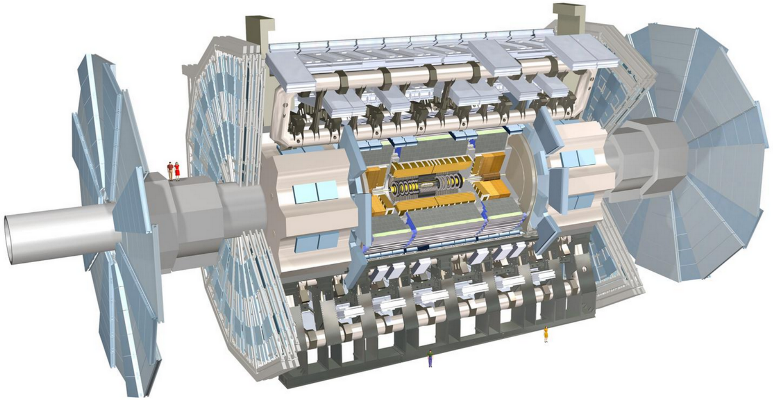
Beyond simple DNN regression calibration of hadronic jets in ATLAS

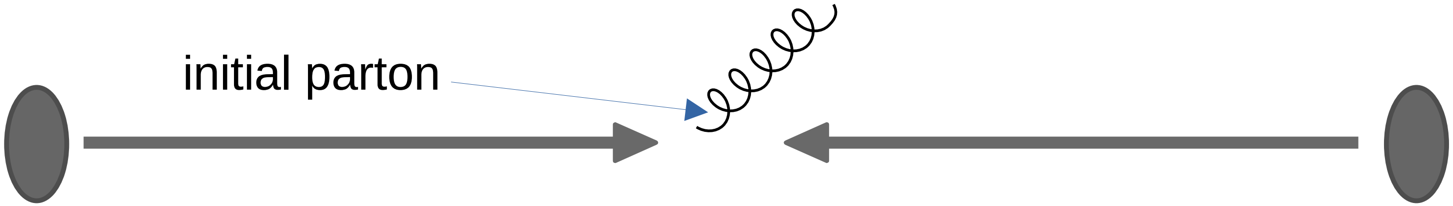
P-A Delsart & **Guillaume albouy**

Experimental context

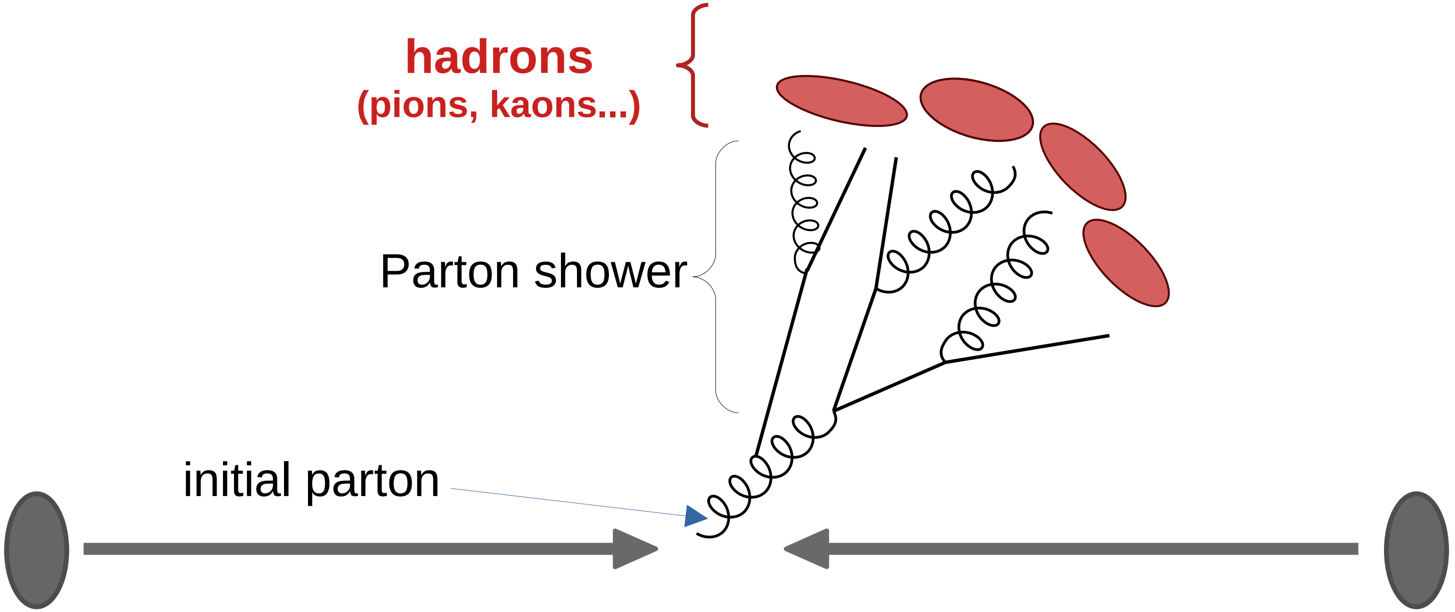
- Proton collision at LHC



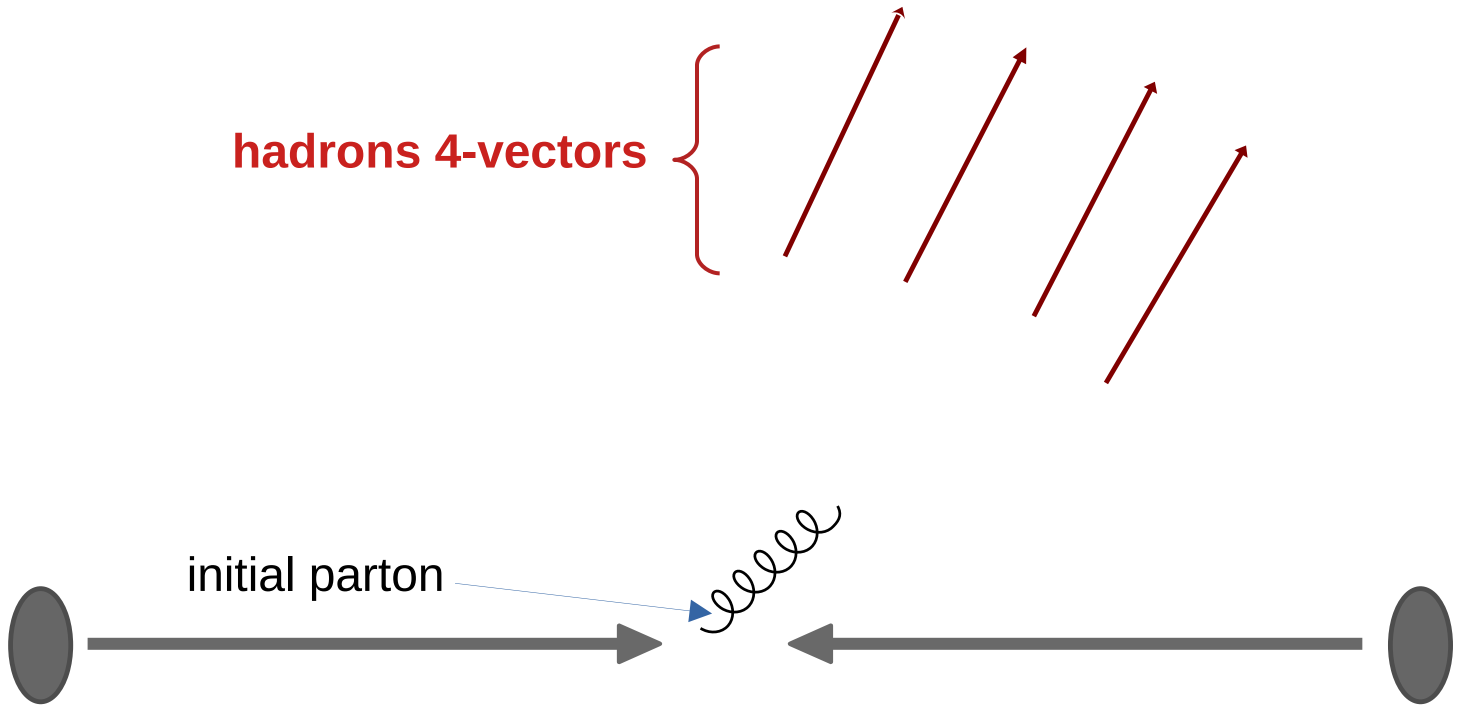
Hadronic jets



Hadronic jets



Hadronic jets

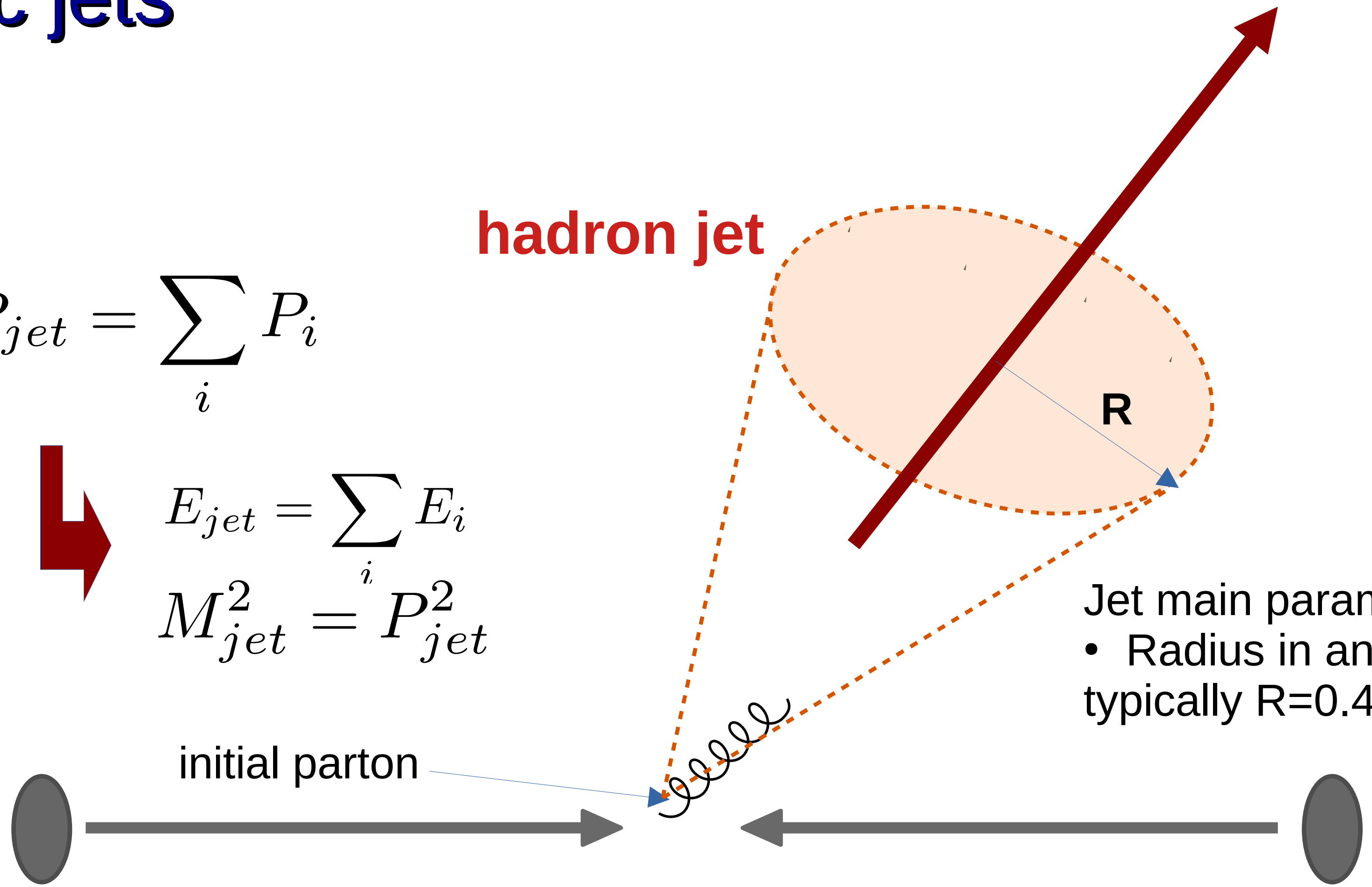


Hadronic jets

