Deep-compression for HEP data

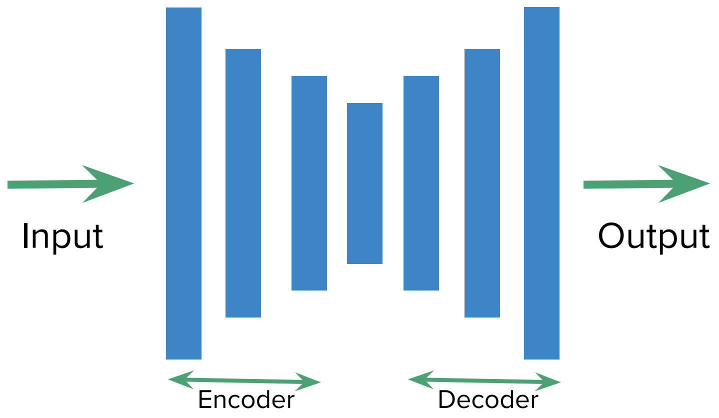
Honey Gupta (hn.gpt1@gmail.com)
Google Summer of Code 2020 & CERN - HSF

Motivation

- There are approximately 1.7 billion events occurring inside the ATLAS detector, each second.
- Storage of these events is limited by the event size and a reduction of the event size will allow for searches that were not previously possible.

Deep-compression

- Deep compression refers to usage of autoencoders for performing data compression.
- Learn the data distribution by projecting it to a lower-dimension and then reprojecting.
- The idea is to use deep compression for High Energy Physics (HEP) data and check their efficacy.



A typical autoencoder (encoder+decoder) network

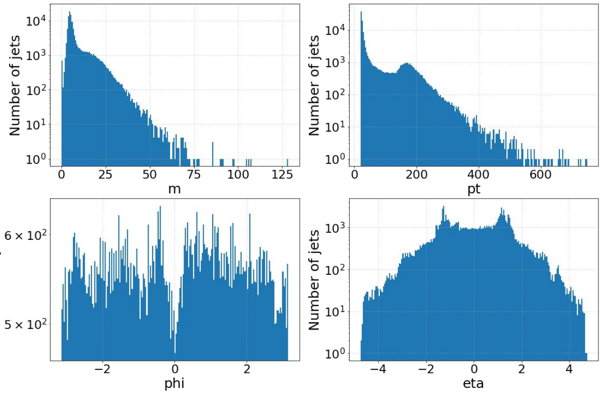
Experiments with 4D data

Normalization

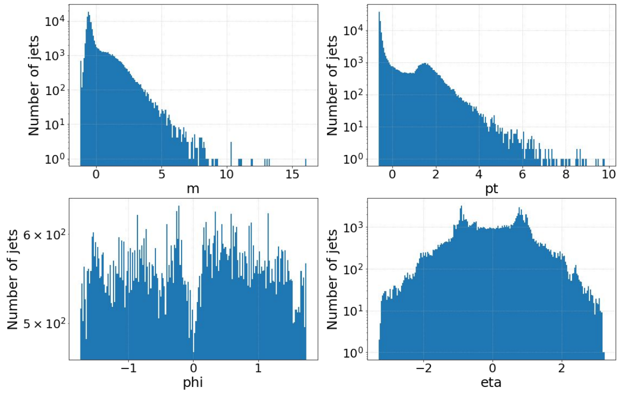
An important part of training any machine learning algorithm is data preparation and analysis. Hence, we analyse the data distribution of the provided 4 dimensional data containing the parameters: **m, p_t, phi** and **eta** when normalized using different normalization methods.

$$x = \frac{x - \mu_x}{\sigma_x};$$
$$x \in [m, p_t, \eta, \phi]$$

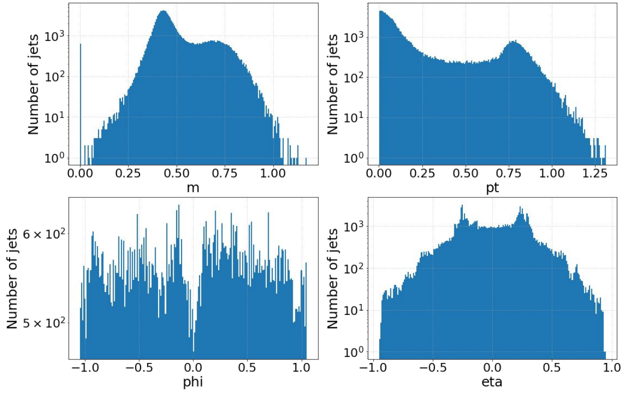
$$m \longrightarrow ([log_{10}(m/1000)] + 1)/1.8$$
$$p_t \longrightarrow ([log_{10}(p_t/1000)] - 1.3)/1.2$$
$$\eta \longrightarrow \eta/5$$
$$\phi \longrightarrow \phi/3$$



No normalization



Standard normalization



Custom normalization

Data distribution for 4D data

Experiment details

To analyze effects of normalization on compression

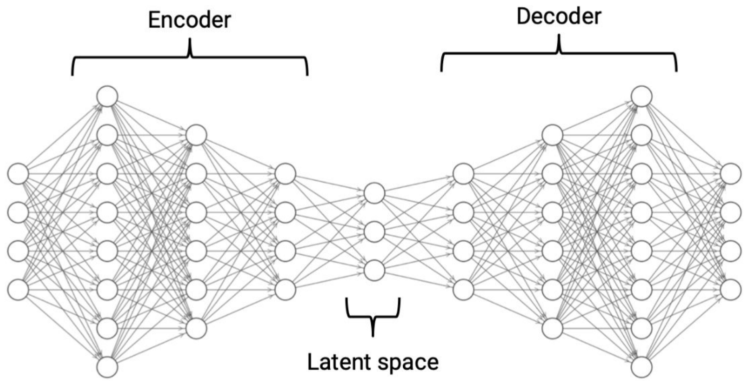
- trained and tested a simple model - latent space of 3
- input - 4D data

Ideally, the decompressed 4D data should be same as the input data and hence we train the autoencoder by using a reconstruction loss as the optimization loss function:

$$\mathrm{MSE} = \frac{1}{N} \sum_{n=1}^{N} \sum_{i=1}^{P} (y_i(\vec{x}_n) - x_{ni})^2 \ ,$$

Model description:

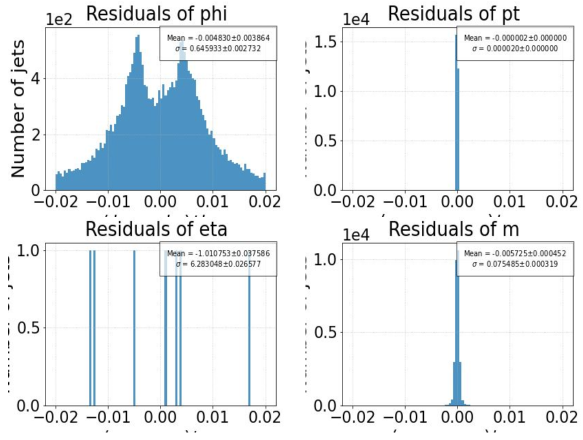
The model contains 7 fully-connected layers with 200, 100, 50, 3, 50, 100, 200 nodes and Tanh activation layer after each layer.



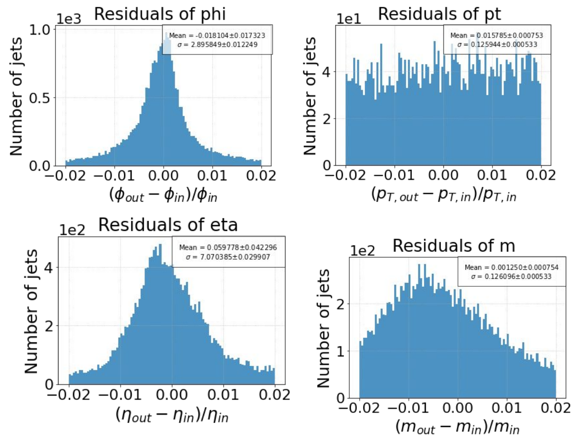
Normalization experiments

Plots of the relative reconstruction error for each compressed variable for the entire test-set:

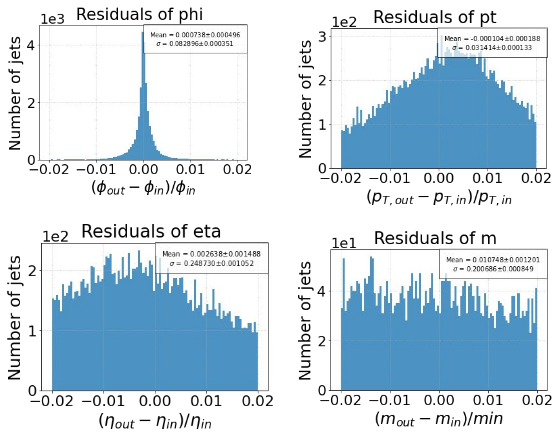
No normalization



Standard normalization



Custom normalization



Observations:

- High bias in the non-normalised model. pt has low error whereas eta very high reconstruction error.
- Standard normalization has highest error for most of the parameters among the three models
- Custom normalization has better performance for most of the parameters as compared to the others

The custom norm data produced the lowest mean squared error on the test-set and is able to capture correlations among variables in a better way, I chose custom normalization for all further experiments.

Normalization type	MSE on the test-set
None	0.5181
Standard	0.01111
Custom/	**0.0007314**

Different variants of compression network

Next, I tried 2 variants of the autoencoder models that had the same 7 layer autoencoder with similar node configuration as the previous model. The dissimilarity is in the activation layer and batchnorm layer.

- The first model has tanh activation and no batchnorm layer (the base model)
- The second has Leaky Rectified Linear Unit (ReLU) [1] activation and batchnorm [2] after each layer
- The third model has Exponential Linear Unit (ELU) [1] and batchnorm after each layer

Note that the MSE for the test-set is calculated on the custom normalized data, whereas the individual mean relative reconstruction errors are calculated on unnormalized data.

Mean of relative reconstruction error

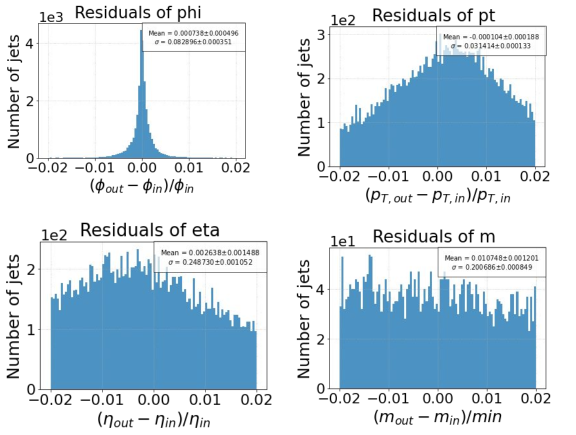
Model	m	pt	phi	eta	MSE on the test-set
Tanh, no BN	0.010748	-0.0001039	0.0007383	0.002638	0.0007314
LeakyReLU, BN	0.004853	-0.001126	0.008089	-0.02475	0.0005750
ELU, BN	0.005528	-0.0007568	-0.003144	-0.0002156	0.0005754

[1] https://en.wikipedia.org/wiki/Rectifier_(neural_networks) [2] https://en.wikipedia.org/wiki/Batch_normalization

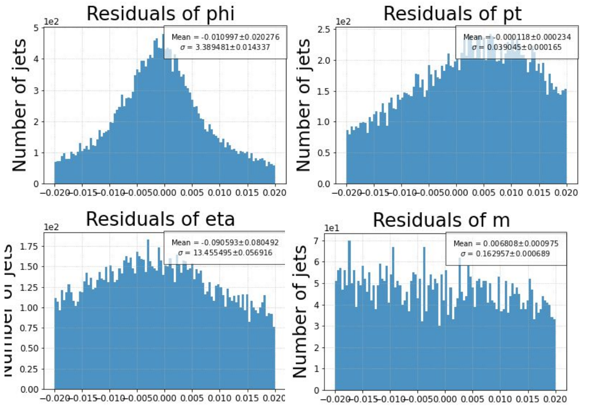
Variants of compression network

Plots of the relative reconstruction error for each compressed variable for the entire test-set:

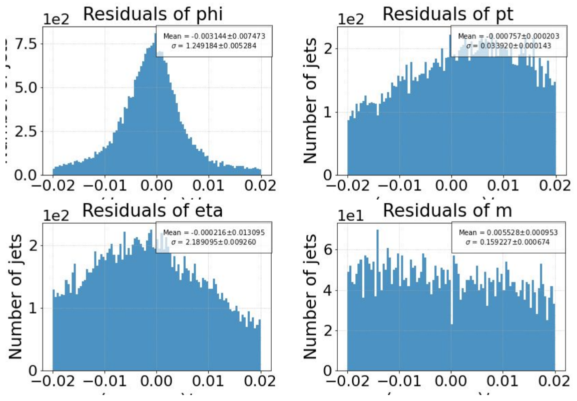
Tanh, no BN



LeakyReLU, BN



ELU, BN



Observations:

- LeakyReLU model has moderate performance (based on the variance of relative error and MSE on the test-set).
- Tanh and ELU have comparable performance. Tanh has lower variance and mean for the relative error but ELU has lower MSE.
- Hence ELU model can be said to be the better among by considering both the relative error and MSE, since there is not much difference between ReLU and ELU's MSE for the test-set.

Execution time and memory allocation

Test-set size: 27945
Hardware for computation: Intel(R) Xeon(R) CPU E5-1620 v4 @ 3.50GHz, 8 cores

Data loading execution time: 0.10674s
Data loading memory: 3.668 MBs

Model	Model initialization time (s)	Model load time (s)	Encoding time (s)	Decoding time (s)	Encoding memory alloc (MB)	Decoding memory alloc (MB)
Tanh, no NN	2.5174	**0.05932**	**0.03934**	**0.02113**	**0.0024**	**0.1780**
LeakyReLU, BN	2.5903	0.08180	0.12613	0.10033	0.0076	0.2240
ELU, BN	**2.4373**	0.07940	0.16376	0.15237	0.0064	0.1945

Observations:

- ELU has exponential component, hence it's runtime is higher
- Tanh also has exponential component, but the batchnorm layer in other two models increases execution time.
- **Overall, we can say that a model without the exponential component i.e. with a ReLU activation and without Batch-normalisation should perform the best in terms of execution time**

Experiment with the loss function

- No re-training done for this network, original training from 4D network (MSE)
- Trained another model with the same configuration but with L1 loss

Model details:

- LeakyReLU, Batch Normalization
- Custom-norm, 4D data

Variance of relative reconstruction error

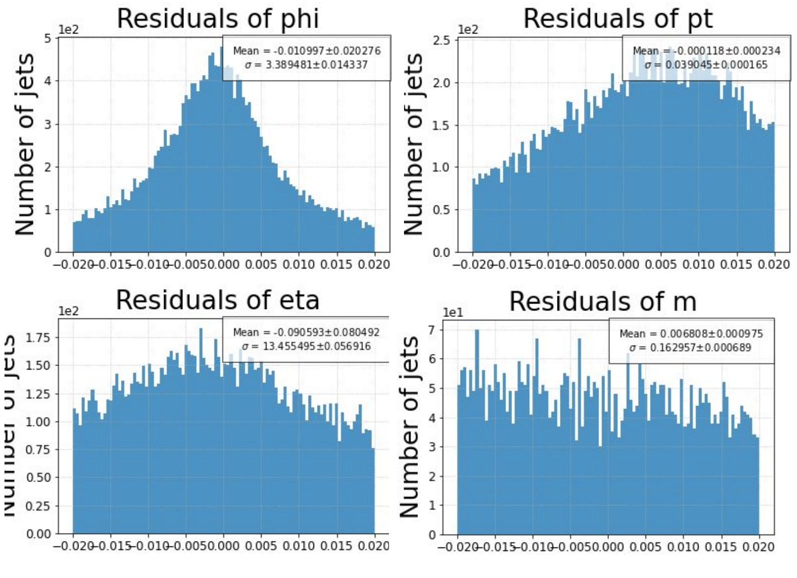
Model	**m**	**pt**	**phi**	**eta**	**MSE on the test-set**
MSE	0.162	0.0369	1.642	7.206	0.0005750
L1	0.302	0.0203	1.385	2.364	0.01211852

Observations:

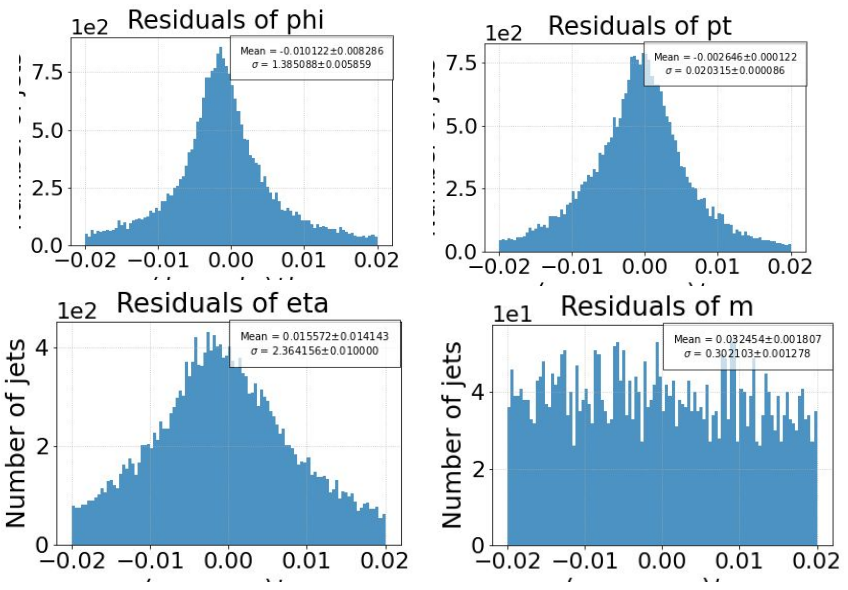
- Since MSE is used as a metric for analysing the test-set error, MSE as a loss function has much less error
- However, the compression performance could be better judged by using the response plots which show the distribution of the relative residual error

MSE vs L1 - residual plots

LeakyReLU, BN, MSE



LeakyReLU, BN, L1



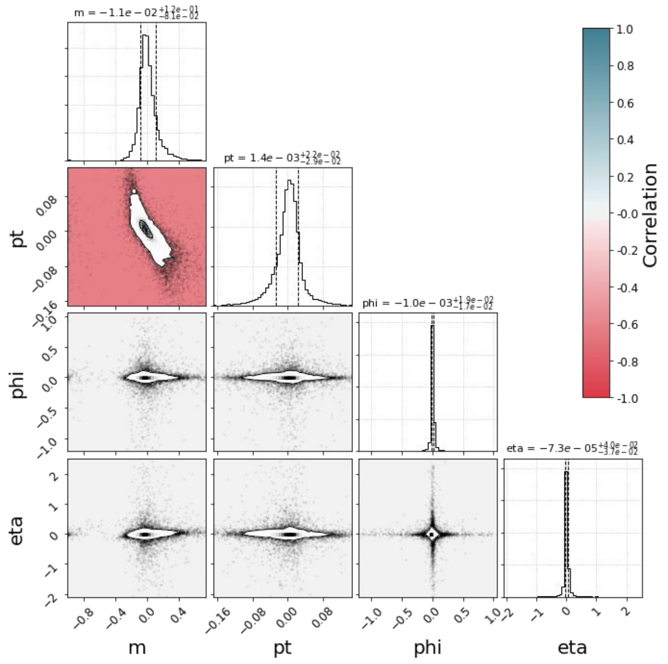
Observations:

- No significant difference between the results with two different loss functions

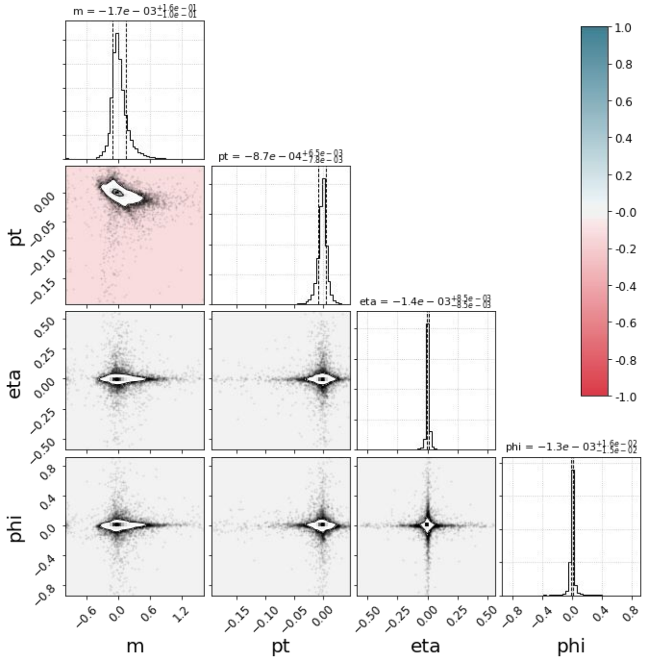
Note that the y-axis of the plots is different

MSE vs L1 - Correlation plots

LeakyReLU, BN, MSE



LeakyReLU, BN, L1



Observation:

- Higher correlation between pt and m for MSE mean that the relative error of reconstruction for these variables are correlated

Note that the axes of the plots is different

Experiments with 27D data

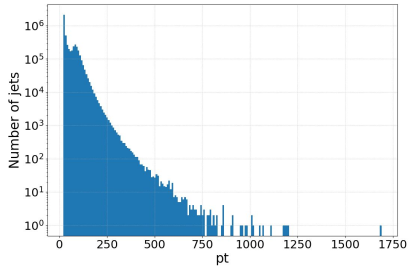
Tasks

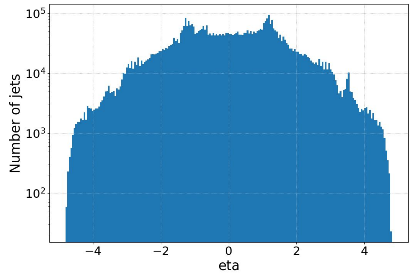
In order to transition to 27D data, we formulated the following tasks:

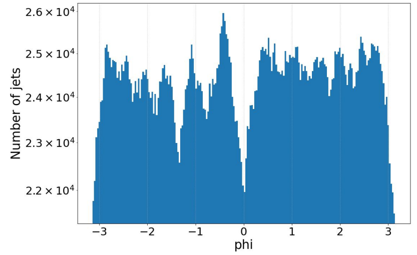
- Analyse the available data
    - plot the distribution of each variable
    - compare the plots with the plots mentioned in prior experiments (Eric Wullf’s thesis, a Masters student who worked on the project earlier)
- Test the available pre-trained model
    - create plots from the pre-trained model
    - compare and validate the published results
- Train the model on the available data
    - create response and correlation plots
    - analyse the performance

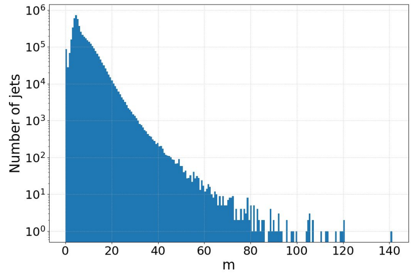
1. Data distribution - 27D

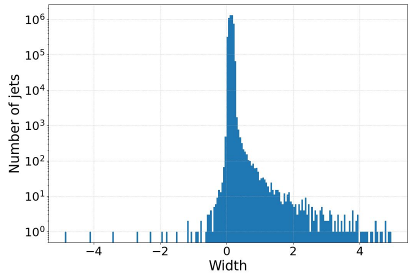
We visualize the data distribution for each variable in the training set

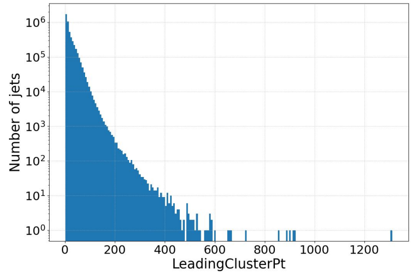


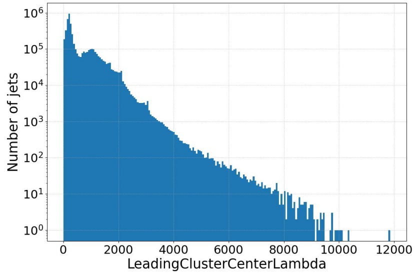


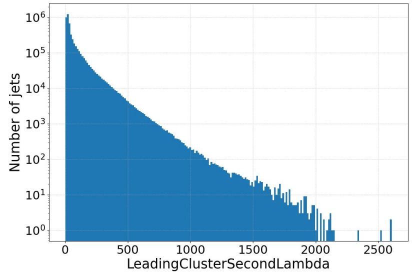


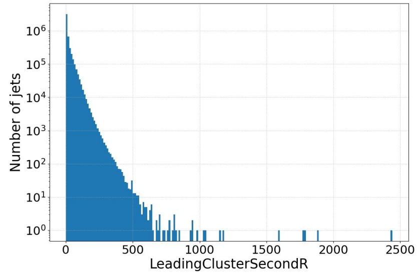




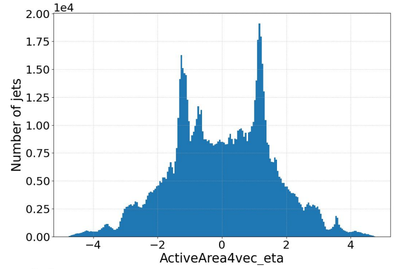


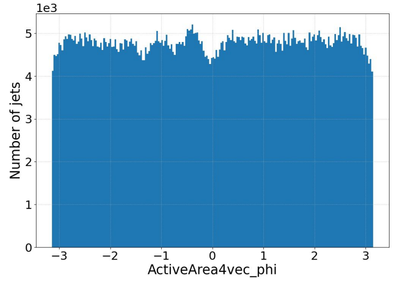


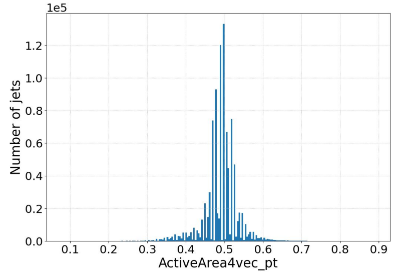


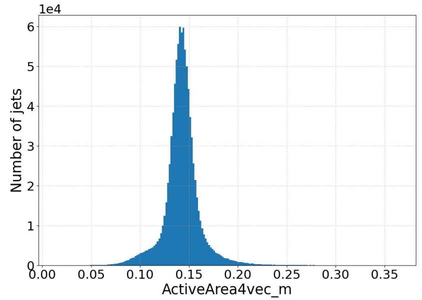


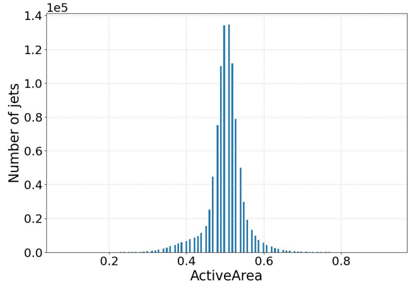
Data distribution - 27D (cont.)

