Muon momentum estimation in scattering tomography

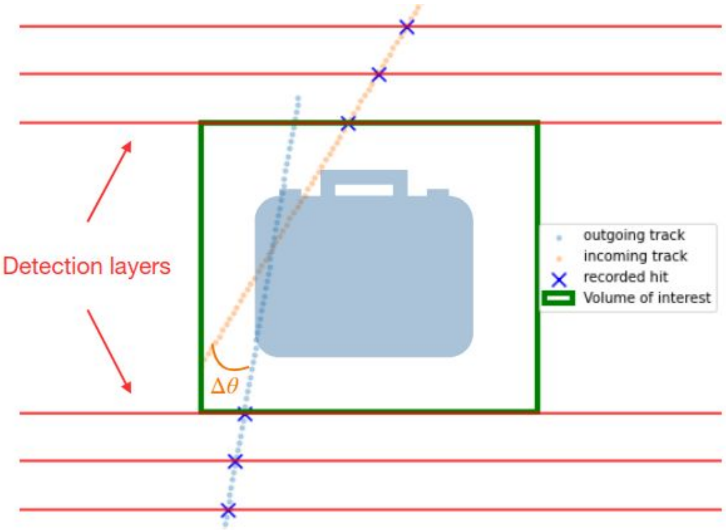
Fourth MODE Workshop on Differentiable Programming for Experiment Design

23-25 September 2024 Valencia

Florian Bury, Maxime Lagrange, for the TomOpt team

Thank you for being here at 9am !!!

Scattering tomography



For a given :

- Material with $X_0(A,\rho)$
- Momentum p

The deflection angle is a centred Gaussian with

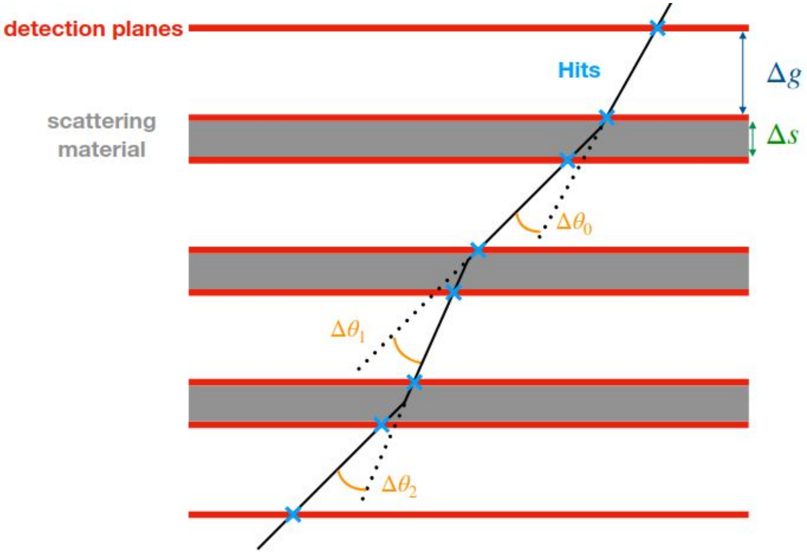
$$\sigma_\theta = \frac{13.6MeV}{p}\sqrt{\frac{x}{X_0}}$$

Measuring track deflection gives access to X_0

Measurement method : $\sigma_\theta \sim \theta_{RMS} = \frac{1}{N}\sum_i^N \Delta\theta_i^2$

Example : yesterday's great talk by Zahraa !

Scattering tomography



For a given :

- Material with $X_0(A,\rho)$
- Momentum p

The deflection angle is a centred Gaussian with

$$\sigma_\theta = \frac{13.6MeV}{p}\sqrt{\frac{x}{X_0}}$$

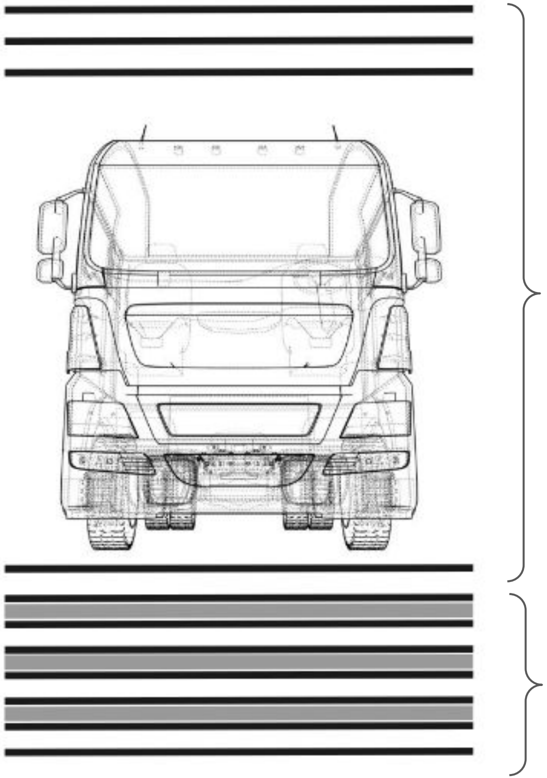
Measuring track deflection gives access to X_0

Measurement method : $\sigma_\theta \sim \theta_{RMS} = \frac{1}{N}\sum_i^N \Delta\theta_i^2$

… but this assumes knowledge of p

Possible solution : inverse the formula $p = \frac{13.6MeV}{\theta_{RMS}}\sqrt{\frac{x}{X_0}}$

Full detector objective



Tomography section

$$\sigma_\theta \sim \theta_{RMS} = \frac{1}{N}\sum_i^N \Delta\theta_i^2$$

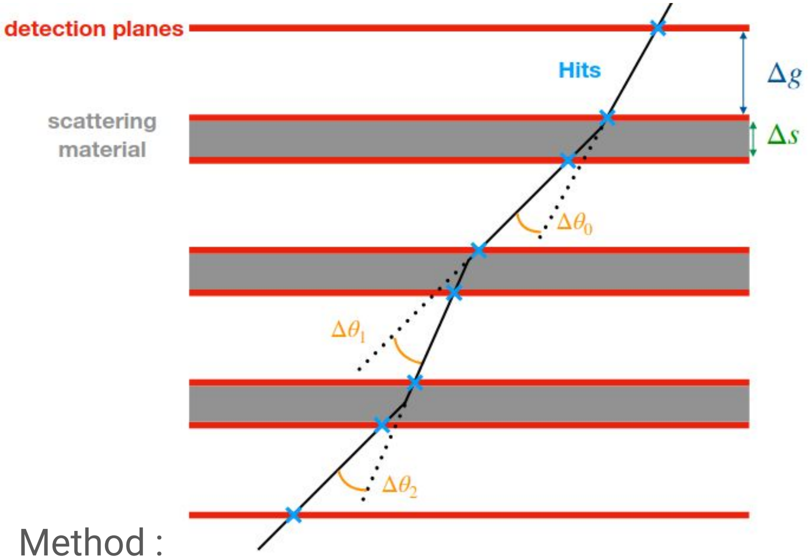
$$\sigma_\theta = \frac{13.6MeV}{p}\sqrt{\frac{x}{X_0}}$$

Measurement objective

Momentum section

$$p = \frac{13.6MeV}{\underbrace{\theta_{RMS}}_{\text{Measured}}}\underbrace{\sqrt{\frac{x}{X_0}}}_{\text{Known by construction (lead, iron)}}$$

Momentum measurement

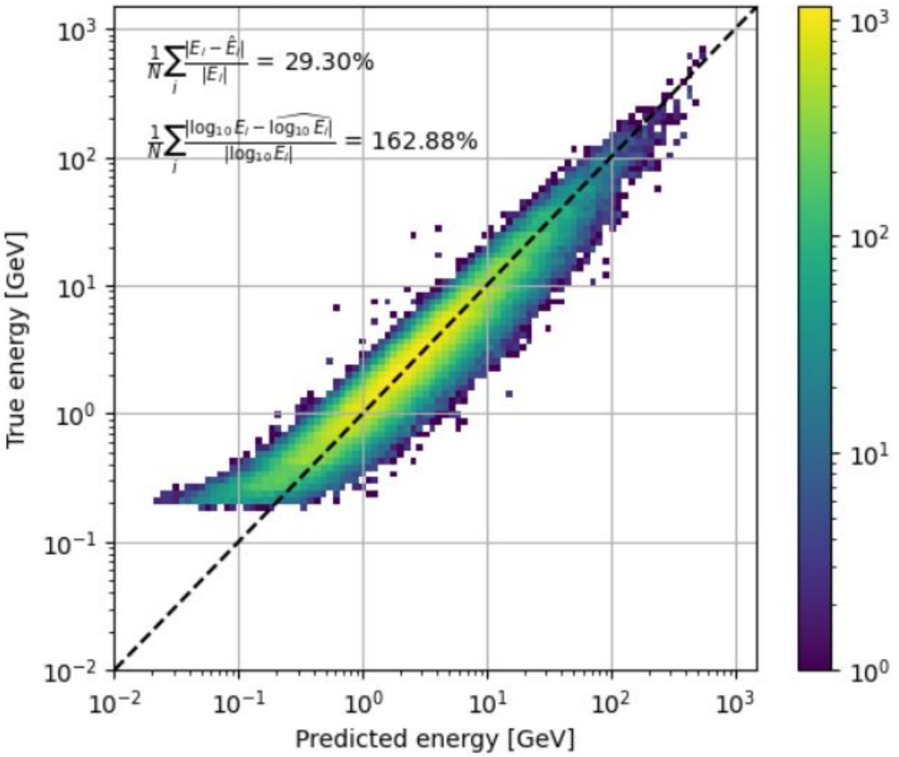


Method :

- From hits obtain segments
- From segments obtain scattering angles
- Take RMS and plug in

$$p = \frac{13.6MeV}{\theta_{RMS}}\sqrt{\frac{x}{X_0}}$$

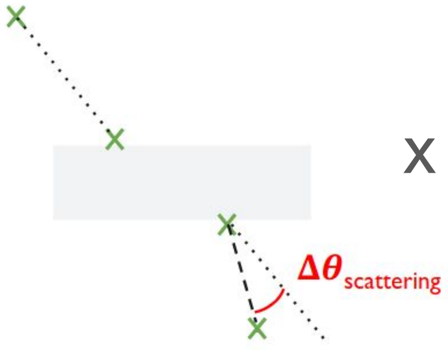
"Energy" = Kinetic energy



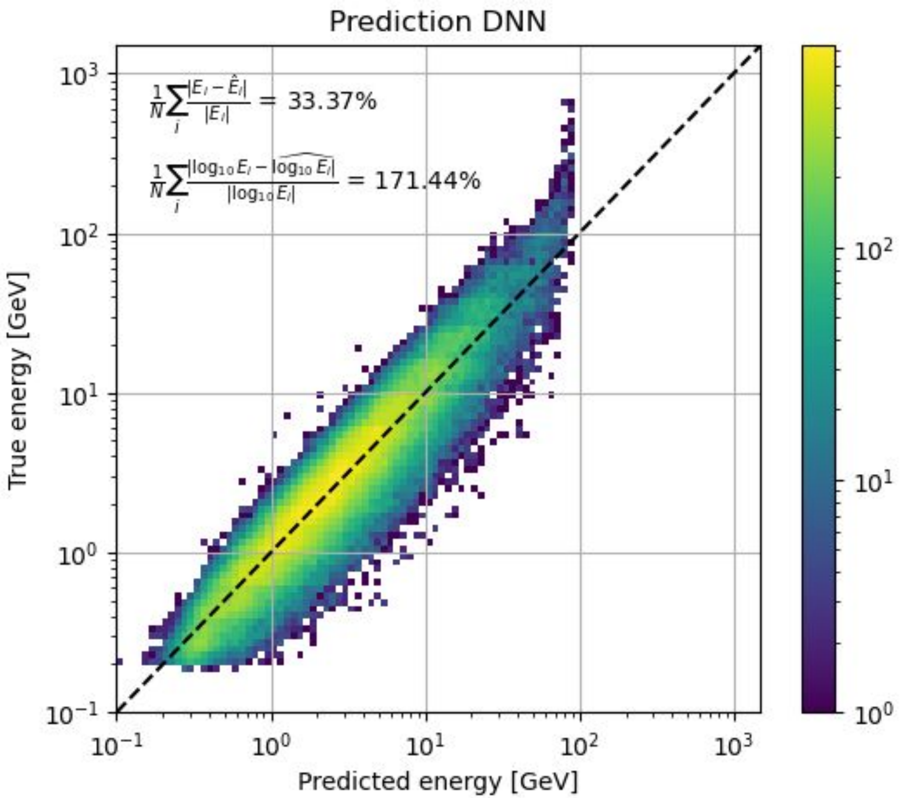
Maybe we can extract more than that

First ML attempt

Let us use a simple 3-layer DNN on the scattering angles



x $N_{scatterer}$



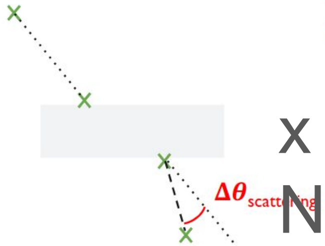
Improves at low energies

… issues at high energies

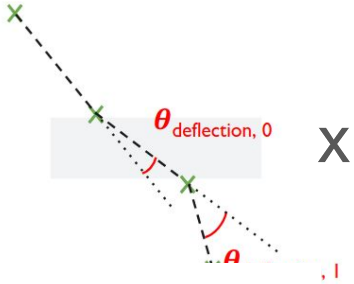
$$\Delta\theta_{\text{scattering}} \xrightarrow{E\to\infty} 0$$