$$P_{jet} = \sum_i P_i$$

$$E_{jet} = \sum_i E_i$$

$$M^2_{jet} = P^2_{jet}$$

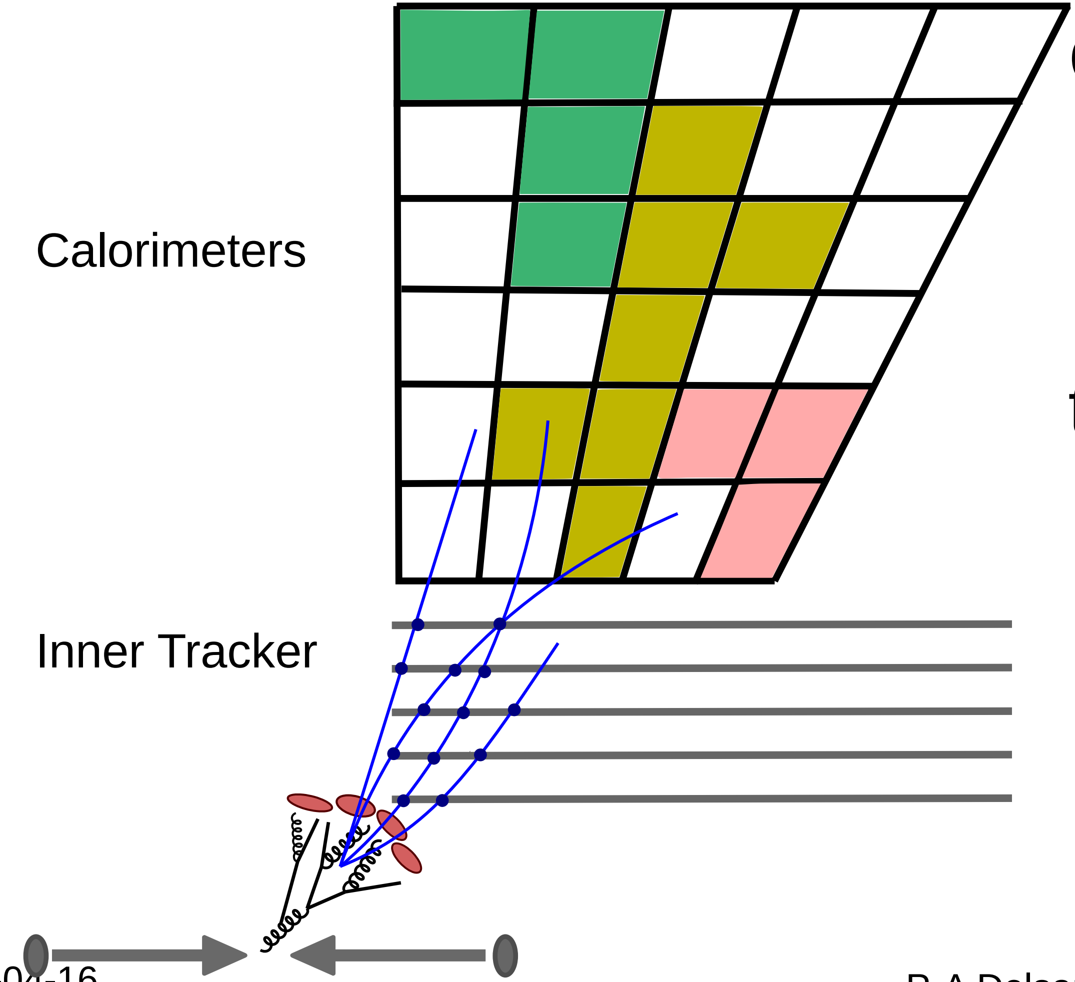


Jet main parameter:

- Radius in angular space

typically R=0.4 or R=1.0

Hadronic jets



Combine

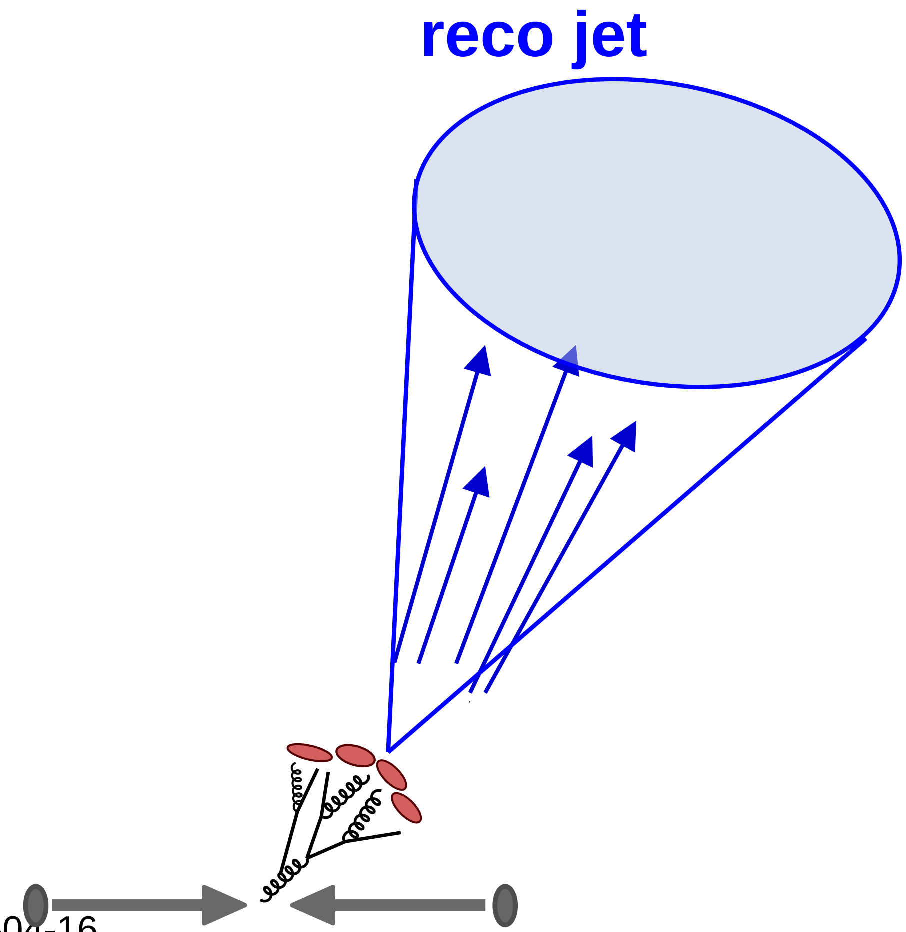
- Tracks
- Calorimeter E clusters

to form

reconstructed 4-vectors

("reco constituent")

Hadronic jets

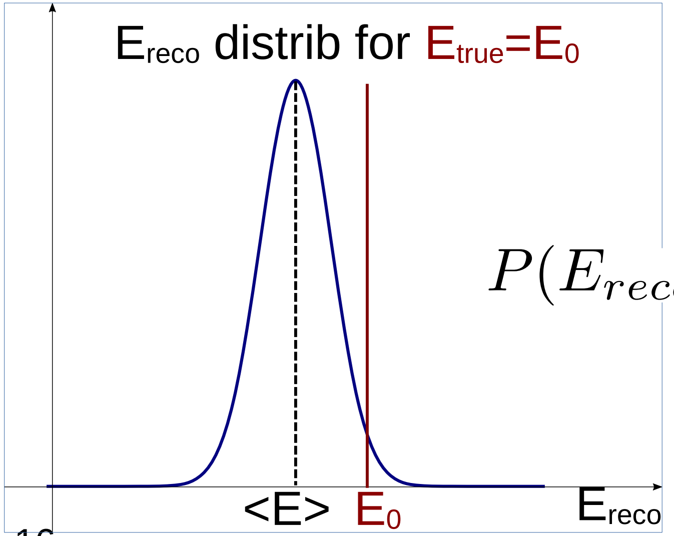