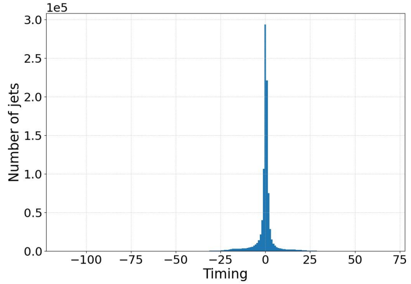


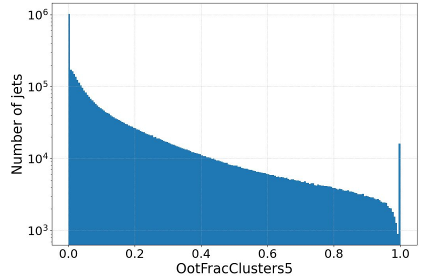


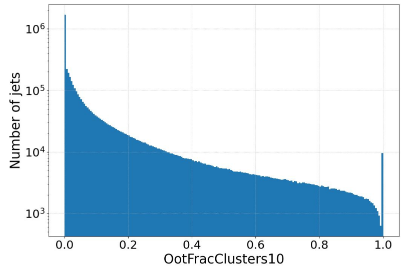


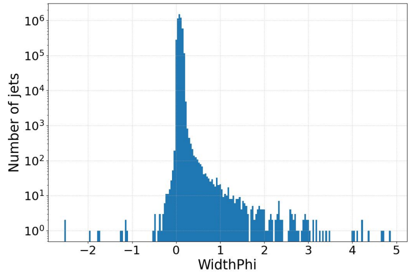




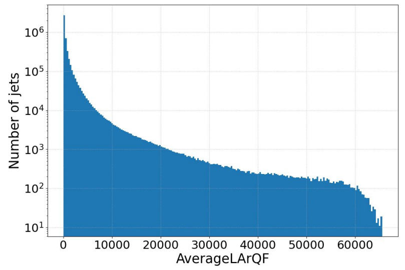


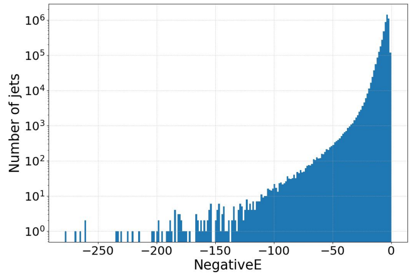


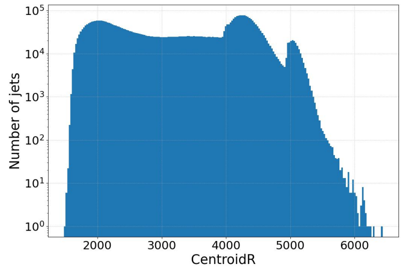


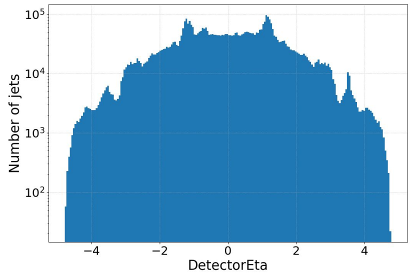


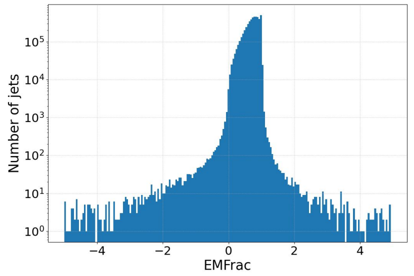
Data distribution - 27D (cont.)

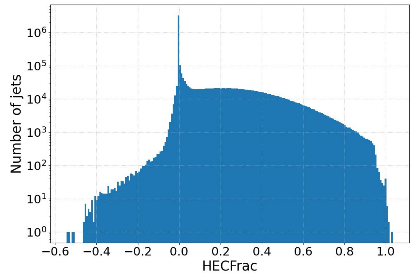


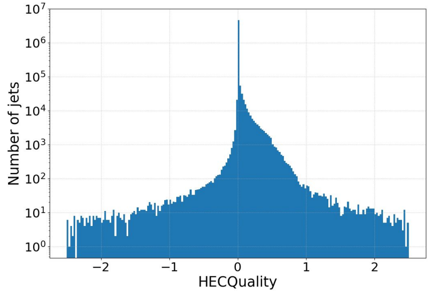


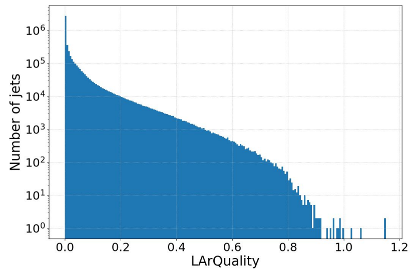


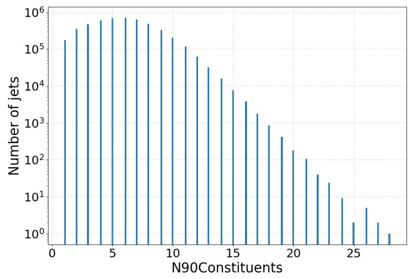












2: Comparison with existing results

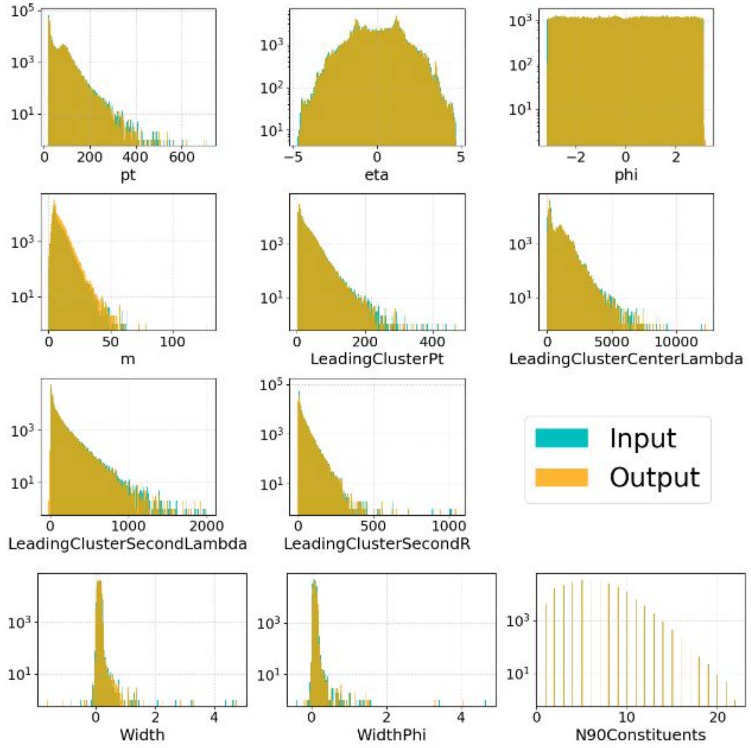
Model details for the available results

- LeakyReLU, BN
- Custom-norm, 27D data
- Latent space = 14
- Model
    - 27-200-200-200-14-200-200-200-27

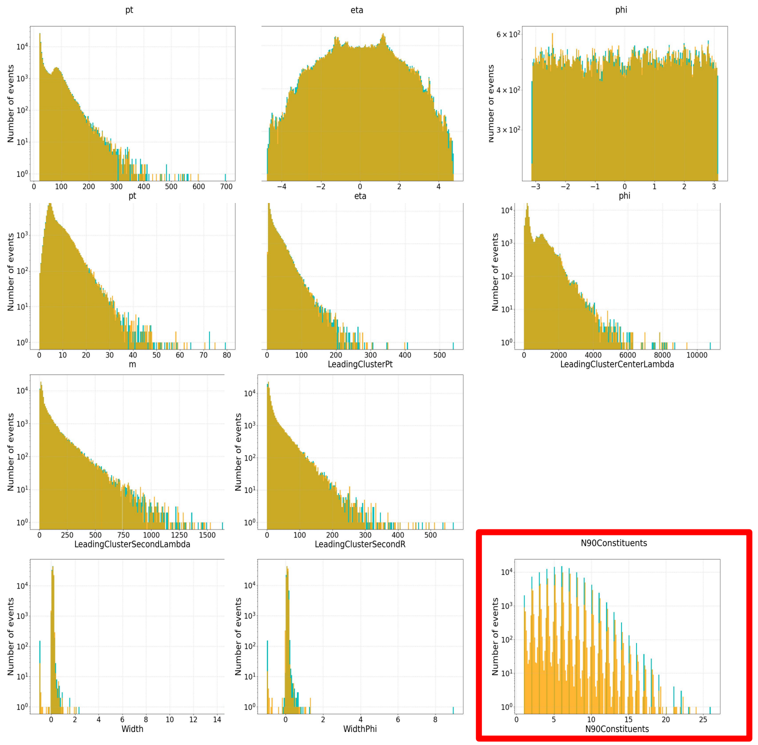
Model details for the available pre-trained model

- LeakyReLU, BN
- Custom-norm, 27D data
- Latent space = 20
- Model
    - 27-200-200-200-20-200-200-200-27

Plots from the available results



Plots for the results from the pre-trained model



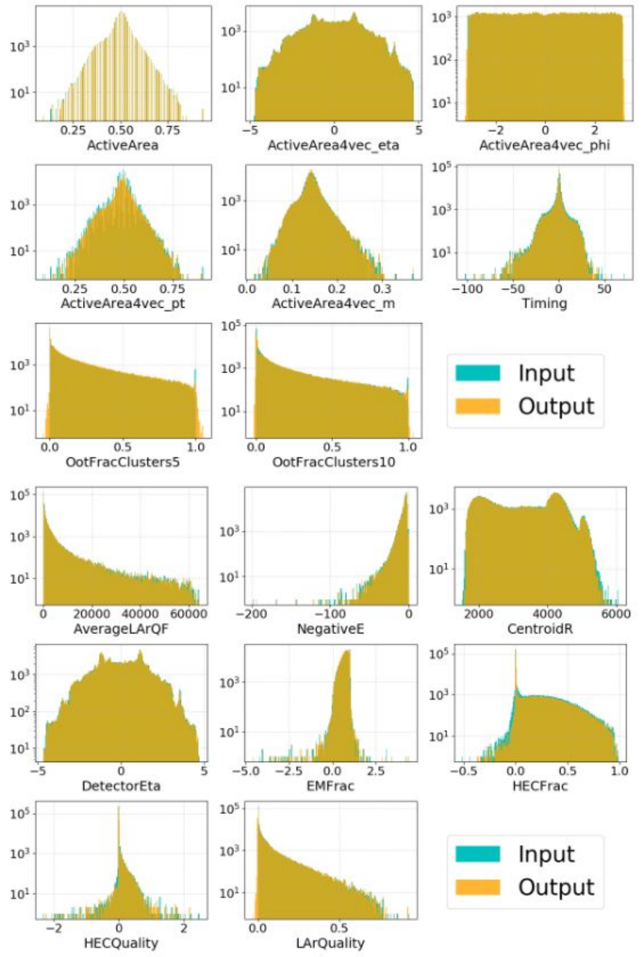
Observations:

- Performance of the available pretrained seem to be very similar to the existing results

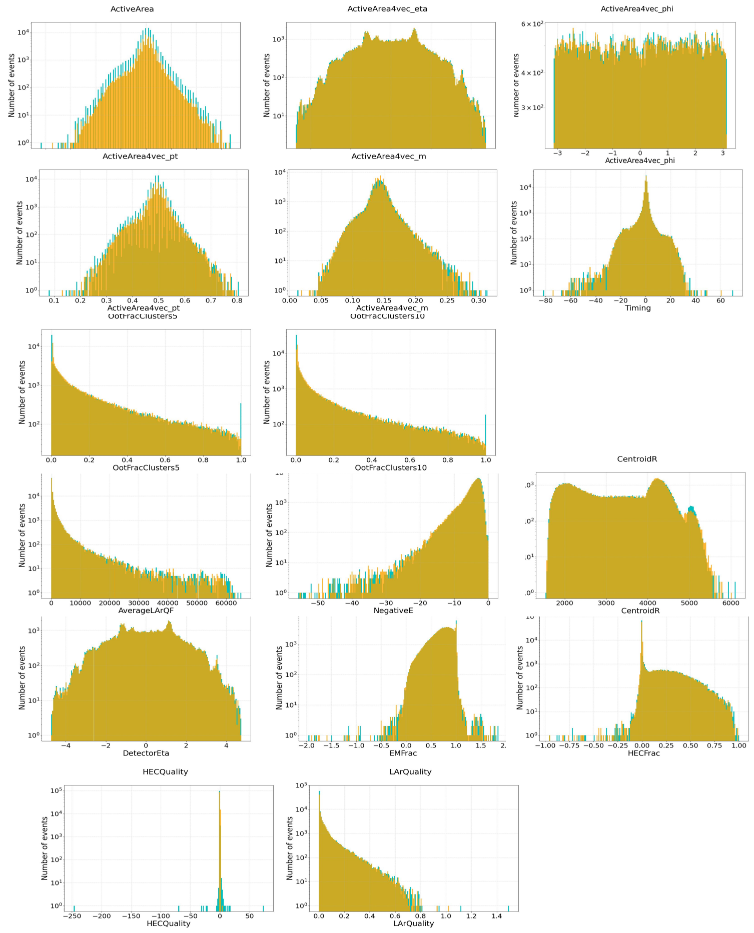
Note:

- *The variable N90Constituents was not casted to int for the pre-trained model, which leads to the difference in the output plots*
- *the axes of the plots are slightly different*

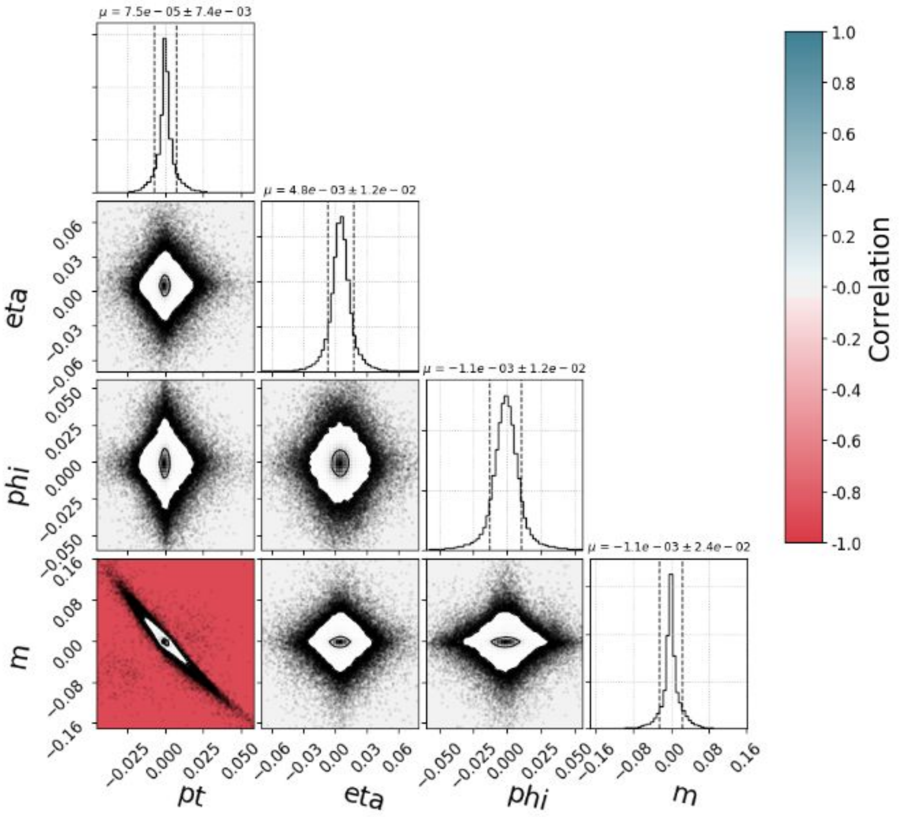
Plots from the available results



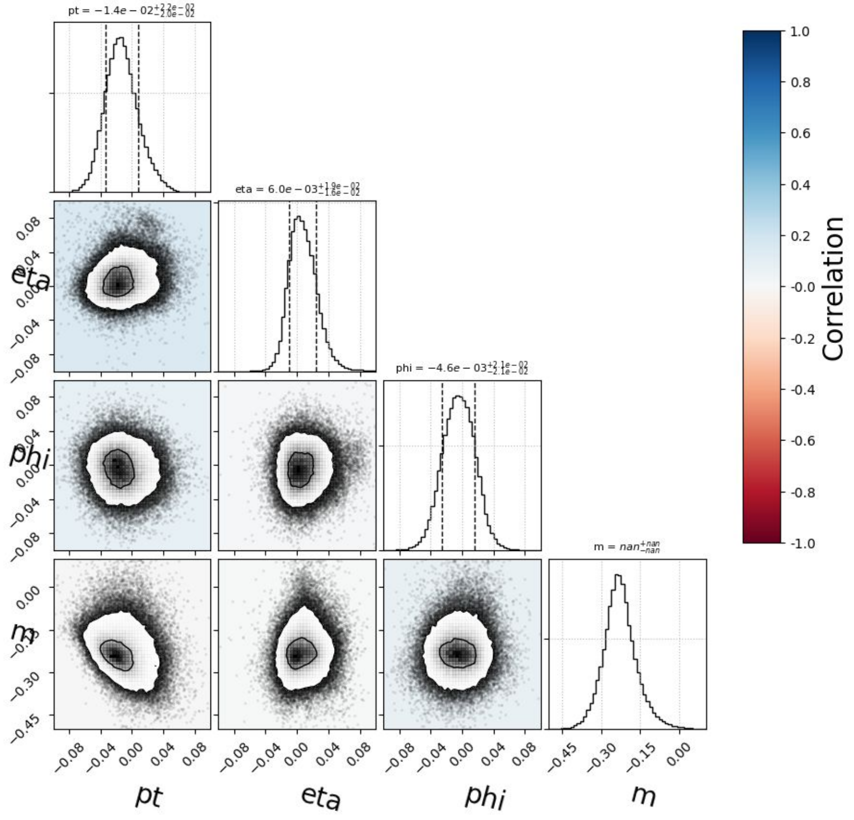
Plots for the results from the pre-trained model



Plots from the available results



Plots for the results from the pre-trained model



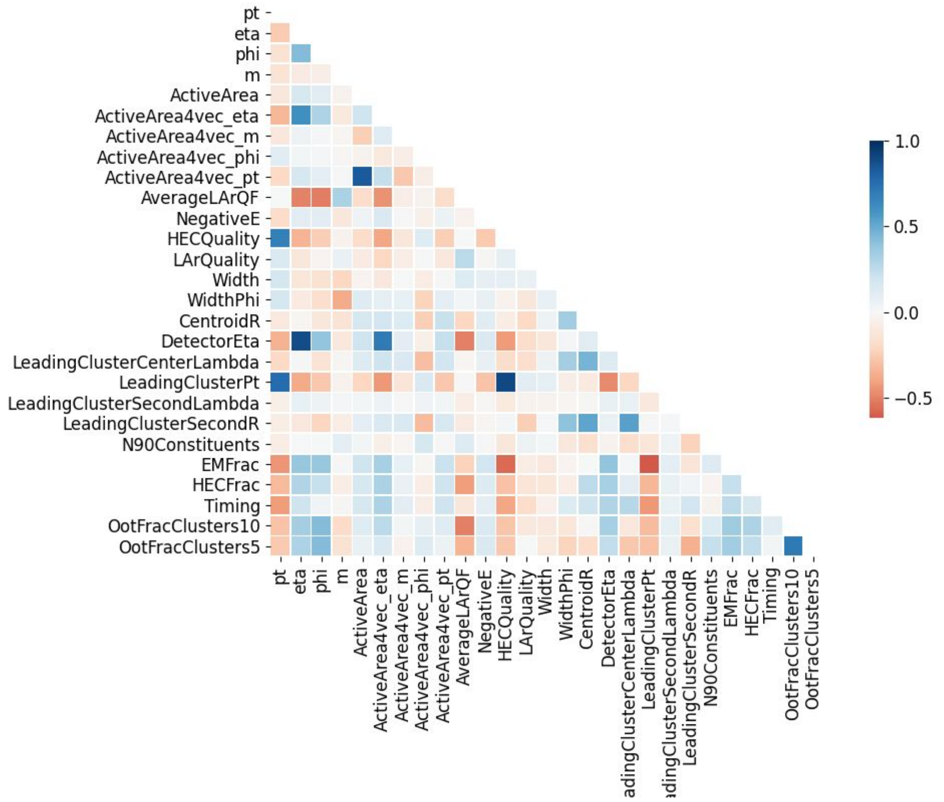
Observations:

- Changes in the latent space dimension changes the correlations

Overall correlation plot for the pre-trained model

Observations:

- We can observe that errors for different variables have considerable correlation among themselves
- This gives a glimpse of the ability of the network to compress different variables and also the tradeoff if some variable is focussed more



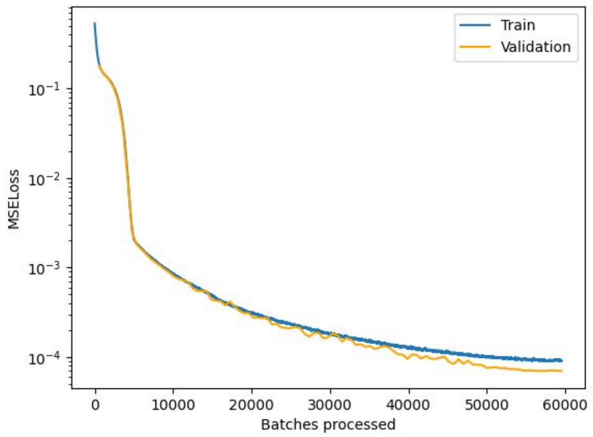
3. Re-training the network on 27D data

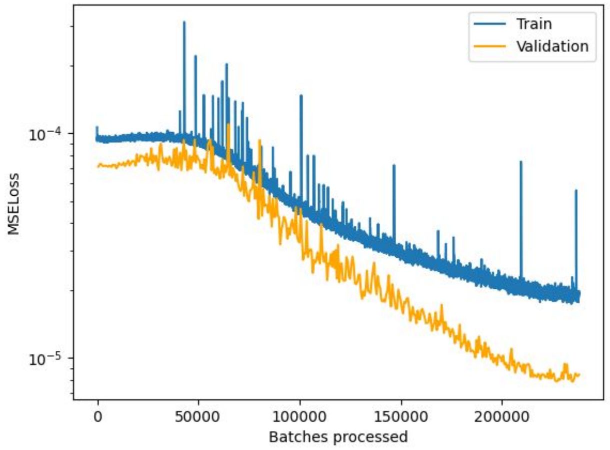
Model details:

- LeakyReLU, BN
- Custom-norm
- Latent space = 20
- Model - [27, 400, 400, 200, 20, 200, 400, 400, 27]

`

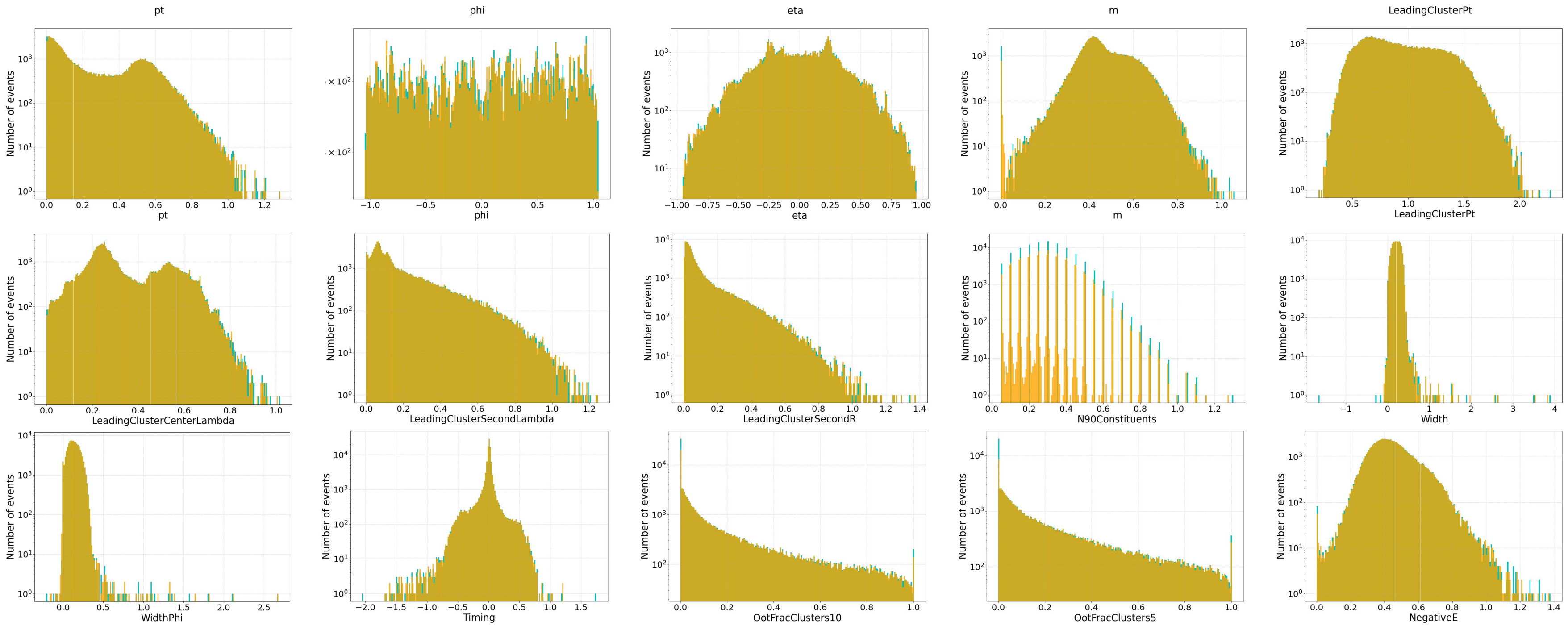
MSE on test-set = 7.844e-06





A comment on the training and validation loss plot - For this particular split and the training settings, the network shows a sharper decrease in the validation loss as compared to the training loss while training. A reason for this could be that the particular split/hyperparameter makes the network learn the data distribution in a better way. An interesting experiment would be to retrain the network with different settings and observe the behaviour.

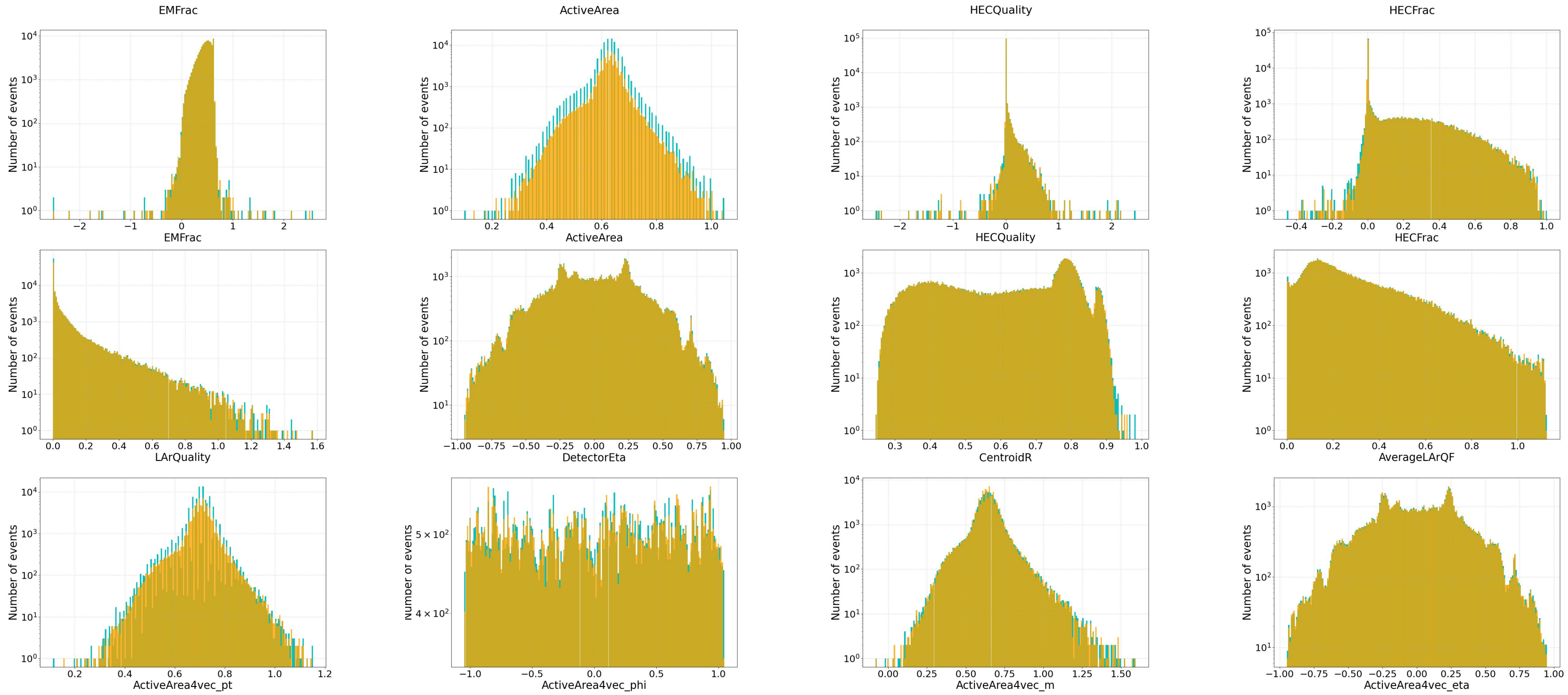
Input-output plots for the retrained model



Observations:

- The plots show that a faithful reconstruction for majority of the variables. Since we did not cast the *N90 Constituents* variable, the reconstructed outputs for this variable seems troublesome.