Improved attempt

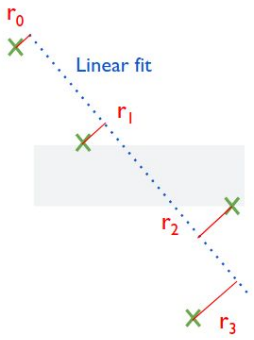
Adding more interesting variables



x $N_{scatterer}$

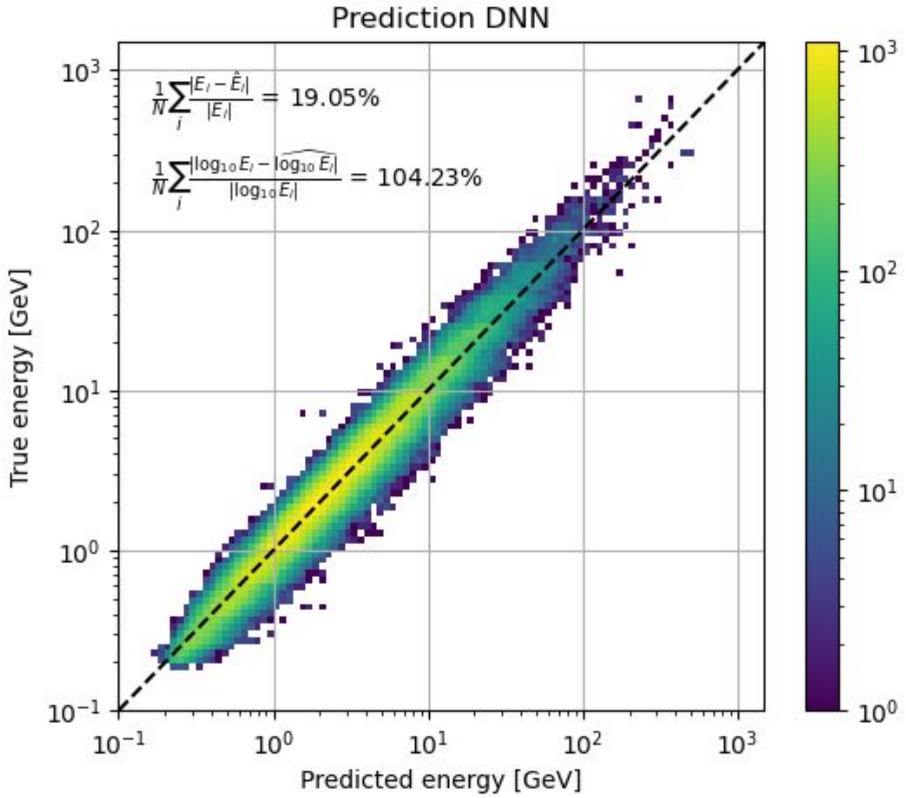


x N_{layer}



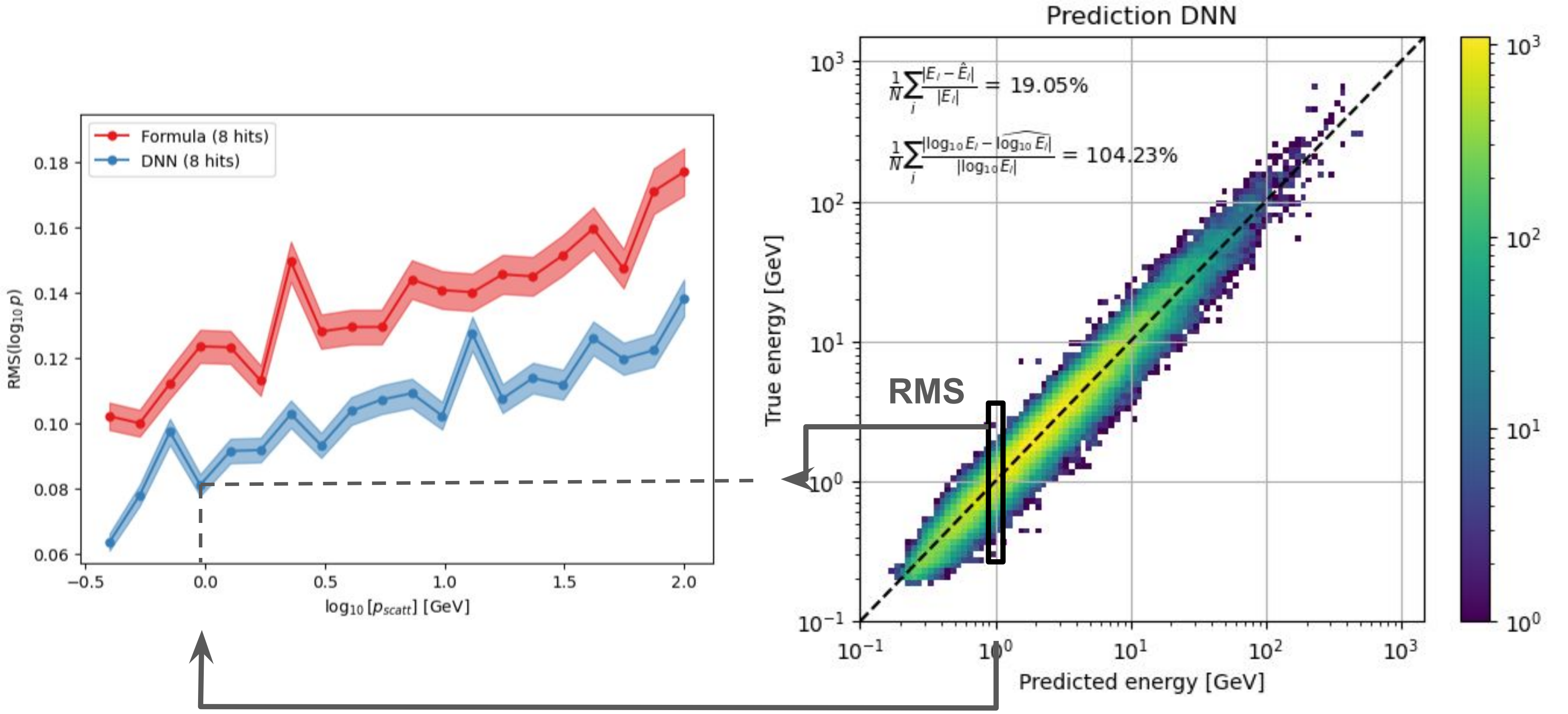
$|r_i|$ x N_{layer}

$$\sum_i r_i^2$$

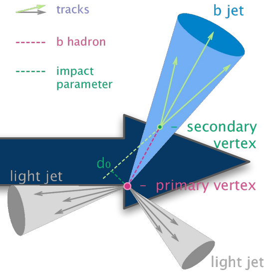


Improved attempt

A more quantitative comparison



Intermezzo : on the story of jet taggers



LHC run-1 | LHC run-2 | LHC run-3

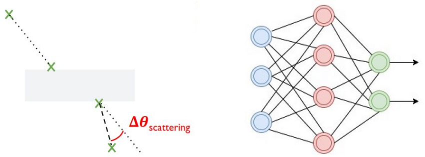
Jet probability taggers [1]

- Purely analytical
- Provides "likelihood" value

$$p = \frac{13.6MeV}{\theta_{RMS}}\sqrt{\frac{x}{X_0}}$$

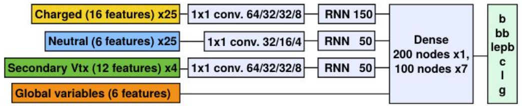
Combined Secondary Vertex (csv) [1]

- Shallow neural network
- Provide a score based on several inputs
- Later on, went deeper (DeepCSV [2])



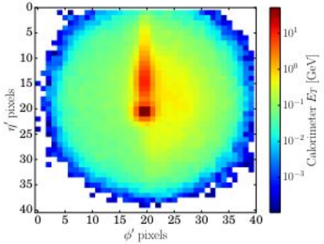
DeepJet [3]

- More complex architecture (CNN,RNN)
- Variable-length inputs (tracks)



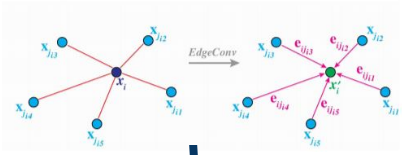
Jet as images [4]

- Jet as image
- CNN architecture
- End-to-end



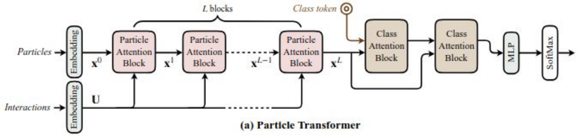
ParticleNet [5]

- Graph-based architecture
- Variable-length
- Sparse inputs



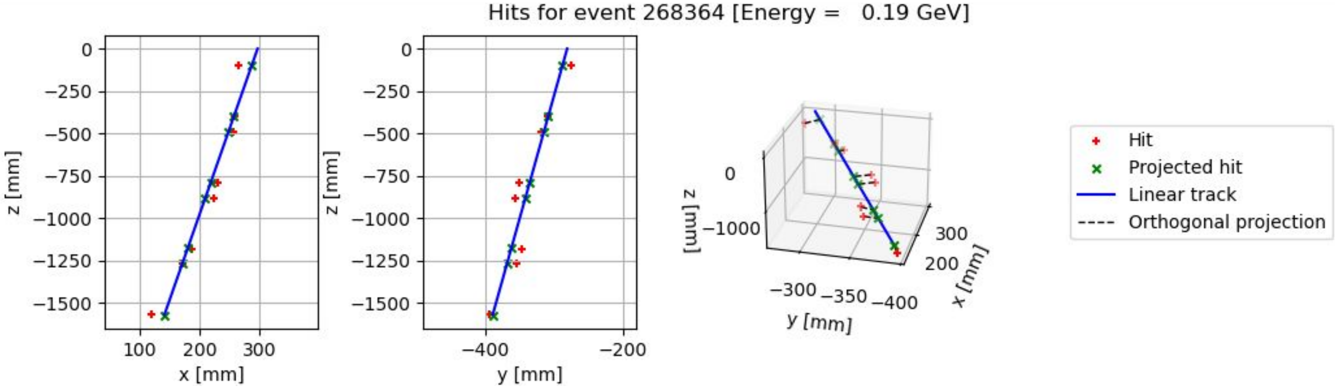
ParticleTransformer [6]

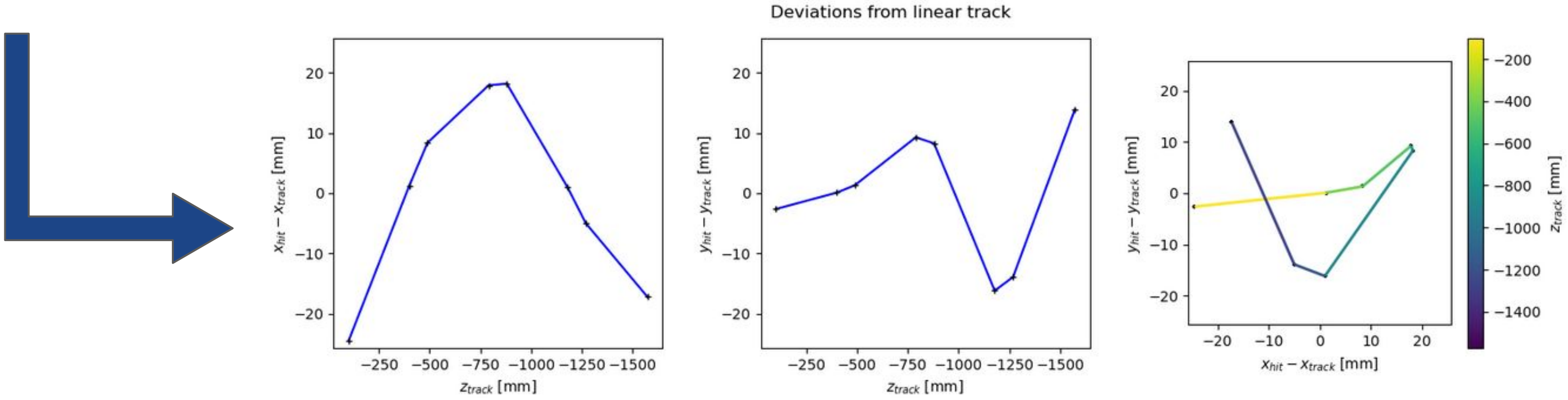
- Transformer (eg GPT) architecture
- Attention mechanism
- Variable-length



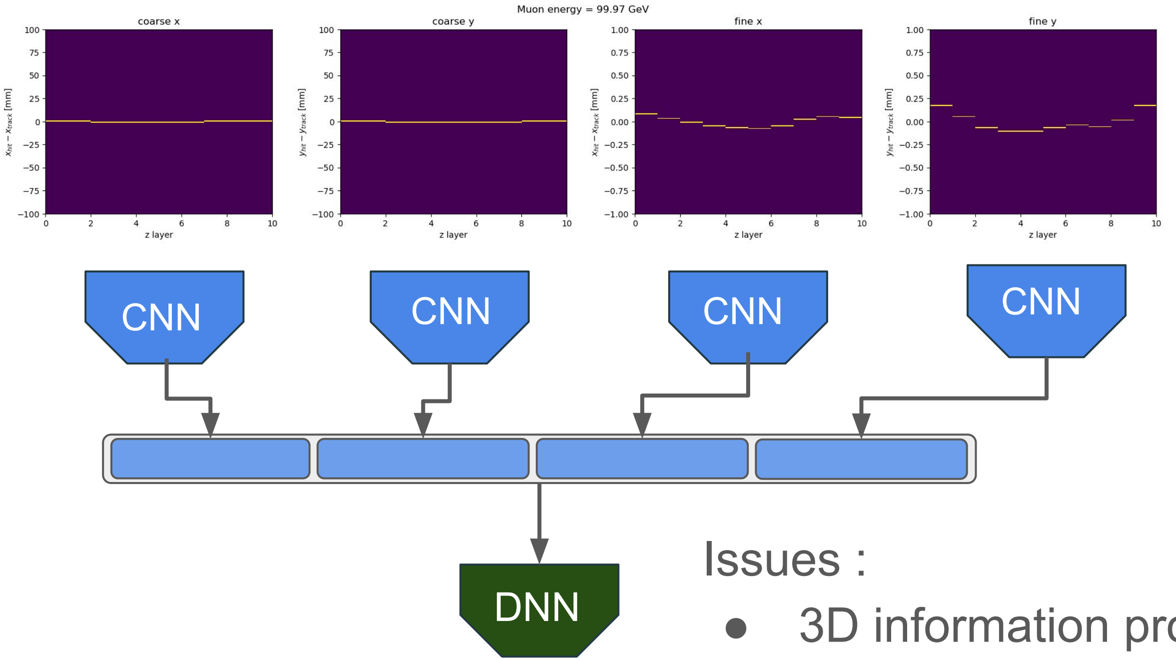
More complex architectures

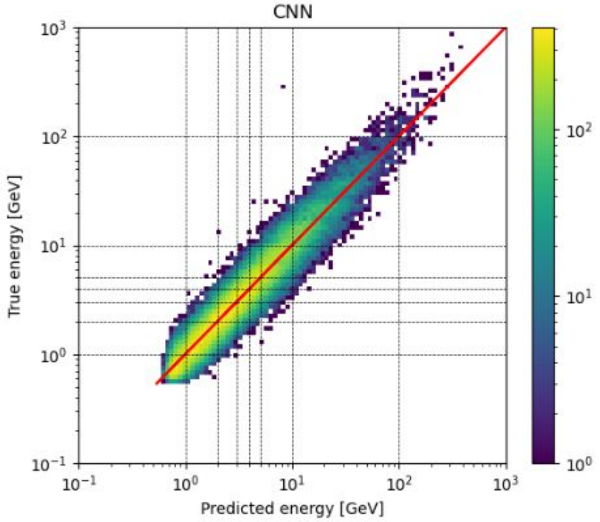
Only deviations from a linear track carry information about the energy