- Correspondence between hadron jets and reco jets

- Simulation can have reference quantities for reco jets

Measuring jets

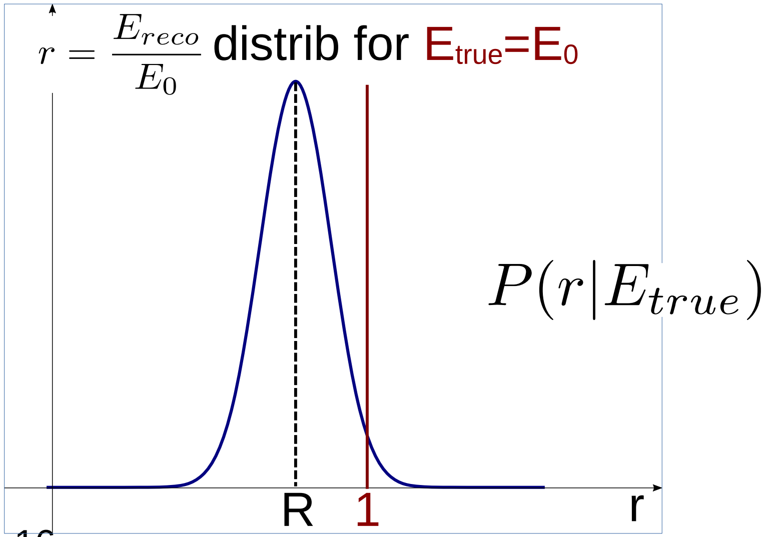
- Reconstructed Jet E and M **require** calibration
- For a given true jet, E_{true}, corresponds a **distribution** of possible E_{reco}
  - due to the nature of QCD and calorimeter showers
  - thus for $E_{reco} \rightarrow$ distribution of possible E_{true}

