape(
        # get a sum of the number of non-zero pixels in the layer (i.e., over the last three axes)
        K.sum(
            # get a one if the pixel is active, 0 otherwise, cast to float
            K.cast(x > 0.0, K.floatx()),
            # broadcast the sum over the entire layer-image
            axis=range(1, len(_shape))
        ),
        (-1, 1) # another weird reshape to make it a column vector
    ) / total # we divide by the total number of pixels to get a sparsity percentage
```

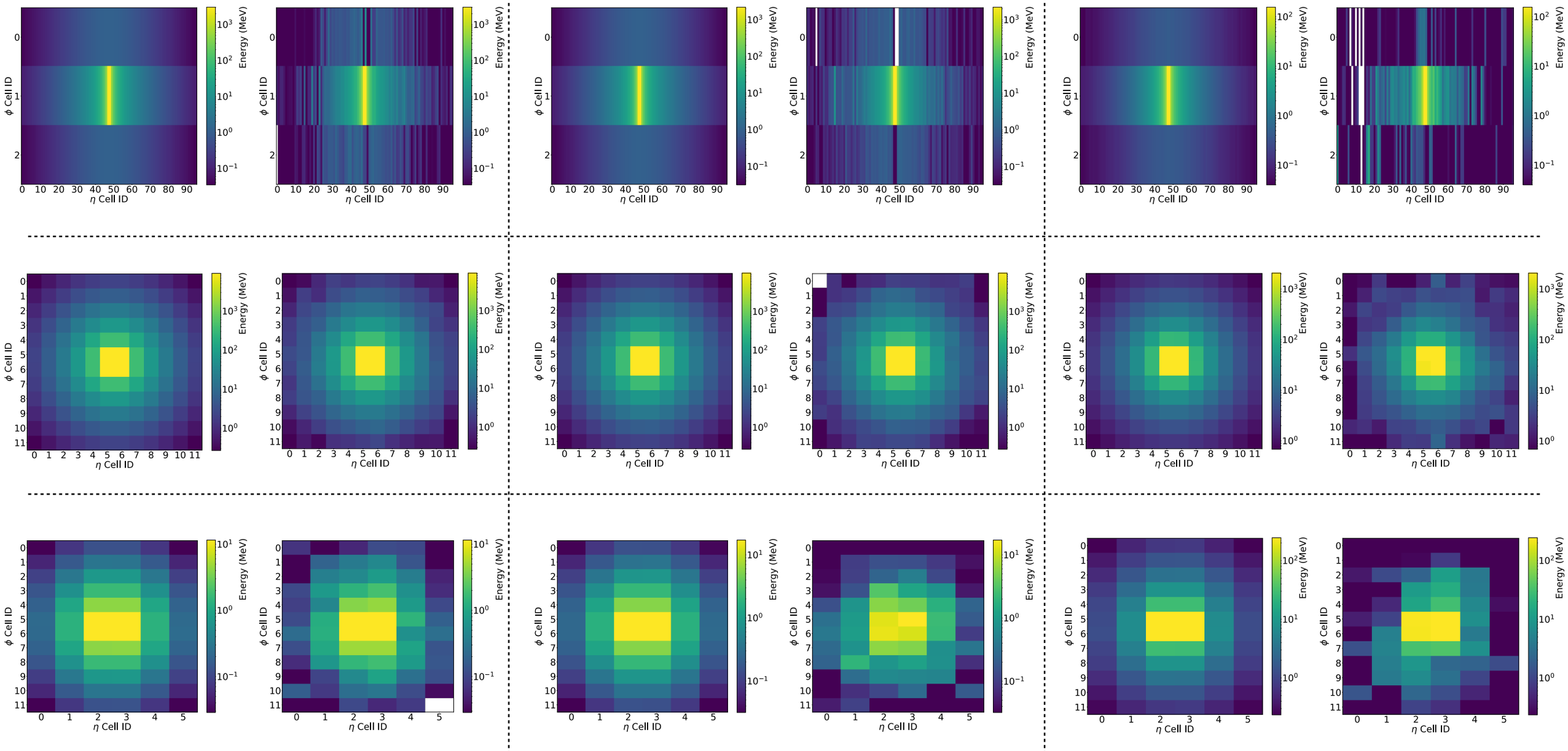
- No need for tension between physics and machine learning. The two can play well together.