More complex architectures : deviations as an image

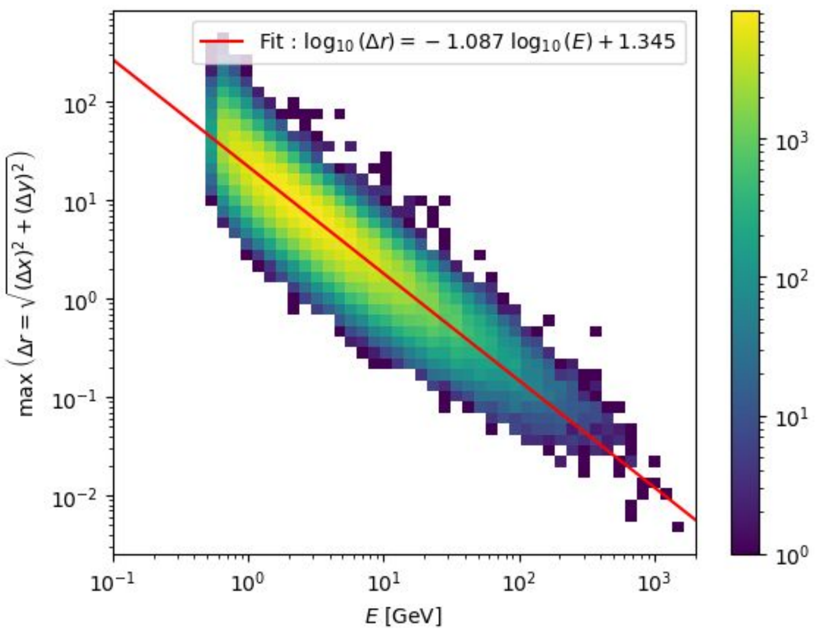




Issues :

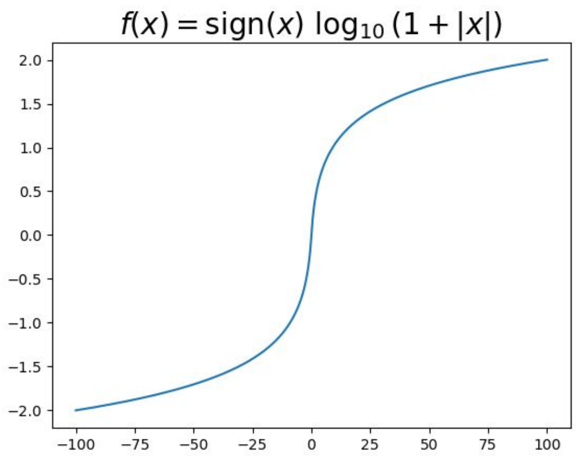
- 3D information projected into 2D
- Pixelized image, need different scales
- Very sparse input (energy/time inefficient)
- 3D images would be too heavy

Dealing with scale

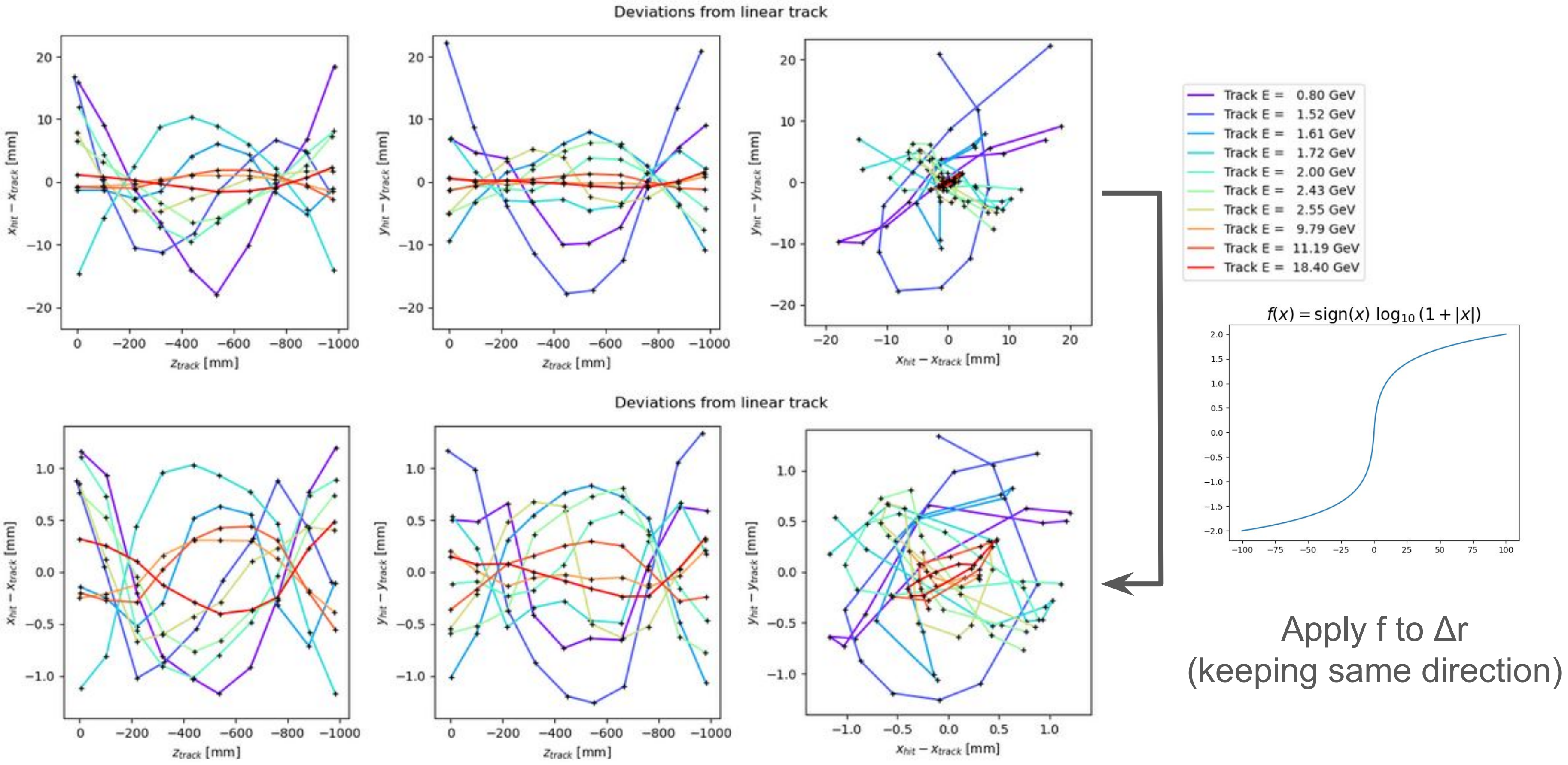


$$\sigma_\theta = \frac{13.6MeV}{p}\sqrt{\frac{x}{X_0}}$$

$$\max_{\text{layer}} \Delta r \propto \frac{1}{p}$$

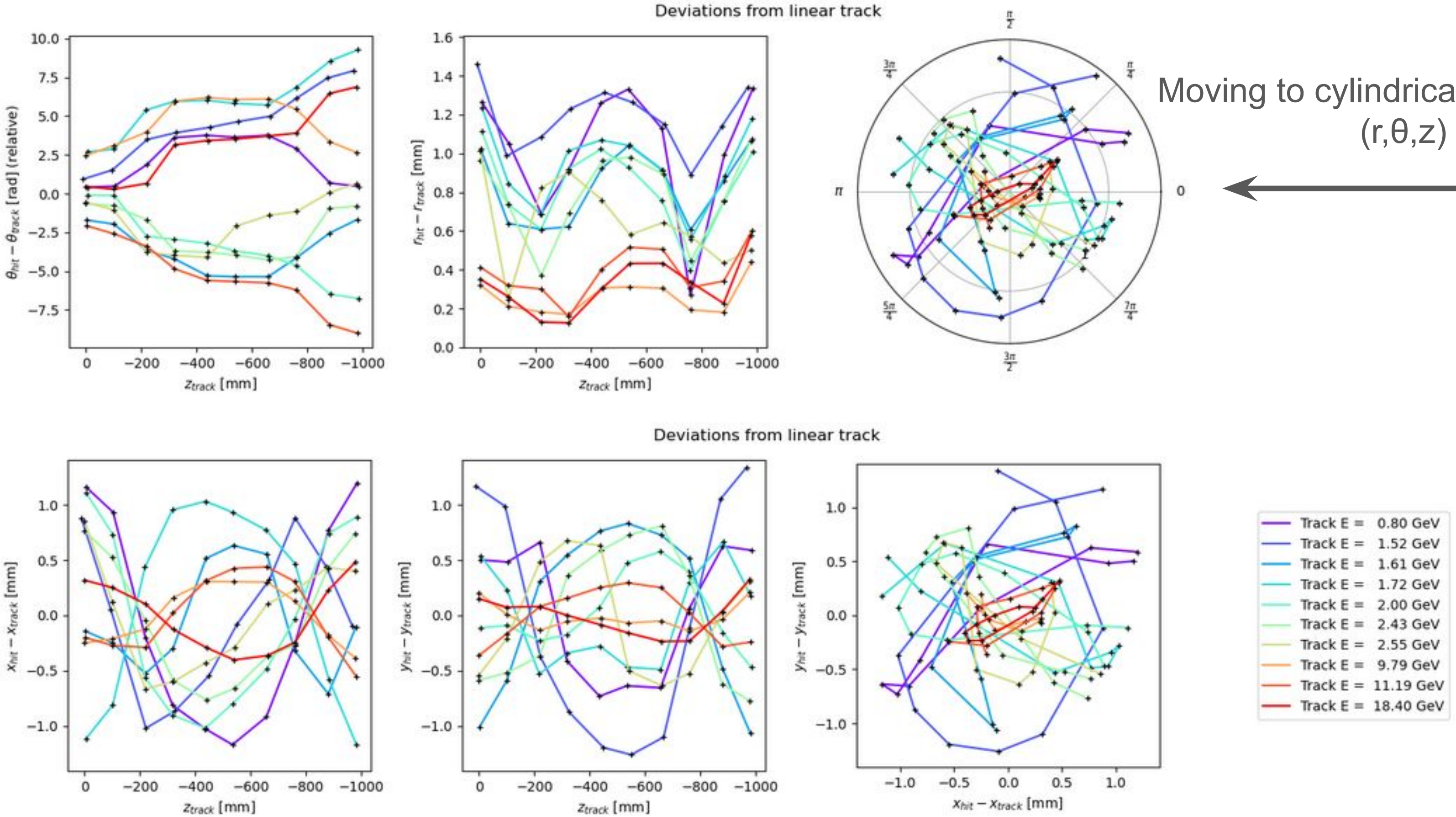


Dealing with scale



Apply f to Δr
(keeping same direction)

Dealing with scale

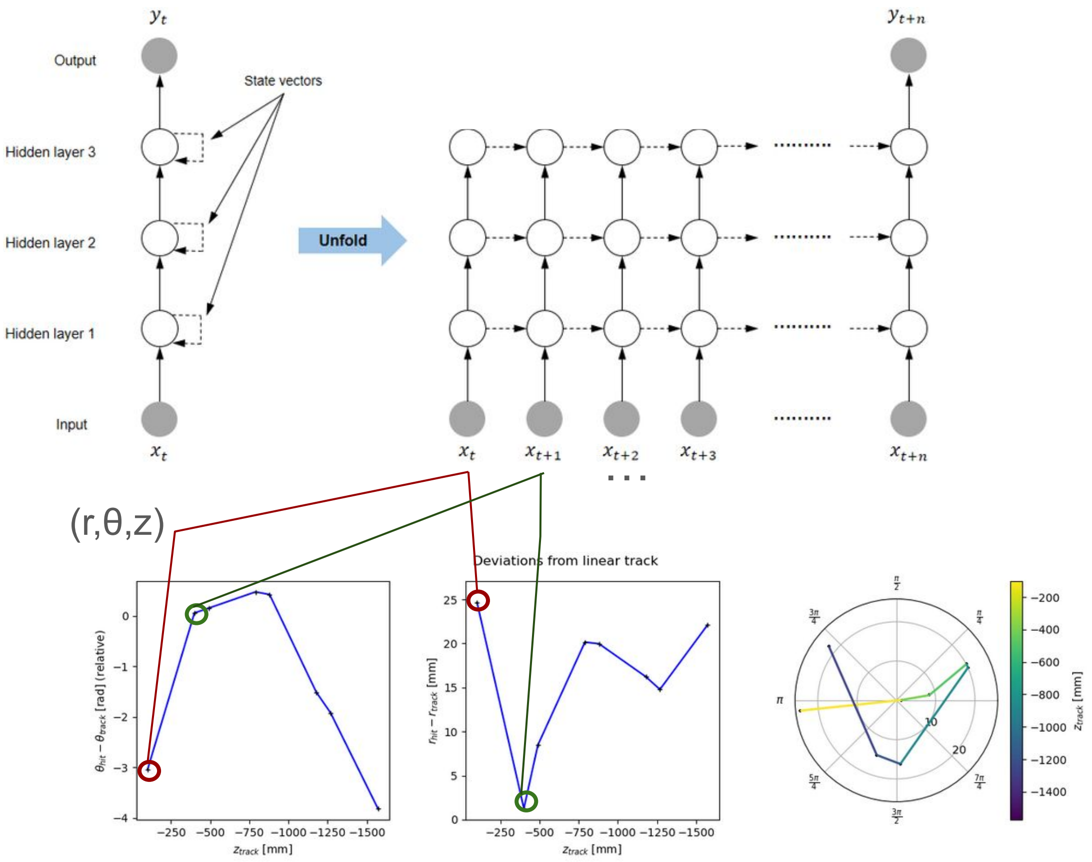


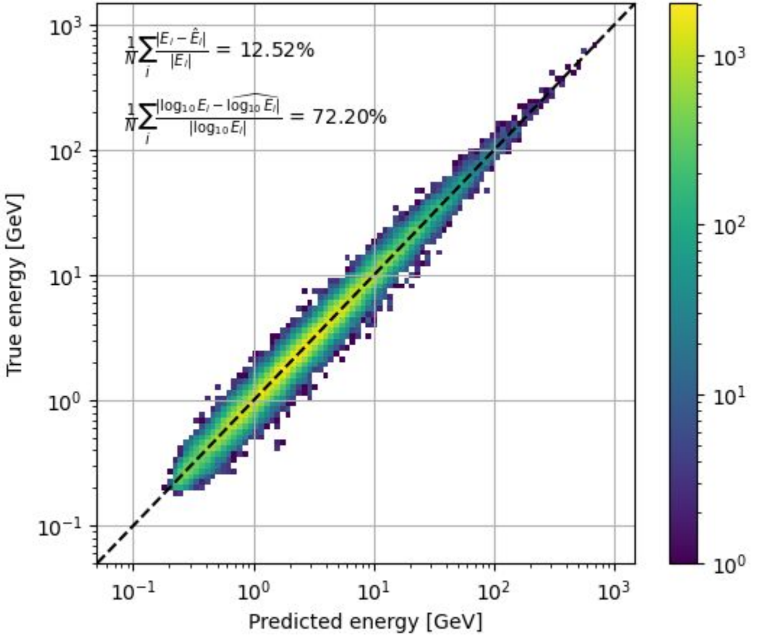
Moving to cylindrical coordinates
(r,θ,z)