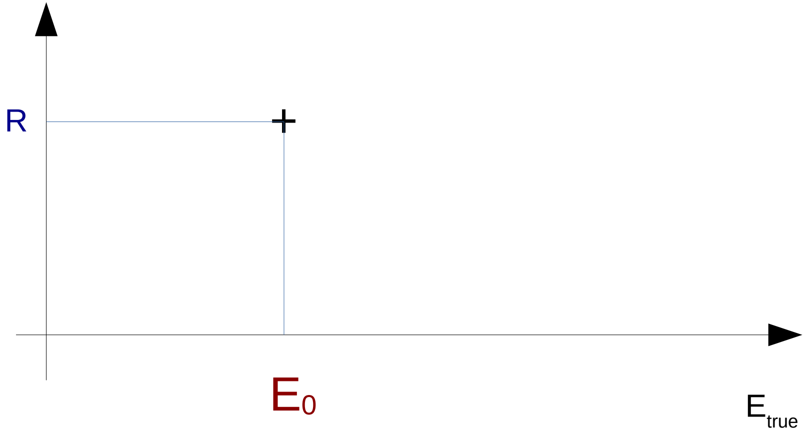


$$P(E_{reco}|E_{true})$$

Measuring jets

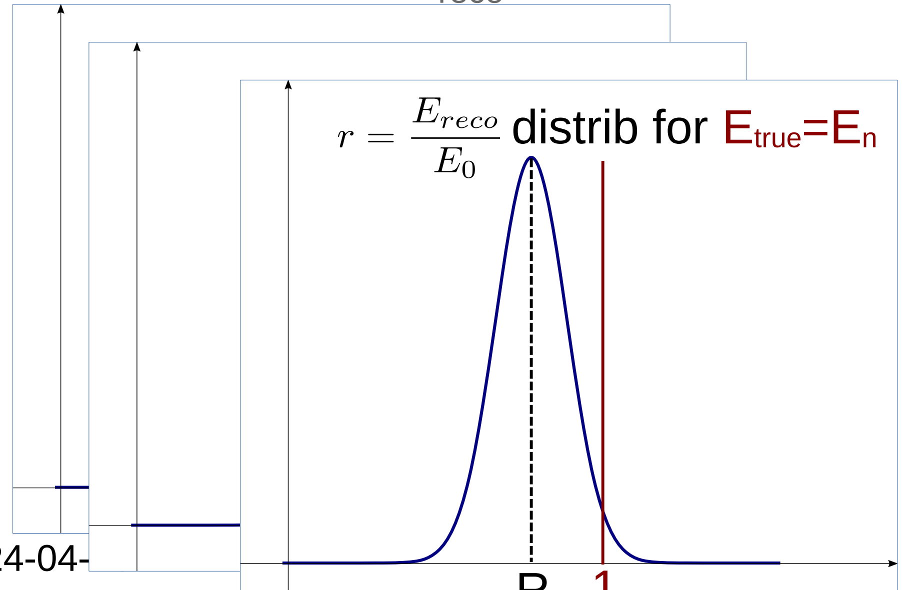
- Reconstructed Jet E and M **require** calibration
- For a given true jet, E_{true}, corresponds a **distribution** of possible E_{reco}
  - due to the nature of QCD and calorimeter showers
  - thus for $E_{reco} \rightarrow$ distribution of possible E_{true}



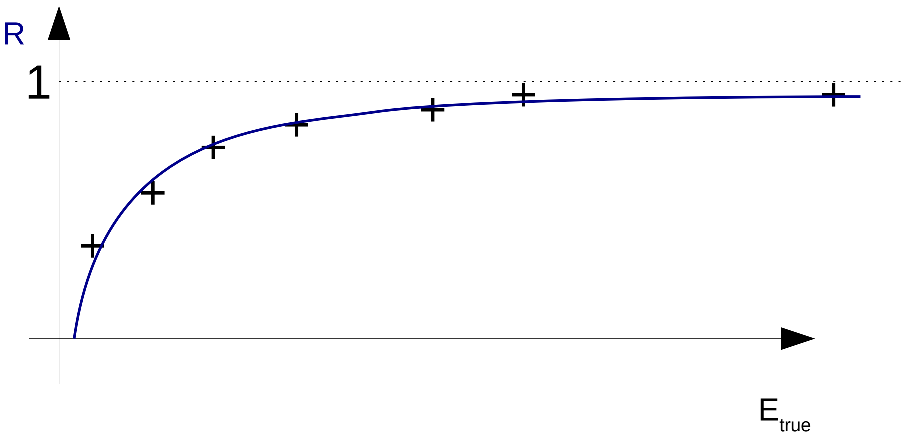


Measuring jets

- Reconstructed Jet E and M **require** calibration
- For a given true jet, E_{true}, corresponds a **distribution** of possible E_{reco}
  - due to the nature of QCD and calorimeter showers
  - thus for E_{reco} → distribution of possible E_{true}

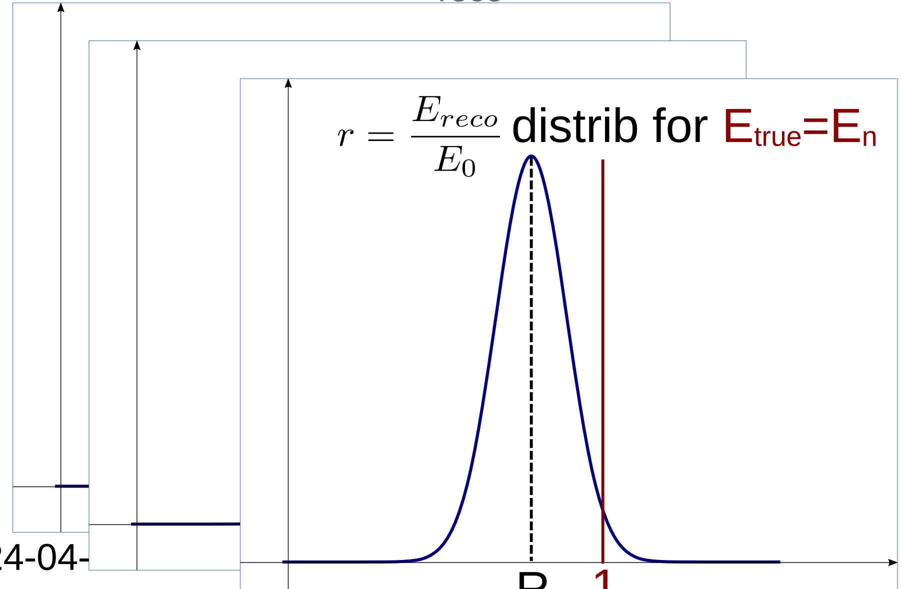


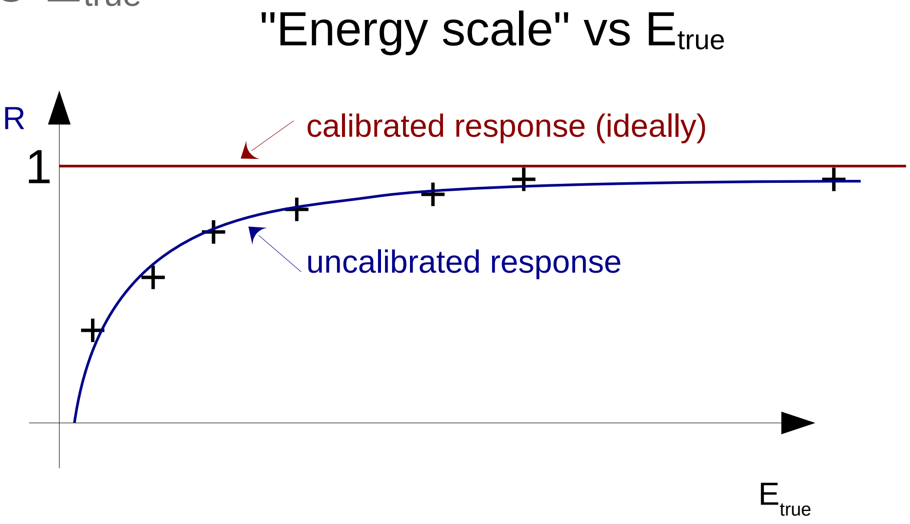
"Energy scale" vs E_{true}



Measuring jets

- Reconstructed Jet E and M **require** calibration
- For a given true jet, E_{true}, corresponds a **distribution** of possible E_{reco}
  - due to the nature of QCD and calorimeter showers
  - thus for $E_{reco} \rightarrow$ distribution of possible E_{true}