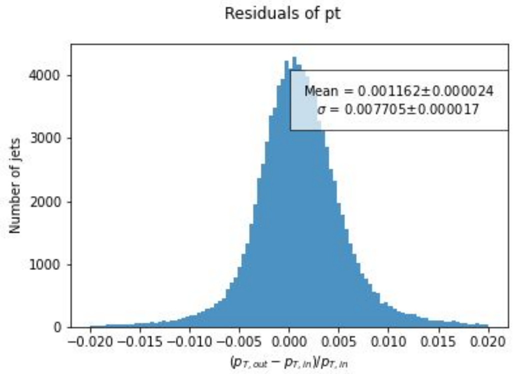
Input-output plots for the retrained model (cont.)

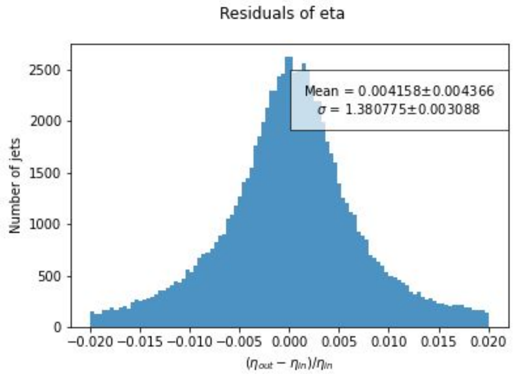


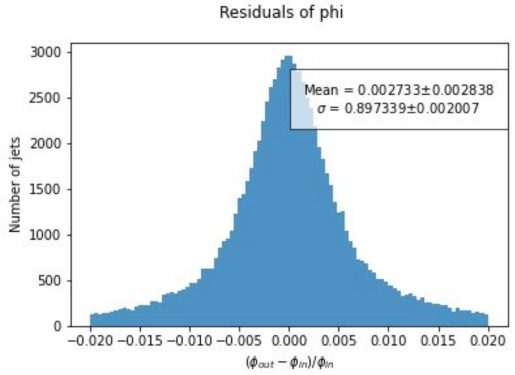
Observations:

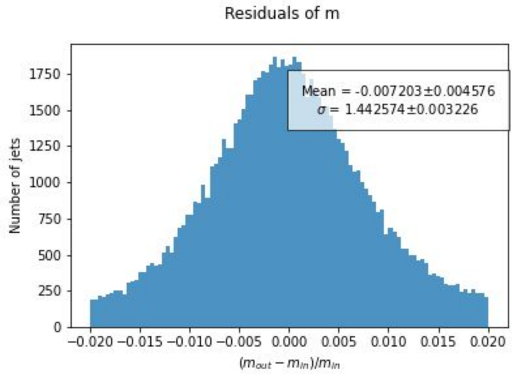
- Similarly in the case of the variables shown here, most of the reconstructions seem good, except for *ActiveArea* and *ActiveArea4vec_pt* variables as we have not casted the output variables to int.

1D response plots for the retrained model





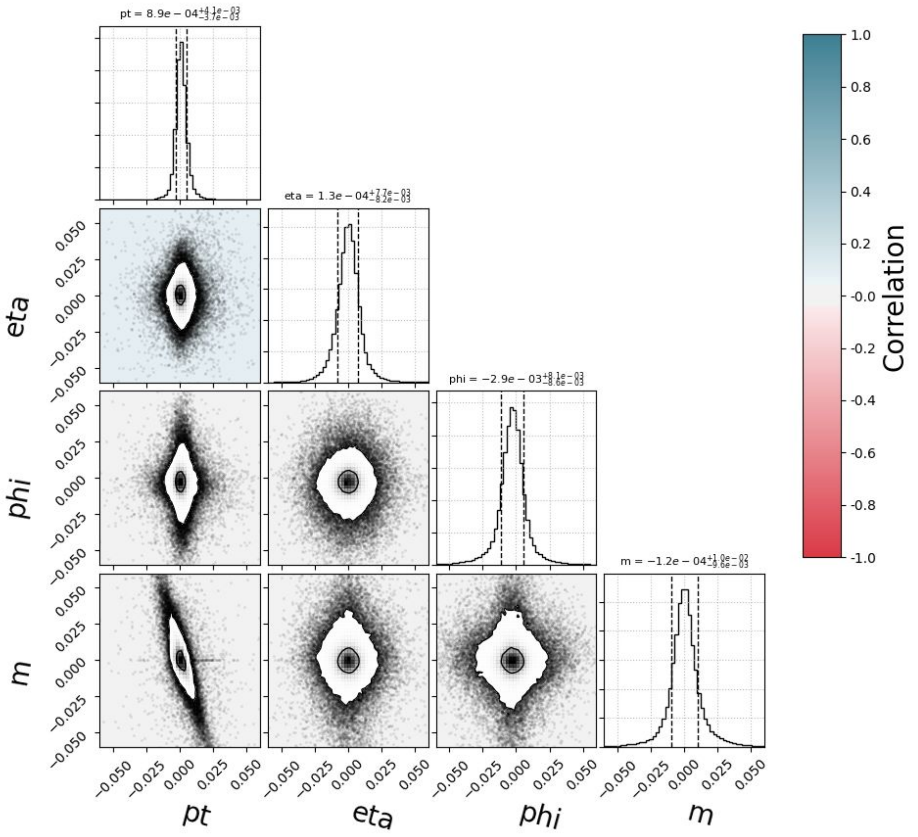


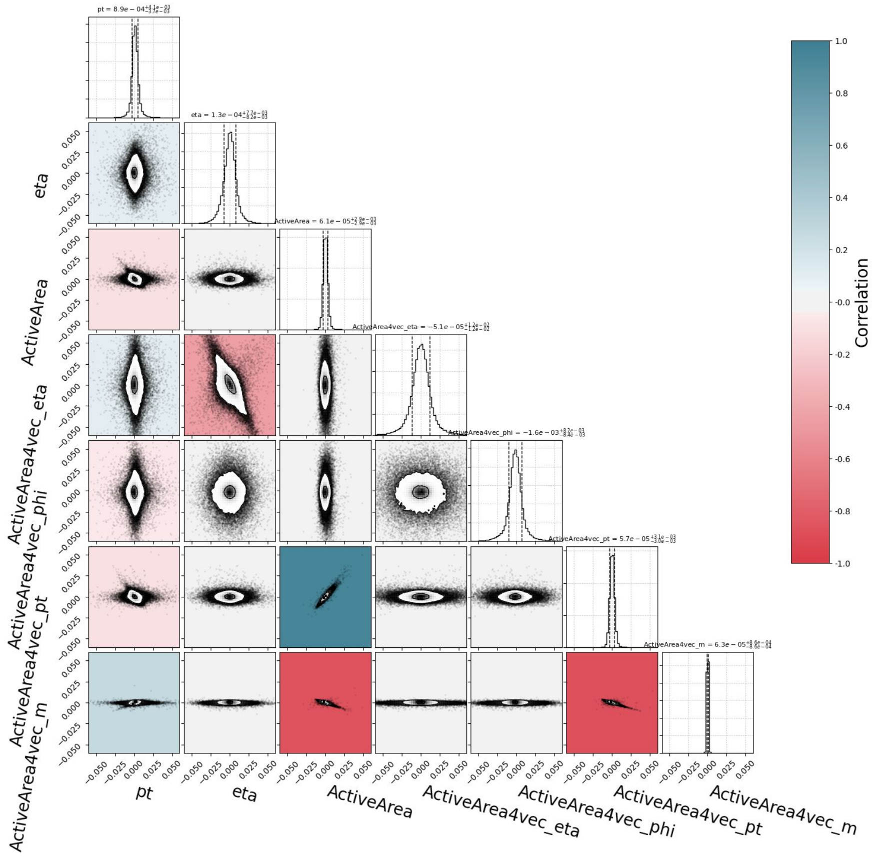


Observations:

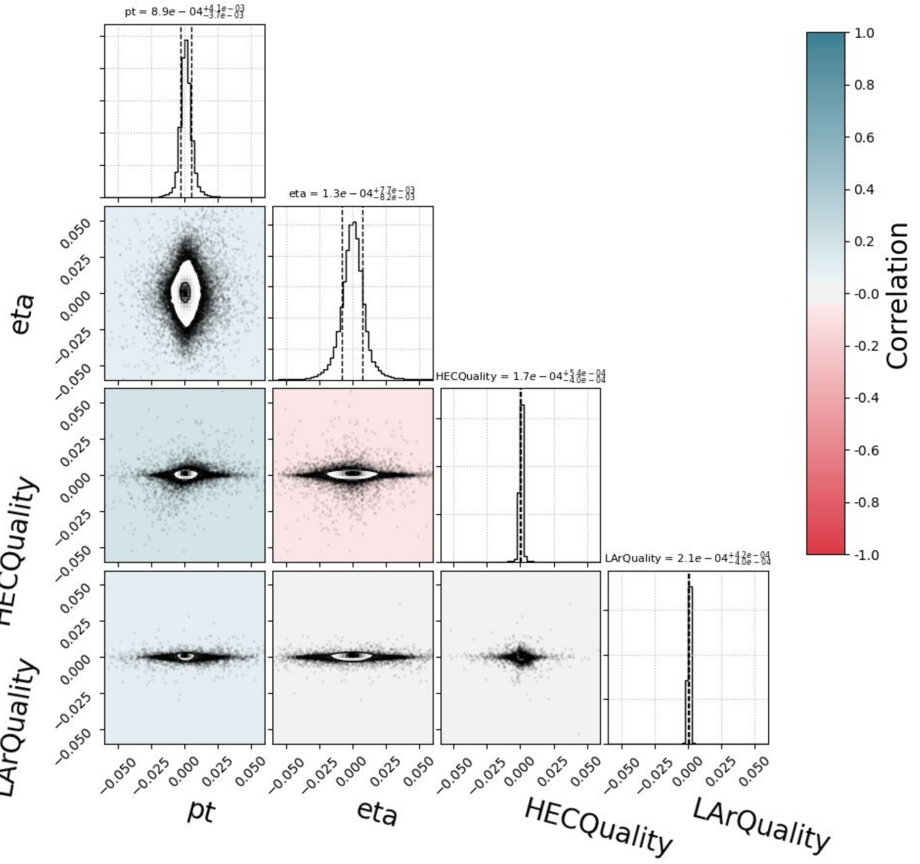
- The 1D response plots here show that the distribution of the relative error. We notice that most of the variables are zero centered and have low variance, which depicts that compression model performs fairly well for these variables.

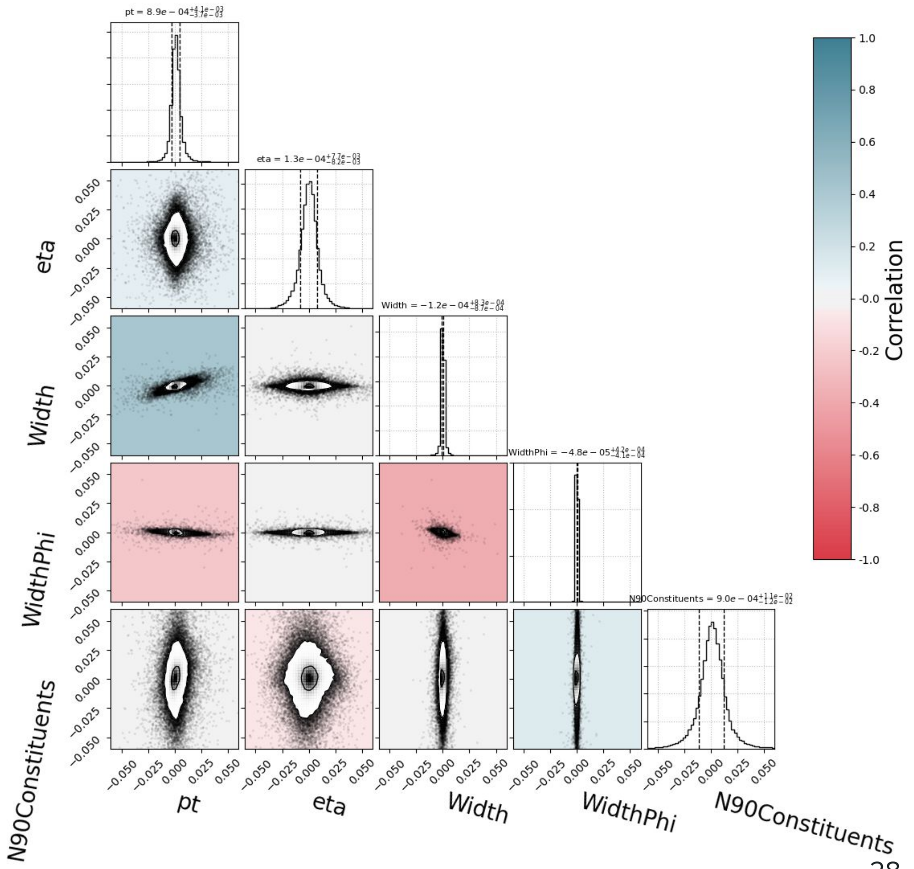
Plots for 27D data - retrained model



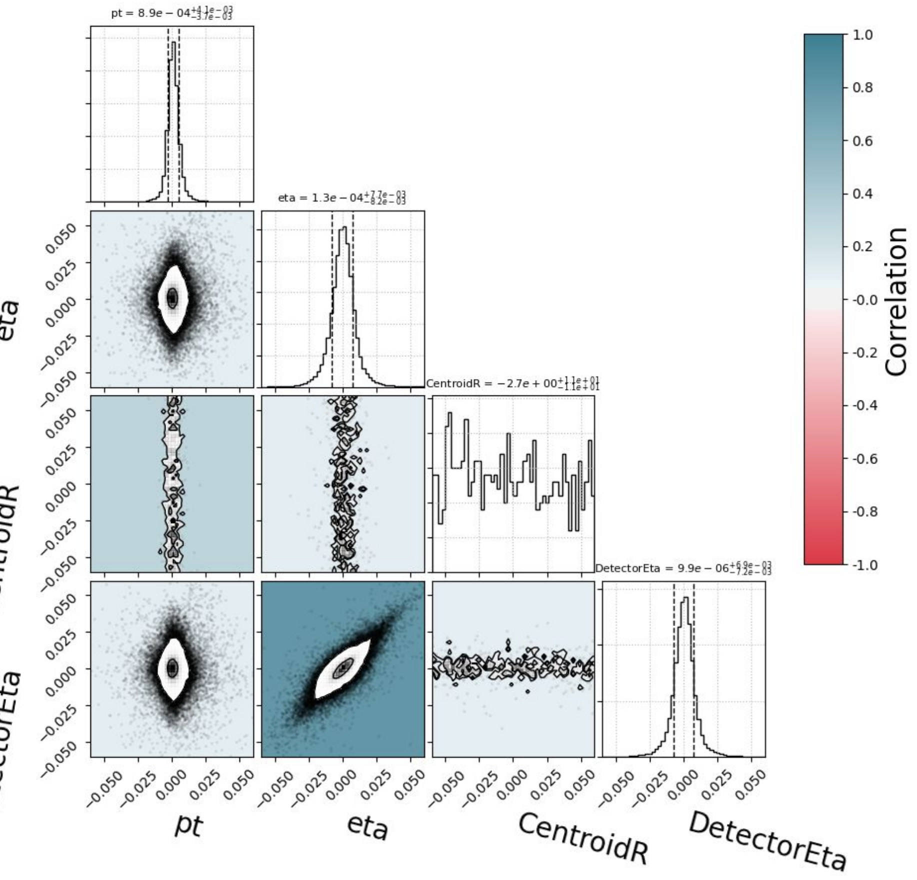


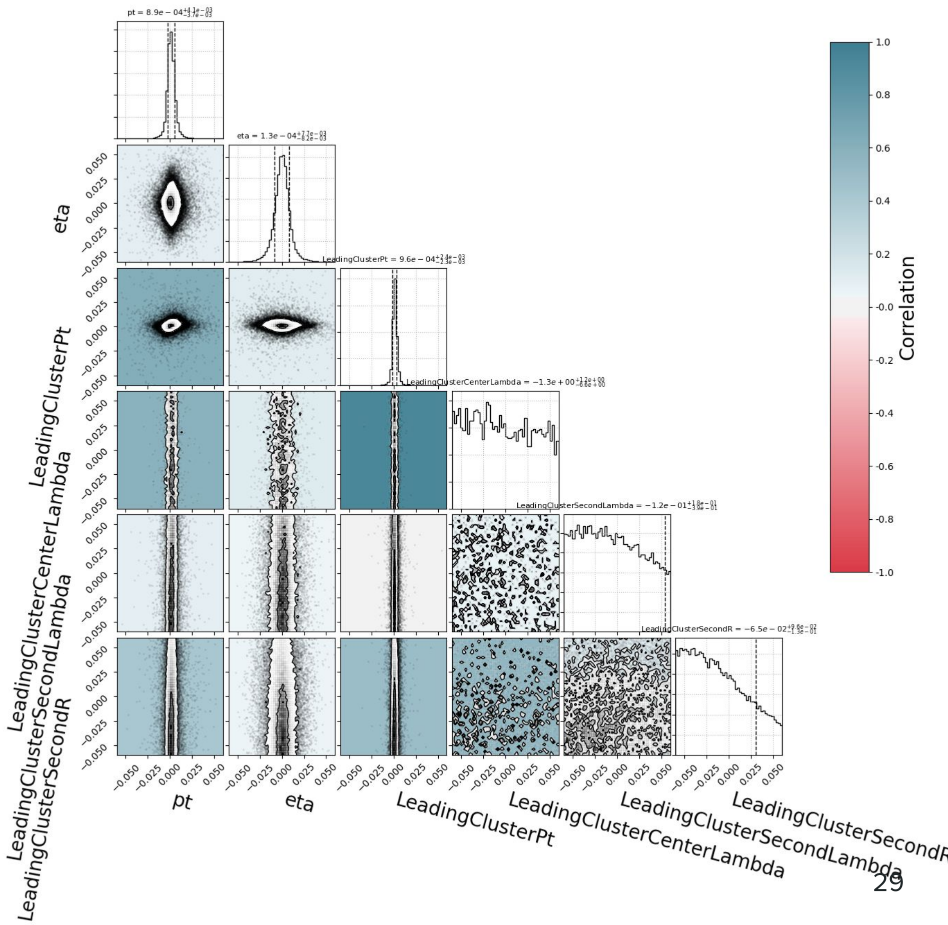
Plots for 27D data - retrained model



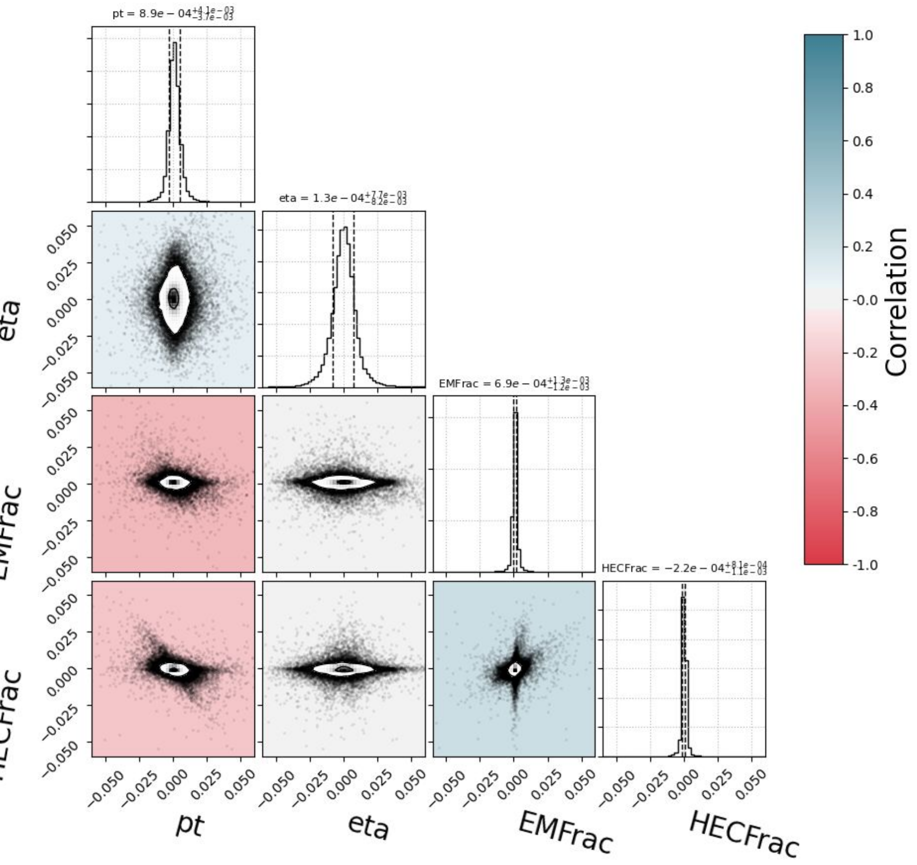


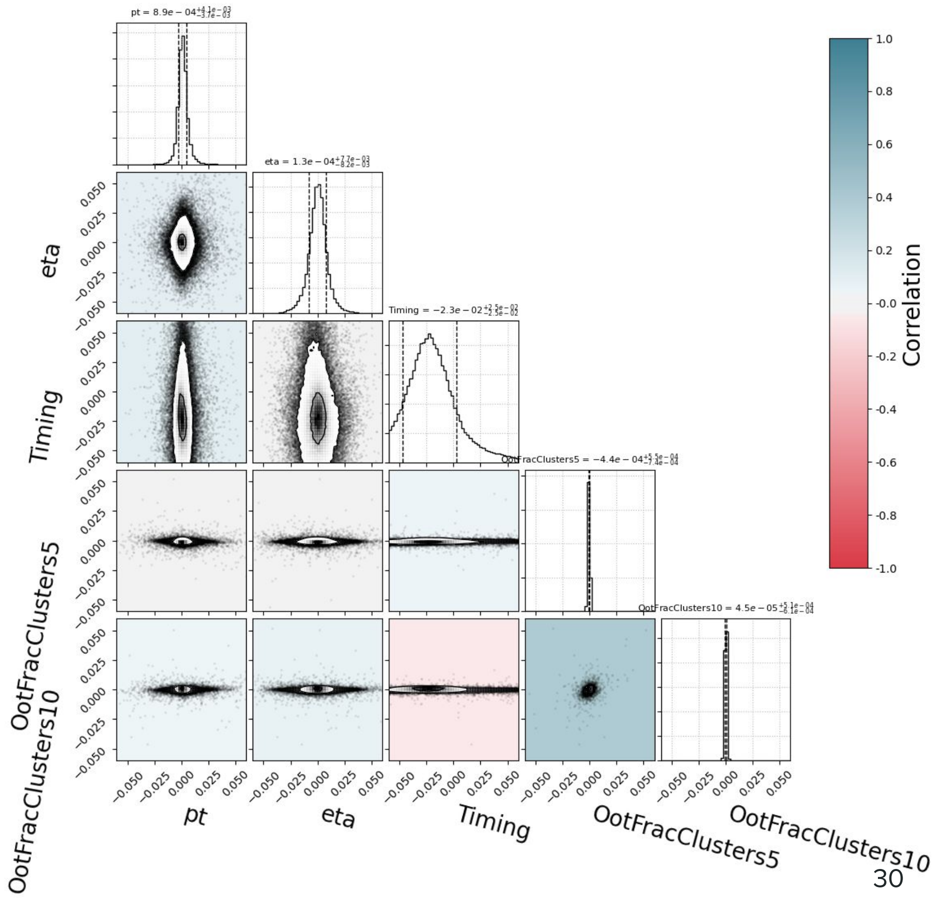
Plots for 27D data - retrained model



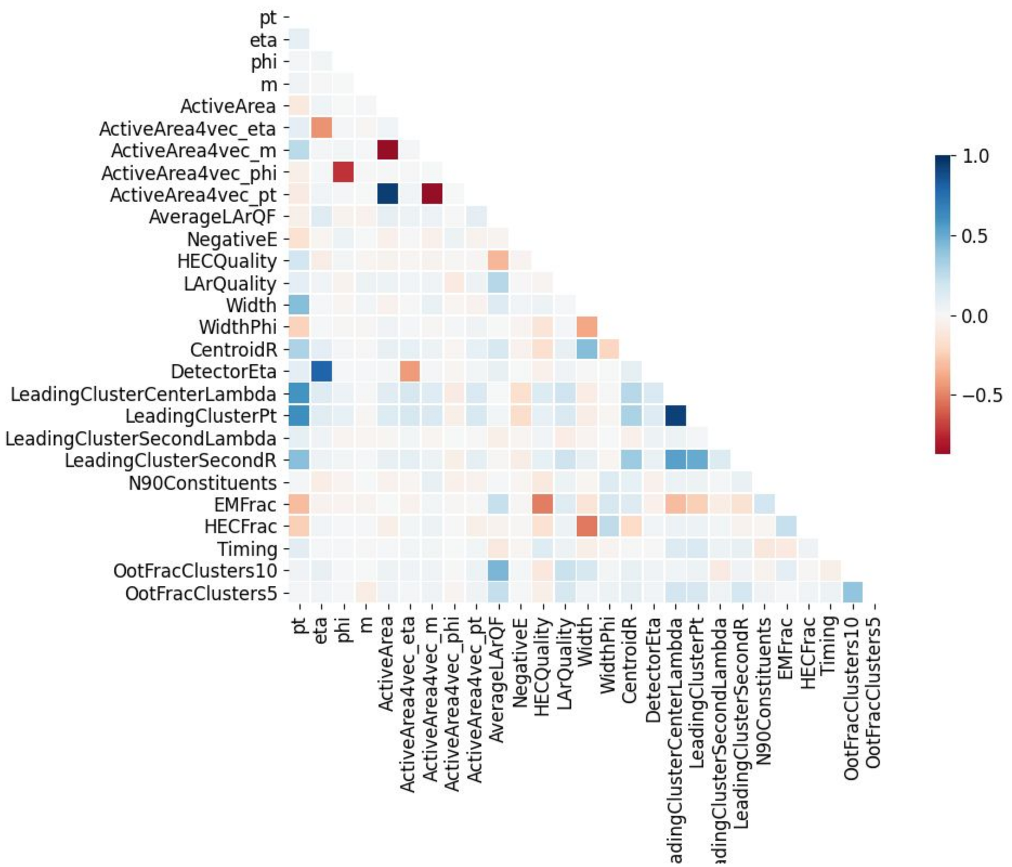


Plots for 27D data - retrained model





Plots for 27D data - retrained model



Standard norm vs Custom norm on PhenoML data

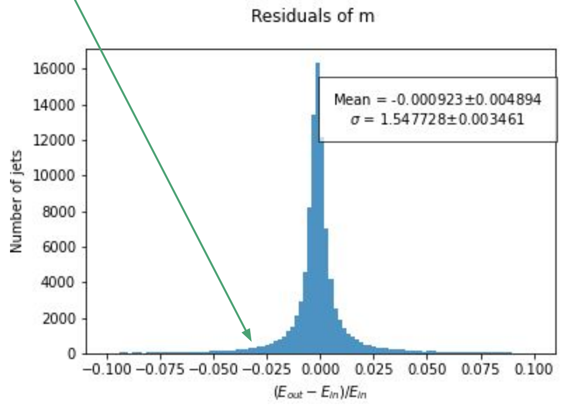
- Performance on the test-set

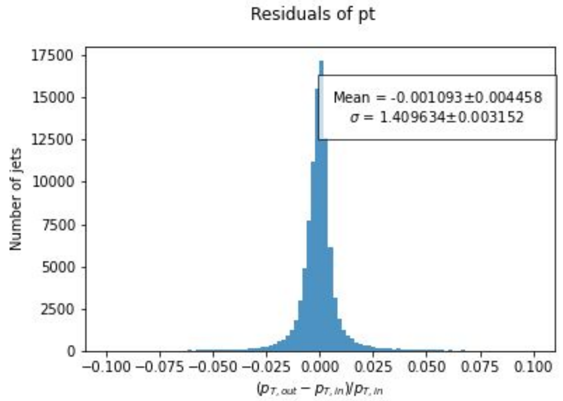
Normalization type	**MSE on the test-set**
Standard	2.7026193e-05
Custom	**8.7358785e-06**