More complex architectures : deviations as a sequence

Recurrent neural network (RNN)

“Many-to-one” scheme



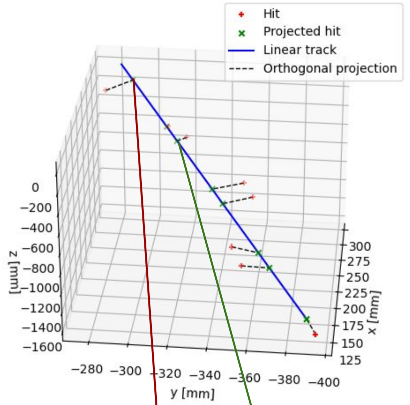


Issues :

- Sequential processing (not parallel, slow)
- No geometrical info

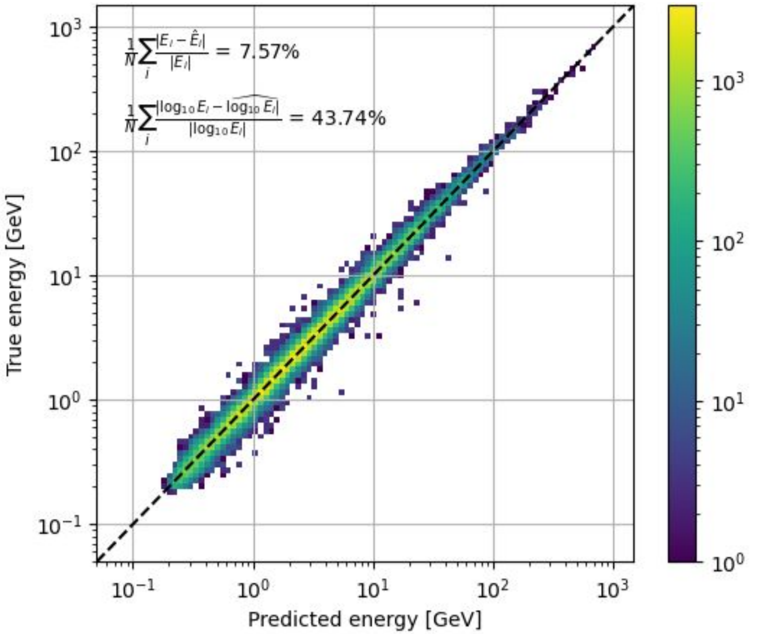
More complex architectures : deviations as a sequence

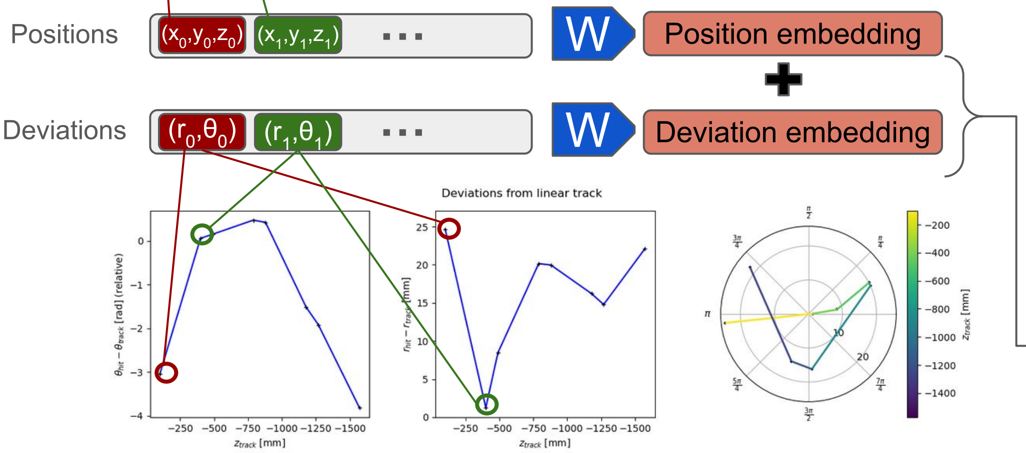
Transformer encoder

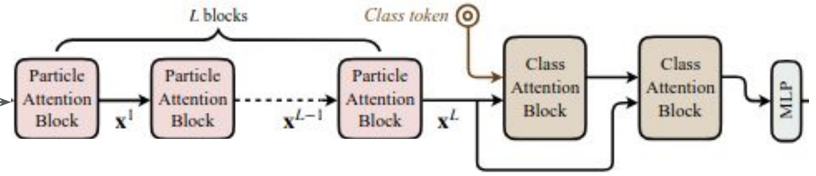


Compared to RNN :

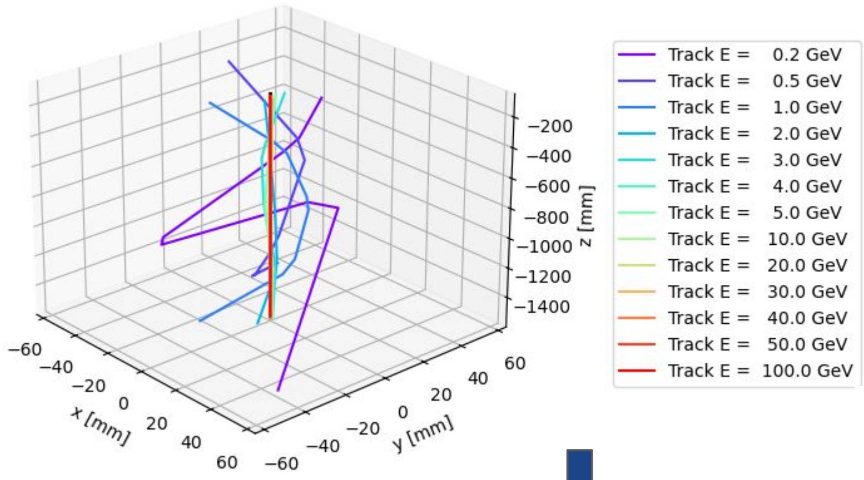
- Parallel processing
- Attention mechanism
- Long+short distance interactions







More complex architectures : deviations as a graph

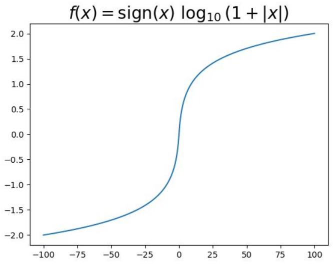


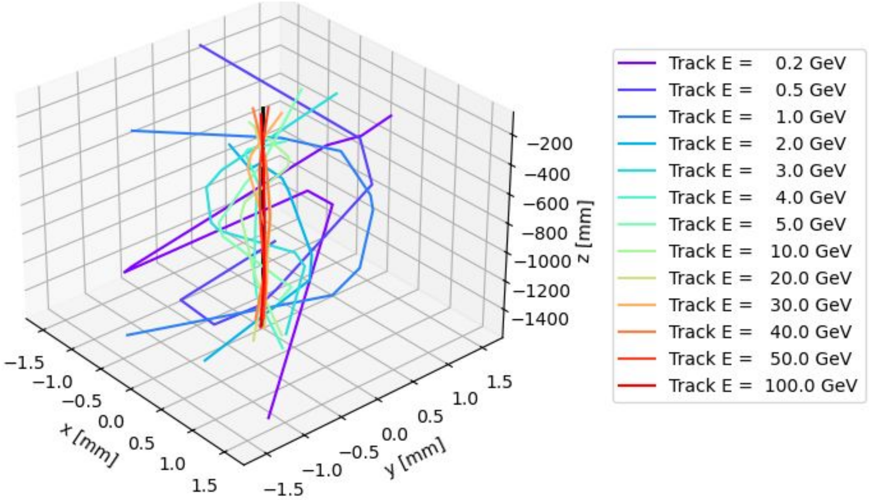
Graph is set of $\{\Delta x_i, \Delta y_i, z_i\}$

($\Delta x_i, \Delta y$ are the deviations at height z_i)

Same problem as before :

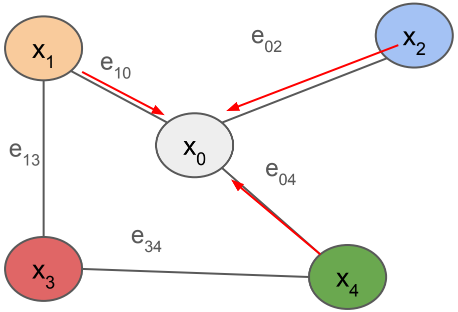
$$\max \Delta x_i \propto \frac{1}{p}$$





(Applied on the radius, keeping same angle)

More complex architectures : deviations as a graph



$$\mathbf{x}_i^{(k)} = \gamma^{(k)} \left(\mathbf{x}_i^{(k-1)}, \underbrace{\bigoplus_{j \in \mathcal{N}(i)} \phi^{(k)}\left(\mathbf{x}_i^{(k-1)}, \mathbf{x}_j^{(k-1)}, \mathbf{e_{ij}}\right)} \right)$$

Optional embedding of the concatenated previous step with aggregate+combine

"Combine" operation

- Applied on the node features and the ones of his immediate neighbours
- Can include edge features
- Usually some ML operation

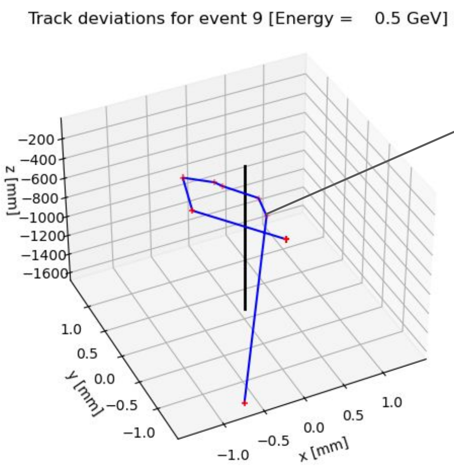
"Aggregate" operation

- Permutation invariant
- Typically min,max, mean, std, …
- Can be more complex (ML)

After *N* steps : $h_G = \bigoplus_{i \in G} x_i^{(N)}$ then fed to a DNN for regression

($\oplus$ can again be simple (eg, mean) or complex (ML))

More complex architectures : deviations as a graph



$$\mathbf{x}_i^{(k)} = \gamma^{(k)} \left(\mathbf{x}_i^{(k-1)}, \bigoplus_{j \in \mathcal{N}(i)} \underbrace{\phi^{(k)}\left(\mathbf{x}_i^{(k-1)}, \mathbf{x}_j^{(k-1)}, \mathbf{e_{ij}}\right)} \right)$$

Optional embedding of the concatenated previous step with aggregate+combine

“Combine” operation

- Applied on the node features and the ones of his immediate neighbours
- Can include edge features
- Usually some ML operation

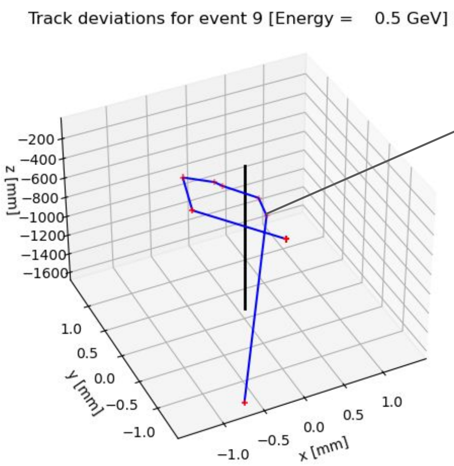
“Aggregate” operation

- Permutation invariant
- Typically min,max, mean, std, …
- Can be more complex (ML)

After *N* steps : $h_G = \bigoplus_{i \in G} x_i^{(N)}$ then fed to a DNN for regression

($\oplus$ can again be simple (eg, mean) or complex (ML))

More complex architectures : deviations as a graph



$$\mathbf{x}_i^{(k)} = \gamma^{(k)}\left(\mathbf{x}_i^{(k-1)}, \bigoplus_{j\in\mathcal{N}(i)} \underbrace{\phi^{(k)}\left(\mathbf{x}_i^{(k-1)}, \mathbf{x}_j^{(k-1)}, \mathbf{e_{ij}}\right)}\right)$$