The calibration problem

Inputs:
reconstructed
quantities

output:
calibrated quantities
(E and Mass)

$$x_{reco} = \begin{pmatrix} E_{reco} \\ M_{reco} \\ \phi \\ \dots \end{pmatrix}$$

???

?

The calibration problem

Inputs:
reconstructed
quantities

output:
calibrated quantities
(E and Mass)

$$x_{reco} = \begin{pmatrix} E_{reco} \\ M_{reco} \\ \phi \\ \ldots \end{pmatrix}$$

???

$$E_{calib} = \frac{E_{reco}}{\mathrm{mode}(P(r|x_{reco}))}$$

most probable response

*this solution is a practical choice & not necessarily mathematically valid

The calibration problem

Inputs:
reconstructed
quantities

output:
calibrated quantities
(E and Mass)

$$x_{reco} = \begin{pmatrix} E_{reco} \\ M_{reco} \\ \phi \\ \ldots \end{pmatrix} \Rightarrow \boxed{\textbf{DNN}} \Rightarrow y_{pred} = \mathrm{mode}(P(r|x_{reco})) \Rightarrow E_{calib} = \frac{E_{reco}}{\mathrm{mode}(P(r|x_{reco}))}$$

most probable response

*this solution is a practical choice &
not necessarily mathematically valid

The ML problem

Design & train a DNN to learn simultaneously the **mode** of the E and mass responses

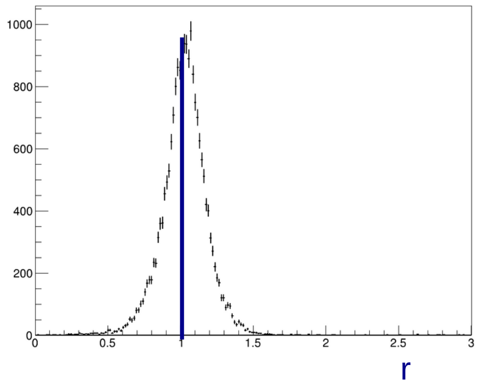
Requirements:
- calibrated E scale = 1 ± 1%
- calibrated M scale = 1 ± 5%
- on all the phase space (E~0.1 → 4TeV)
- with improved resolution

Solutions

- Choose a proper loss function
- Encode the jet angular position
- Network Architecture
- Training procedure

Loss function choice

Learning the mode of distributions



- We want to predict the **mode** of the distribution
  - In training examples, the **target** ($r=E_{reco}/E_{true}$) is just 1 number out of this distribution
  - NOT the mode
- The choice of the loss function is important
  - $L=||r_{target}-r_{pred}||^2$ → learns the mean
  - $L=|r_{target}-r_{pred}|$ → learns the median
  - $L = \delta(r_{target}-r_{pred})$ → learns the mode
    - unusable in practice

Losses for mode learning

- Leaky Gaussian Kernel :

$$L_{LGK} = \exp(-\frac{(x_{\text{target}} - x_{\text{pred}})^2}{2\alpha}) + \beta|x_{\text{target}} - x_{\text{pred}}|$$

  - approximation of Dirac's delta
    - α and β are fixed hyper-parameters

Losses for mode learning

Mixture Density Network

- First, assume distrib is gaussian :
  - μ is the **mode** !
  - μ, σ are estimated by the DNN, functions of θ=(E_{reco},…)

$$P(r|\theta) \simeq e^{\frac{-(r-\mu)^2}{2\sigma^2}}$$

- Given inputs, μ and σ are obtain when maximizing the likelihood :

$$\mathrm{LH} = \prod_{i \in \mathrm{inputs}} P(r_i|\theta_i)$$

- In practice :
  - have the NN predicts both **μ and σ**
  - choose the **log likelihood as the loss**
-

$$\mathrm{loss}((\mu_{\mathrm{pred}}, \sigma_{\mathrm{pred}}), r_{\mathrm{target}}) = \log(\sigma_{\mathrm{pred}}) + \frac{1}{2}(\frac{\mu_{\mathrm{pred}} - r_{\mathrm{target}}}{\sigma_{\mathrm{pred}}})^2$$

Losses for mode learning

- Mixture Density Network
- … but real r distribution are not gaussian ?
- We can use other underlying assumption
  - asymmetric gaussian

$$P_{\text{asym}}(x) \sim \begin{cases} e^{(x-\mu)^2/2\sigma_1^2} & \text{if } x < \mu \\ e^{(x-\mu)^2/2\sigma_2^2} & \text{if } x \geq \mu \end{cases}$$

  - truncated gaussian
    - ignore tails+focus on mode

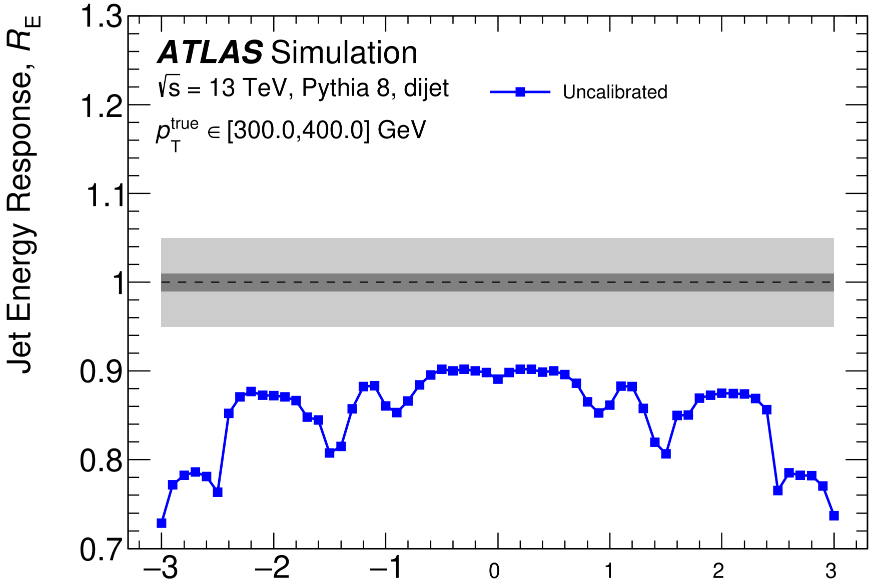
$$P_{\text{trunc}}(x) \sim \begin{cases} e^{(x-\mu)^2/2\sigma^2} & \text{if } |x-\mu| < N\sigma \\ 0 & \text{otherwise} \end{cases}$$

- can change/tune loss during training

Input encoding

Input encoding

- Detector segmented in different subdector
- Strong response variations vs angular position
  - very difficult to model
- Solution : encode the angular position
  - "η annotation"
  - create new inputs out of angular position
    - 1 new input for each detector region