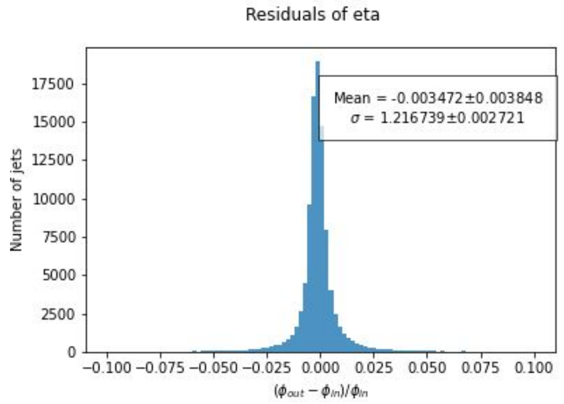
Standard norm on PhenoML data

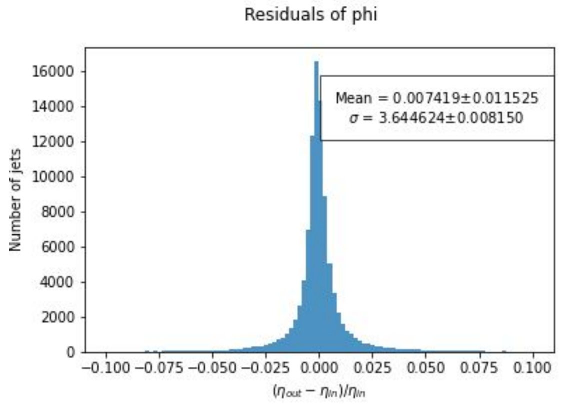
More tails lead to the resolution being worse for standard norm

1D response plots



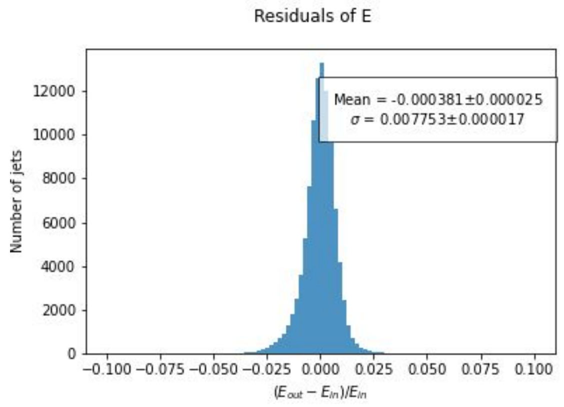


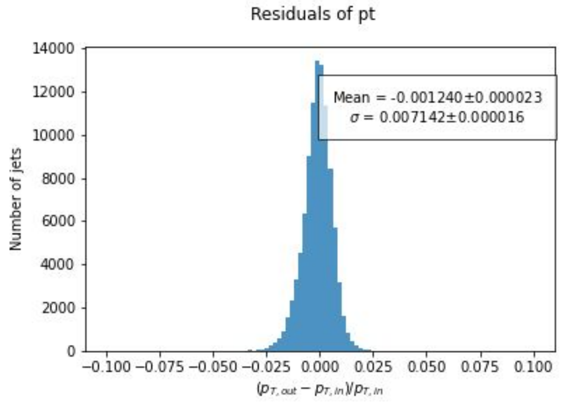


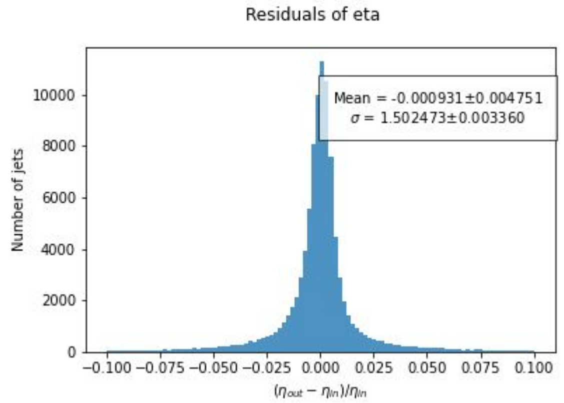


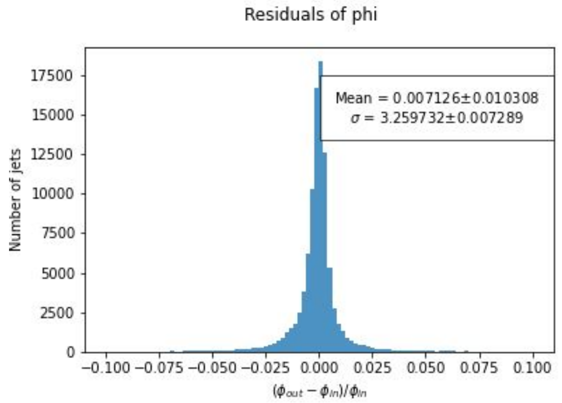
Custom norm on PhenoML data

1D response plots





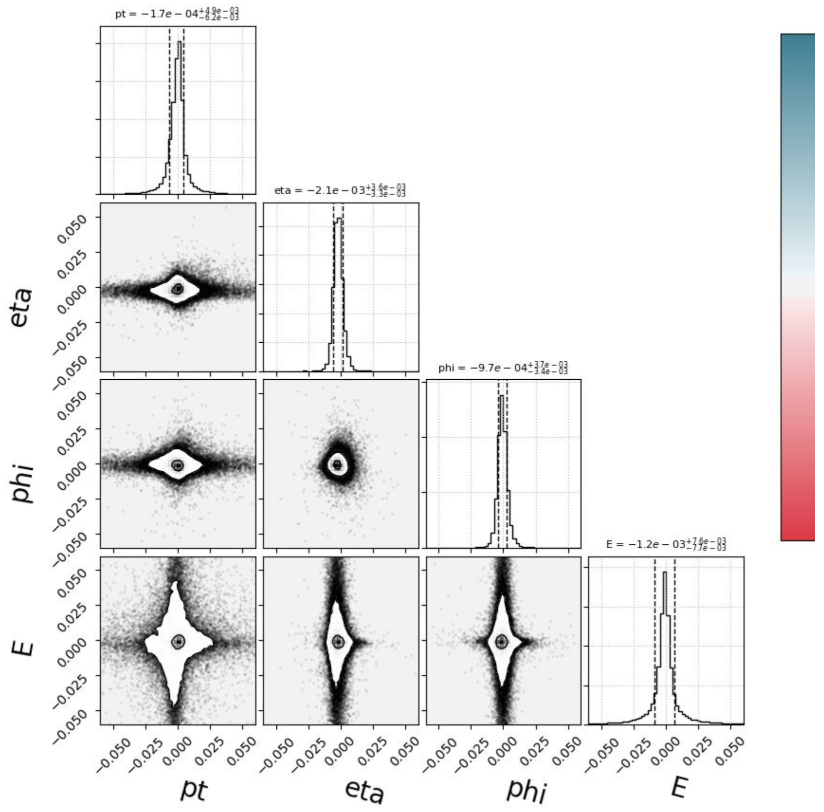




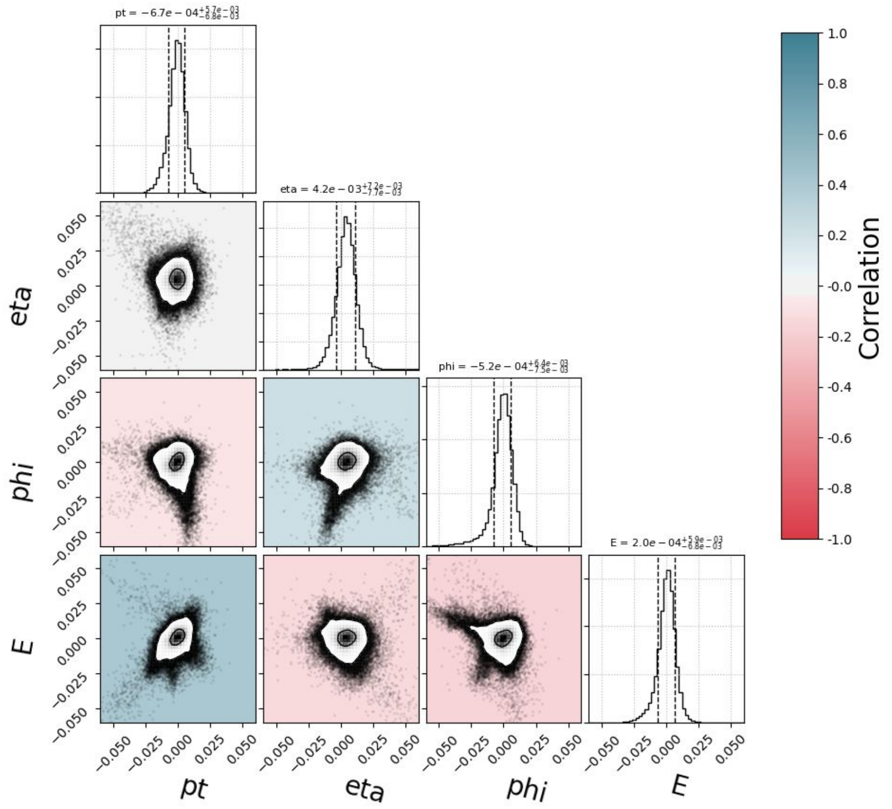
Std vs Custom norm on PhenoML data

Correlation plots

Standard norm



Custom norm



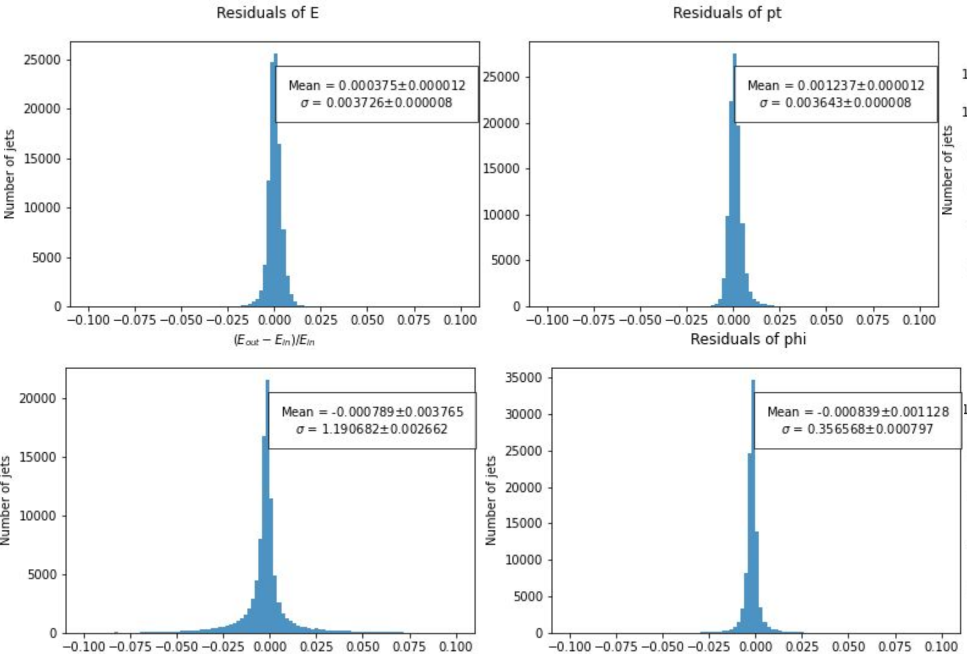
Variation in training-set size

MSE on test-set:

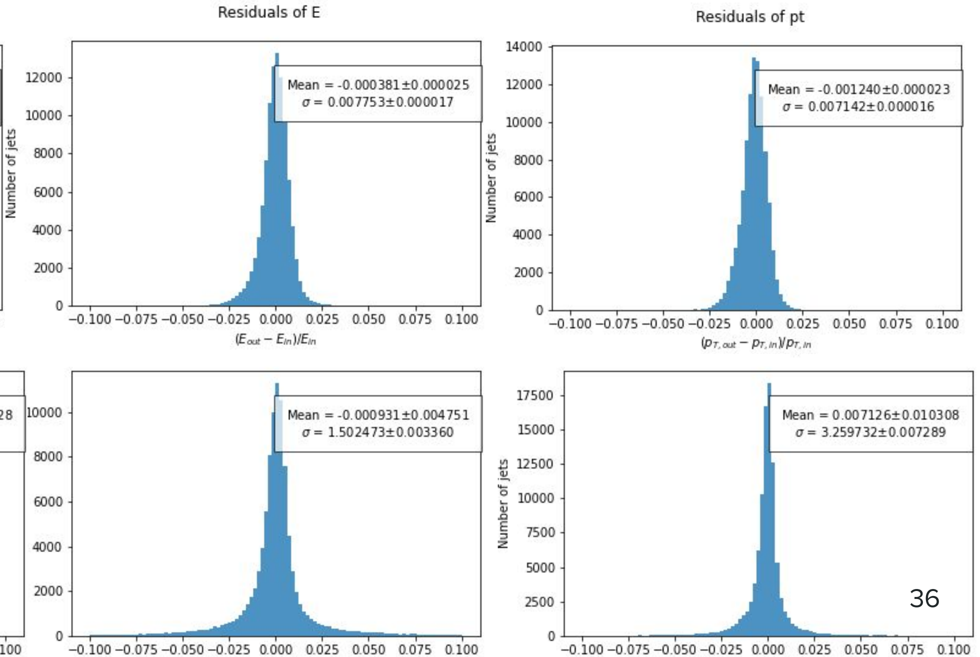
- “Full” dataset (500 MB of jets): 2.485e-06 - 2-3 days, batch size 8k (using the one from before)
  - More requires too much memory to be loaded all at once…
- Half- dataset: 8.735e-06 - 24h
- Question for later: if we increase the batch size && use the full dataset it should take less time…check E. Wulff’s thesis

1D response plots

Full training-set



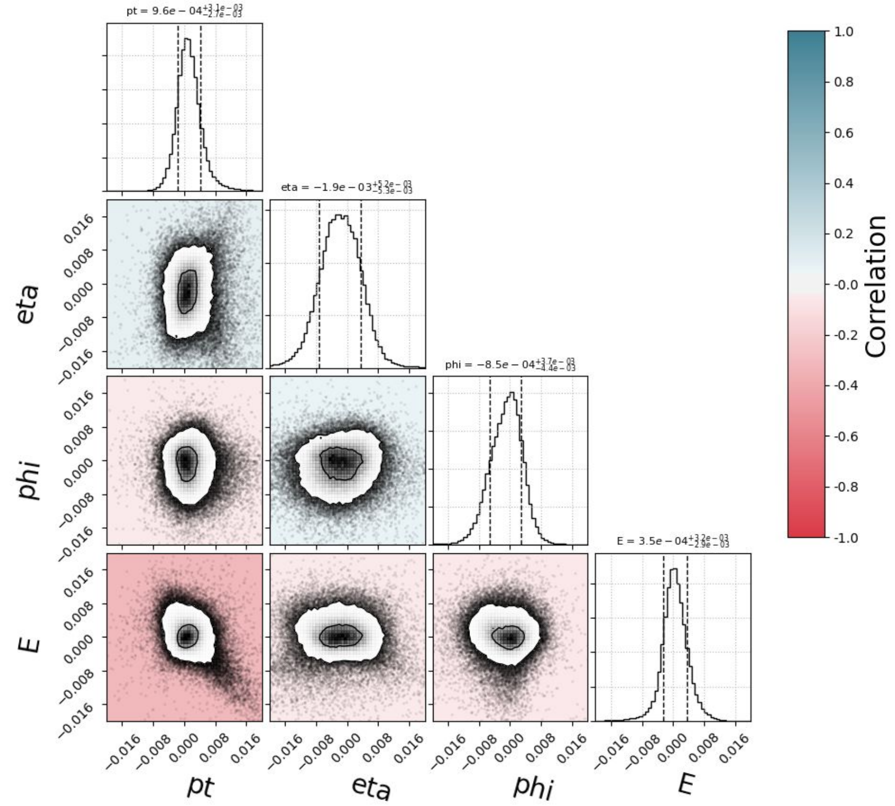
Half training set



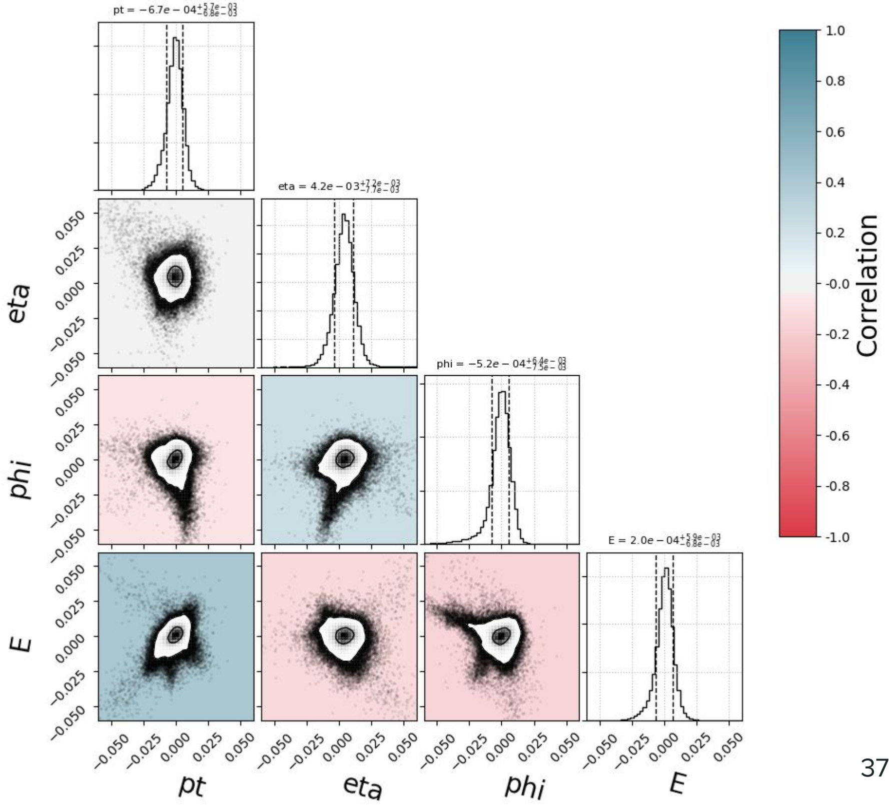
Variation when trained with half data

Correlation plots

Full training-set



Half training set



gamma-top

If any other particle, exclude event

Test with atop_10fb data - events with only jets

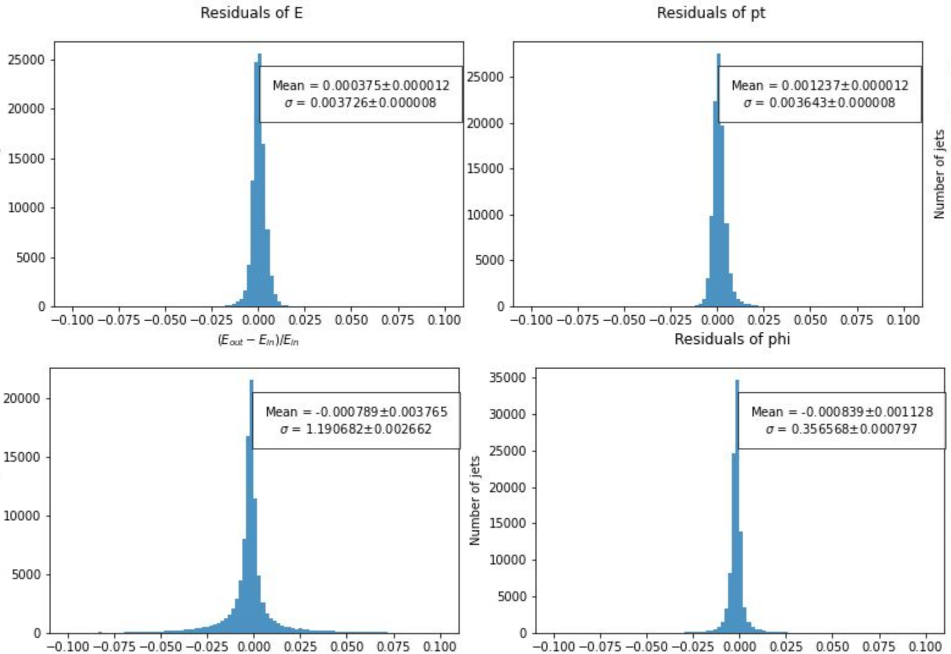
Model details:

- LeakyReLU, BN, Custom-norm
- Latent space = 3
- Model - [4, 400, 400, 200, 3, 200, 400, 400, 4]
- Trained on njets_10fb dataset - containing only jets [number of events?]

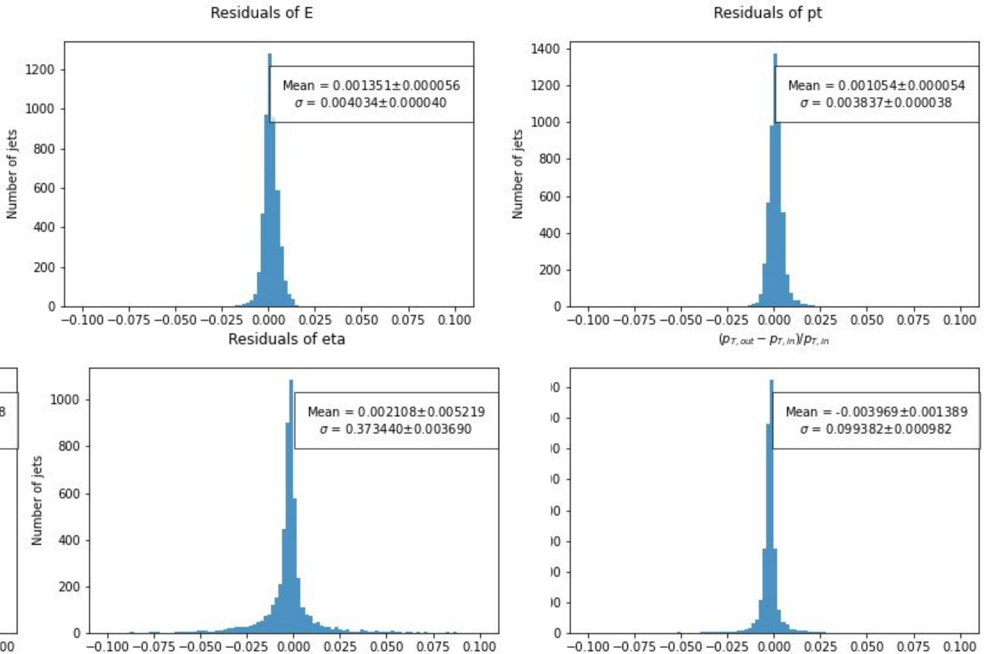
`

Normalization type	MSE on the test-set
atop	2.7047477e-06
Jets	2.4856113e-06

Jets data with custom norm

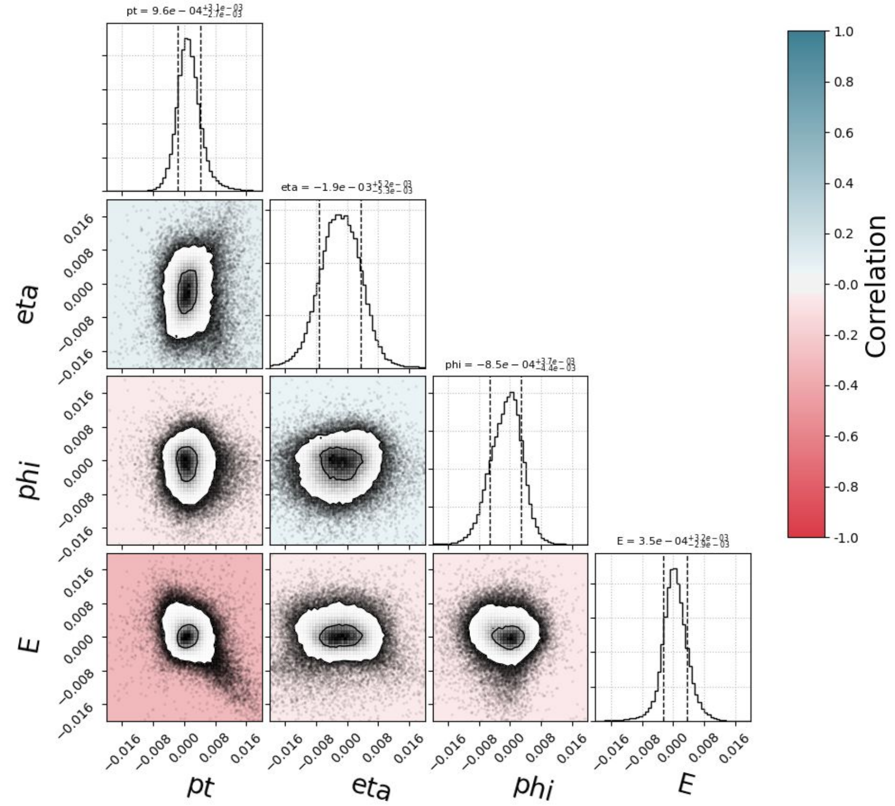


atop events with only jets data with custom norm

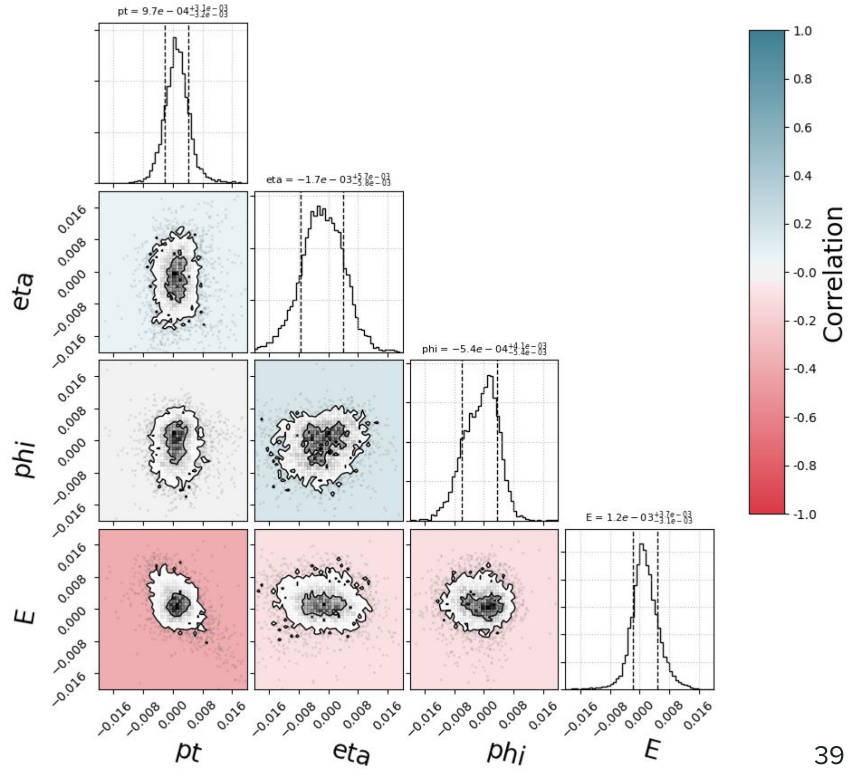


Test with atop_10fb data - events with only jets

Jets data with custom norm



atop events with only jets data with custom norm



Keep all the event, but use only the type of particles == jets in network dataset