Optional embedding of the concatenated previous step with aggregate+combine

"Aggregate" operation

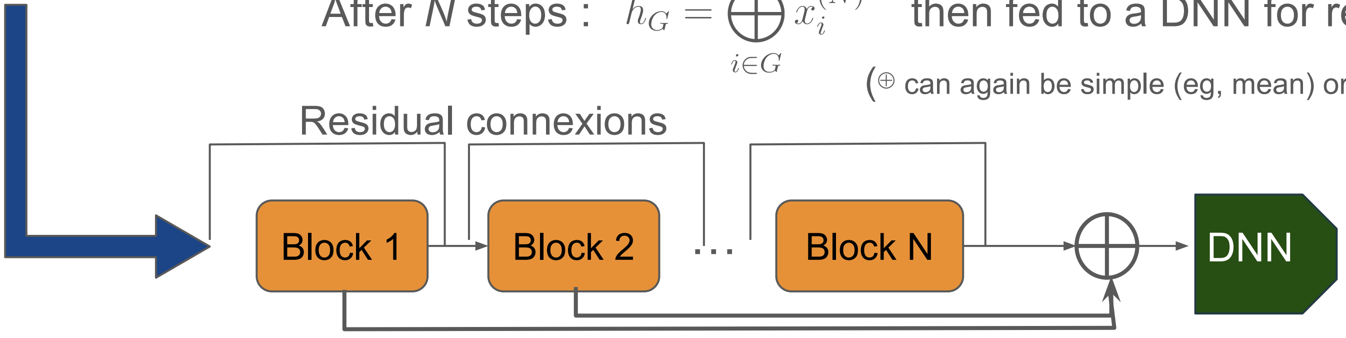
- Permutation invariant
- Typically min,max, mean, std, …
- Can be more complex (ML)

"Combine" operation

- Applied on the node features and the ones of his immediate neighbours
- Can include edge features
- Usually some ML operation

After *N* steps : $h_G = \bigoplus_{i\in G} x_i^{(N)}$ then fed to a DNN for regression

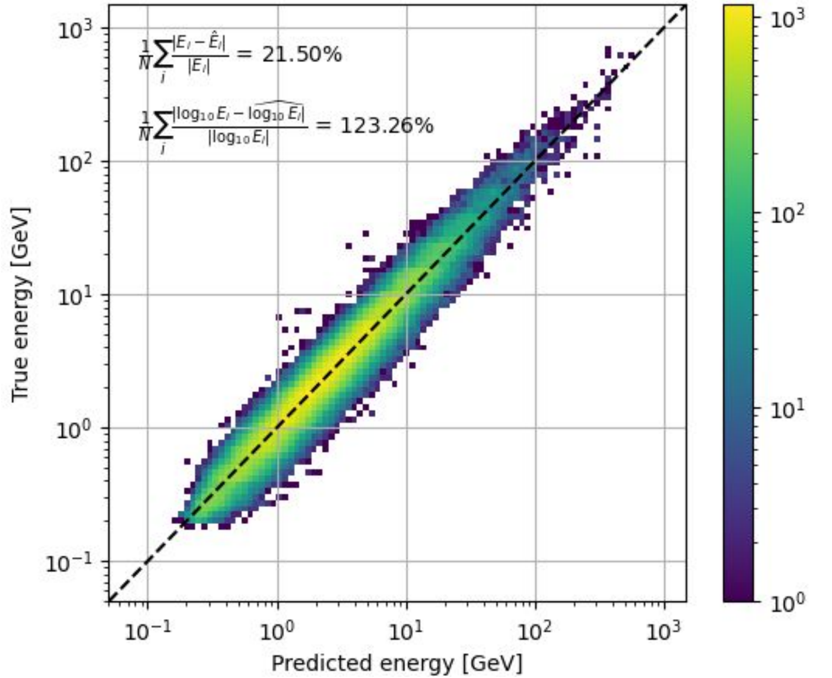
($\oplus$ can again be simple (eg, mean) or complex (ML))



(Note : block models considered in back-up)

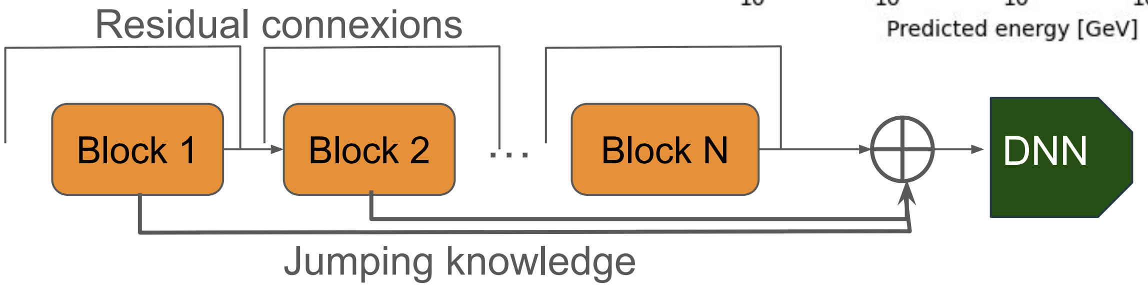
More complex architectures : deviations as a graph

$$\mathbf{x}_i^{(k)} = \gamma^{(k)}\left(\mathbf{x}_i^{(k-1)}, \bigoplus_{j\in\mathcal{N}(i)} \phi^{(k)}\left(\mathbf{x}_i^{(k-1)}, \mathbf{x}_j^{(k-1)}, \mathbf{e_{ij}}\right)\right)$$

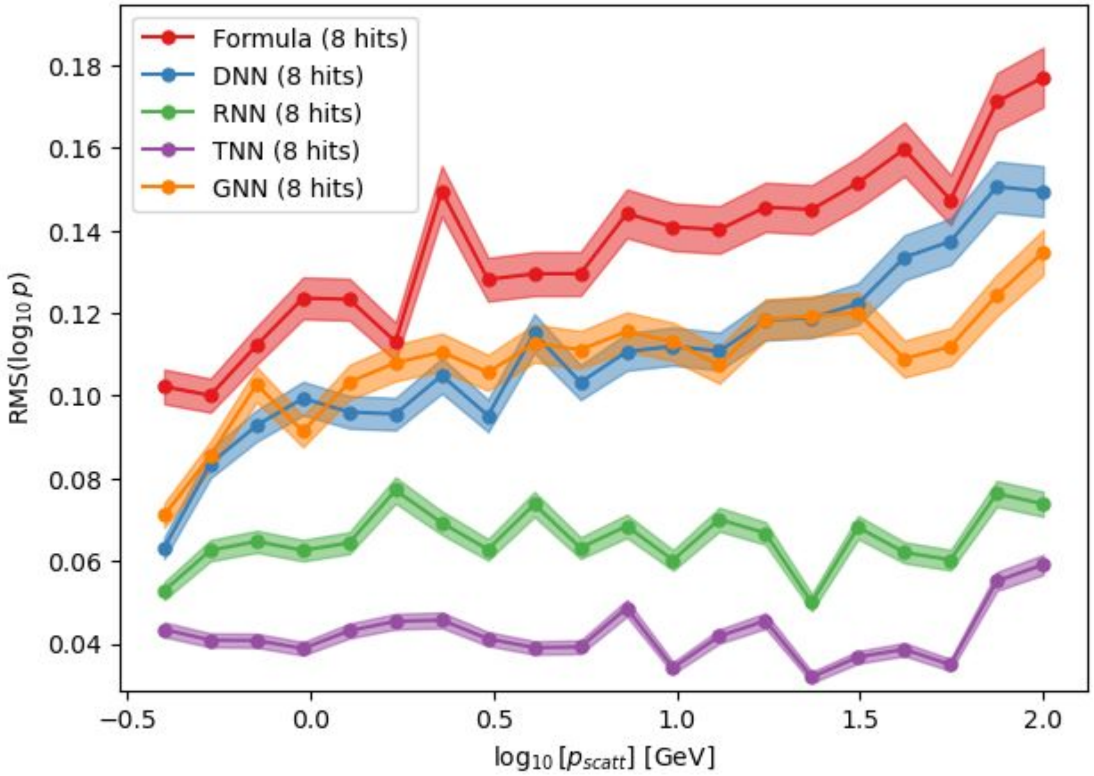


Here PointGCN
(other block models in back-up)

Not satisfactory, still have to play with aggregation and models



More complex architectures : comparing performance



Model summary

- Model size
  - DNN : 18K params / 0.07 MB
  - RNN : 165K params / 0.62 MB
  - TNN : 430K params / 1.64 MB
  - GNN : 100K params / 0.40 MB
- Computation time
  (batch size = 1024, GPU GeForce RTX 2080)
  - DNN : 21 μs / muon
  - RNN : 60 μs / muon
  - TNN : 24 μs / muon
  - GNN : 80 μs / muon

(note : CPU batching not optimised)

Real-life requirements

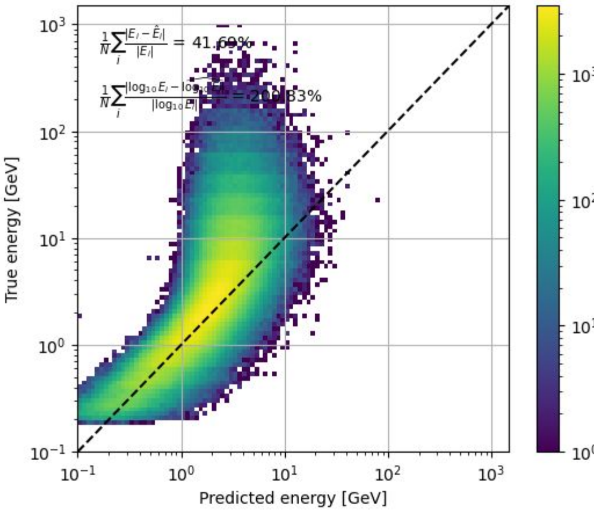
Using a Gaussian smearing on x and y hit position

$$x, y \sim \mathcal{N}(\mu = 0, \sigma = 1\ \mathrm{mm})$$

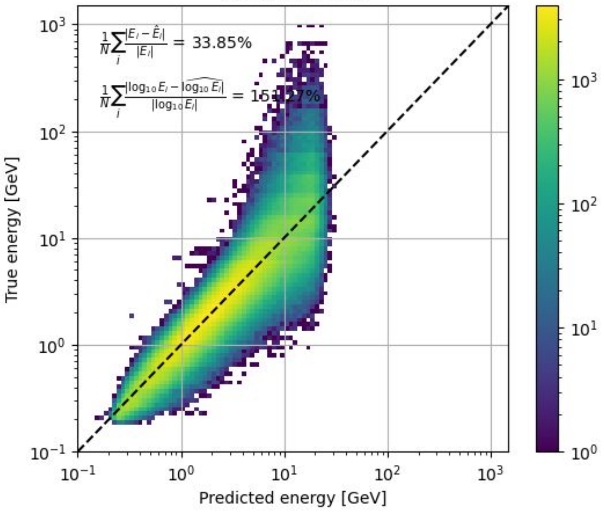
(typical resolution of RPC detectors)

Effect of spatial resolution

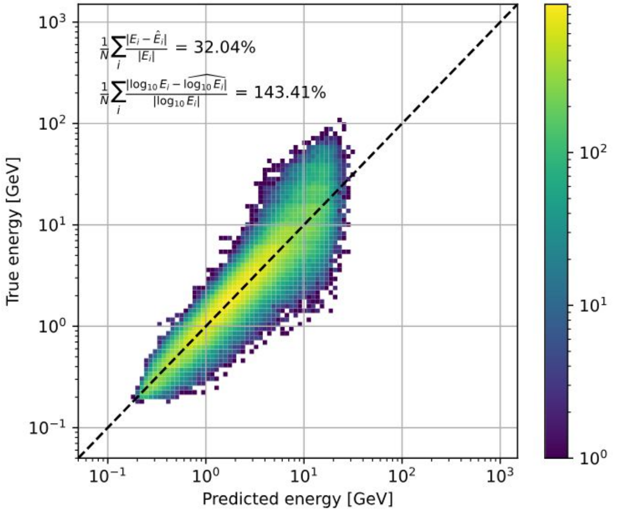
Scattering formula



DNN



TNN



ML methods can limit the issue of the limited precision … up to a point

Real-life requirements

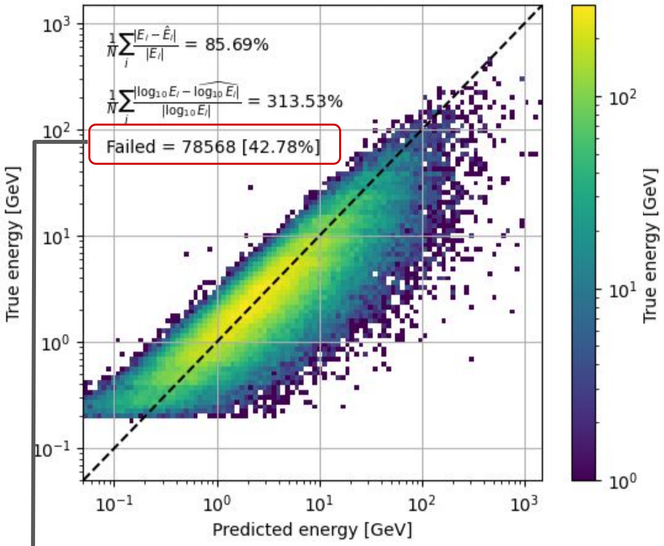
For 8 detector planes (3 scatterer volumes) :

- ε = 90% : $N_{8\ hits}$ = 43%, $N_{7\ hits}$ = 38%, $N_{6\ hits}$ = 15%
- ε = 95% : $N_{8\ hits}$ = 66%, $N_{7\ hits}$ = 28%, $N_{6\ hits}$ = 5%
- ε = 99% : $N_{8\ hits}$ = 92%, $N_{7\ hits}$ = 7%, $N_{6\ hits}$ = <1%

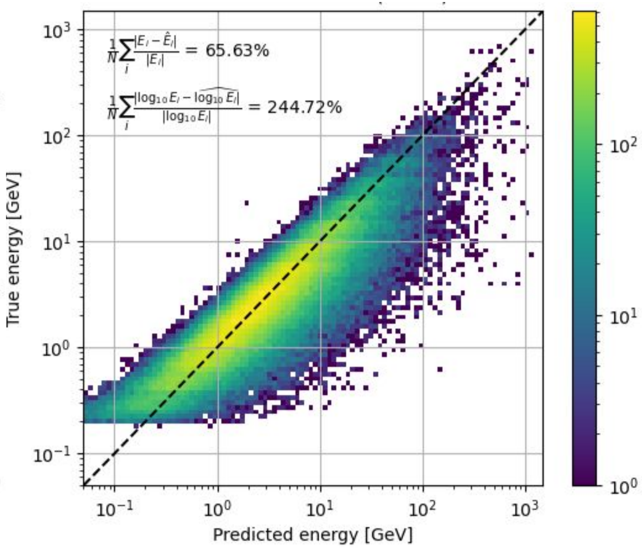
Effect of hit efficiency $$p = \frac{13.6MeV}{\theta_{RMS}}\sqrt{\frac{x}{X_0}}$$