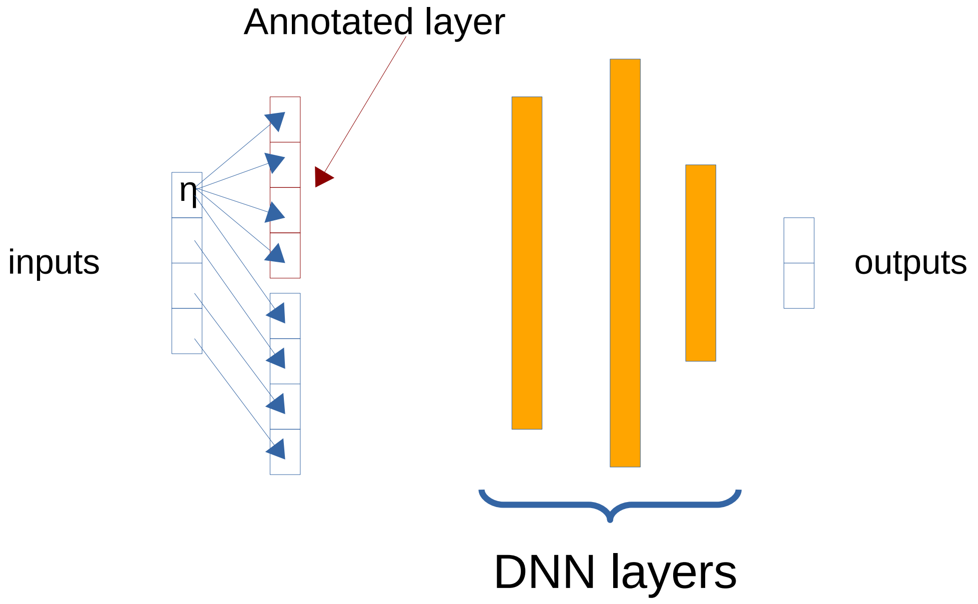


Angular position

Implement detector knowledge into the NN

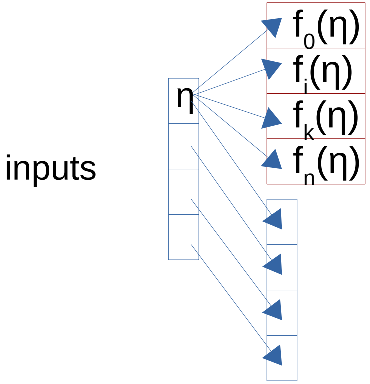
Input Annotation

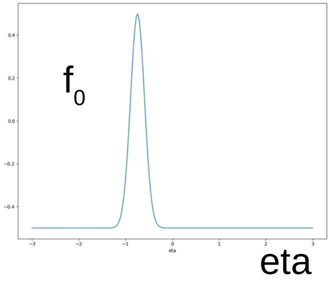
- Increase the input by adding “features”

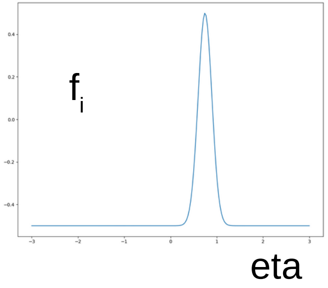


Input Annotation

- Increase the input by adding “features”