Qualitative Performance (1)

Yale

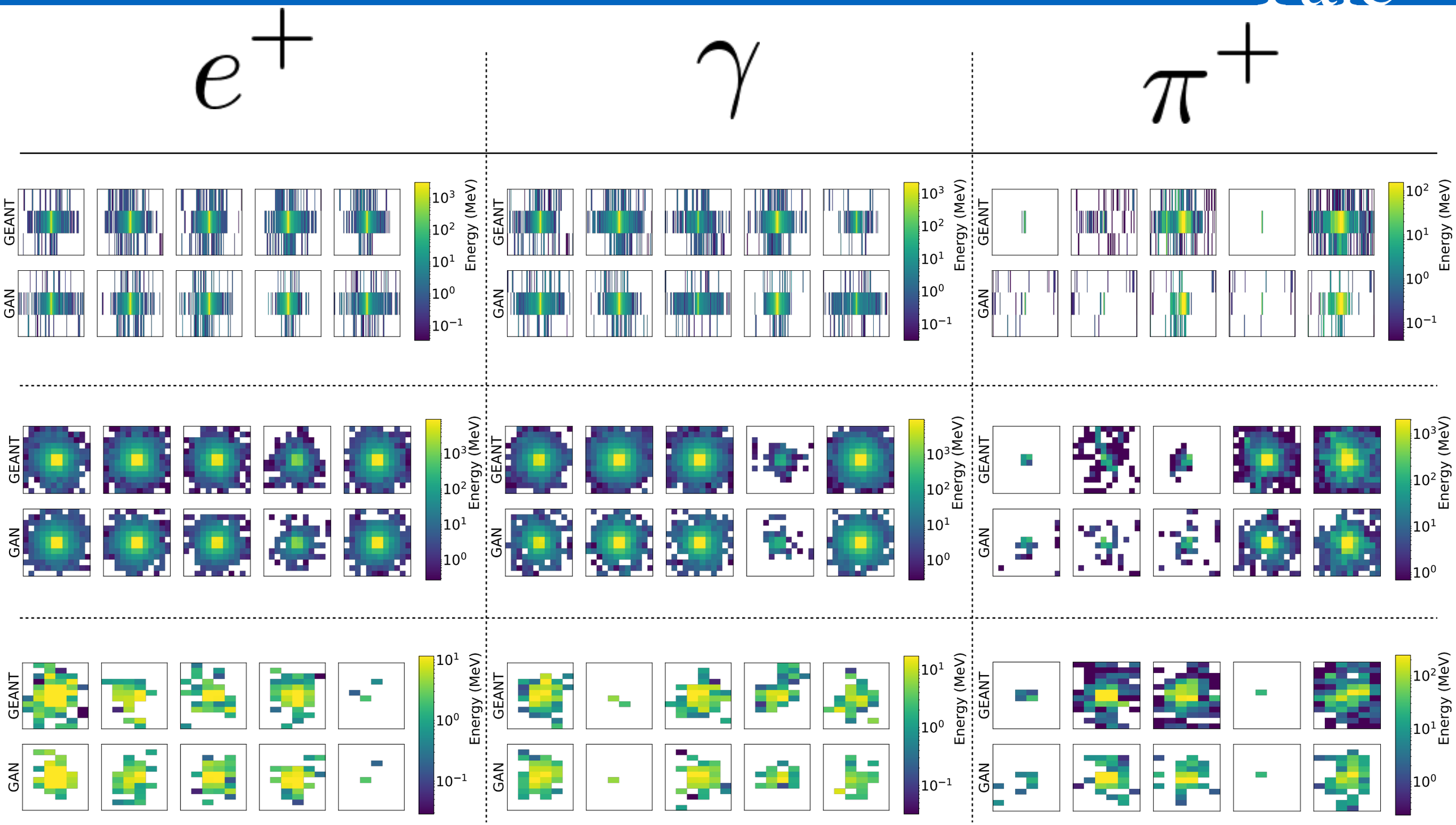
e^+ γ π^+

GEANT GAN GEANT GAN GEANT GAN



Qualitative Performance (2)

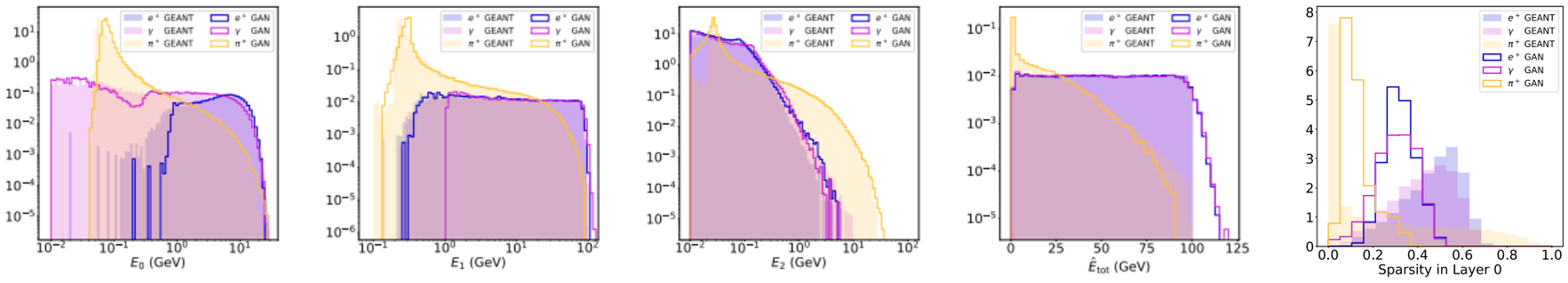
Yale



Shower Shapes

Yale

Check: does the LAGAN recover the true data distribution as projected onto a set of meaningful 1D manifolds?



Conditioning on Energy

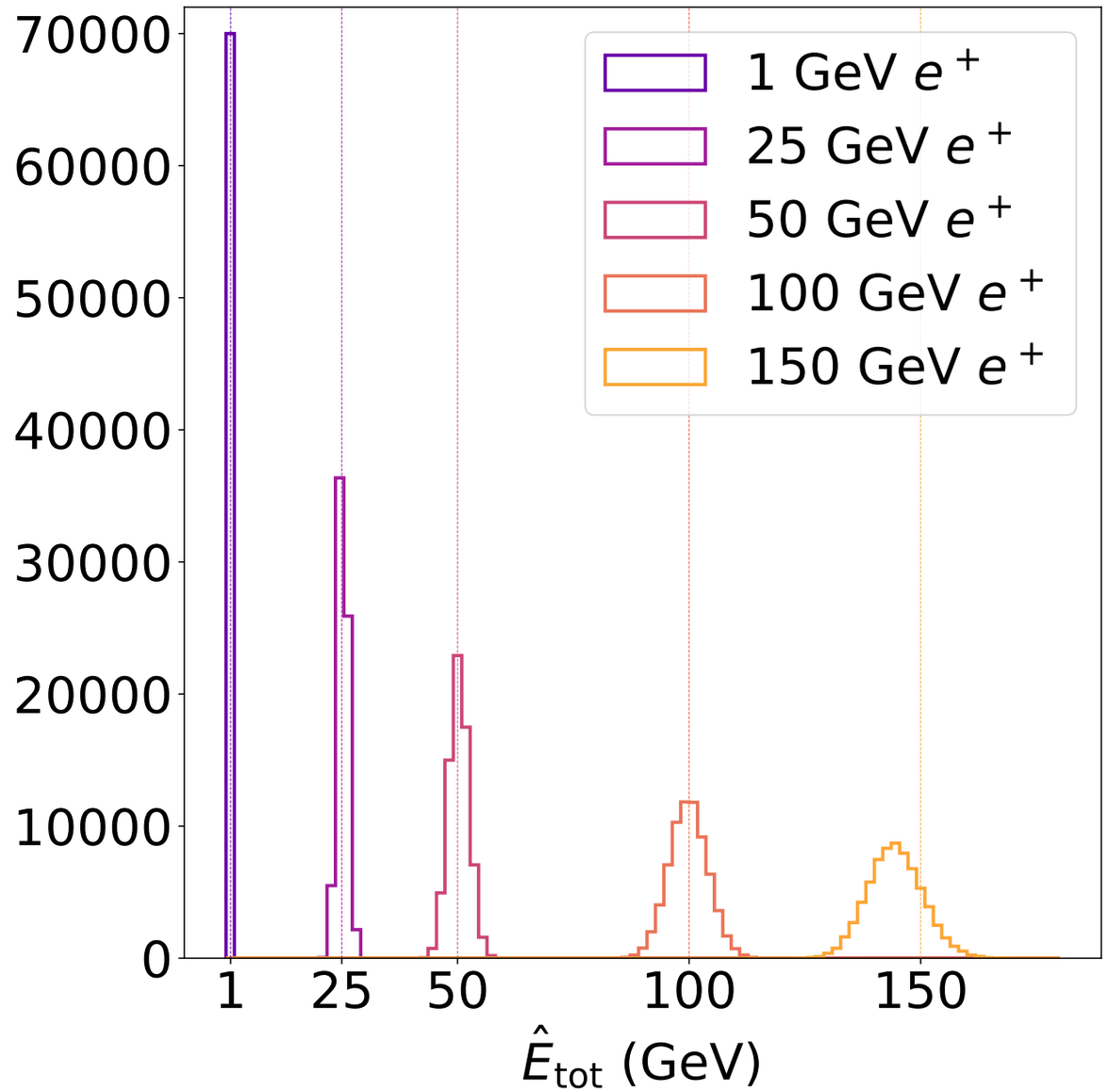
Yale

- Training domain: E ∈ [1, 100] GeV

- At test time, ask for any E

- Great response from the GAN!

- Can even ask for E outside of training domain and get acceptable results —> extrapolation



Computing

Yale

Can obtain an evaluation time speed up of **100,000x** on GPU!

Generation Method	Hardware	Batch Size	milliseconds/shower