3 angles → can be reduced when hit is missed (but worse performance)

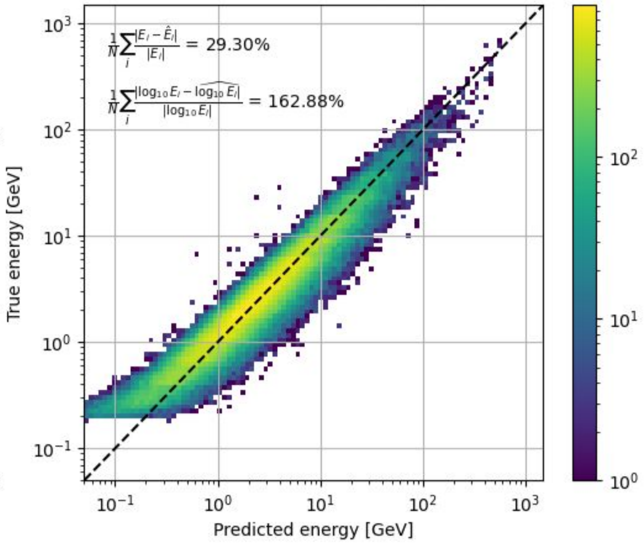
2 missed hits (total = 6)



1 missed hit (total = 7)



0 missed hit (total = 8)



Two missed hits at the wrong places, and no segment can be built around scatterers

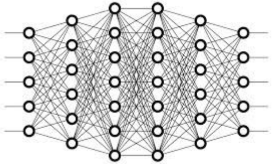
Real-life requirements

For 8 detector planes (3 scatterer volumes) :

- $\varepsilon = 90\%$: $N_{8\ hits} = 43\%$, $N_{7\ hits} = 38\%$, $N_{6\ hits} = 15\%$
- $\varepsilon = 95\%$: $N_{8\ hits} = 66\%$, $N_{7\ hits} = 28\%$, $N_{6\ hits} = 5\%$
- $\varepsilon = 99\%$: $N_{8\ hits} = 92\%$, $N_{7\ hits} = 7\%$, $N_{6\ hits} = <1\%$

Effect of hit efficiency

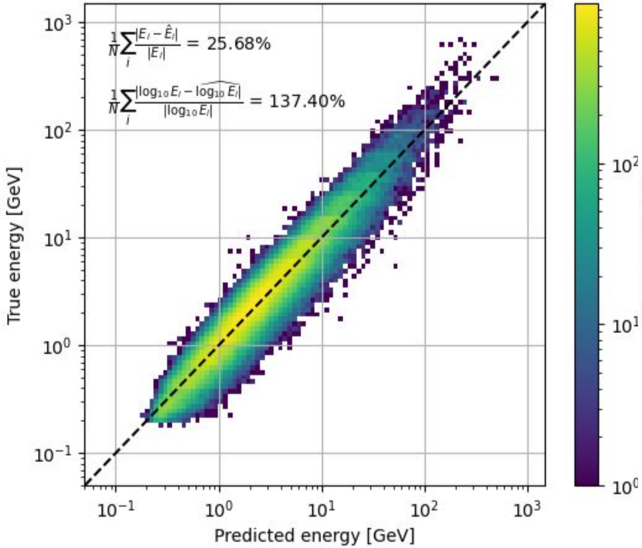
DNN



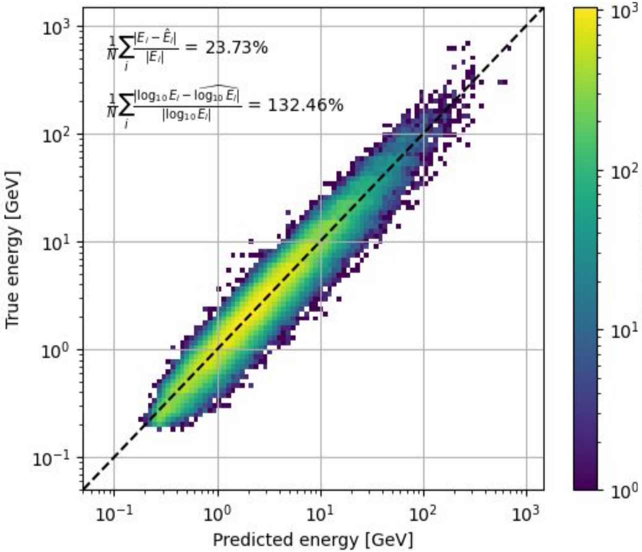
Missed variables are zero-padded

Model is trained on a mix of datasets
(fit and deviations done only on recorded hits)

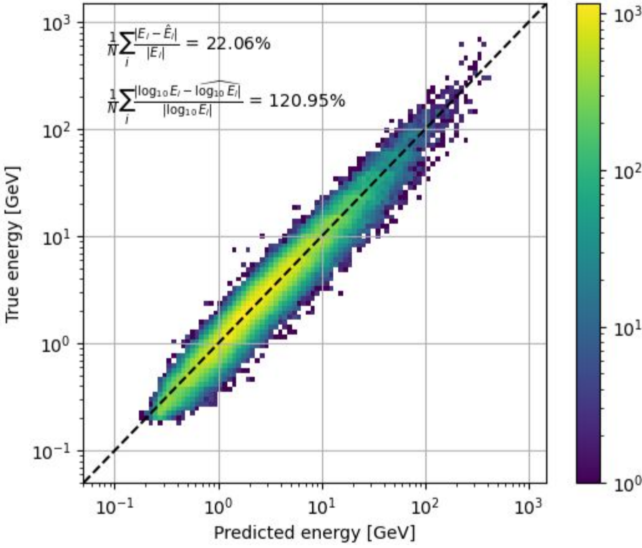
2 missed hits (total = 6)



1 missed hit (total = 7)



0 missed hit (total = 8)



Real-life requirements

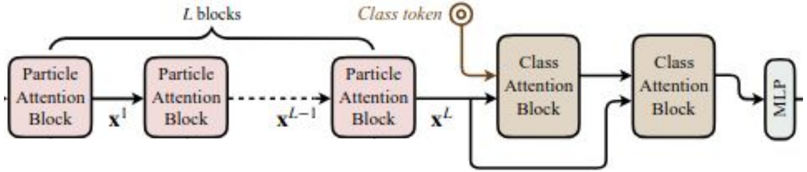
For 8 detector planes (3 scatterer volumes) :

- ε = 90% : $N_{8\ hits}$ = 43%, $N_{7\ hits}$ = 38%, $N_{6\ hits}$ = 15%
- ε = 95% : $N_{8\ hits}$ = 66%, $N_{7\ hits}$ = 28%, $N_{6\ hits}$ = 5%
- ε = 99% : $N_{8\ hits}$ = 92%, $N_{7\ hits}$ = 7%, $N_{6\ hits}$ = <1%

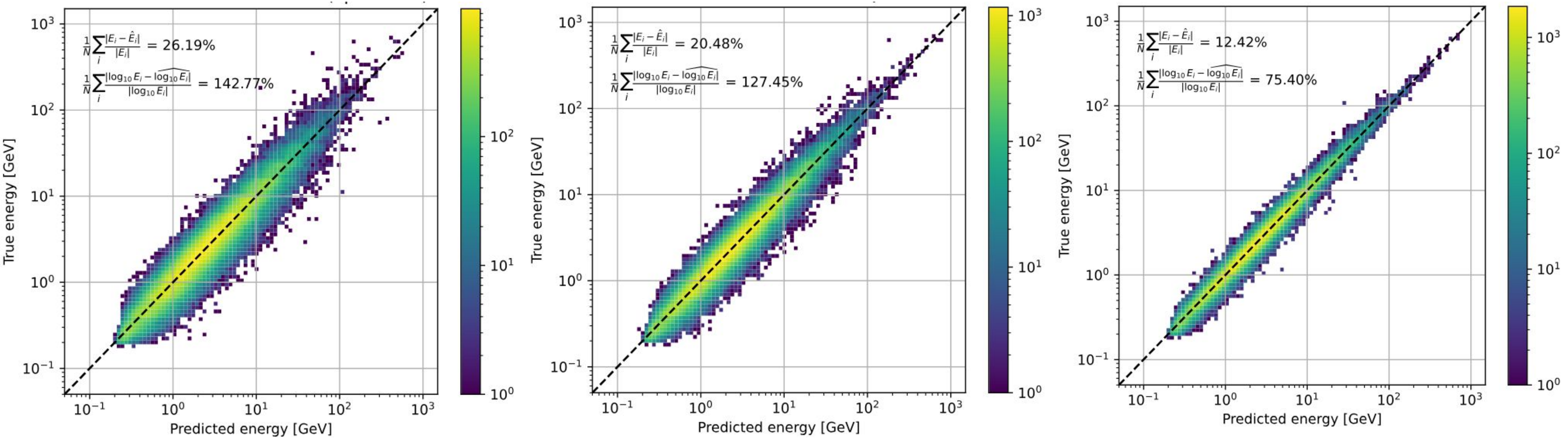
Effect of hit efficiency

Using the variable-length feature

TNN



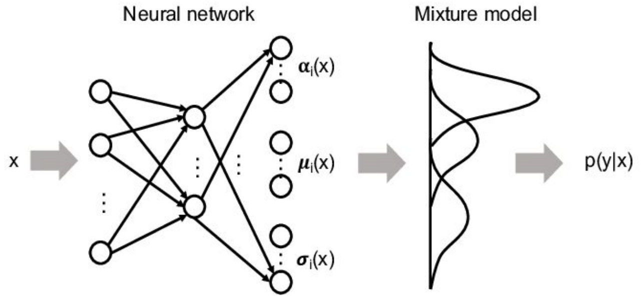
2 missed hits (total = 6) | 1 missed hit (total = 7) | 0 missed hit (total = 8)



Real-life requirements

Resolution estimation

Mixture density network (MDN)

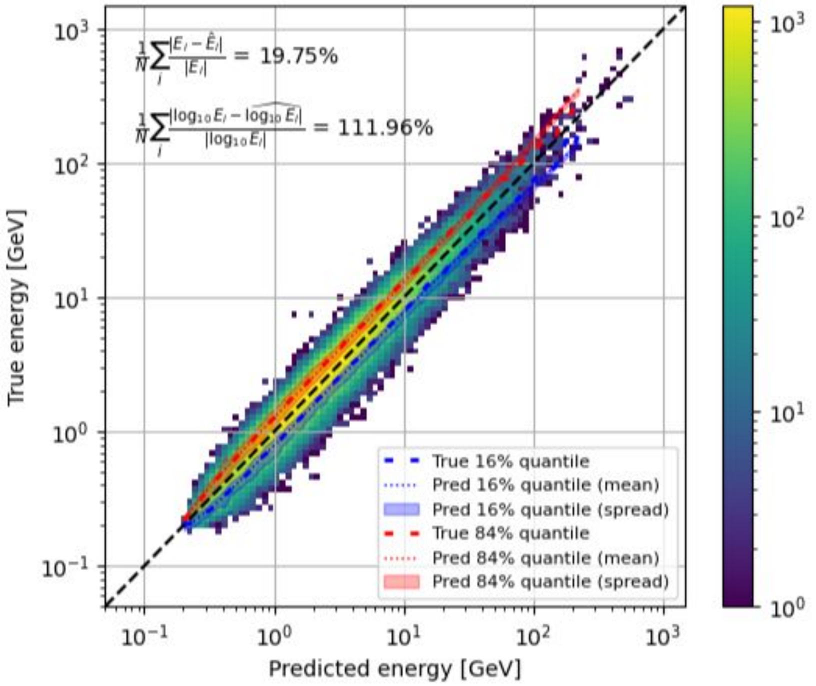


$$p(E|x,\theta) = \sum_i \pi_i \frac{1}{\sqrt{2\pi\sigma_i^2}} e^{-\frac{(x-\mu_i)^2}{2\sigma_i^2}}$$

$$\mu_i \equiv \mu_i(x|\theta),\ \sigma_i \equiv \sigma_i(x|\theta),\ \pi_i \equiv \pi_i(x|\theta)$$

$$L(t, \{\mu_i, \sigma_i, \pi_i\}) = -\log\left(\sum_i \exp(\log(\pi_i) - \log\sigma_i - \frac{1}{2}(\frac{t-\mu_i}{\sigma_i})^2)\right)$$