Test with atop_10fb data - all events but only jet particles

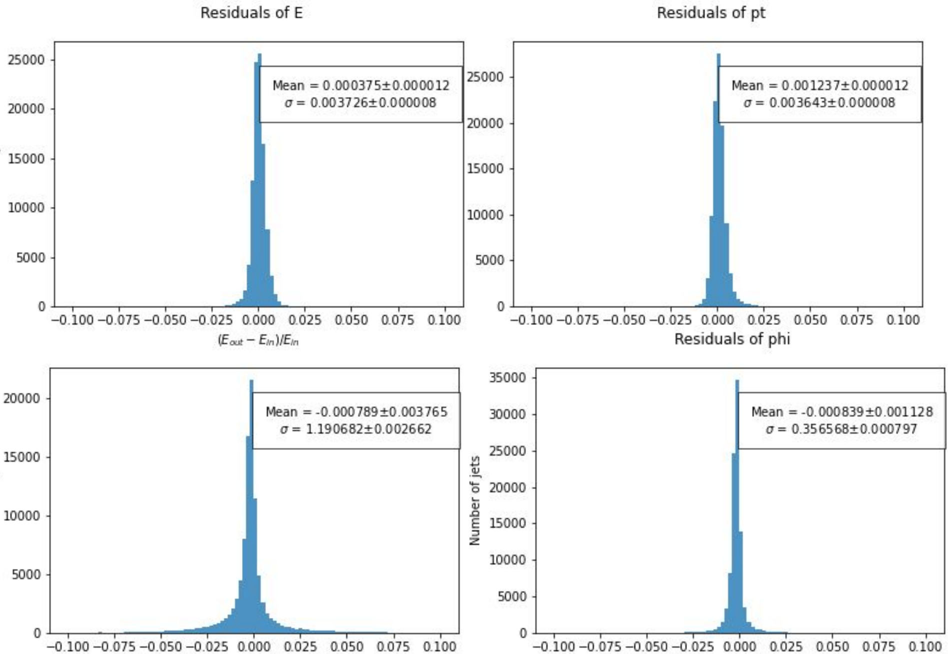
Model details:

- LeakyReLU, BN, Custom-norm
- Latent space = 3
- Model - [4, 400, 400, 200, 3, 200, 400, 400, 4]
- Trained on njets_10fb dataset - containing only jets

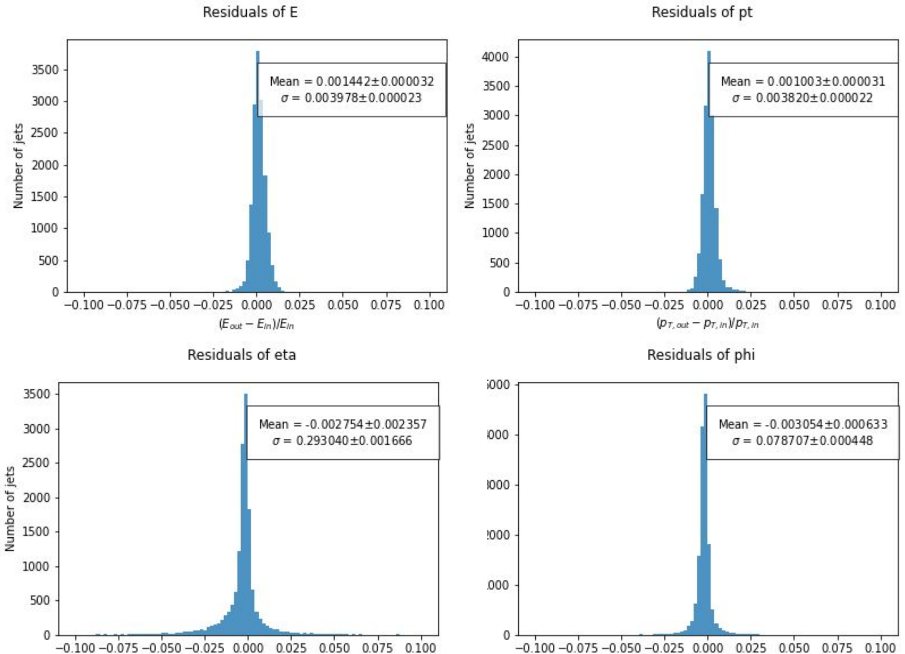
`

Normalization type	MSE on the test-set
atop	2.6954153e-06
Jets	2.4856113e-06

Jets data with custom norm

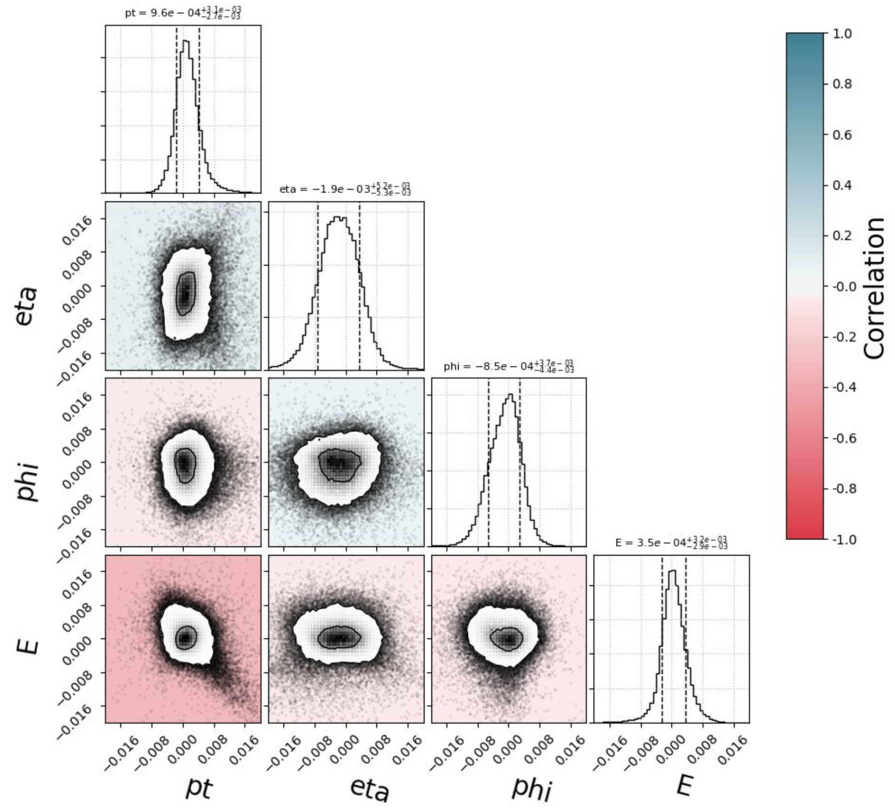


all events but only jet particles

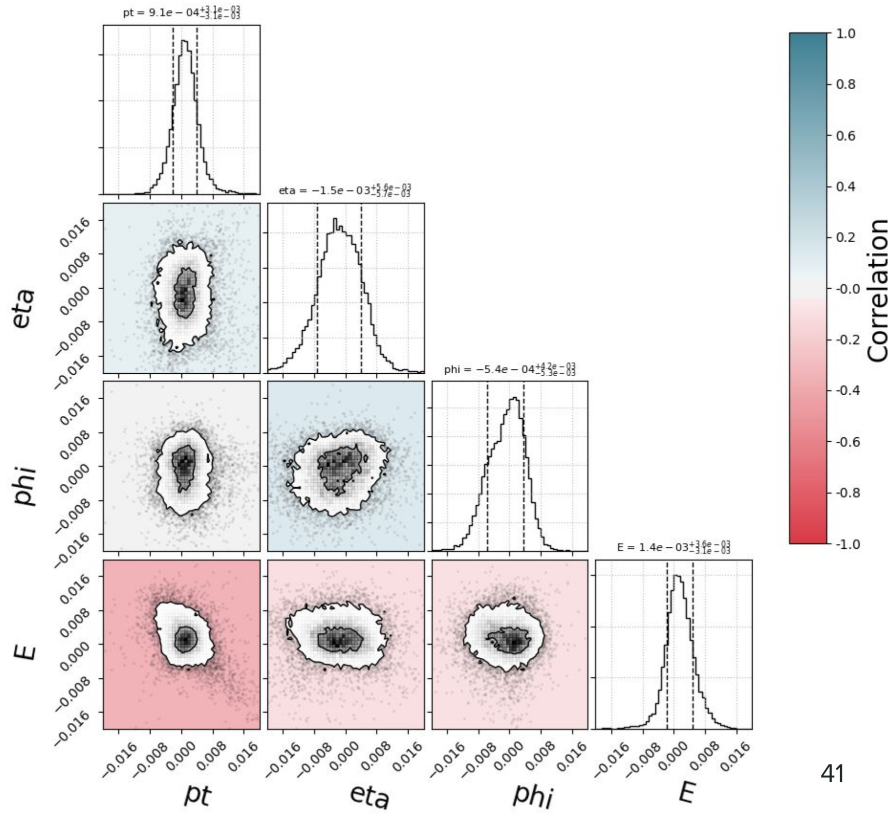


Test with atop_10fb data - all events but only jet particles

Jets data with custom norm



all events but only jet particles



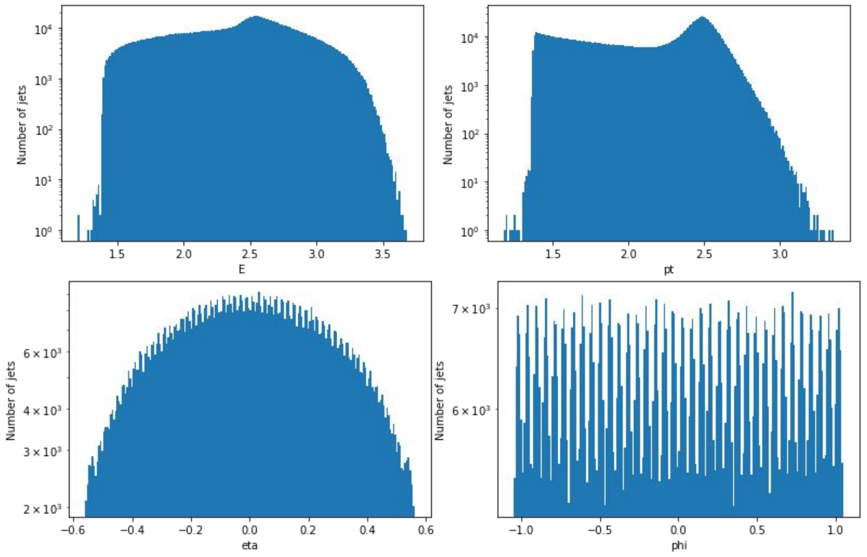
All particles are there

Test with atop_10fb data - all events

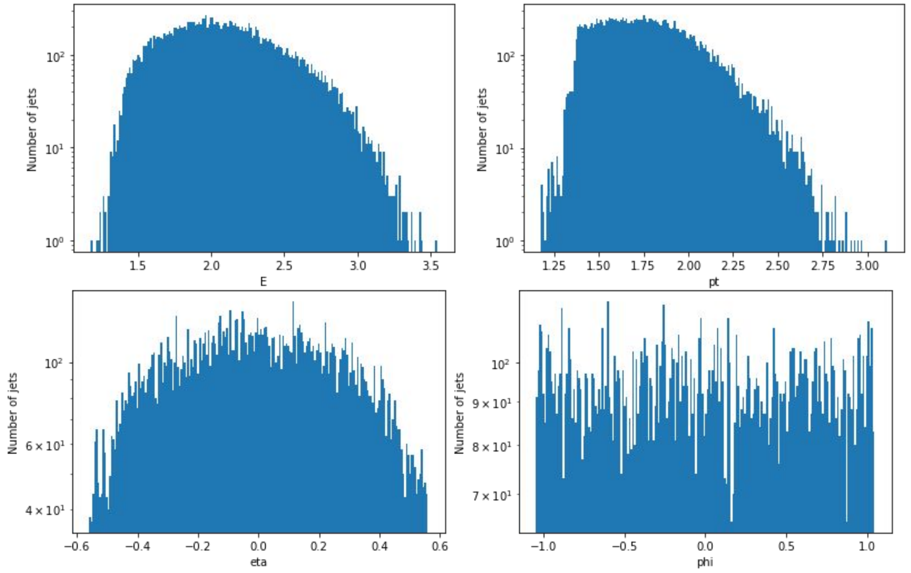
Model details:

- LeakyReLU, BN, Custom-norm
- Latent space = 3
- Model - [4, 400, 400, 200, 3, 200, 400, 400, 4]
- Trained on njets_10fb dataset - containing only jets

Jets data with custom norm



atop data with custom norm



All particles are there

Test with atop_10fb data - all events

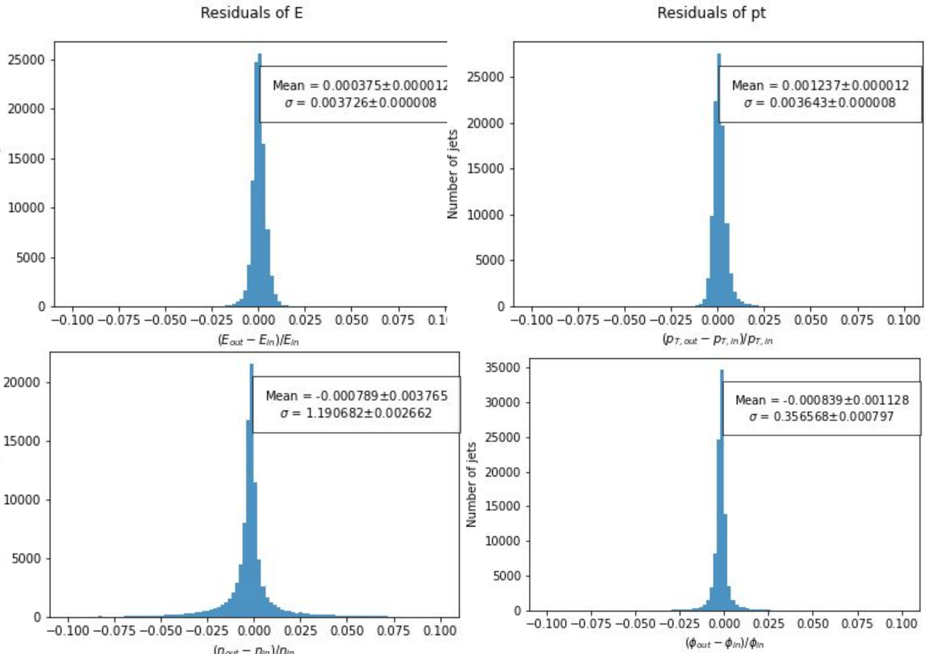
Model details:

- LeakyReLU, BN, Custom-norm
- Latent space = 3
- Model - [4, 400, 400, 200, 3, 200, 400, 400, 4]
- Trained on njets_10fb dataset - containing only jets

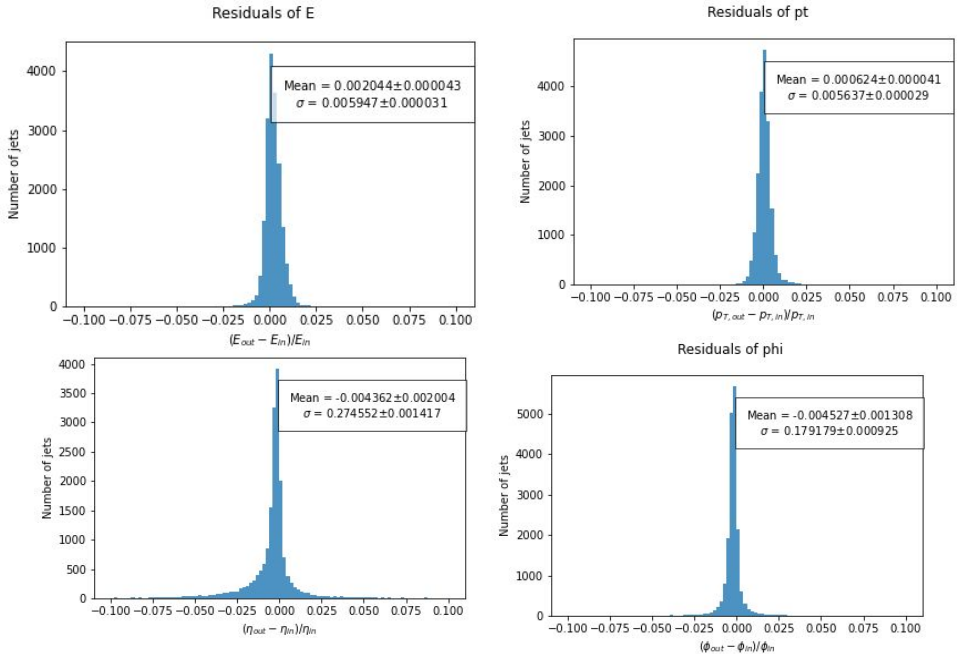
`

Normalization type	MSE on the test-set
atop	6.7475153e-06
Jets	2.4856113e-06

Jets data with custom norm

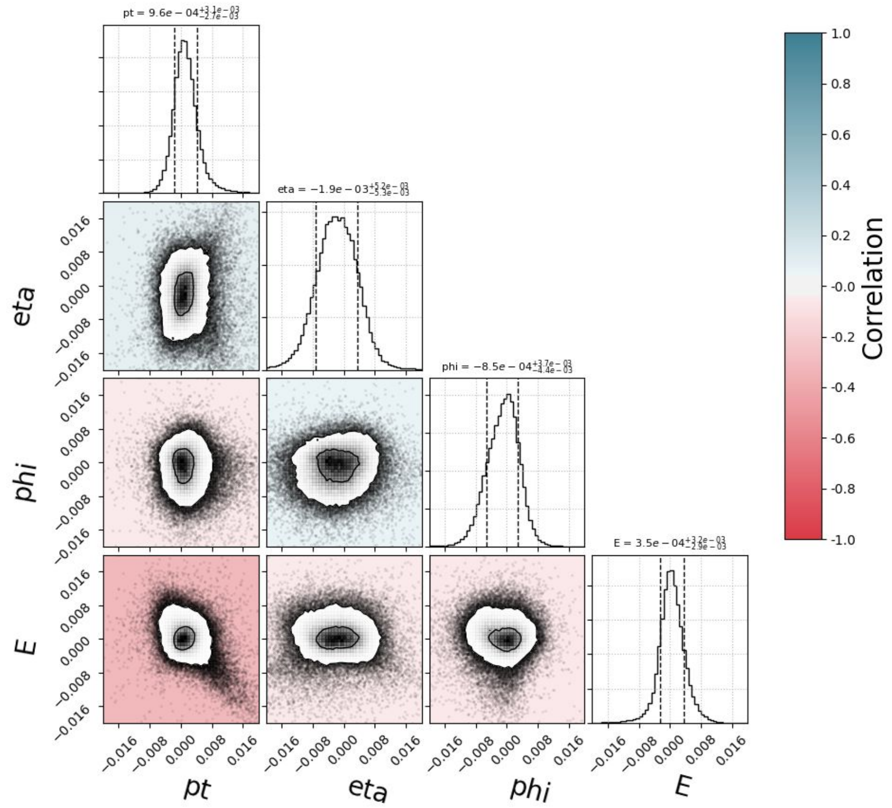


atop data with custom norm

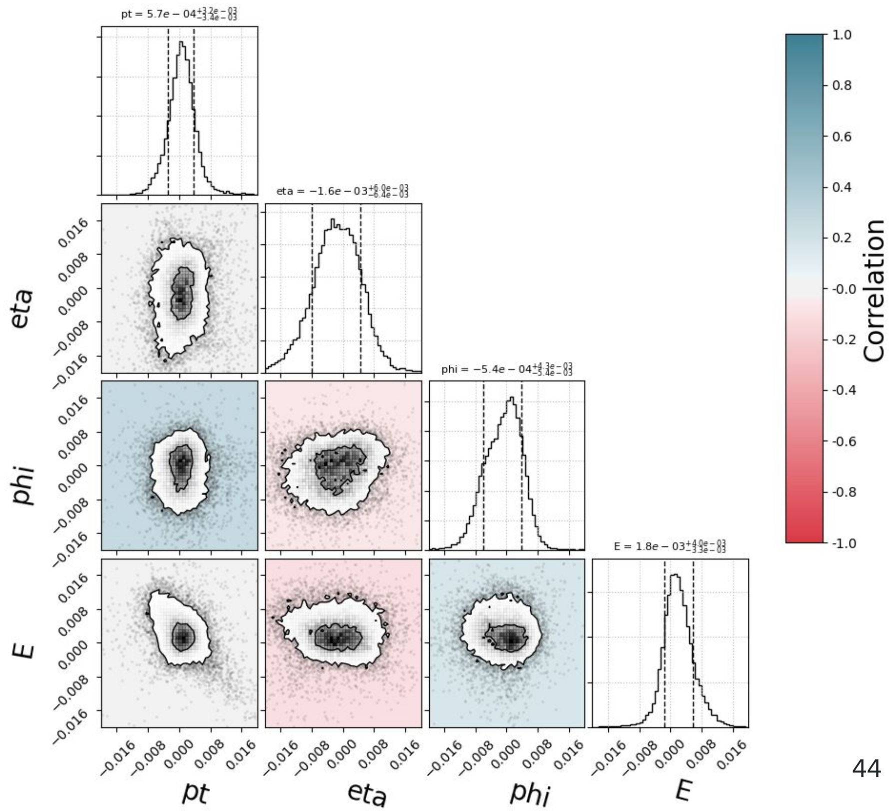


Test with atop_10fb data - all events

Jets data with custom norm



atop data with custom norm



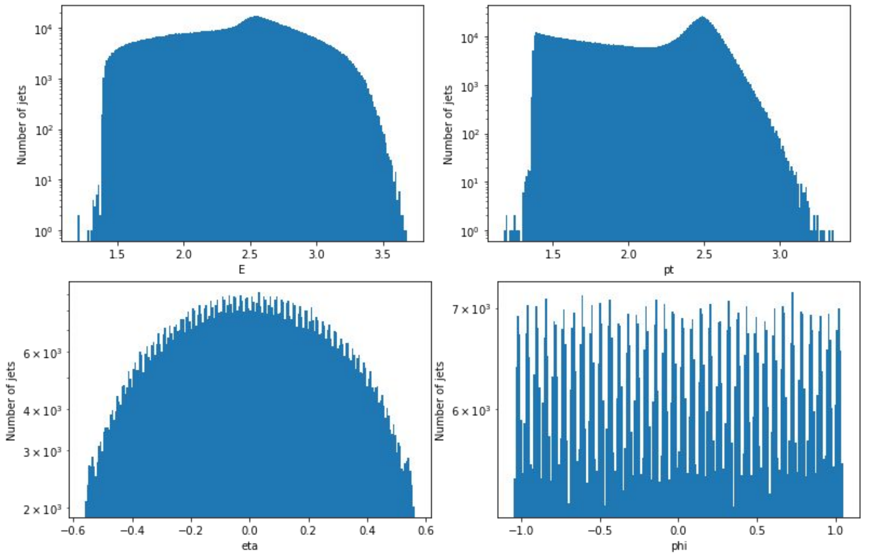
Test with atopbar_10fb data

Model details:

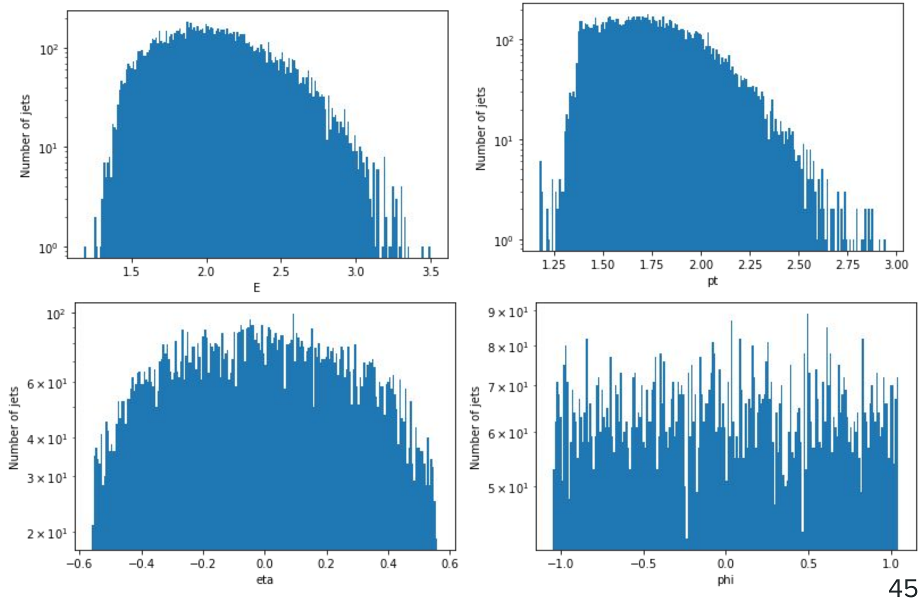
- LeakyReLU, BN, Custom-norm
- Latent space = 3
- Model - [4, 400, 400, 200, 3, 200, 400, 400, 4]
- Trained on njets_10fb dataset - containing only jets

`

Jets data with custom norm



atopbar data with custom norm



Test with atopbar_10fb data

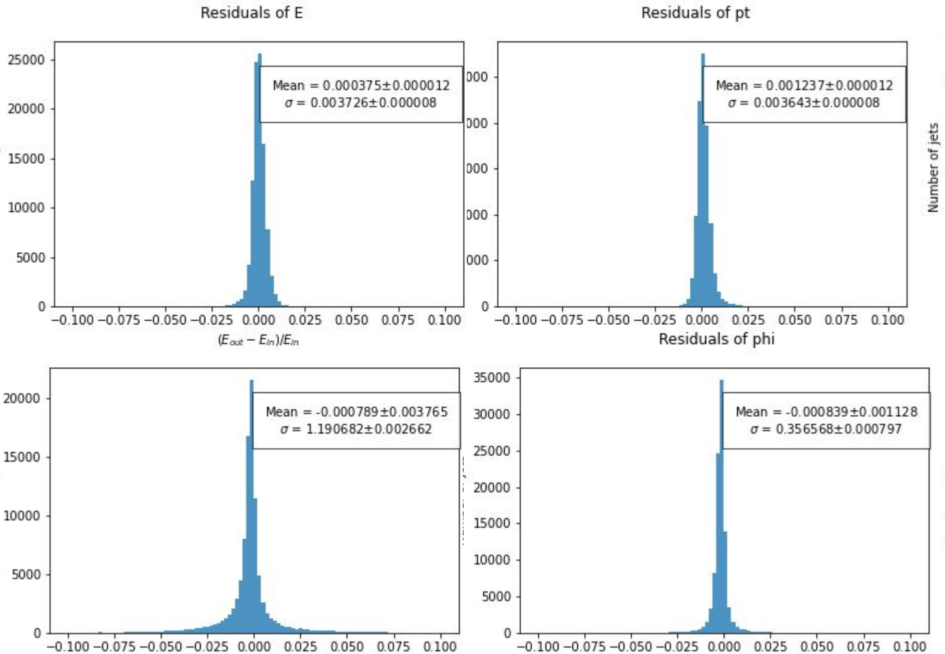
Model details:

- LeakyReLU, BN, Custom-norm
- Latent space = 3
- Model - [4, 400, 400, 200, 3, 200, 400, 400, 4]
- Trained on njets_10fb dataset - containing only jets

