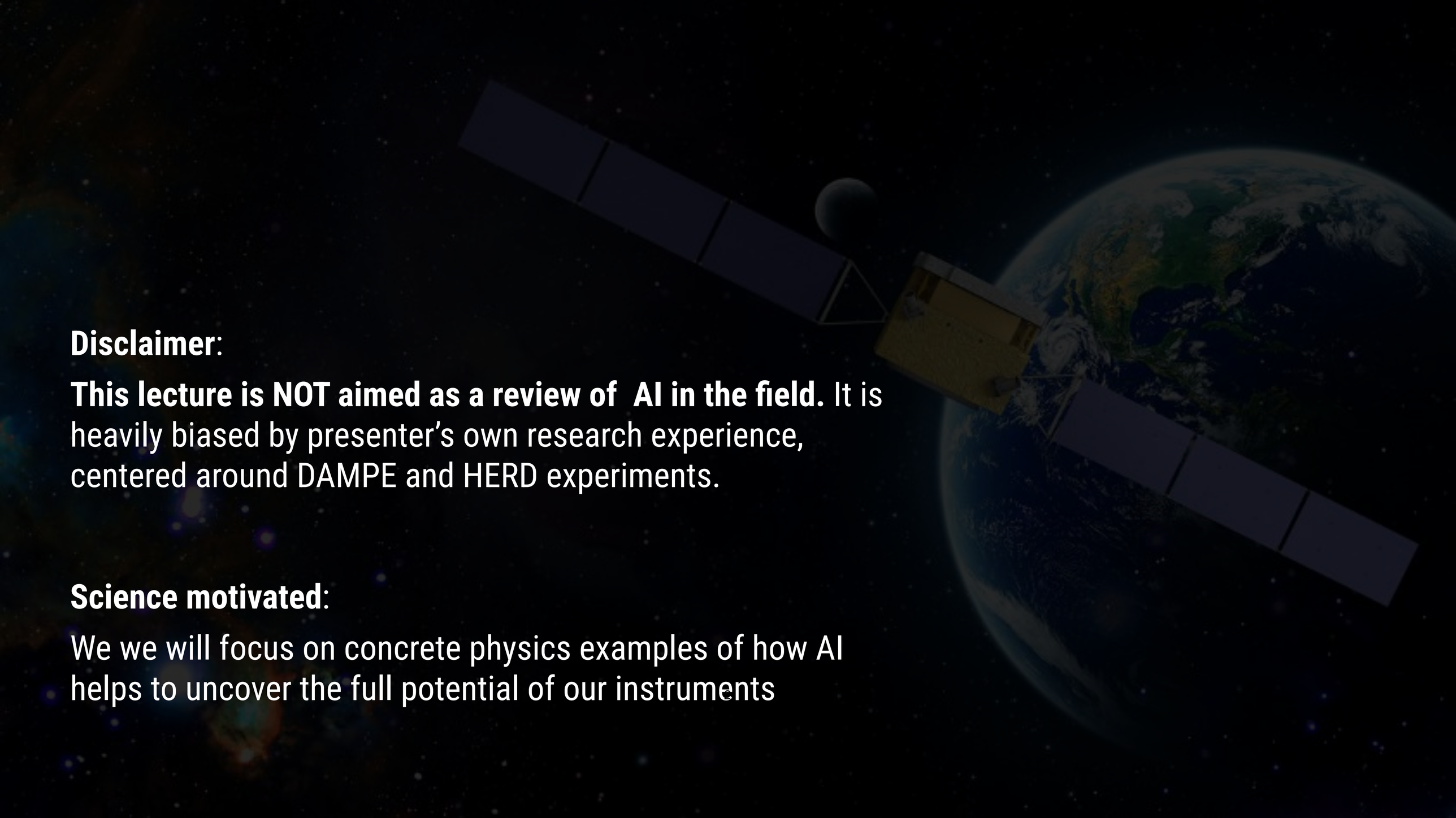
Deep Learning & cosmic ray detection in space at high-energy frontier