GEANT4	CPU	N/A	1772
CALOGAN	CPU	1	13.1
		10	5.11
		128	2.19
		1024	2.03
	GPU	1	14.5
		4	3.68
		128	0.021
		512	0.014
		1024	0.012

Classification

Yale

Super quick classification task using simple fully-connected net on 70k showers for each particle type

```
net = Sequential()
net.add(Dropout(0.5, input_shape=(504, )))

net.add(Dense(512, activation='relu'))
net.add(Dropout(0.5))

net.add(Dense(512, activation='relu'))
net.add(Dropout(0.5))

net.add(Dense(512, activation='relu'))
net.add(Dropout(0.5))

net.add(Dense(512, activation='relu'))
net.add(Dropout(0.5))

net.add(Dense(512, activation='relu'))
net.add(Dropout(0.5))
net.add(Dense(1, activation='sigmoid'))
```

e^+ vs. π^+

trained on	tested on	accuracy
GAN	GAN	~99%
GAN	GEANT	~96%
GEANT	GEANT	~99%

e^+ vs. γ

trained on	tested on	accuracy
GAN	GAN	~99%
GAN	GEANT	~54%
GEANT	GEANT	~54%

Conclusion and Outlook

Yale

- ☑ generated 3D EM showers in a multi-layer sampling LAr calorimeter with uneven spatial segmentation
- ☑ attempted to preserve spatio-temporal relation among layers
- ☑ infused Physics domain knowledge
- ☑ up to 5 orders of magnitude increase in computational efficiency
- ☐ work with GEANT + experiments
- ☐ expanding the complexity of the training dataset:
  - ☐ incoming particles at different locations and angles
  - ☐ hadronic calorimeter
  - ☐ computational performance HPC clusters

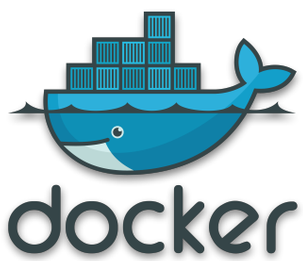
Backup

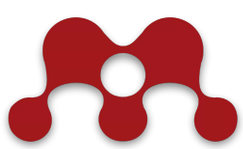
Reproduce the LAGAN work!