Correctly predicts the 1σ range of E_{true}

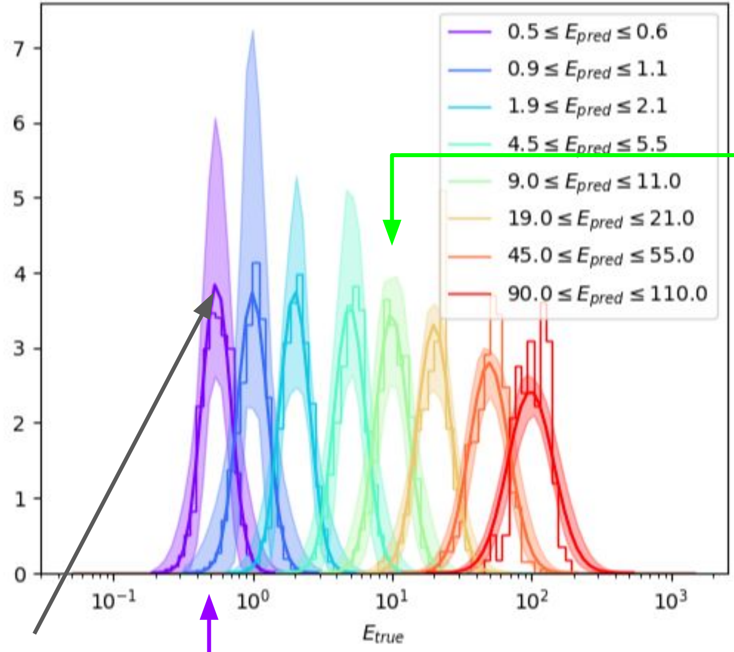


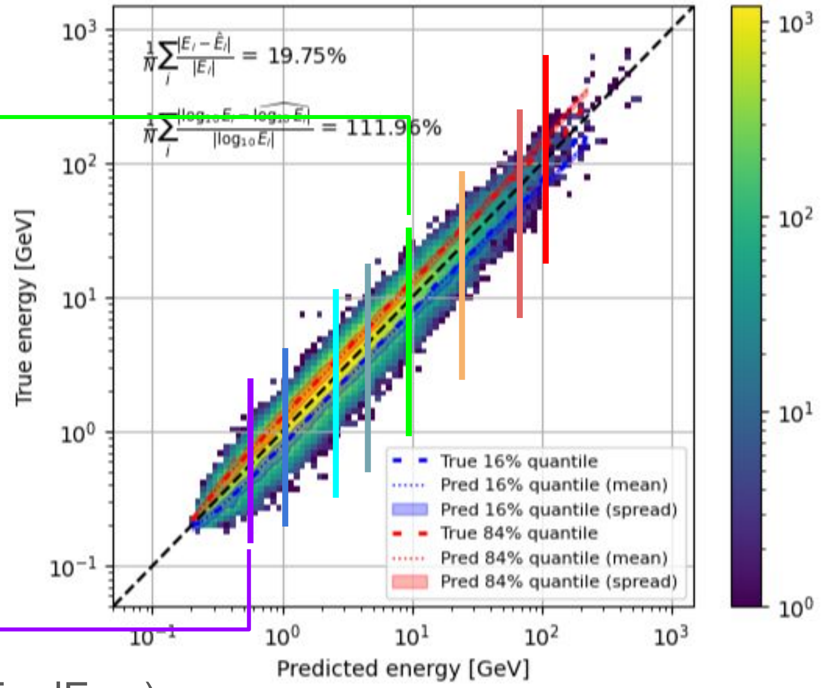
Note : can also be done with TNN or GNN as backbone

Real-life requirements

Resolution estimation

Correctly predicts the 1σ range of E_{true}



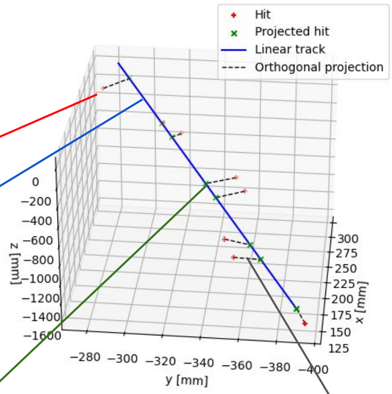


MDN mean and spread

Good reconstruction of $P(E_{true}|E_{pred})$

Real-life requirements

End-to-end model



Hits → Fit → Projected hits → Deviations

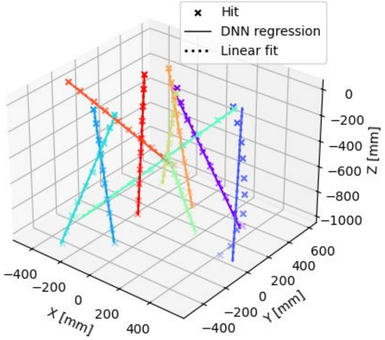
Cumbersome and potentially time-consuming

RNN
TNN
GNN

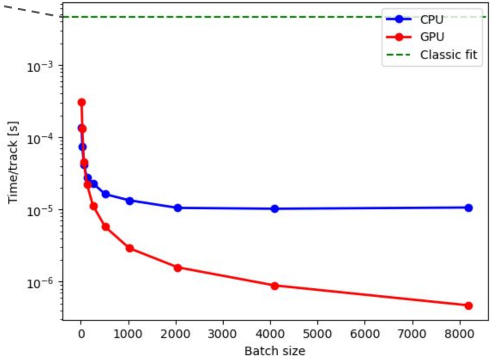
$\log_{10}E$

Fitter (DNN)
m_x, m_y, b_x, b_y

$$x = m_{xz}z + b_x$$
$$y = m_{yz}z + b_y$$



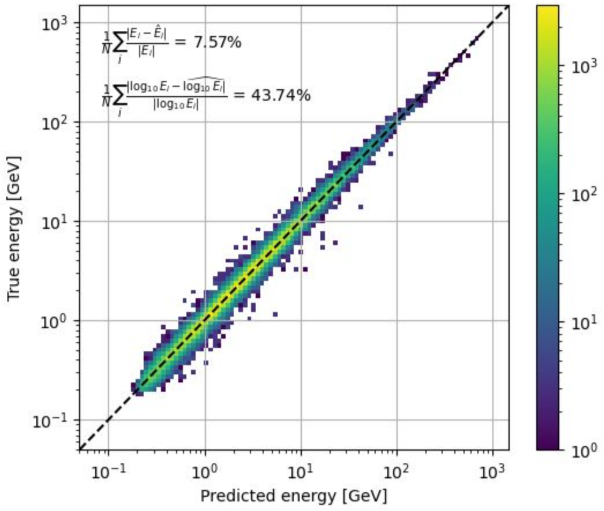
Deviations (analytical)



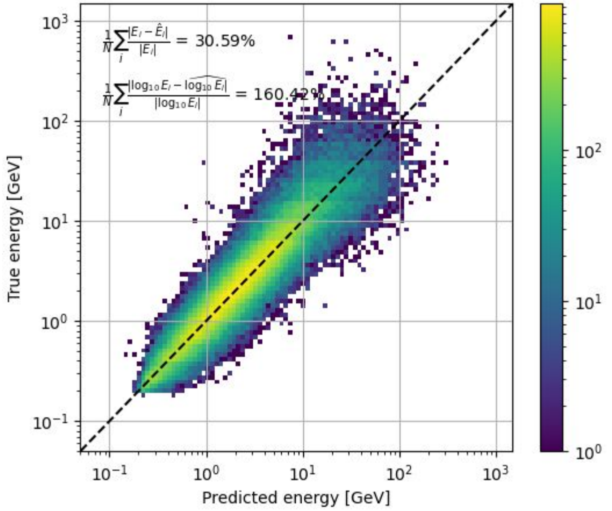
Real-life requirements

End-to-end model

Classic fit + TNN (trained on classic fit data)

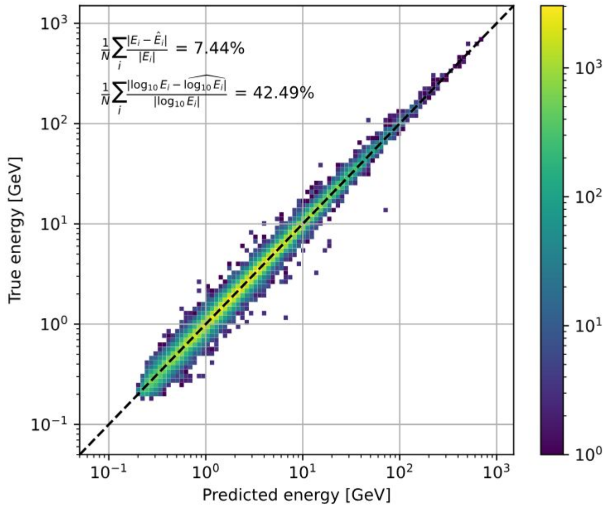


DNN fit + TNN (trained on classic fit data)



DNN fitter not perfect (especially at high energies)

DNN fit + TNN (trained on DNN fit data)



Can be recovered by specific training

Conclusion

$$p = \frac{13.6MeV}{\theta_{RMS}}\sqrt{\frac{x}{X_0}}$$

Advanced ML architectures allow extracting more information from the scattering

Detector-specific predictions

Done :

- Hit efficiency
- Hit resolution
- Posterior $P(E_{true}|E_{pred})$
- End-to-end model

To-dos :

- Modify E distributions
- Different detector setups
- Optimize architectures

Detector agnostic predictions

- Exploit symmetries more explicitly (contrastive learning)
- Generative pre-training
- Test within TomOpt

Conclusion

Advanced ML architectures allow extracting more information from the scattering

Experimental requirements are met :

$$p = \frac{13.6MeV}{\theta_{RMS}}\sqrt{\frac{x}{X_0}}$$

- Hit efficiency
- Hit resolution
- Posterior $P(E_{true}|E_{pred})$
- End-to-end model

Future checks :

- Modify E spectrum in training
- Hyperparameter scans
- Test different setups (#planes, X_0)

Specific detector

Future developments :

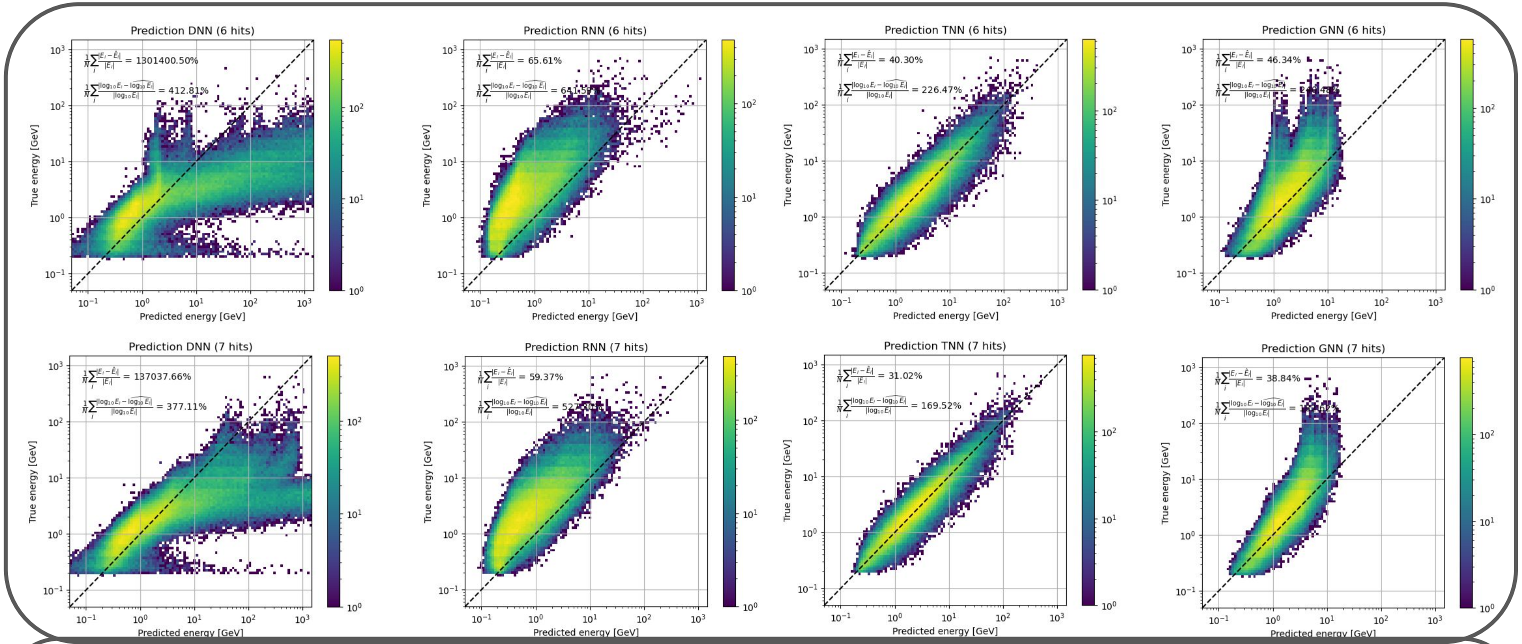
- Include symmetry (contrastive learning)
- Generative pre-training

Detector-agnostic predictions
(TomOpt)