Andrii Tykhonov

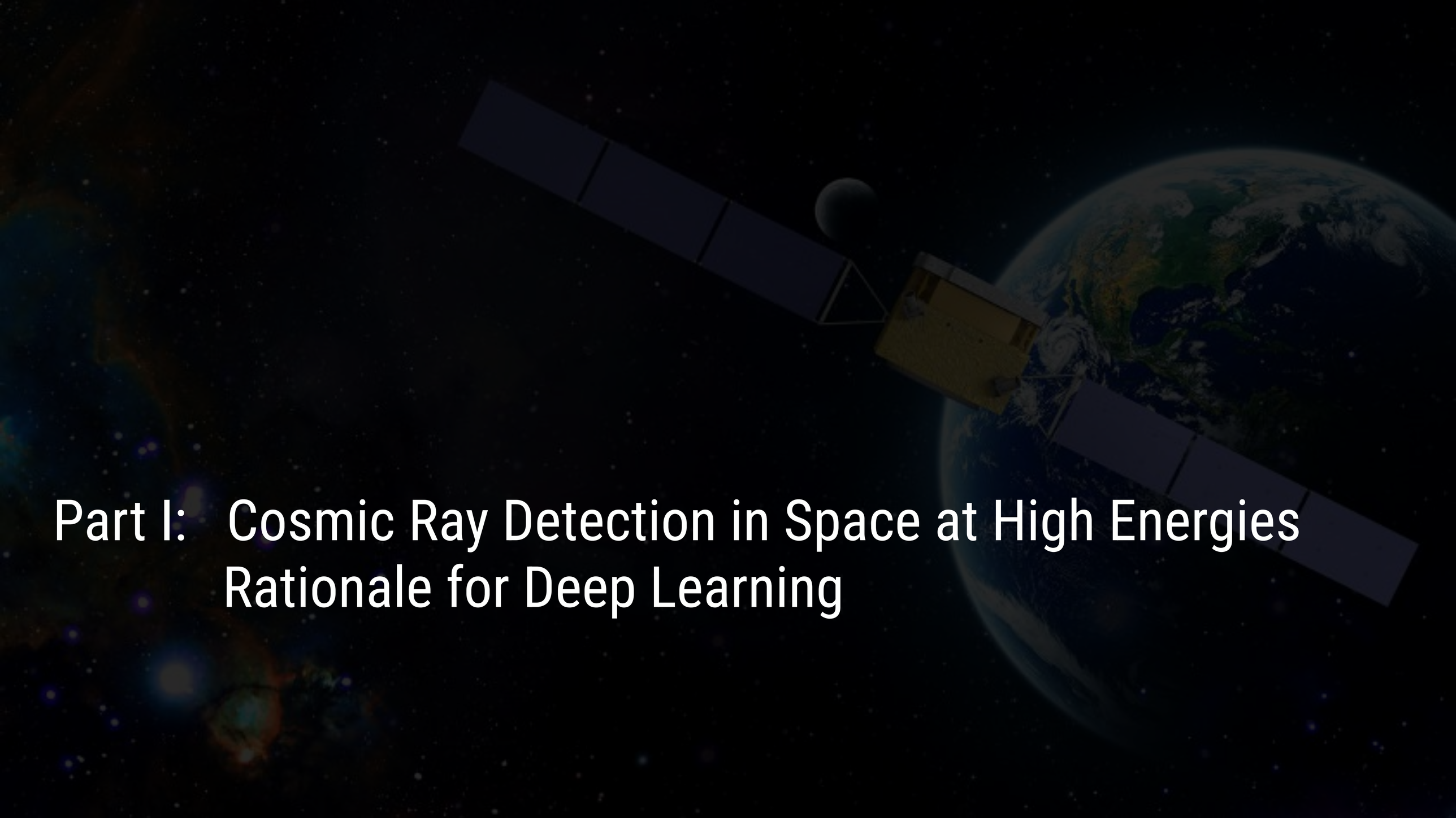


Disclaimer:

This lecture is NOT aimed as a review of AI in the field. It is heavily biased by presenter's own research experience, centered around DAMPE and HERD experiments.

Science motivated:

We we will focus on concrete physics examples of how AI helps to uncover the full potential of our instruments



Part I: Cosmic Ray Detection in Space at High Energies Rationale for Deep Learning

Cosmic Ray detection

Space *(satellites, ISS, balloons)*

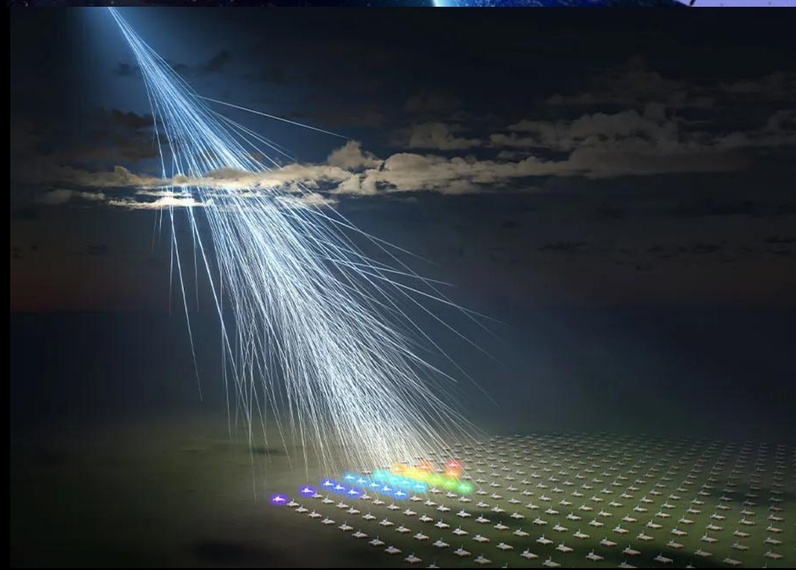
Very precise — measurement errors reaching 1% level

Limited in energy up to ~hundreds of TeV

Ground-based

Cover very large ares ~O(100) km^2 and reach maximum possible energies — up to 10^{19} eV (~millions of TeV)

Not very precise — measurement errors more than 100%!