Yale

- This paper on the arXiv [1701.05927]
- Github repository: centralized location with all links
- Download the **training dataset** of Pythia jet images, or generate them yourself using this Docker image
- Train a DCGAN, fully-connected GAN, hybrid GAN, or LAGAN using the code we provide in `models`, or load our pre-trained weights
- Use the jupyter notebook in `analysis` to make plots like ours



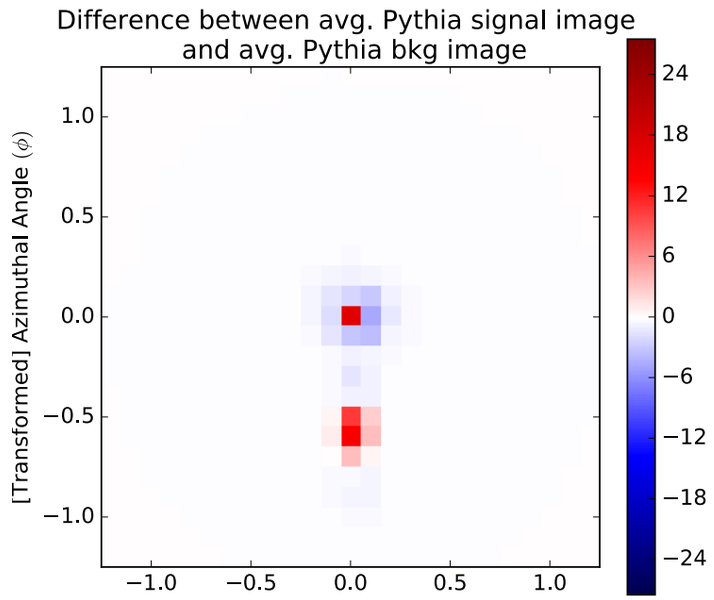


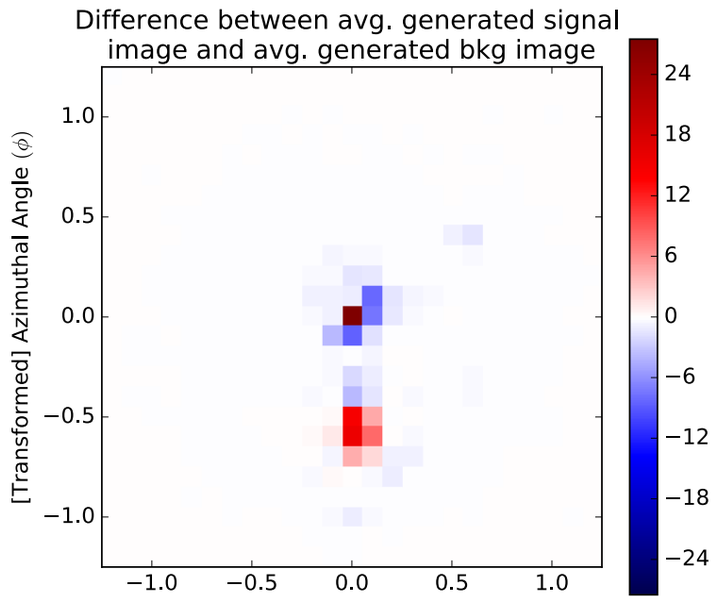


LAGAN Performance

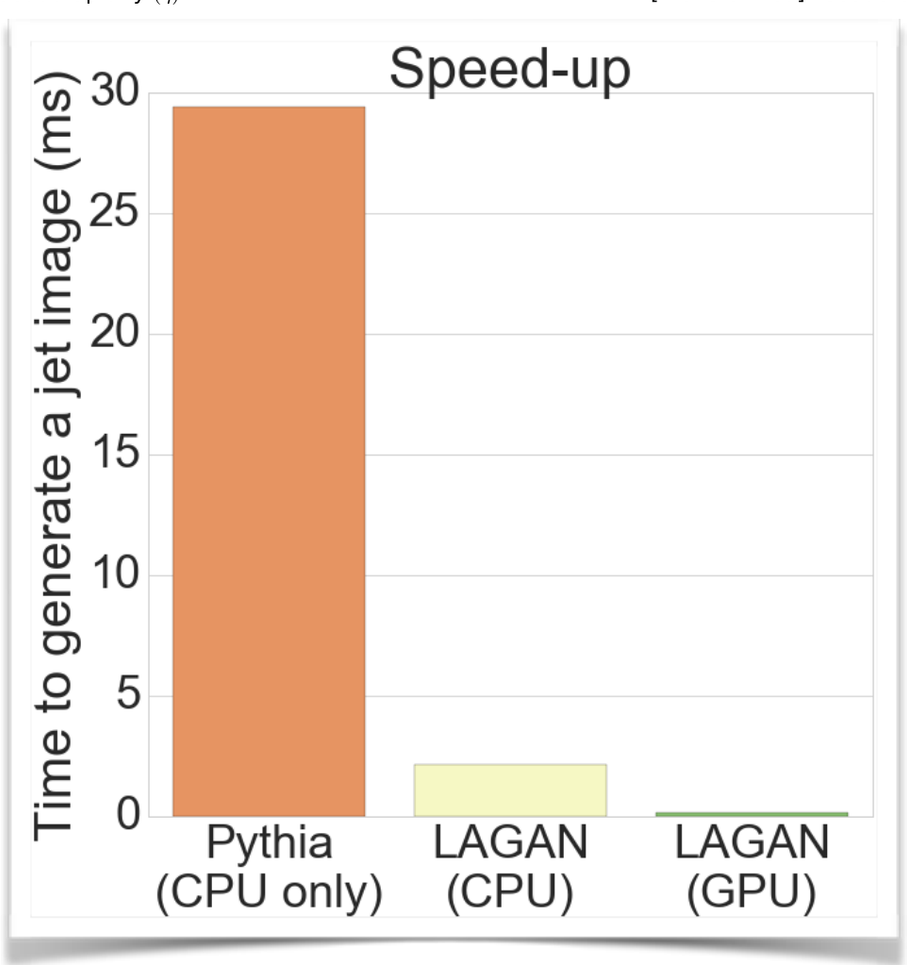
Yale

in Pythia



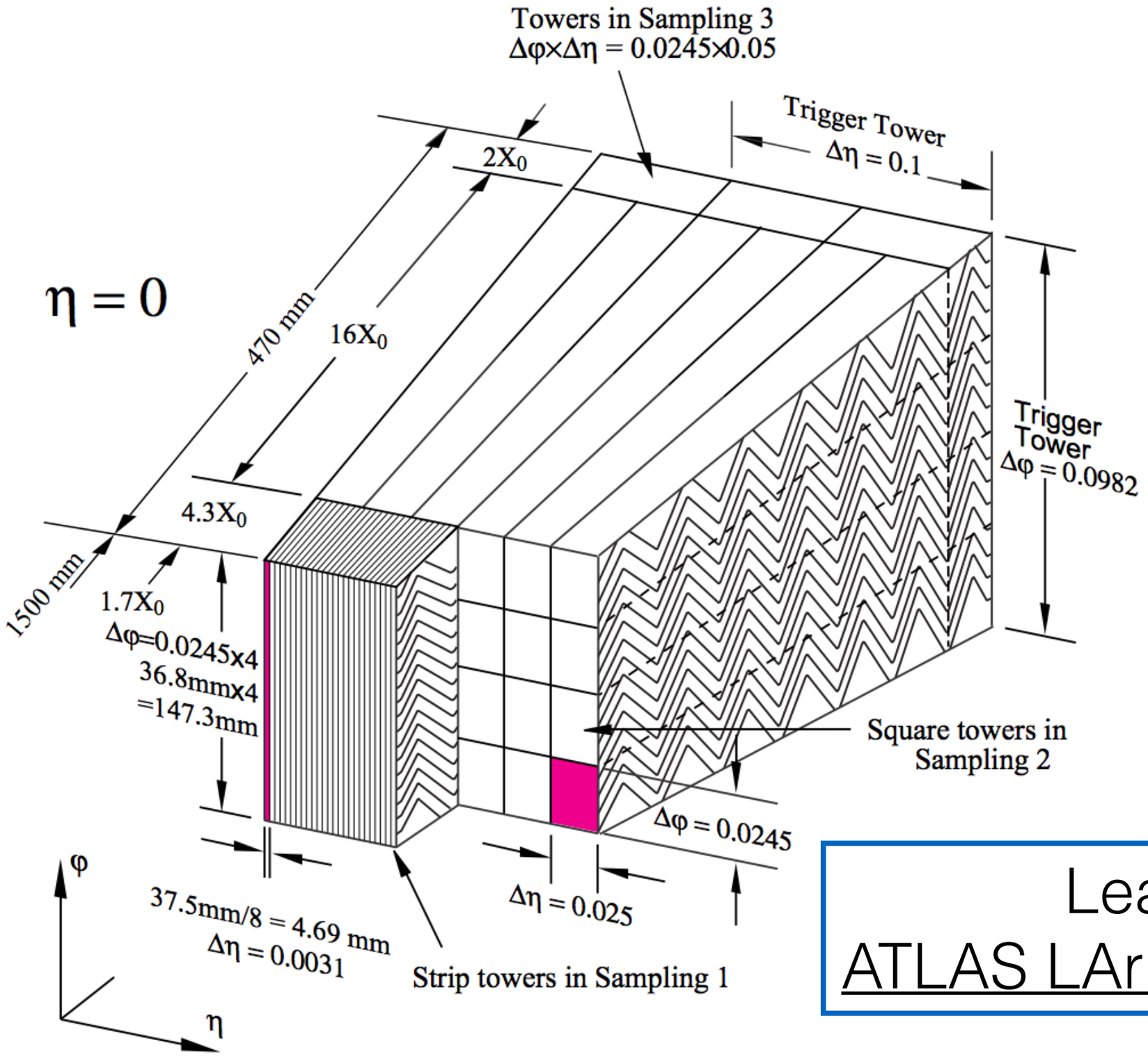


in LAGAN



ATLAS EM Calorimeter

Yale



Learn more:
ATLAS LAr Calorimeter TDR

1D Manifolds

Yale

Shower Shape Variable	Formula	*Notes*
E_i	$E_i = \sum_{\text{pixels}} \mathcal{I}_i$	Energy deposited in the i^{th} layer of calorimeter
E_{tot}	$E_{\text{tot}} = \sum_{i=0}^{2} E_i$	Total energy deposited in the electromagnetic calorimeter