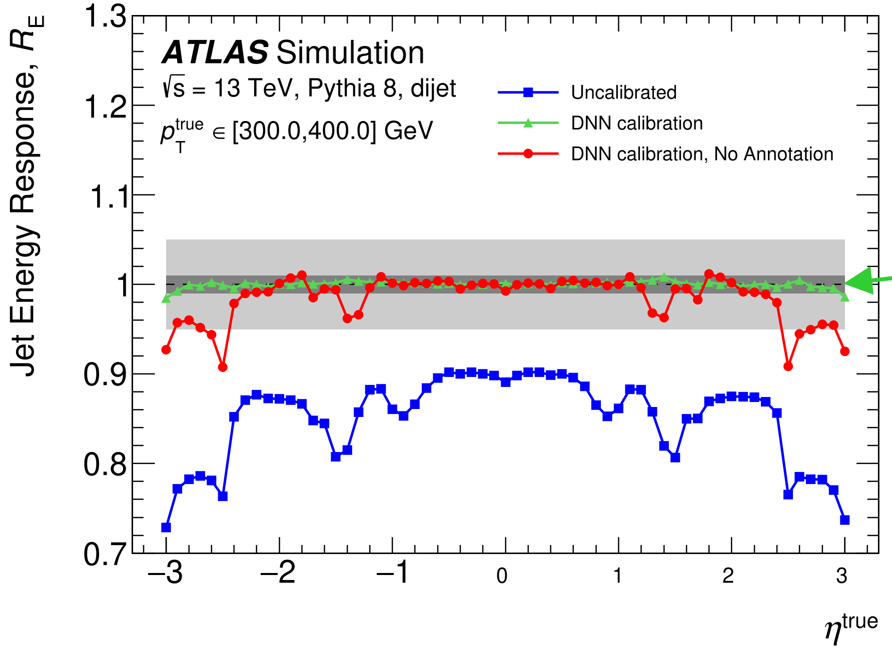


Gaussian Annotation

- Gaussian centers set on each detector region

Intention : add the “distance to the region” information to the NN

Input Annotation

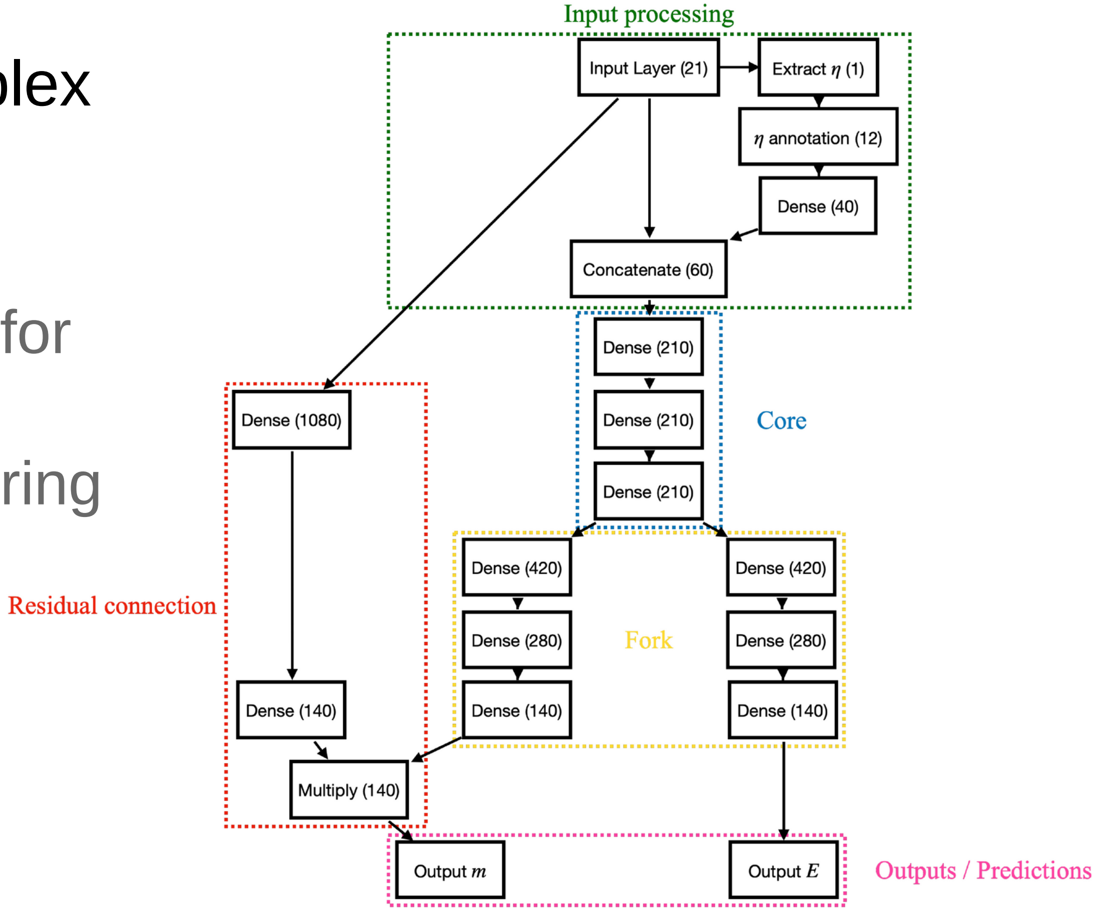


Input encoding allows to perfectly recover from detector boundaries

Network architecture

Network architecture

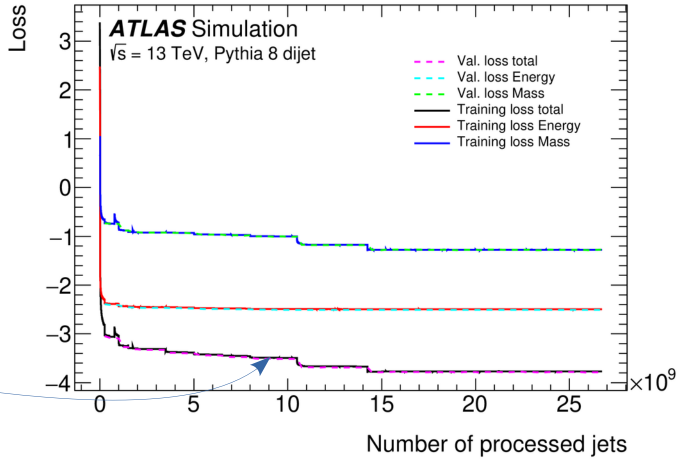
- Requirements impose a complex architecture
- Fork
  - specialize different weights for E and mass calib
  - allow to "freeze" weights during training
- Residual connection
  - mass calib much harder
  - help to converge on better weights



Training procedure

Training procedure

- Naive training for N epochs is not enough
  - ex: stopping here doesn't work
  - response≠1 in some region of phase space
  - mass response not calibrated enough

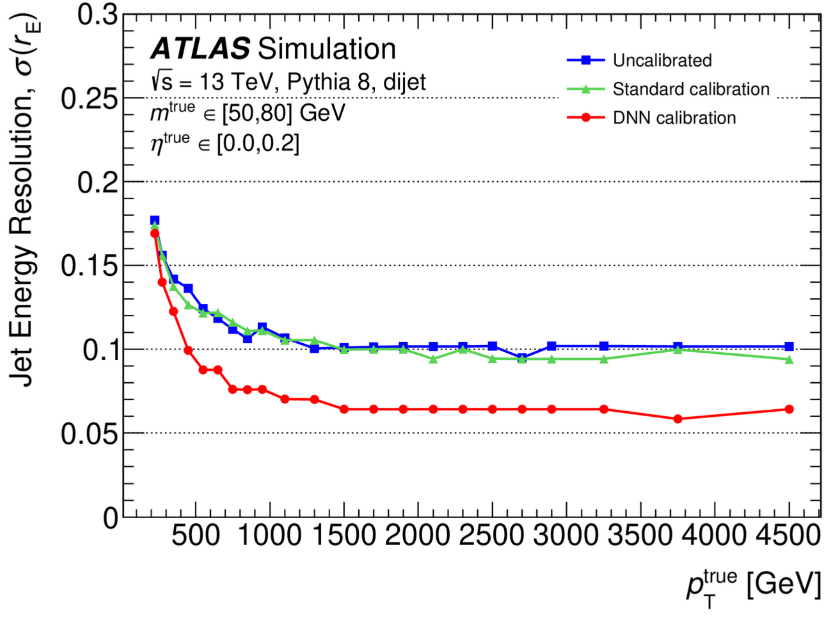


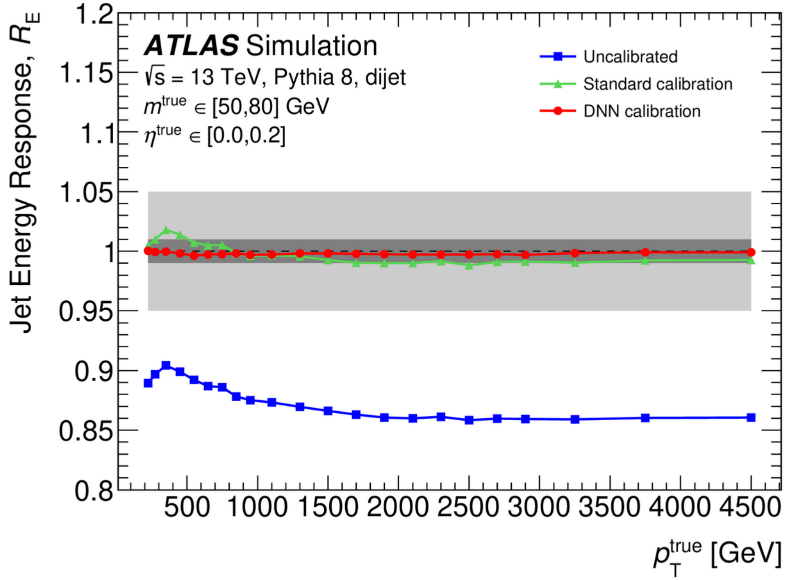
- Evolve loss functions to help with convergence to the mode everywhere
- Freeze all weights related to E and continue mass weights trainings

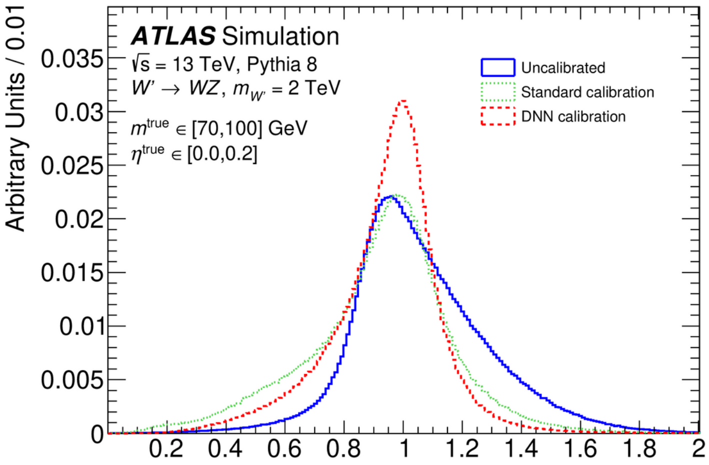
Steps	N°	Number of epochs	Batch size	Learning rate	Loss
Initialisation	1	2	15 000	10^{-3}	MDNA
	2	2	25 000	10^{-3}	MDNA
	3	2	35 000	10^{-3}	MDNA truncated (4.0σ)
	4	2	15 000	10^{-3}	MDNA truncated (3.5σ)
Common training	5	6	95000	10^{-3}	MDNA truncated (3.5σ)
	6	6	95 000	10^{-3}	MDNA truncated (3.5σ)
	7	6	125 000	10^{-3}	MDNA truncated (3.2σ)
	8	6	125 000	10^{-3}	MDNA truncated (3.2σ)
	9	10	155 000	5.10^{-4}	MDNA truncated (3.0σ)
	10	15	95 000	10^{-5}	MDNA truncated (E: 3.0σ, m: 2.0σ)
Exclusive mass training	11	50	95 000	10^{-5}	MDN truncated (1.0σ)

Results

Excellent Performances !