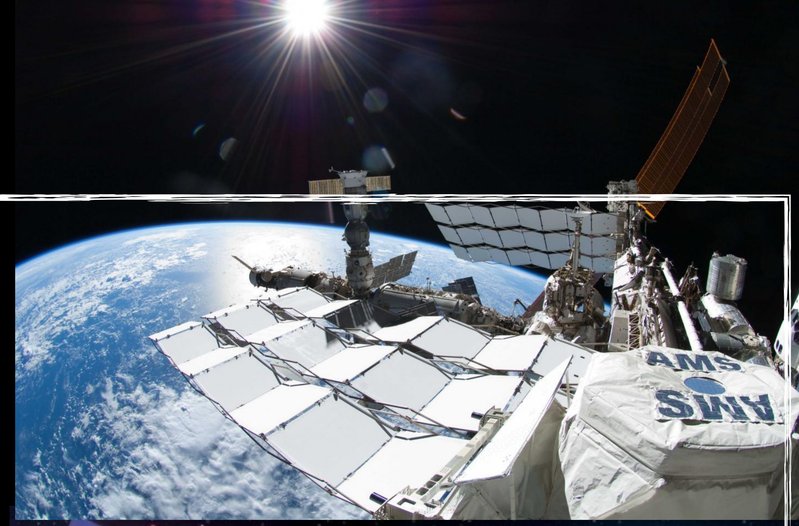


Cosmic Ray detection

Space *(satellites, ISS, balloons)*

Very precise — measurement errors reaching 1% level

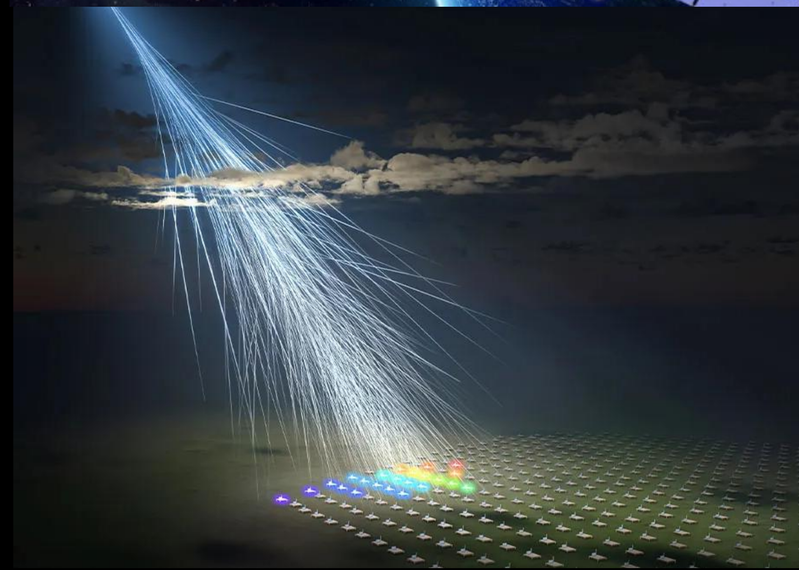
Limited in energy up to ~hundreds of TeV