f_i	$f_i = E_i / E_{\text{tot}}$	Fraction of measured energy deposited in the i^{th} layer of calorimeter
$E_{\text{ratio},i}$	$\frac{\mathcal{I}_{i,(1)} - \mathcal{I}_{i,(2)}}{\mathcal{I}_{i,(1)} + \mathcal{I}_{i,(2)}}$	Difference in energy between the highest and second highest energy deposit in the cells of the i^{th} layer, divided by the sum
d	$d = \max\{i : \max(\mathcal{I}_i) > 0\}$	Deepest calorimeter layer that registers non-zero energy
Depth-weighted total energy, l_d	$l_d = \sum_{i=0}^{2} i \cdot \mathcal{I}_i$	The sum of the energy per layer, weighted by layer number.
Shower Depth, s_d	$s_d = l_d / E_{\text{tot}}$	The energy-weighted depth in units of layer number.
Shower Depth Width, σ_{s_d}	$\sigma_{s_d} = \sqrt{\frac{\sum_{i=0}^{2} i^2 \cdot \mathcal{I}_i}{E_{\text{tot}}} - \left(\frac{\sum_{i=0}^{2} i \cdot \mathcal{I}_i}{E_{\text{tot}}}\right)^2}$	The standard deviation of s_d in units of layer number.
i^{th} Layer Lateral Width, σ_i	$\sigma_i = \sqrt{\frac{\mathcal{I}_i \odot H^2}{E_i} - \left(\frac{\mathcal{I}_i \odot H}{E_i}\right)^2}$	The standard deviation of the transverse energy profile per layer, in units of cell numbers.

Implementation Note

Yale

"Separating per-particle-type CaloGAN architectures and implementations affords many benefits. It is easy to imagine a situation where the life cycles surrounding models for different particle types are very different. In addition, this allows for total independence of versioning, framework, or language – i.e., an ecosystem of RPC clients around various generators for specific tasks.

When possible, any GAN should maximally employ batching – we imagine most applications can request all showers from one event simultaneously, maximally taking advantage of CPU/GPU while minimizing data transfer overhead."