Conclusions

- Solved 4 difficulties to implement a complete E&mass calibration of ATLAS
- Excellent performances → public result (to be plublished in MSLT)
  - including on "types" of jet not seen during training
- Remaining work lines
  - what/how input variables impact predictions ?
  - robust criteria for stopping training procedure ?

Technical details & difficulties

Technicalities

- Framework : own code build on keras/tensorflow
- Data flow : custom solution
  - O(100M) examples x N features > available memory
  - ROOT ntuple → read by uproot → numpy array → tensorflow
  - Other better technical solutions ?
- Computing : using CC-IN2P3 GPU farm
  - works very well !
  - good interactions with CC experts

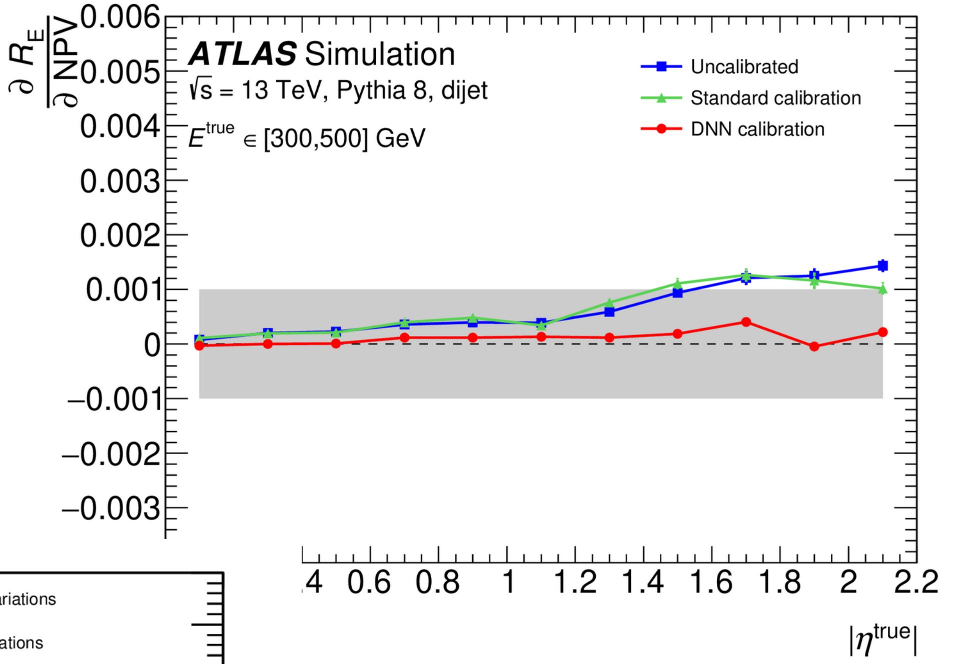
Difficulties

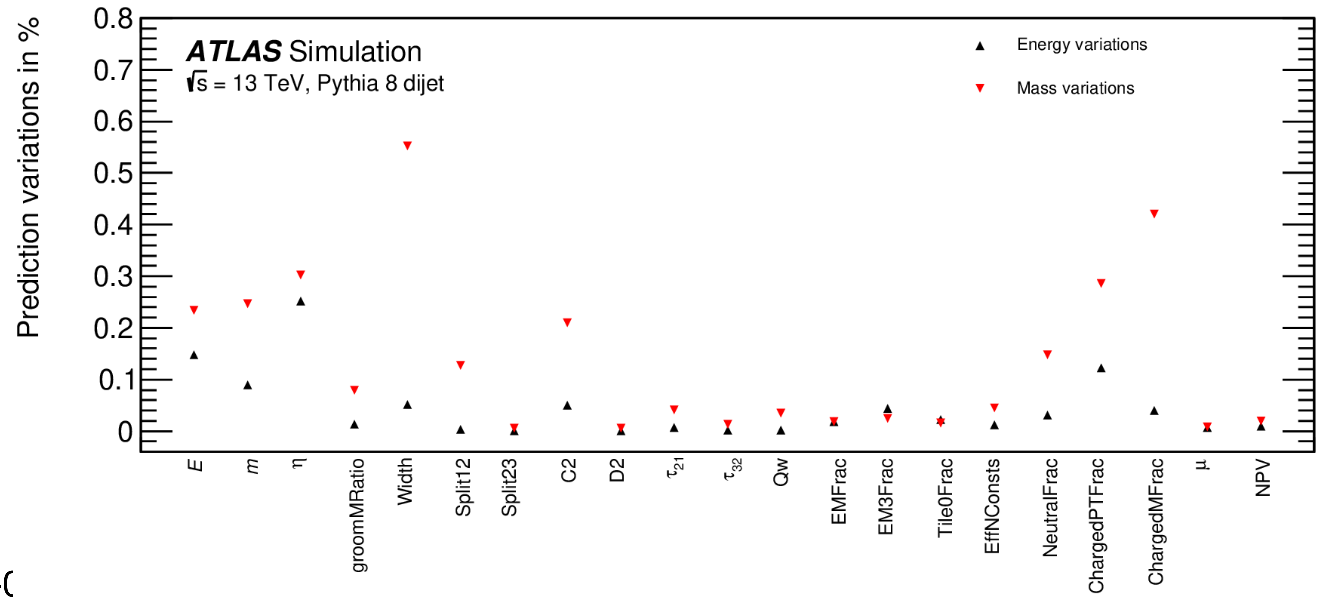
- NN convergence issues solved with
  - Inputs & targets normalization
  - use of weights regularization (l2 or max-norm)
  - paying attention to activation functions !
- The Loss is not enough
  - different NN can reach similar minimal loss YET having different perfs according to other metrics
    - workaround by complex training procedures
  - Is it a sign something is wrong ?

Difficulties

- GraphNN difficulties/open questions
  - Isn't the system "underconstrained" ?
    - tuning constituent-level corrections from jet-level constraints
    - will it be able to converge to a physical solution ?
    - How to enforce valid/useful constraints if needed ?
  - Sometimes training Loss starts to **increase** continuously
    - yet model weights seem reasonable
  - In some setup GNN converges to ~constant correction factors for most of the input constituents...

Back-up


ATLAS Simulation
√s = 13 TeV, Pythia 8, dijet
E^{true} ∈ [300,500] GeV
Uncalibrated
Standard calibration
DNN calibration
$\frac{\partial R_E}{\partial \mathrm{NPV}}$
$|\eta^{true}|$


ATLAS Simulation
√s = 13 TeV, Pythia 8 dijet
Energy variations
Mass variations
Prediction variations in %
E
m
η
groomMRatio
Width
Split12
Split23
C2
D2
τ_{21}
τ_{32}
Qw
EMFrac
EM3Frac
Tile0Frac
EffNConsts
NeutralFrac
ChargedPTFrac
ChargedMFrac
μ
NPV