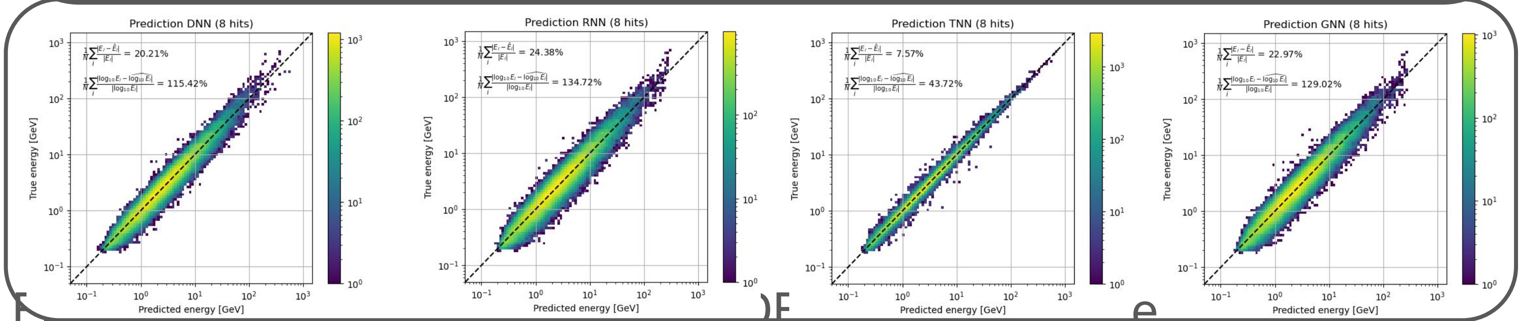
Back-up

Effect of hit efficiency

Model only trained on 8 hits, evaluated on 6, 7 and 8 hits

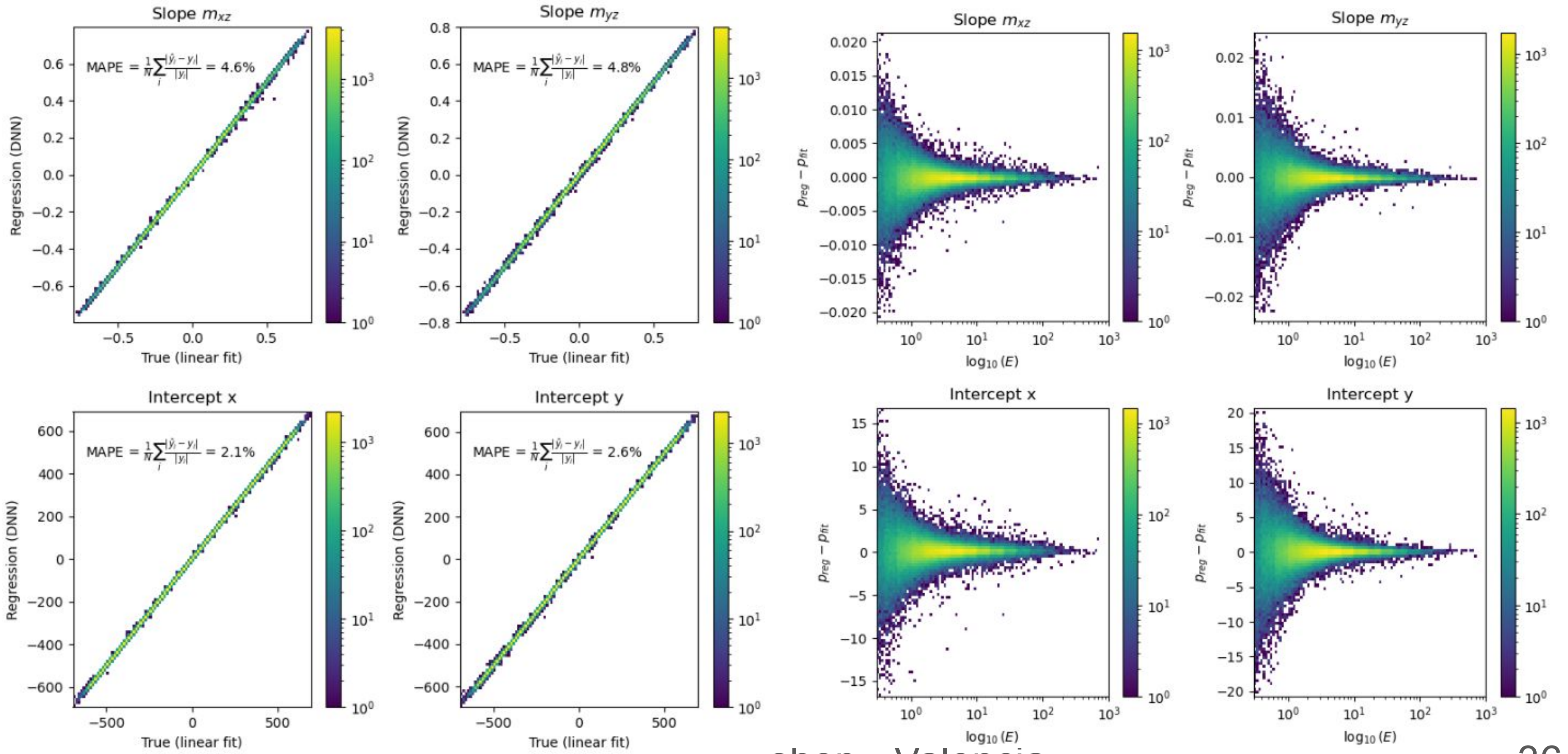


Not seen during training



Seen during training

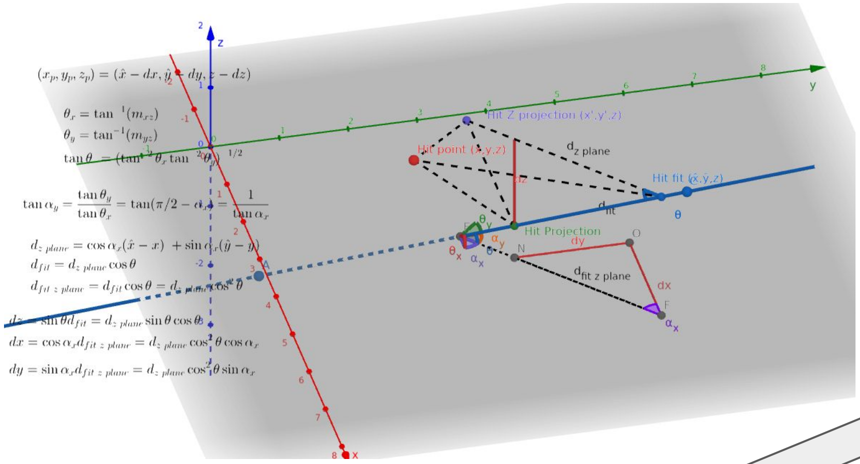
DNN fitter



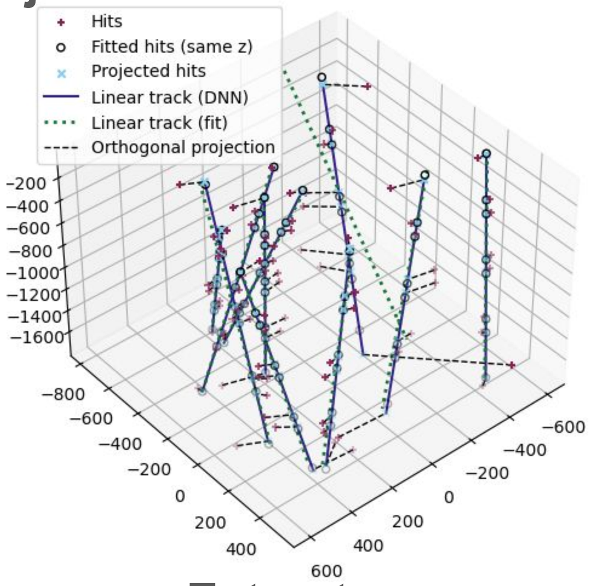
Slope m_{xz}
Slope m_{yz}
Intercept x
Intercept y
MAPE = $\frac{1}{N}\sum_i \frac{|\hat{y}_i - y_i|}{|y_i|}$ = 4.6%
MAPE = $\frac{1}{N}\sum_i \frac{|\hat{y}_i - y_i|}{|y_i|}$ = 4.8%
MAPE = $\frac{1}{N}\sum_i \frac{|\hat{y}_i - y_i|}{|y_i|}$ = 2.1%
MAPE = $\frac{1}{N}\sum_i \frac{|\hat{y}_i - y_i|}{|y_i|}$ = 2.6%
Regression (DNN)
True (linear fit)
$p_{reg} - p_{fit}$
$\log_{10}(E)$

DNN fitter

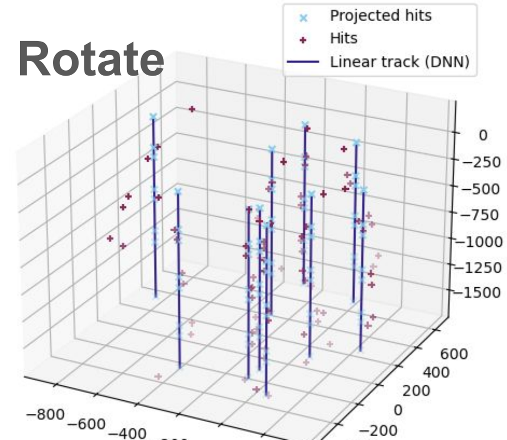
Calculate



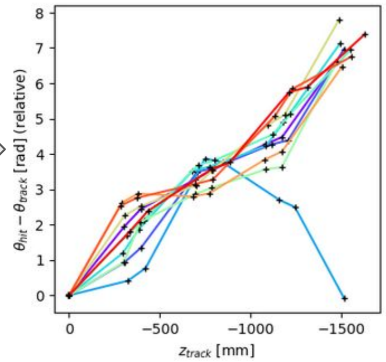
Project

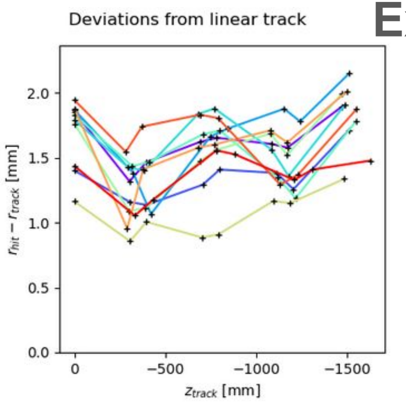


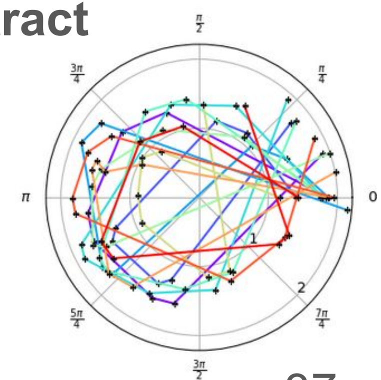
Rotate



Extract

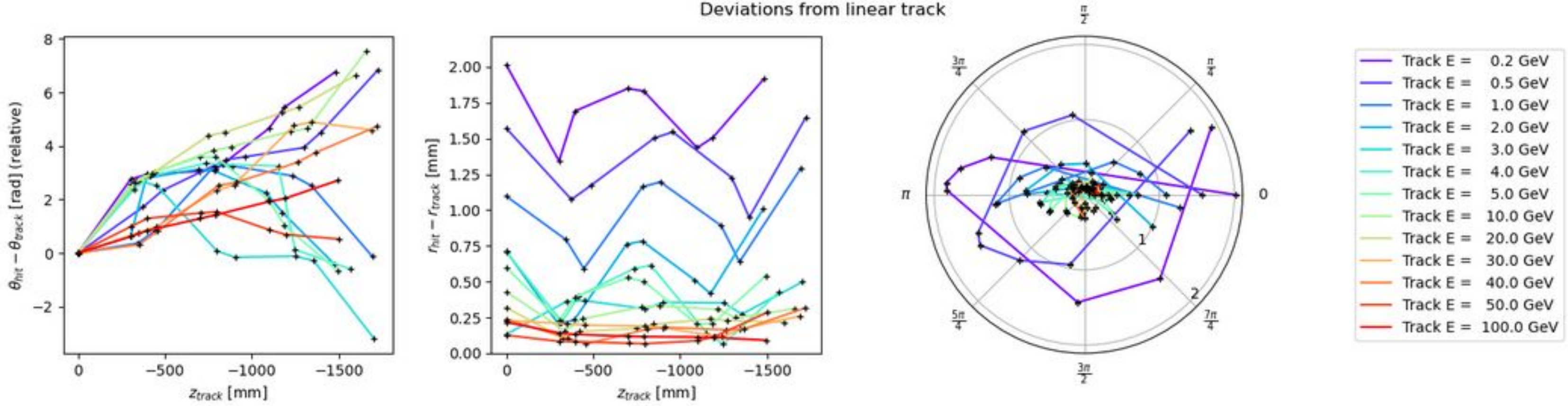




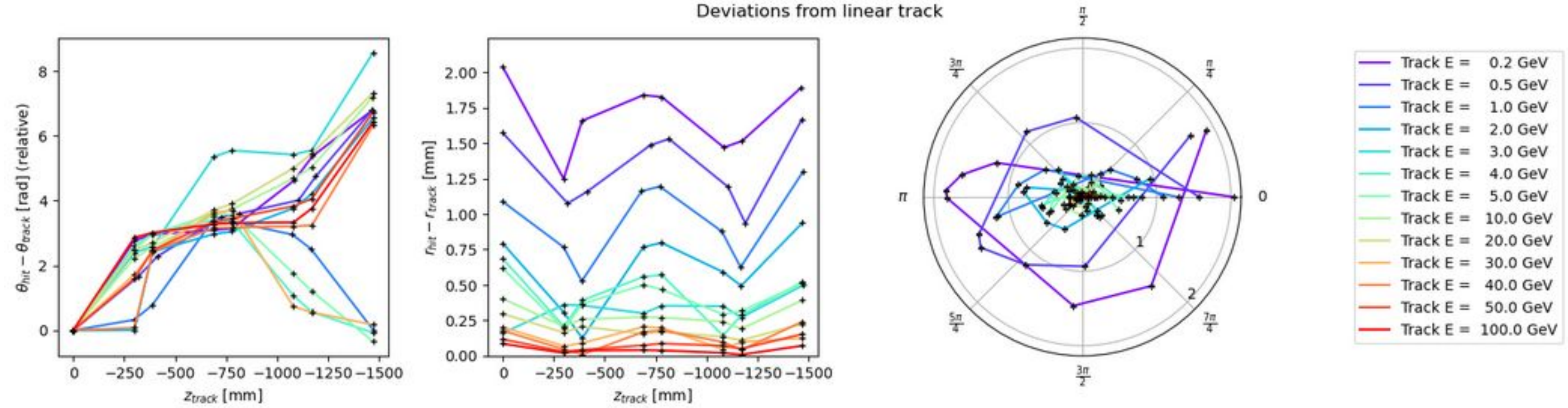


DNN fitter

DNN fitter



Classic fitter



More complex architectures : deviations as a graph

$$\mathbf{x}_i^{(k)} = \gamma^{(k)} \left(\mathbf{x}_i^{(k-1)}, \bigoplus_{j \in \mathcal{N}(i)} \phi^{(k)} \left(\mathbf{x}_i^{(k-1)}, \mathbf{x}_j^{(k-1)}, \mathbf{e_{ij}} \right) \right)$$

Many models on the market (see pytorch-geometric), the following are considered :

- Convolutional-based : in feature space $\boldsymbol{x}$ (in my case $\boldsymbol{x}_i$ = deviations $\{\Delta x_i, \Delta y_i, z_i\}$)
  - GCN [1] (2016) : $\oplus$ = mean and ϕ = linear layer
  - GAT(v2) [2][3] (2017,2021) : $\oplus$ = mean (concat per head) and ϕ = attention mechanism
  - GraphSAGE [4] (2017) : $\oplus$ = max, ϕ = linear layer + ReLU and γ = linear layer on concat
  - EdgeConv [5] (2018): $\oplus$ = mean/sum/max/… and ϕ = NN applied on concat($\boldsymbol{x}_i$,$\boldsymbol{x}_i$-$\boldsymbol{x}_j$)
  - GIN [6] (2018) : $\oplus$ = sum and γ = NN on (variable) weighted sum of $\boldsymbol{x}_i$ and the aggregation
  - PNA [7] (2020) : $\oplus$ = several aggregations combined and scaled, γ, ϕ = NN
- Point-based : depends on positions $\boldsymbol{p}_i$ (with potential addition of node features $\boldsymbol{x}_i$)
  - PointNet [8] (2016) : $\oplus$ = max, γ=NN and ϕ = NN applied on concat($\boldsymbol{x}_i$,$\boldsymbol{p}_i$-$\boldsymbol{p}_j$)
  - PointTransformer [9] (2020) : more complex attention mechanism based on $\boldsymbol{x}_i$ and $\boldsymbol{p}_i$-$\boldsymbol{p}_j$
  - PointGCN [10] (2020) : more complex with 3 NNs based on $\boldsymbol{x}_i$ and $\boldsymbol{p}_i$-$\boldsymbol{p}_j$