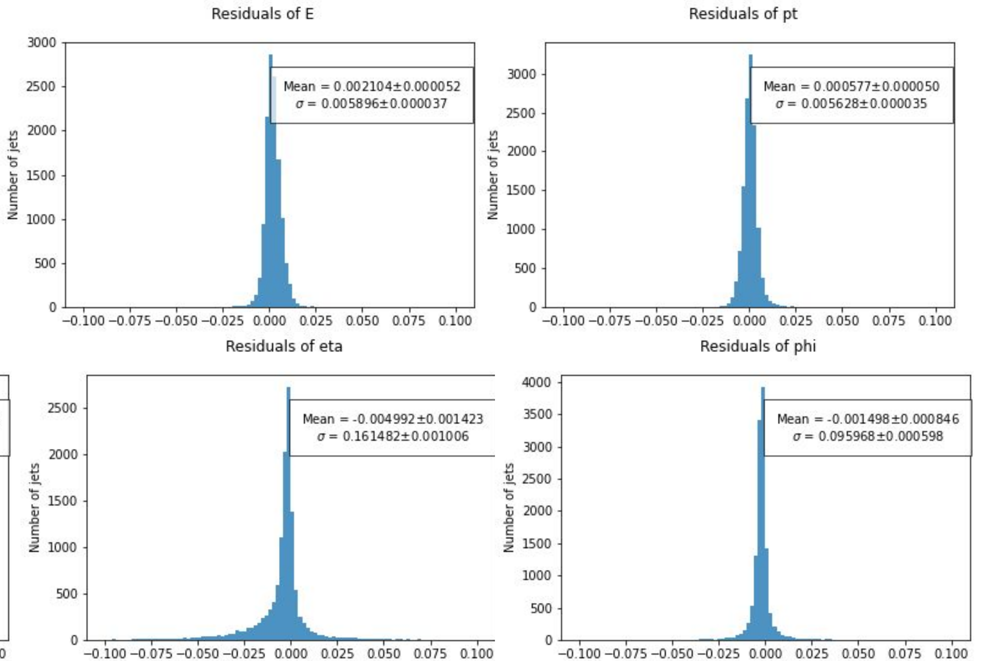
`

Normalization type	MSE on the test-set
atopbar	6.674753e-06
Jets	2.4856113e-06

Jets data with custom norm

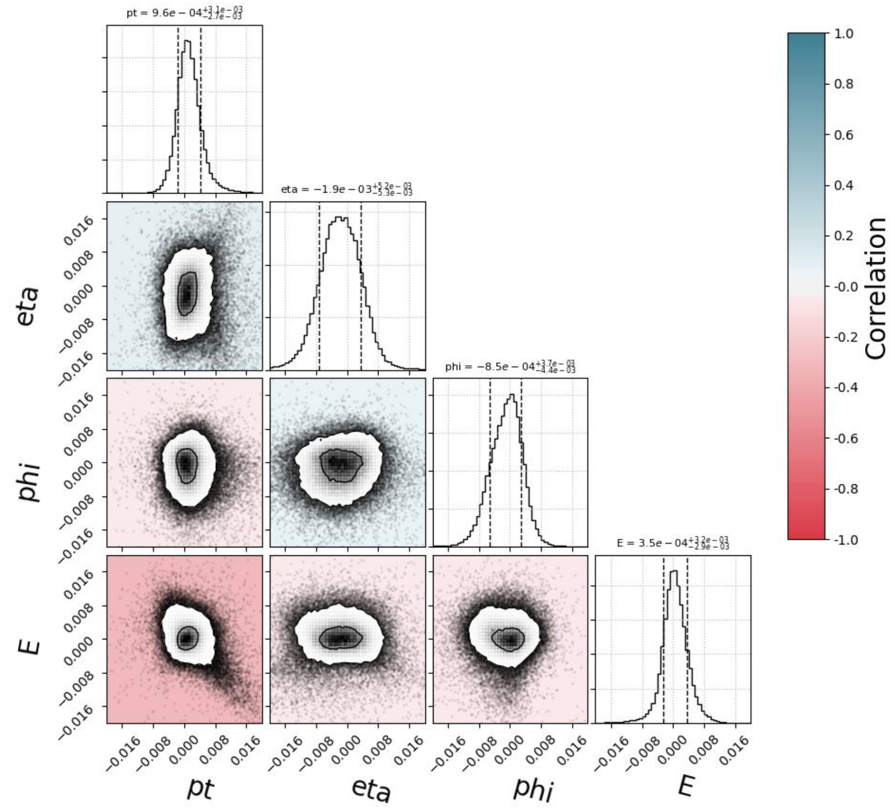


atopbar data with custom norm

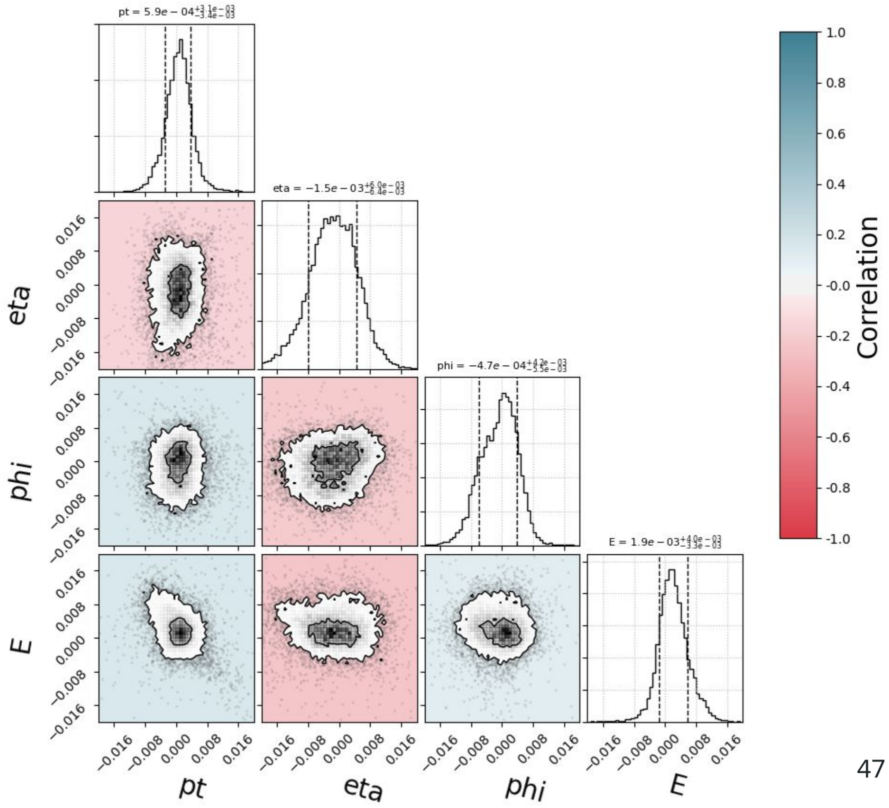


Test with atop_10fb data

Jets data with custom norm



atopbar data with custom norm



Test with ttbar_10fb data - events with only jets

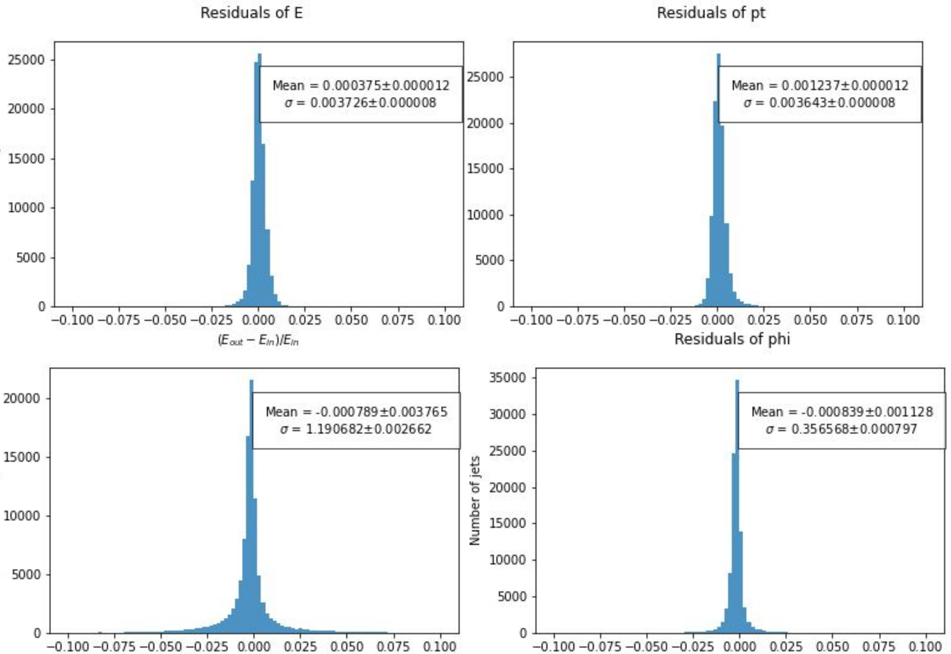
Model details:

- LeakyReLU, BN, Custom-norm
- Latent space = 3
- Model - [4, 400, 400, 200, 3, 200, 400, 400, 4]
- Trained on njets_10fb dataset - containing only jets

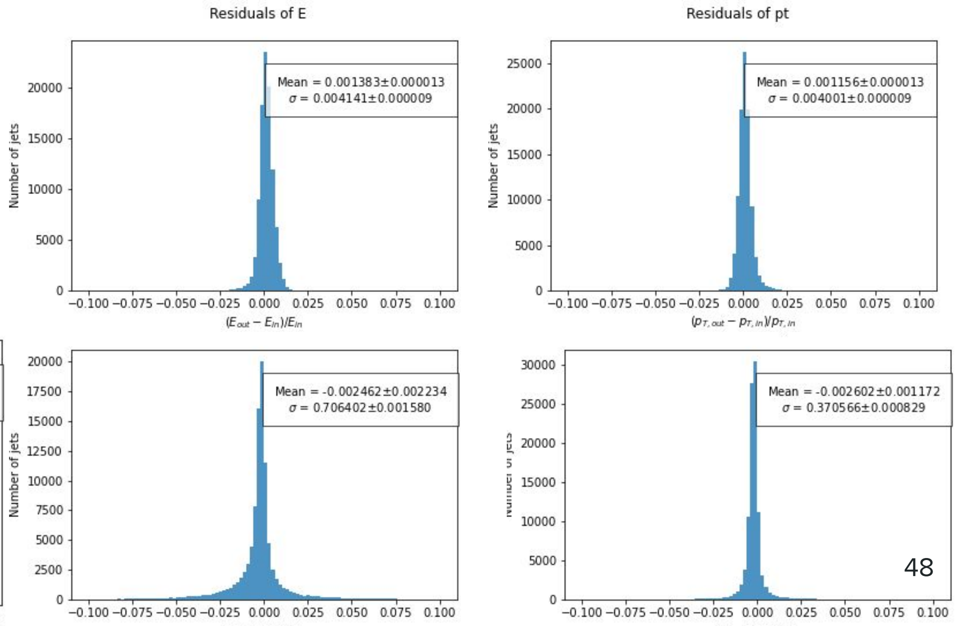
`

Normalization type	MSE on the test-set
ttbar	2.915699e-06
Jets	2.4856113e-06

Jets data with custom norm

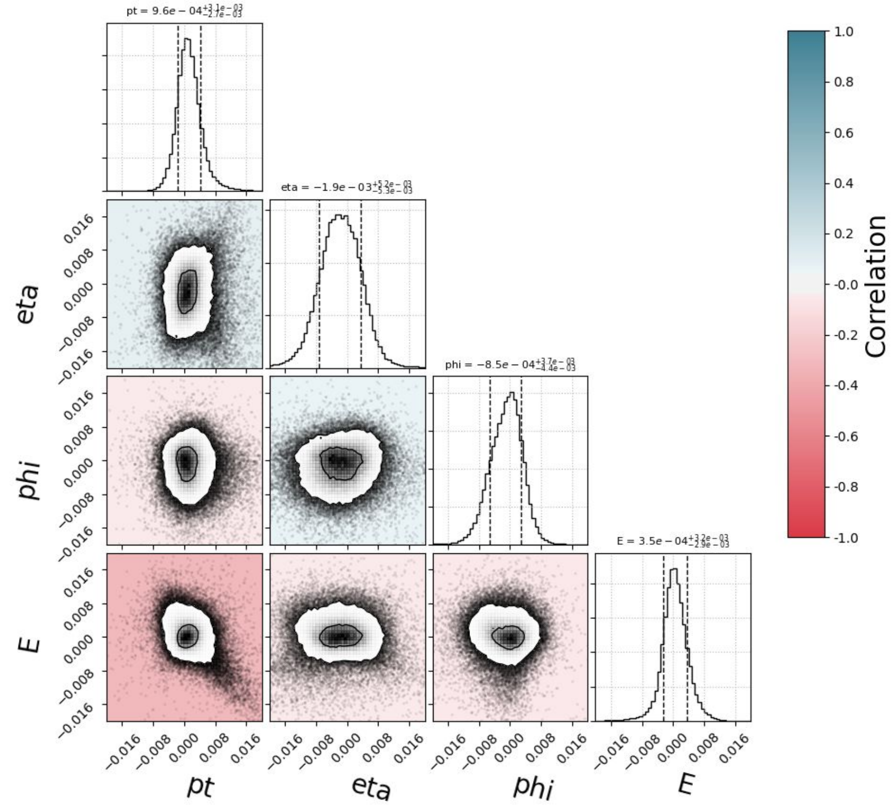


ttbar events with only jets data with custom norm

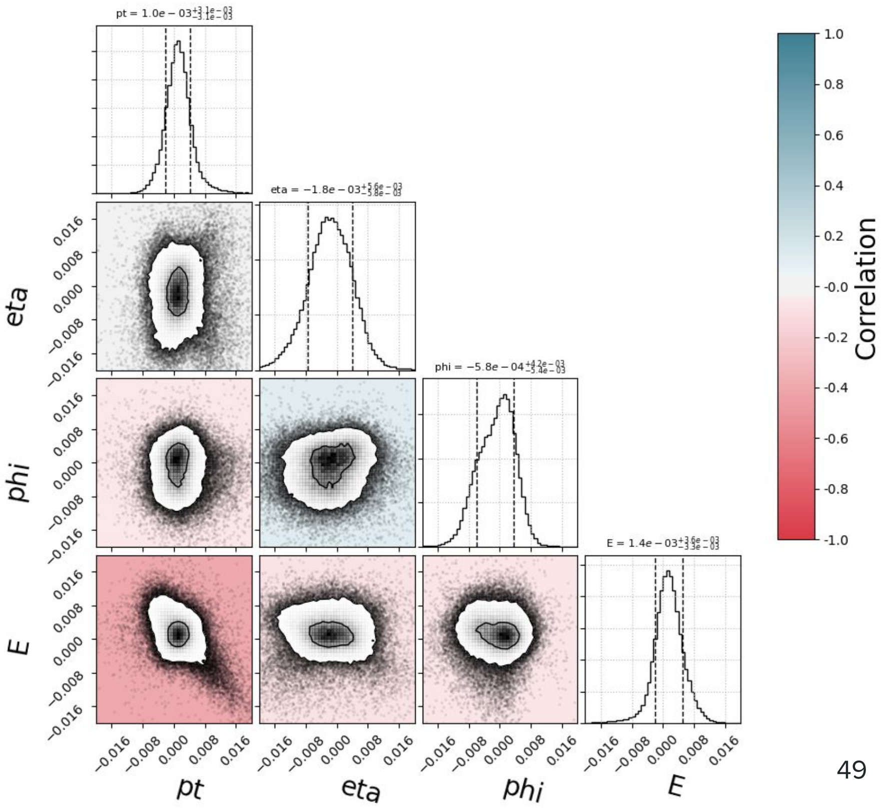


Test with ttbar_10fb data - events with only jets

Jets data with custom norm



ttbar events with only jets data with custom norm



Test with ttbar_10fb data - all events but only jet particles

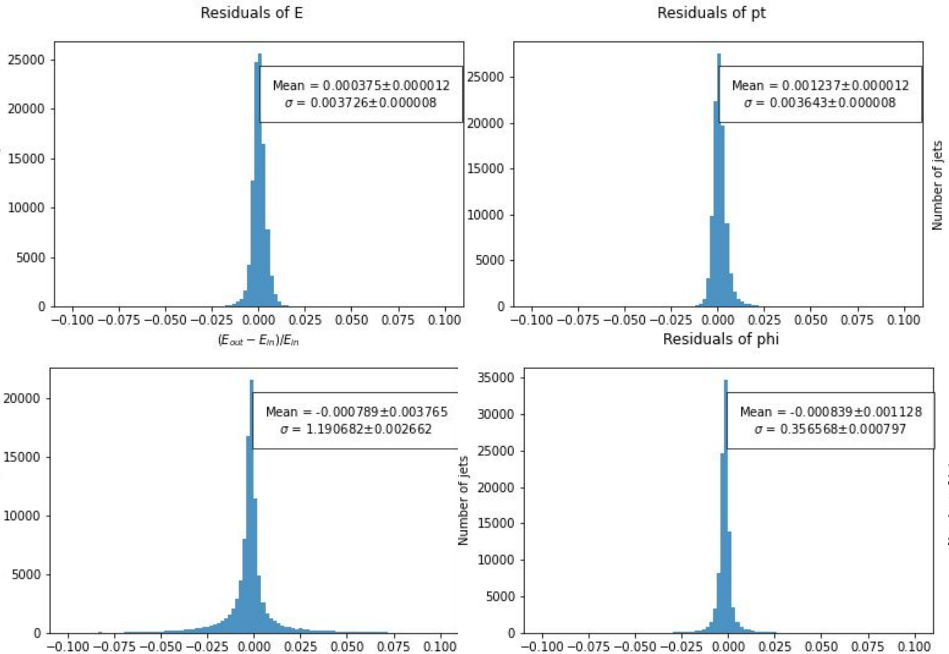
Model details:

- LeakyReLU, BN, Custom-norm
- Latent space = 3
- Model - [4, 400, 400, 200, 3, 200, 400, 400, 4]
- Trained on njets_10fb dataset - containing only jets

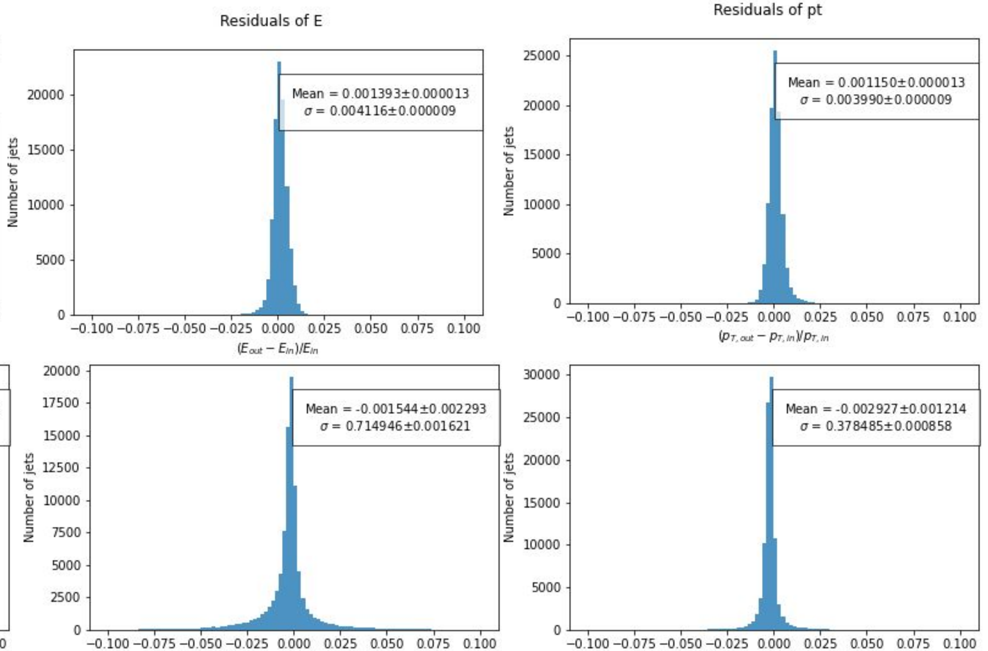
`

Normalization type	MSE on the test-set
ttbar	3.0117515e-06
Jets	2.4856113e-06

Jets data with custom norm

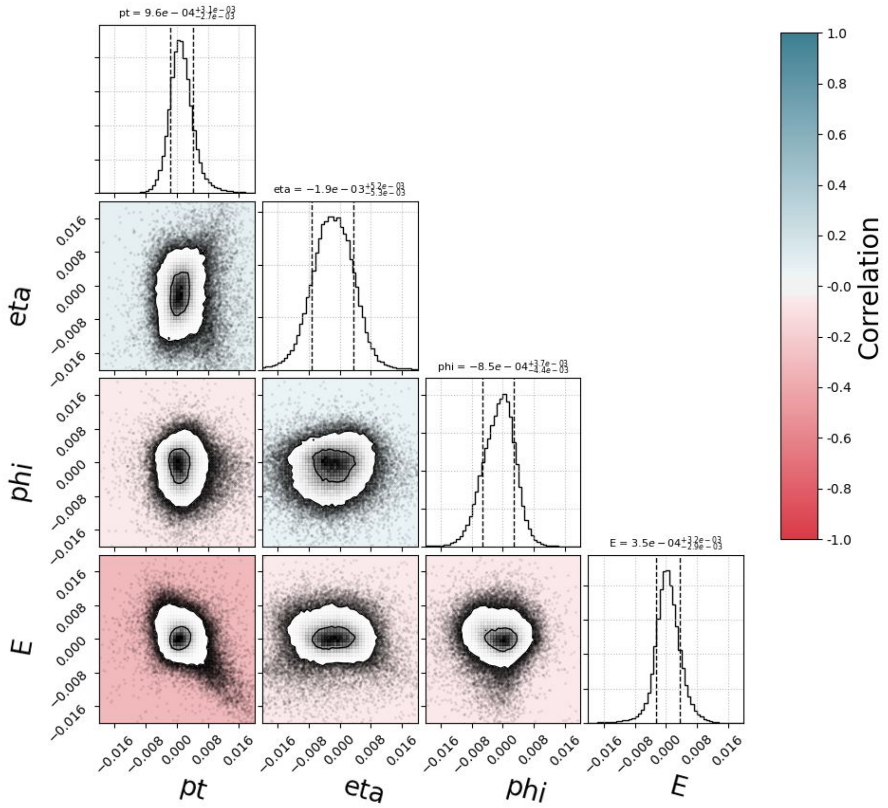


all events but only jet particles

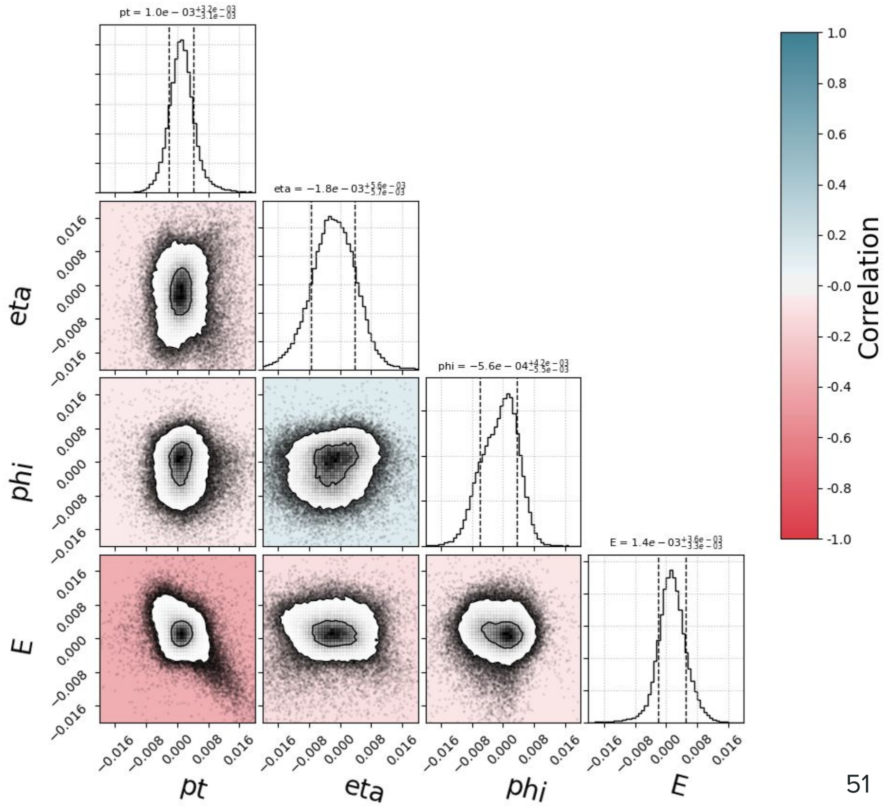


Test with ttbar_10fb data - all events but only jet particles

Jets data with custom norm



all events but only jet particles



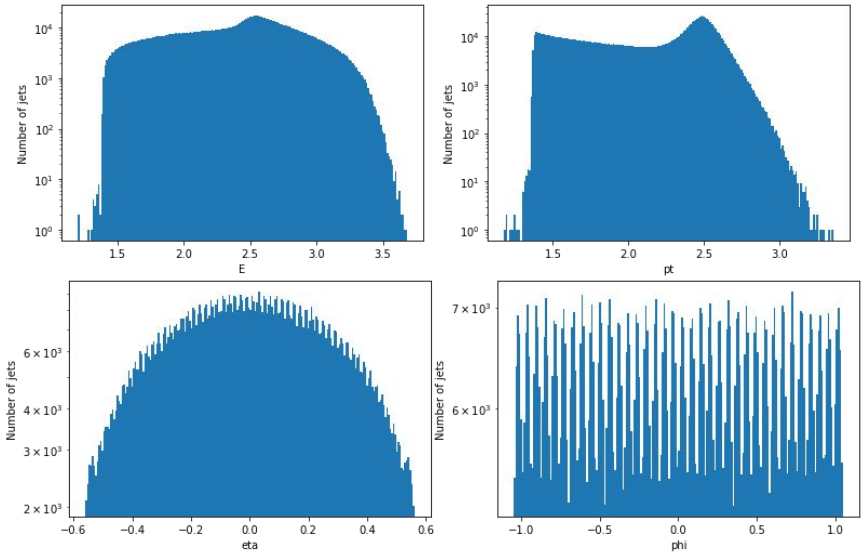
Test with ttbar_10fb data - all events

Model details:

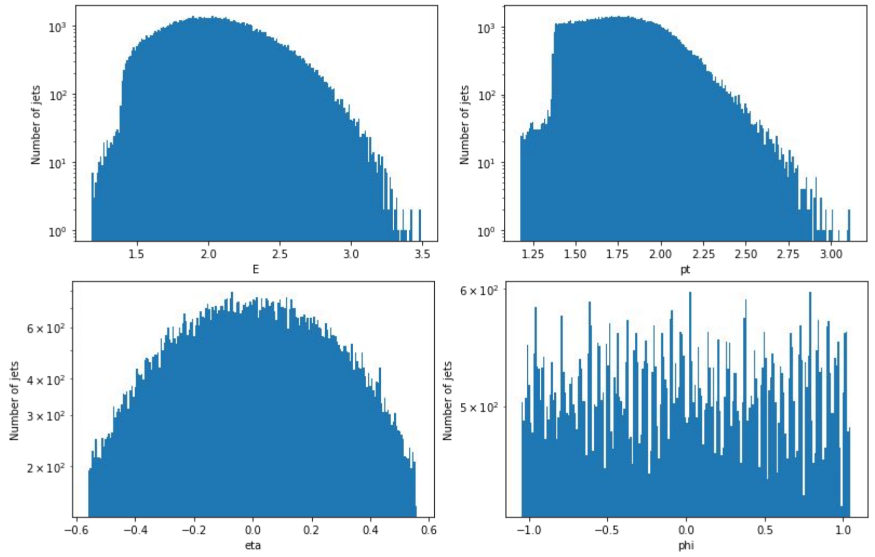
- LeakyReLU, BN, Custom-norm
- Latent space = 3
- Model - [4, 400, 400, 200, 3, 200, 400, 400, 4]
- Trained on njets_10fb dataset - containing only jets

`

Jets data with custom norm



ttbar data with custom norm



Test with atop_10fb data - all events

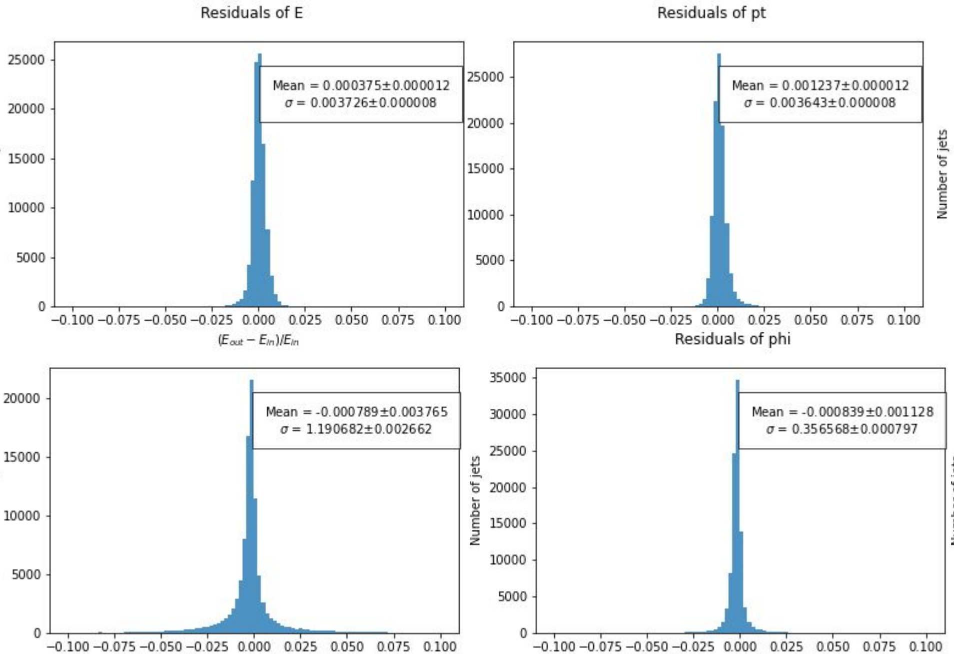
Model details:

- LeakyReLU, BN, Custom-norm
- Latent space = 3
- Model - [4, 400, 400, 200, 3, 200, 400, 400, 4]
- Trained on njets_10fb dataset - containing only jets

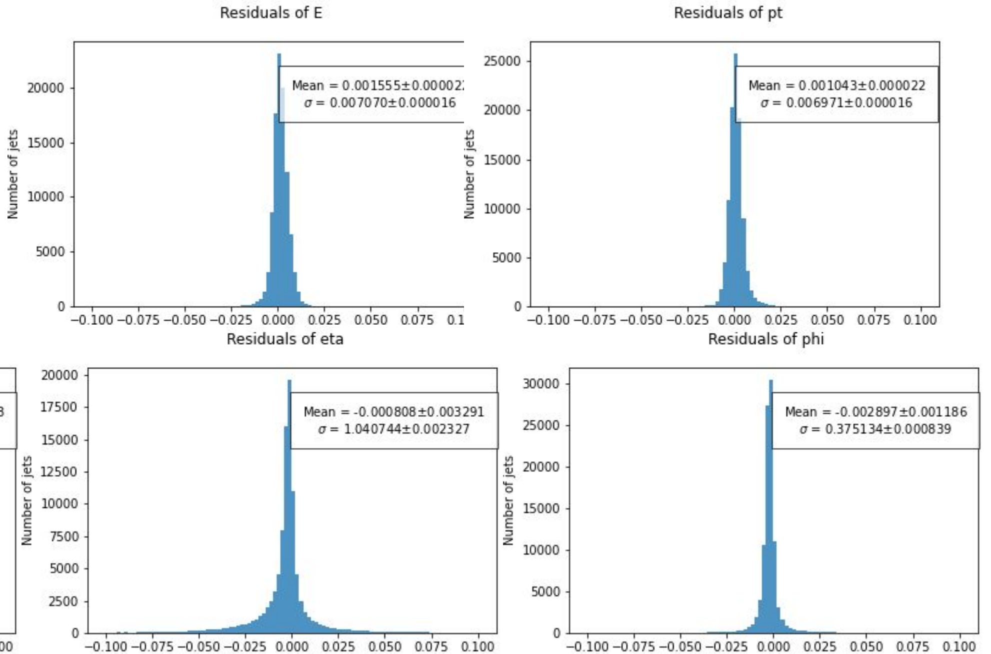
`

Normalization type	**MSE on the test-set**
ttbar	8.184939e-06
Jets	2.4856113e-06

Jets data with custom norm

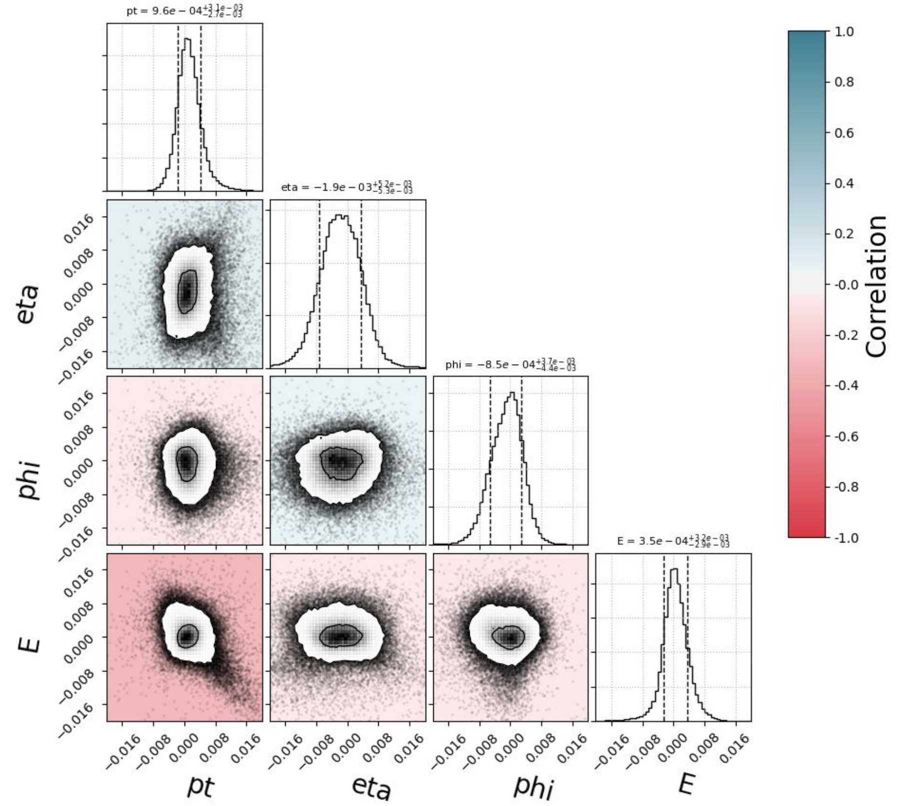


ttbar data with custom norm

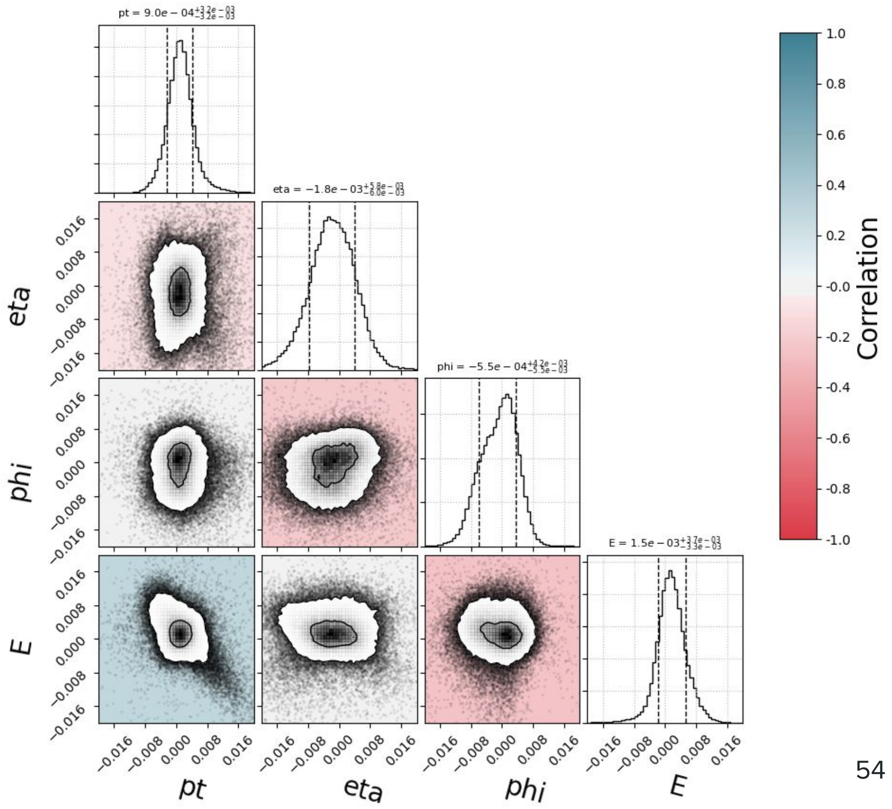


Test with ttbar_10fb data - all events

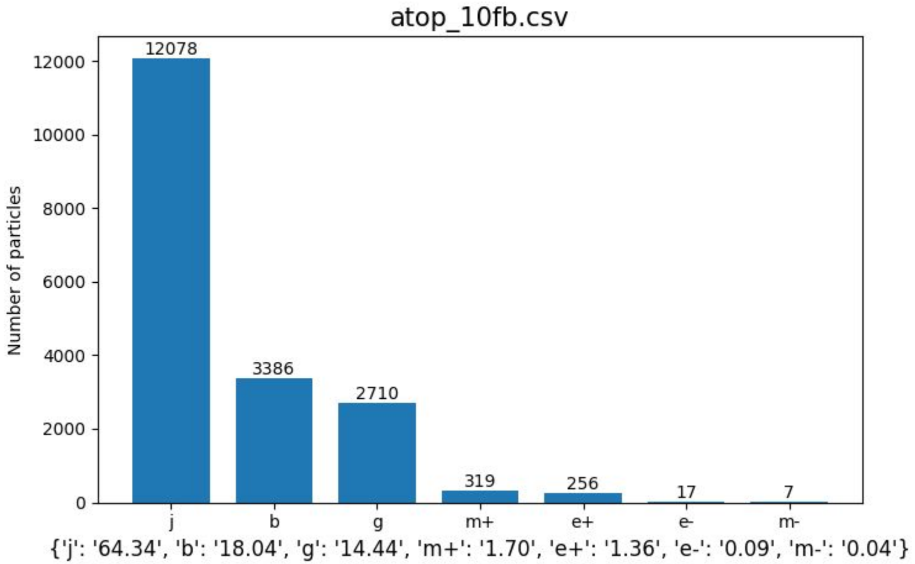
Jets data with custom norm



ttbar data with custom norm

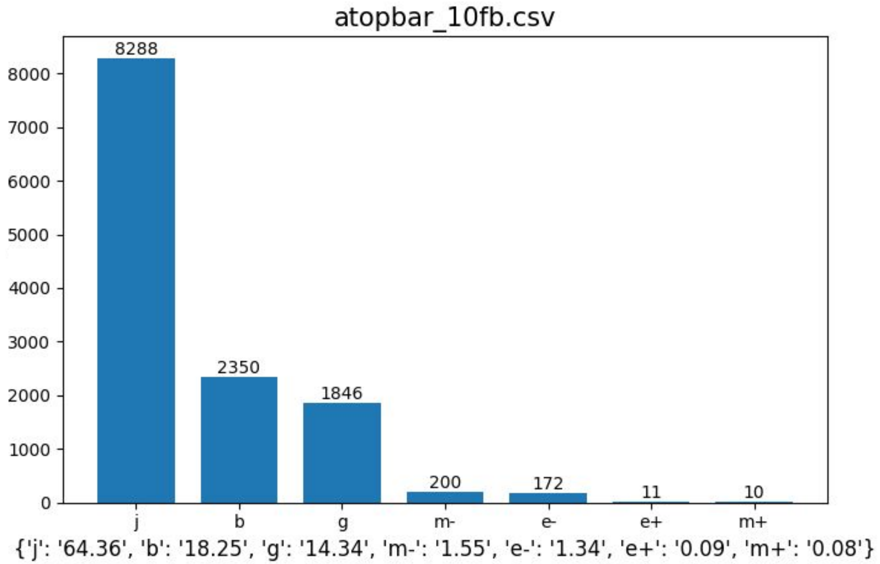


Particle distribution plot for different processes in phenoML dataset - SM



{'j': '64.34', 'b': '18.04', 'g': '14.44', 'm+': '1.70', 'e+': '1.36', 'e-': '0.09', 'm-': '0.04'}

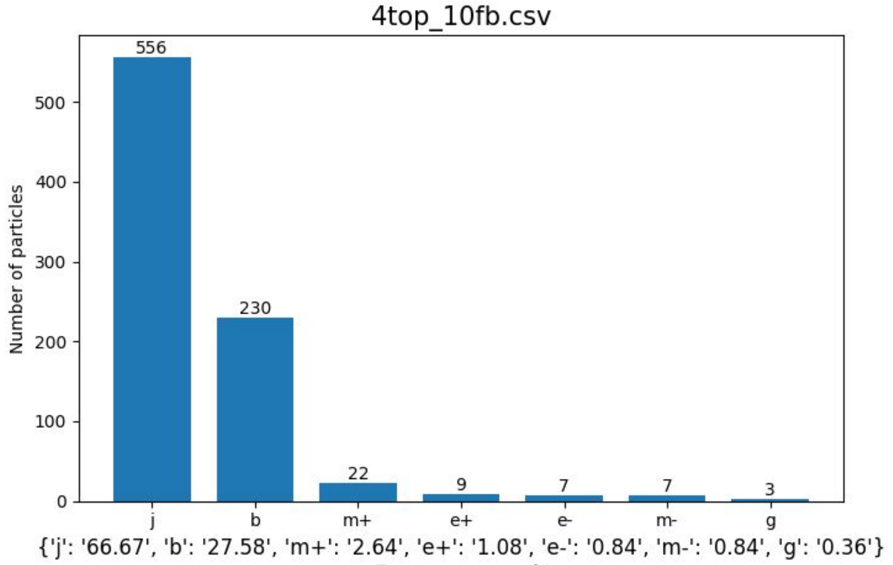
Percentage %



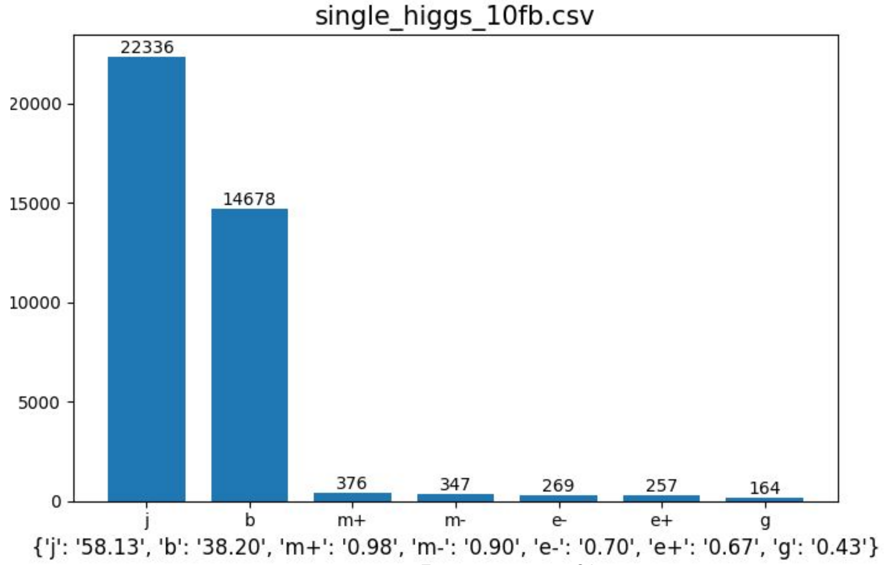
{'j': '64.36', 'b': '18.25', 'g': '14.34', 'm-': '1.55', 'e-': '1.34', 'e+': '0.09', 'm+': '0.08'}

Percentage %

Particle distribution plot for different processes in phenoML dataset - SM

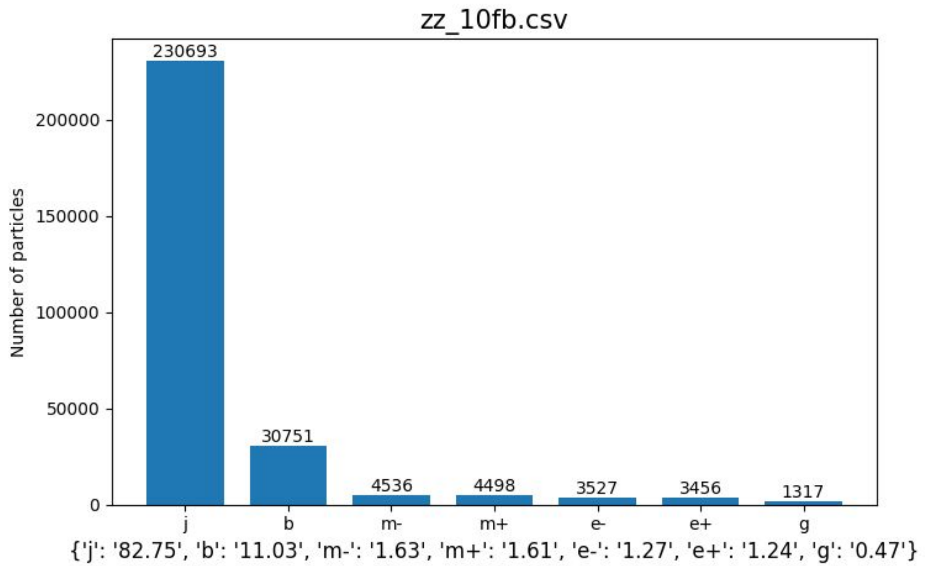


{'j': '66.67', 'b': '27.58', 'm+': '2.64', 'e+': '1.08', 'e-': '0.84', 'm-': '0.84', 'g': '0.36'}
Percentage %



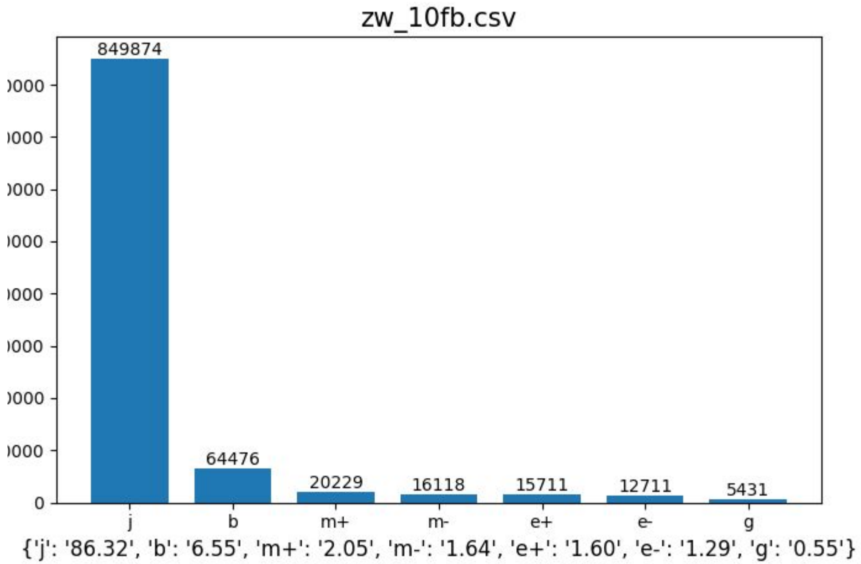
{'j': '58.13', 'b': '38.20', 'm+': '0.98', 'm-': '0.90', 'e-': '0.70', 'e+': '0.67', 'g': '0.43'}
Percentage %

Particle distribution plot for different processes in phenoML dataset - SM



{'j': '82.75', 'b': '11.03', 'm-': '1.63', 'm+': '1.61', 'e-': '1.27', 'e+': '1.24', 'g': '0.47'}

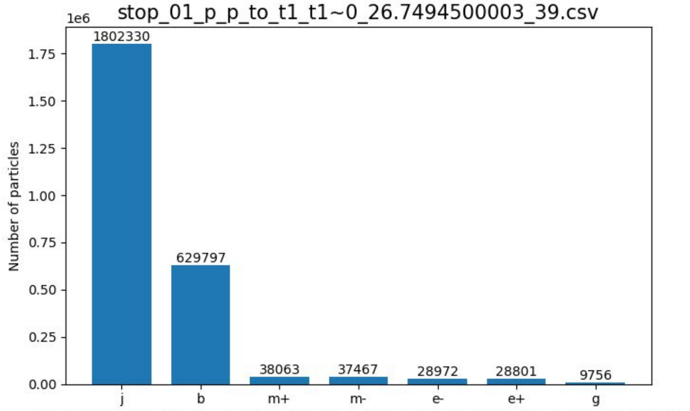
Percentage %



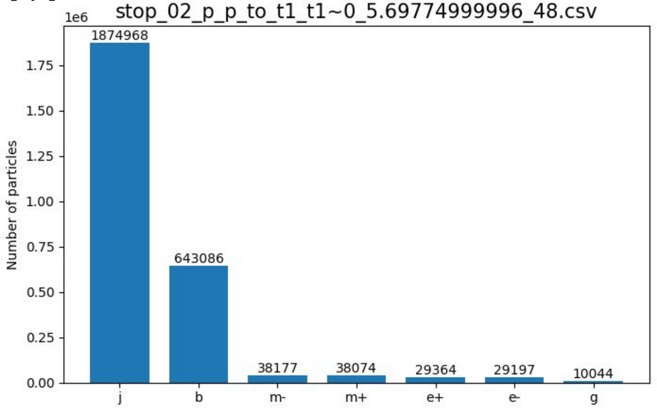
{'j': '86.32', 'b': '6.55', 'm+': '2.05', 'm-': '1.64', 'e+': '1.60', 'e-': '1.29', 'g': '0.55'}

Percentage %

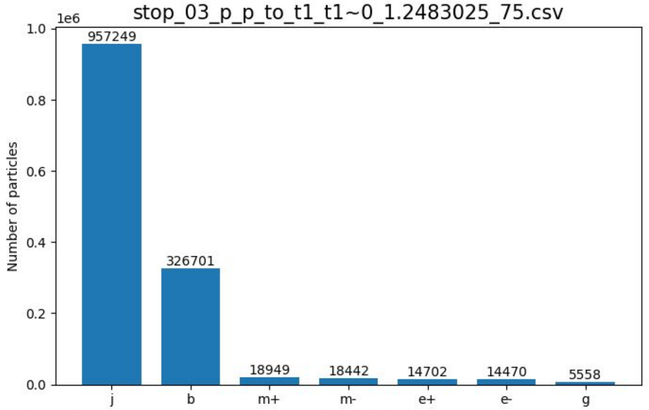
Particle distribution plot for different processes in phenoML dataset - BSM



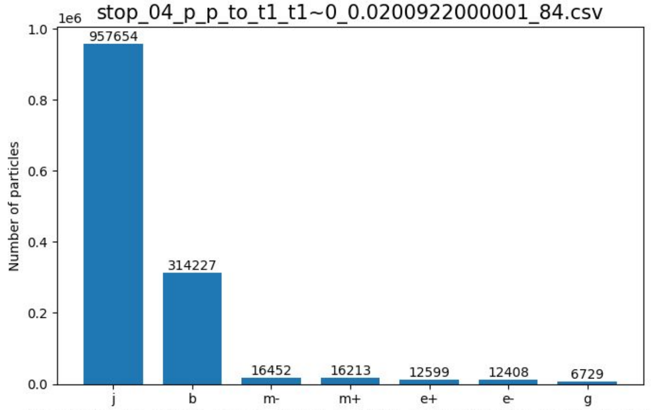
{'j': '69.99', 'b': '24.46', 'm+': '1.48', 'm-': '1.45', 'e-': '1.13', 'e+': '1.12', 'g': '0.38'}



{'j': '70.41', 'b': '24.15', 'm-': '1.43', 'm+': '1.43', 'e+': '1.10', 'e-': '1.10', 'g': '0.38'}

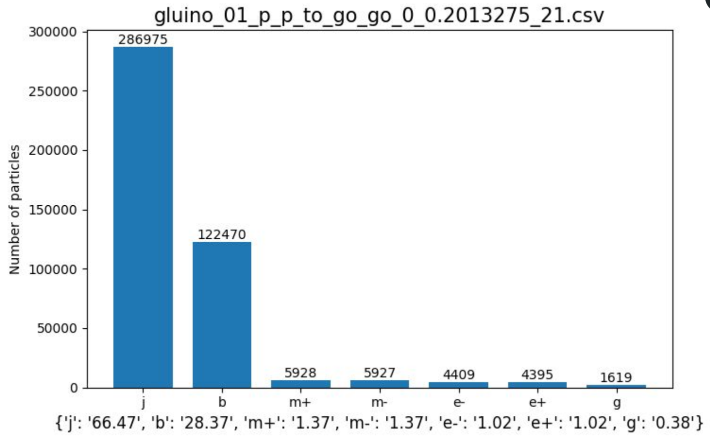


{'j': '70.59', 'b': '24.09', 'm+': '1.40', 'm-': '1.36', 'e+': '1.08', 'e-': '1.07', 'g': '0.41'}

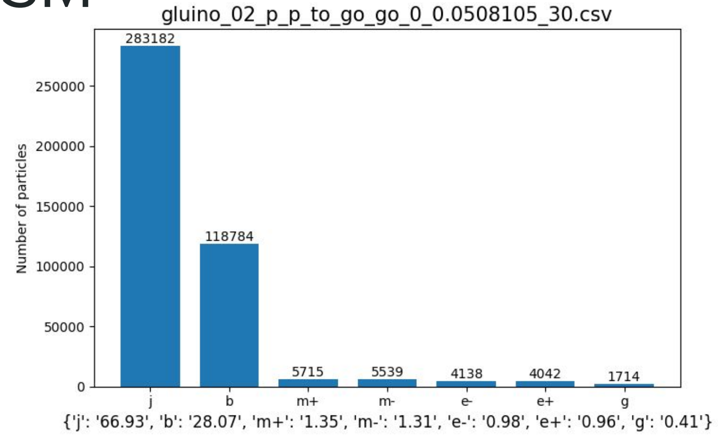


{'j': '71.67', 'b': '23.52', 'm-': '1.23', 'm+': '1.21', 'e+': '0.94', 'e-': '0.93', 'g': '0.50'}

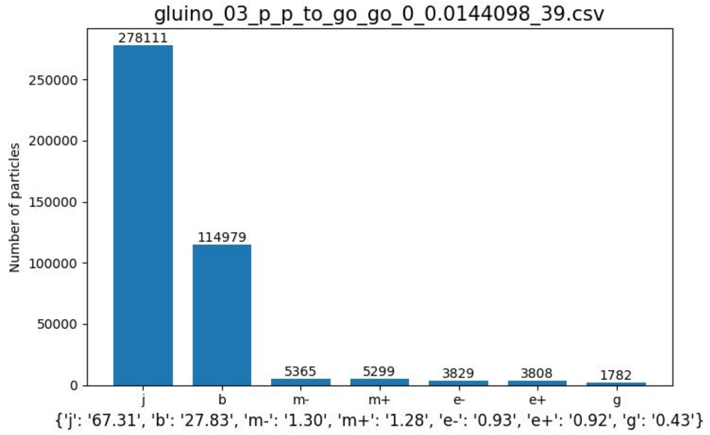
Particle distribution plot for different processes in phenoML dataset - BSM



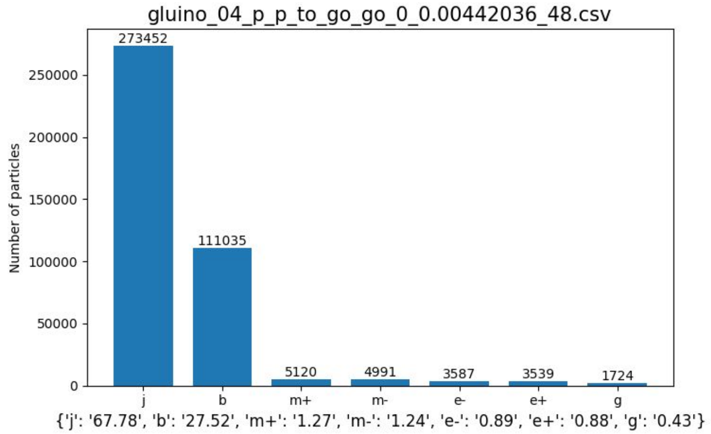
{'j': '66.47', 'b': '28.37', 'm+': '1.37', 'm-': '1.37', 'e-': '1.02', 'e+': '1.02', 'g': '0.38'}



{'j': '66.93', 'b': '28.07', 'm+': '1.35', 'm-': '1.31', 'e-': '0.98', 'e+': '0.96', 'g': '0.41'}

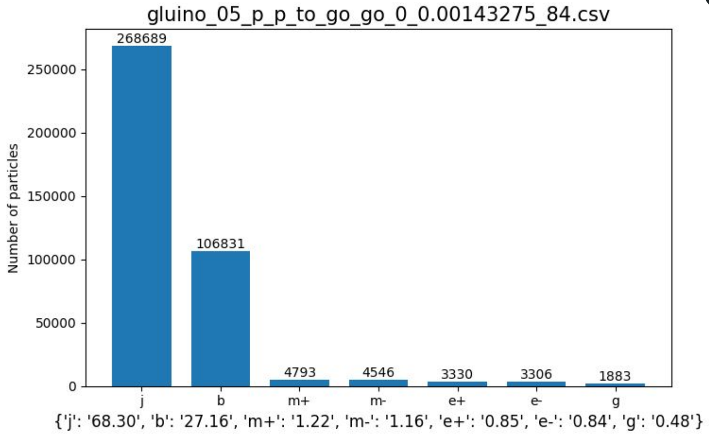


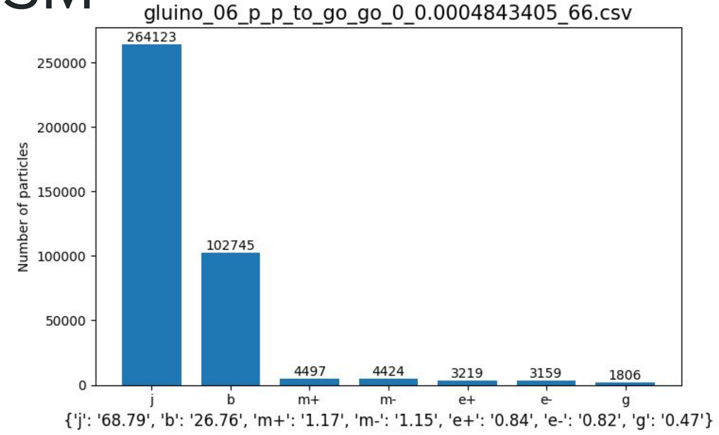
{'j': '67.31', 'b': '27.83', 'm-': '1.30', 'm+': '1.28', 'e-': '0.93', 'e+': '0.92', 'g': '0.43'}

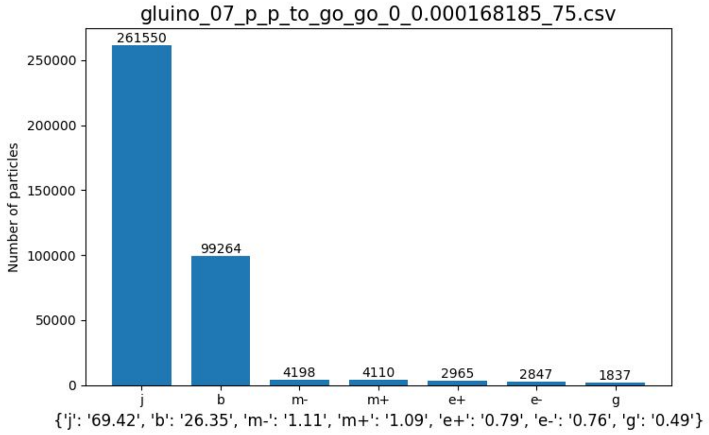


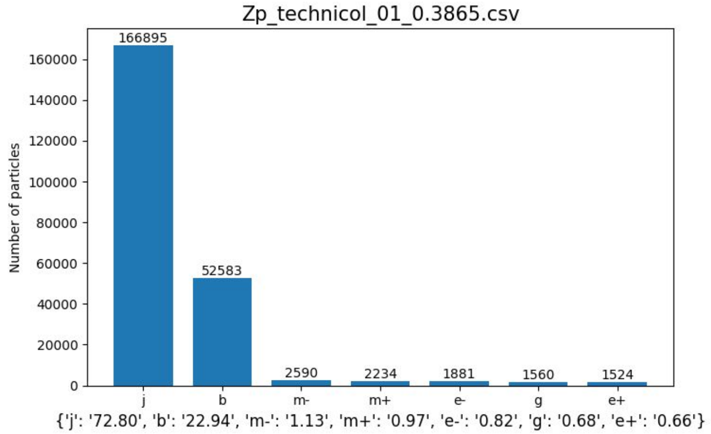
{'j': '67.78', 'b': '27.52', 'm+': '1.27', 'm-': '1.24', 'e-': '0.89', 'e+': '0.88', 'g': '0.43'}

Particle distribution plot for different processes in phenoML dataset - BSM

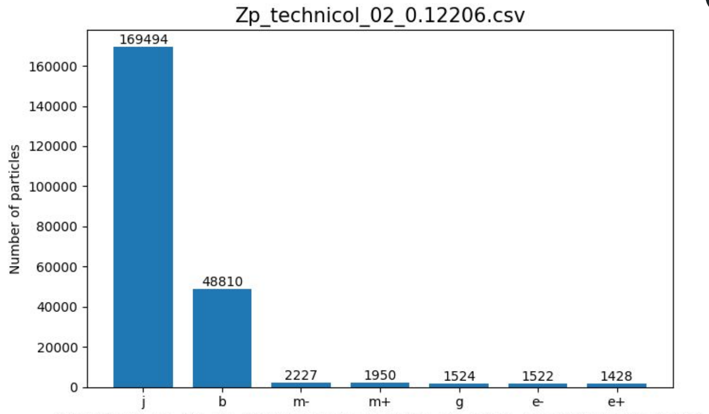




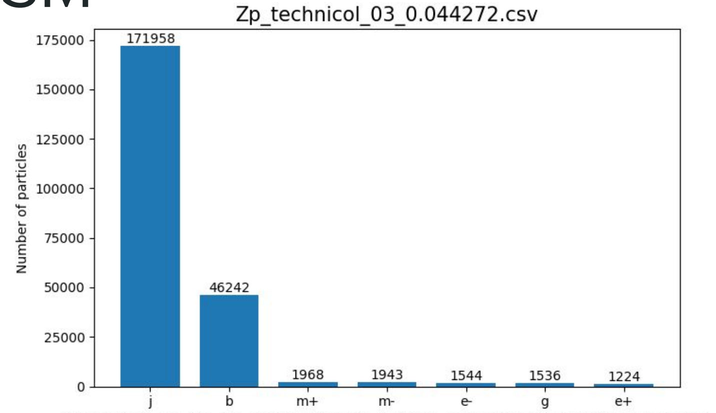




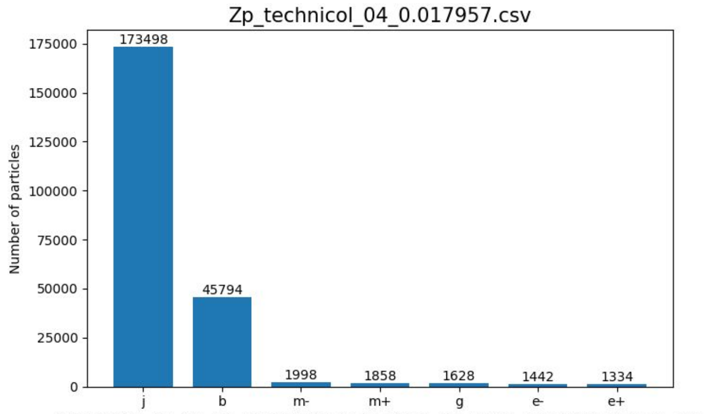
Particle distribution plot for different processes in phenoML dataset - BSM



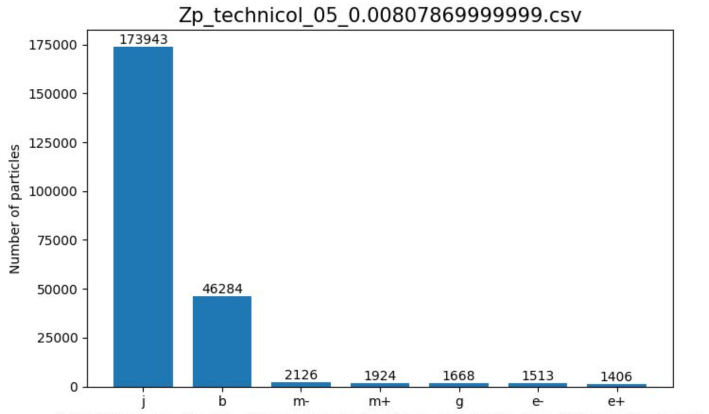
{'j': '74.68', 'b': '21.51', 'm-': '0.98', 'm+': '0.86', 'g': '0.67', 'e-': '0.67', 'e+': '0.63'}



{'j': '75.95', 'b': '20.42', 'm+': '0.87', 'm-': '0.86', 'e-': '0.68', 'g': '0.68', 'e+': '0.54'}



{'j': '76.25', 'b': '20.12', 'm-': '0.88', 'm+': '0.82', 'g': '0.72', 'e-': '0.63', 'e+': '0.59'}



{'j': '76.00', 'b': '20.22', 'm-': '0.93', 'm+': '0.84', 'g': '0.73', 'e-': '0.66', 'e+': '0.61'}

Testing a Jets-trained-model on different particles

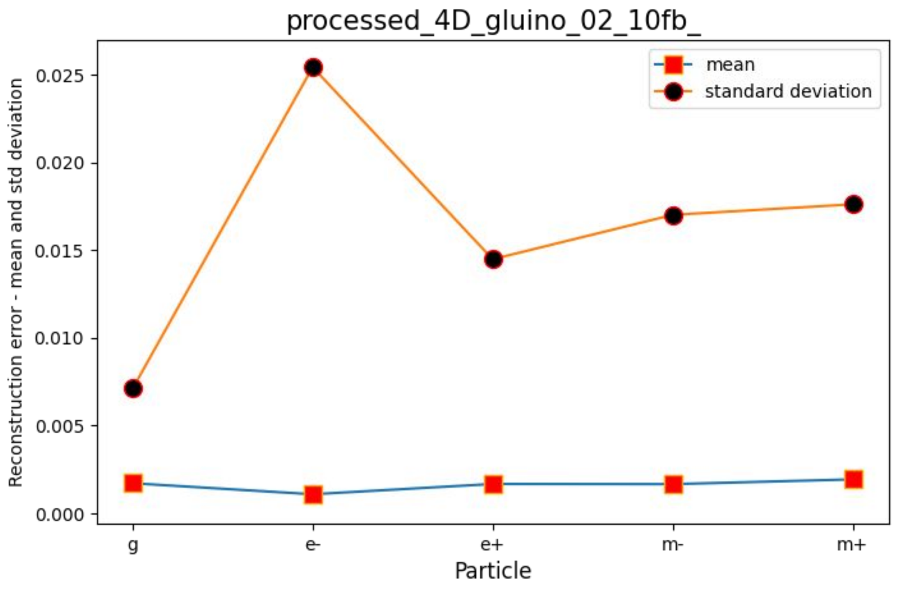
Model details:

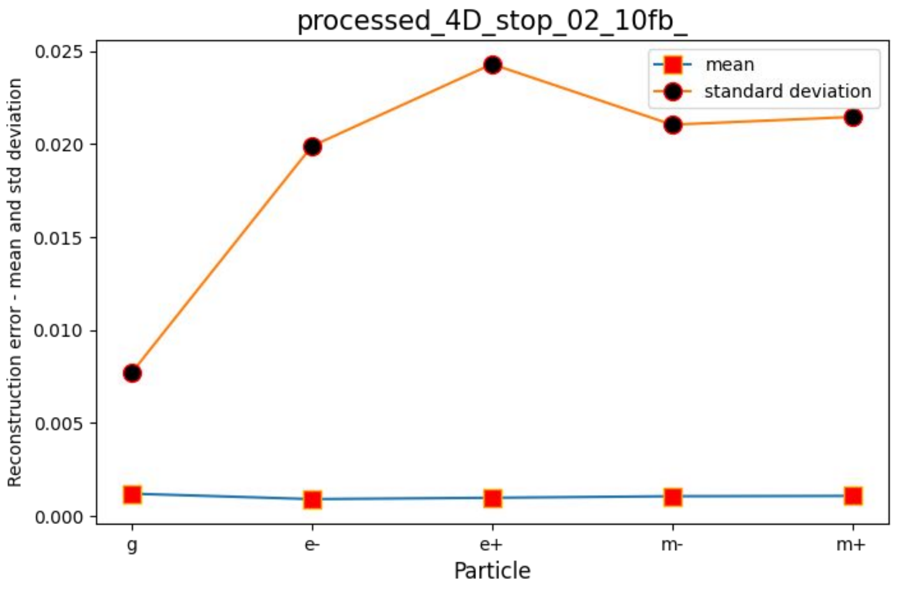
- LeakyReLU, BN, Custom-norm
- Latent space = 3
- Model - [4, 400, 400, 200, 3, 200, 400, 400, 4]
- Trained on njets_10fb dataset - containing only jets

`

Process	**e+**	**e-**	**m+**	**m-**	**\gamma**
stop_02	8.737900e-05	8.628796e-05	9.500347e-05	0.0001134310	1.1838656e-05
gluino_02	6.45536e-05	5.087386e-05	9.127547e-05	7.675688e-05	1.2573055e-05

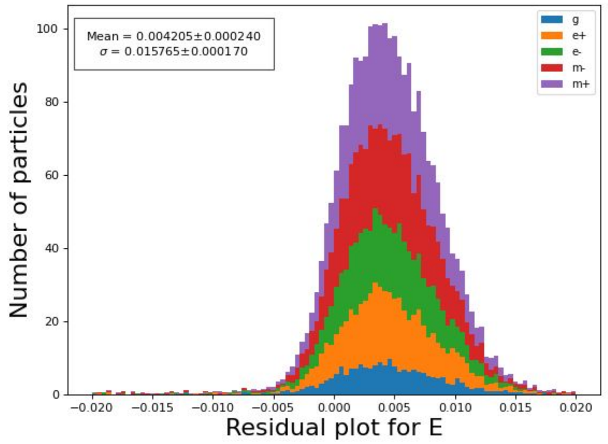
Testing a Jets-trained-model on different particles

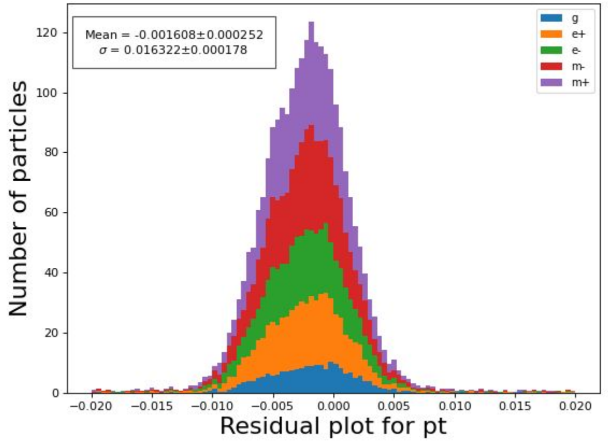




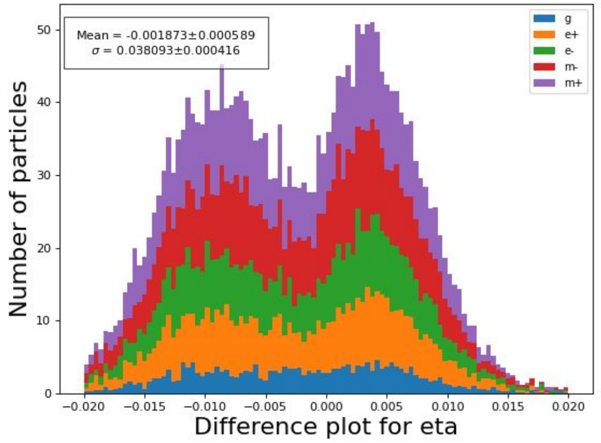
Mean and std-dev for the residuals of pt

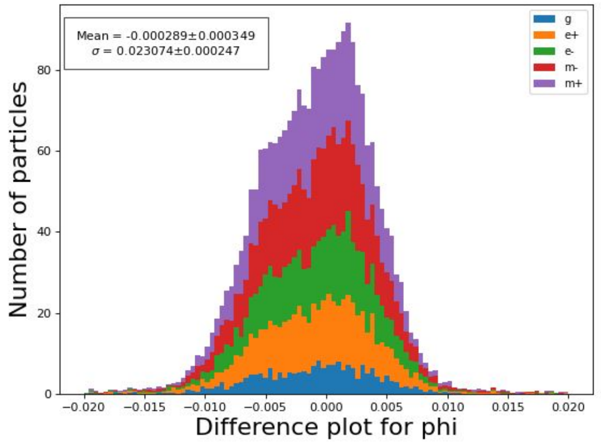
Testing a Jets-trained-model on different particles



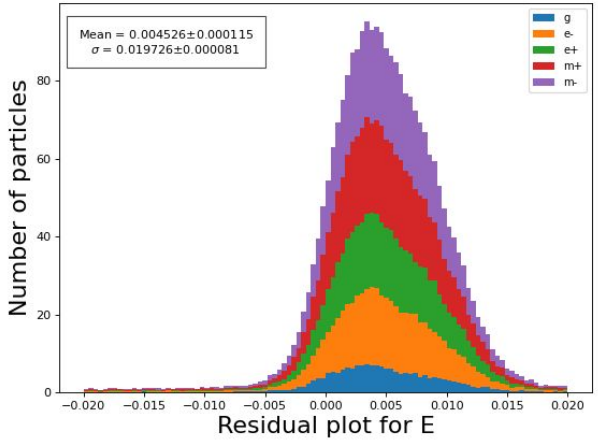


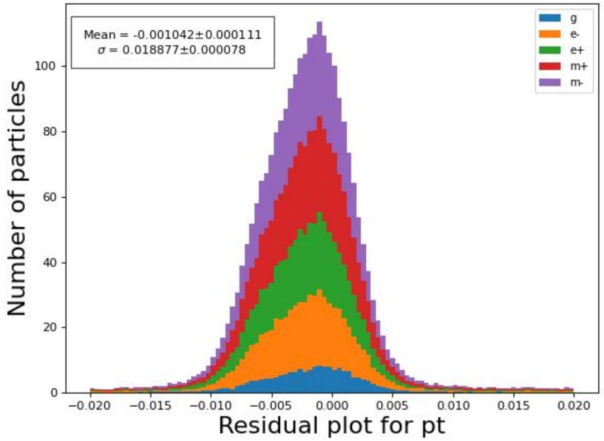
gluino_02



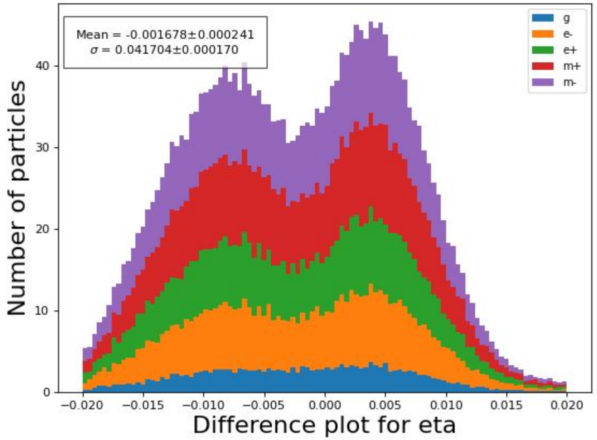


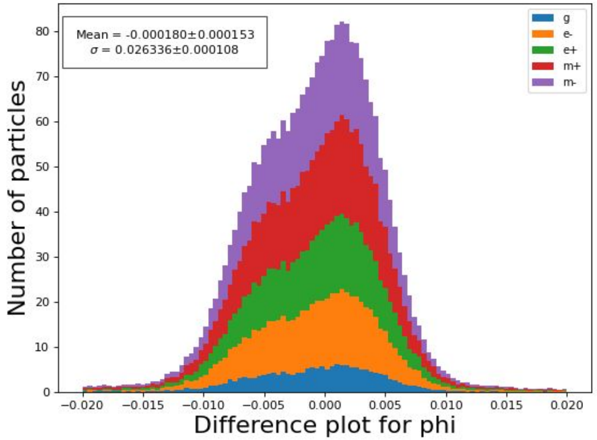
Testing a Jets-trained-model on different particles



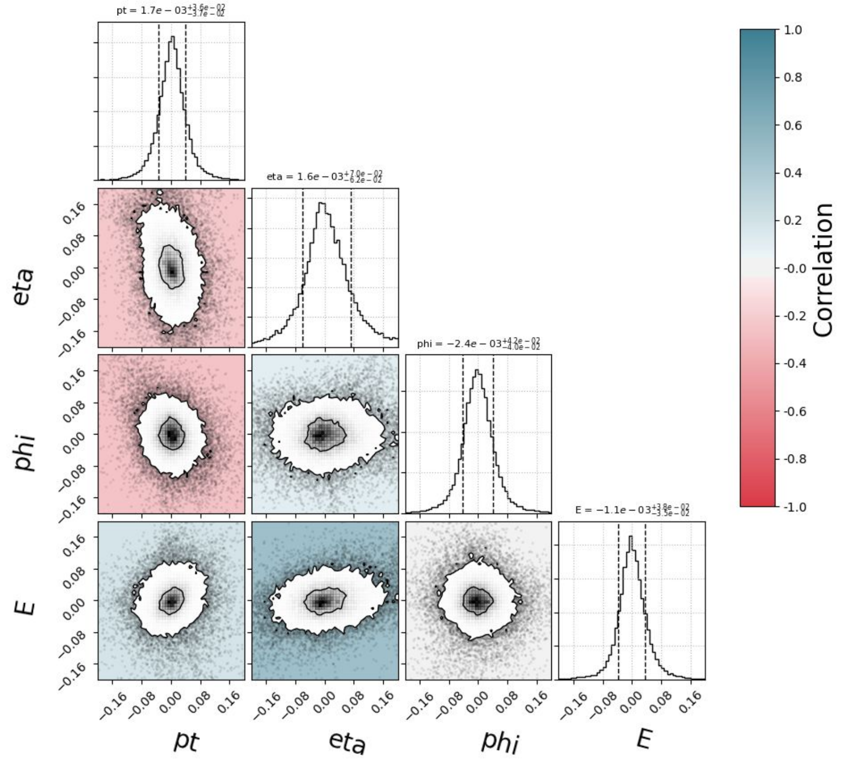


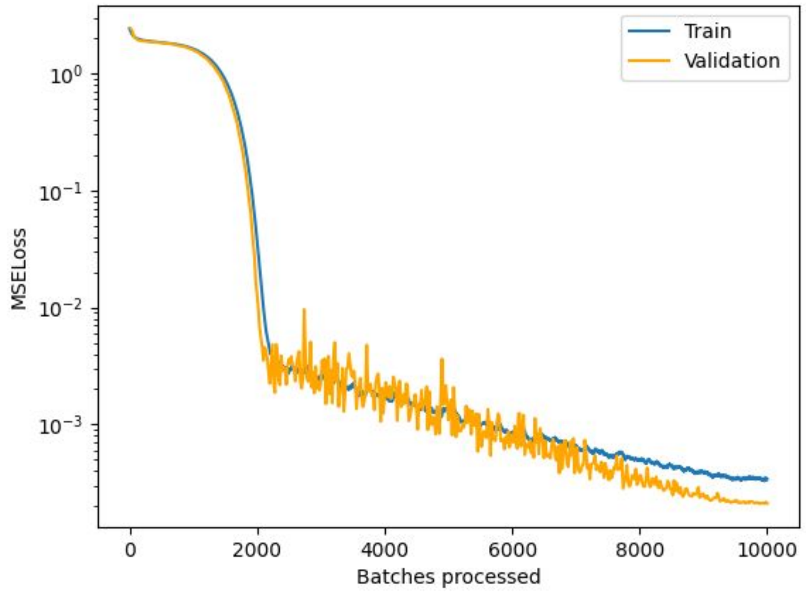
stop_02





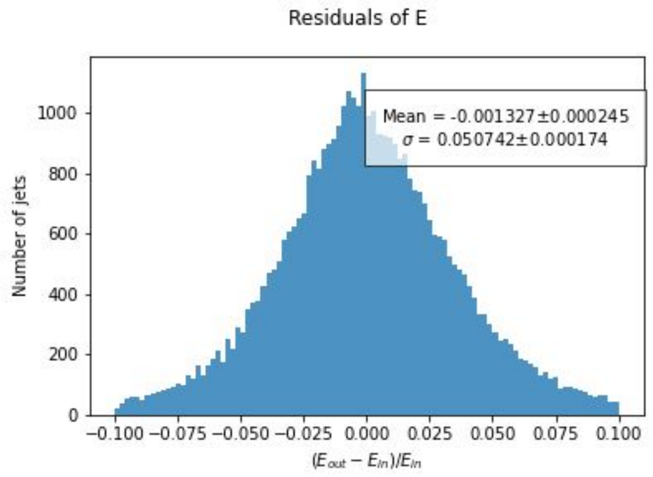
Testing a chan2a-trained-model on chan3 of DarkMachines data

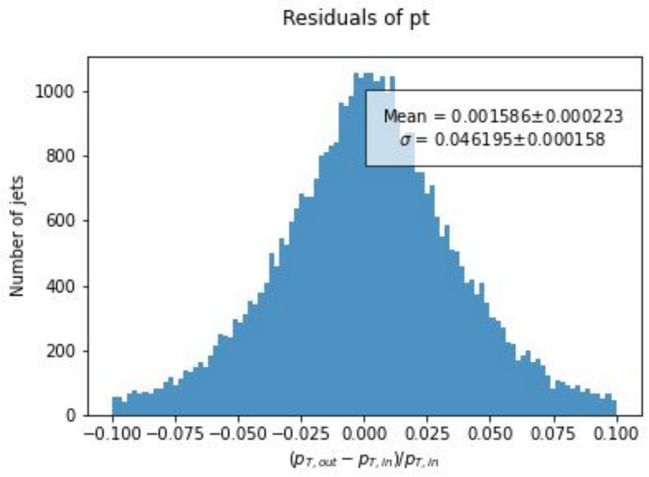


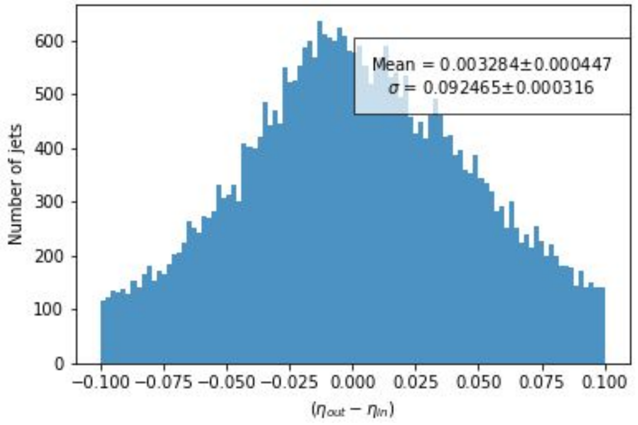


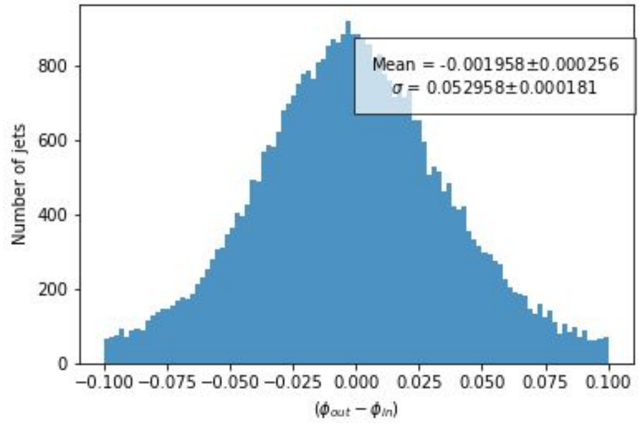
Model trained on chan2a doesn't converge that well, and performs moderately on jets

Testing a chan2a-trained-model on chan3 of DarkMachines data









Detailed summary

- Fixed the GitHub repo by adding missing references to the files
    - Commit links: [a558e70, 8f1253d, 0faabf1]
- Extracted data from ROOT files and created binary (pickle) files for processing
- Plotted the data distribution to validate that the working data is correct
    - Link to the plots
- Created plots from the pre-trained model available to compare and validate the published results (thesis of Eric Wulff)
    - Link to the plots
- Understood the functioning of HTCondor
- Wrote documents (starting-guide) to work on HTCondor
    - Link to the doc

Detailed summary (cont.)

- Created scripts for
  - Processing and saving the ROOT files [0531239]
  - Scaling the data by fitting scaling models for each data [06161dc]
  - Custom normalising the data [a93afeb]

  on HTCondor
- Modified and created training scripts for 27D data to run on the GPUs available at HTCondor
  - [f169d01, ef03a0e]
- Checked the training strategy for the 27D model
  - Number of epochs or stopping point was not mentioned anywhere
  - Found that an MSE loss around 1e-6 is sufficient for stopping
  - For custom-normalized data, it corresponds to around 500 epochs @ LR 1e-4

Detailed summary (cont.)

- Trained the model for 27D - [27, 400, 400, 200, 20, 200, 400, 400, 27]
    - scaled data - (link to the training summary and models)
    - custom normalized data - (link to the training summary and models)
- Checked the training time for the models
    - On an average - 3.12 minutes per epoch
    - For 500ep - 25:35 hours on a GPU @ batch-size of 8192
- Created plots for the trained models
    - Scaled data - (link to the plots)
    - Custom-normalized data - (link to the plots)
- Compared L1 with MSE loss for 4D data
    - Link to the plots