Ground-based

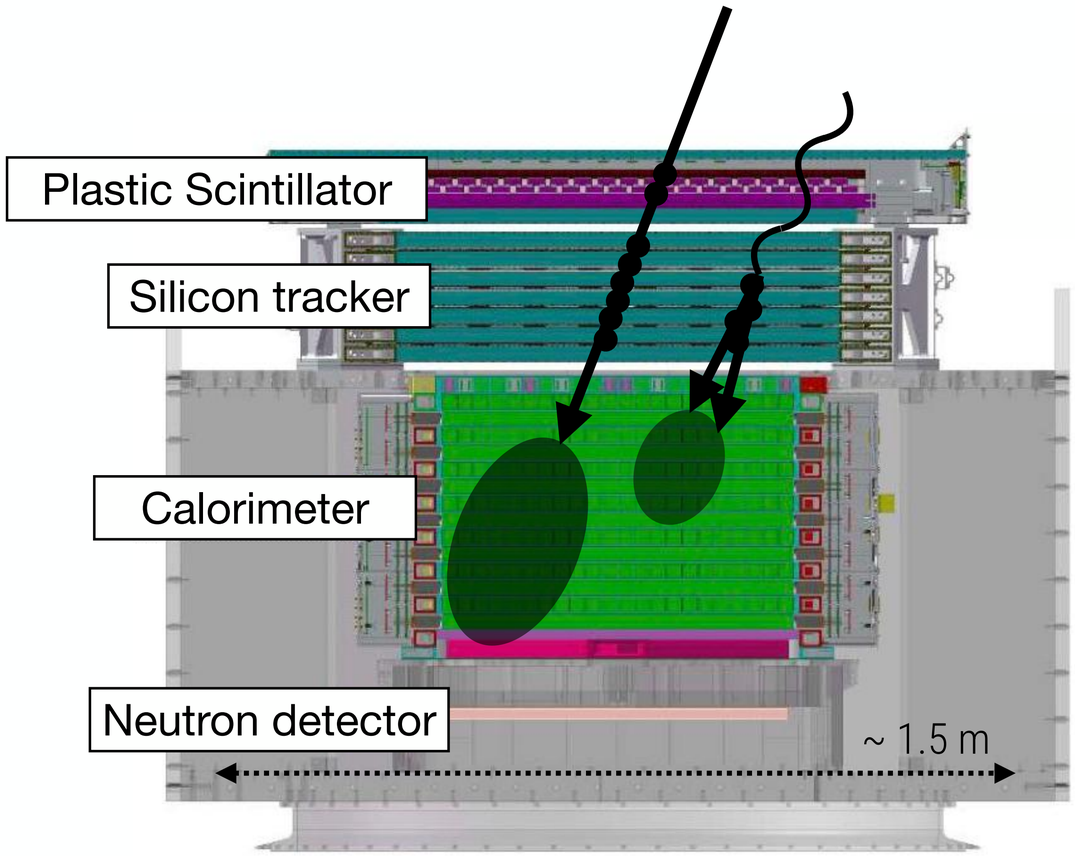
Cover very large ares ~O(100) km^2 and reach maximum possible energies — up to 10^{19} eV (~millions of TeV)

Not very precise — measurement errors more than 100%!

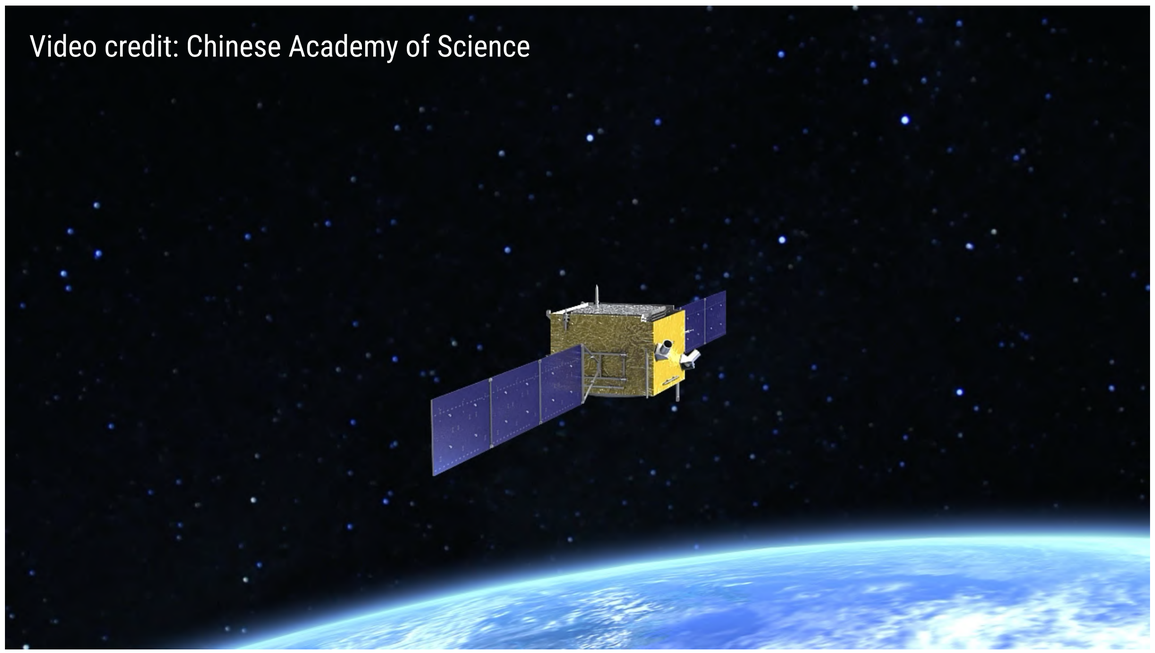


DArk Matter Particle Explorer (DAMPE)

DAMPE – a high-energy cosmic ray space mission

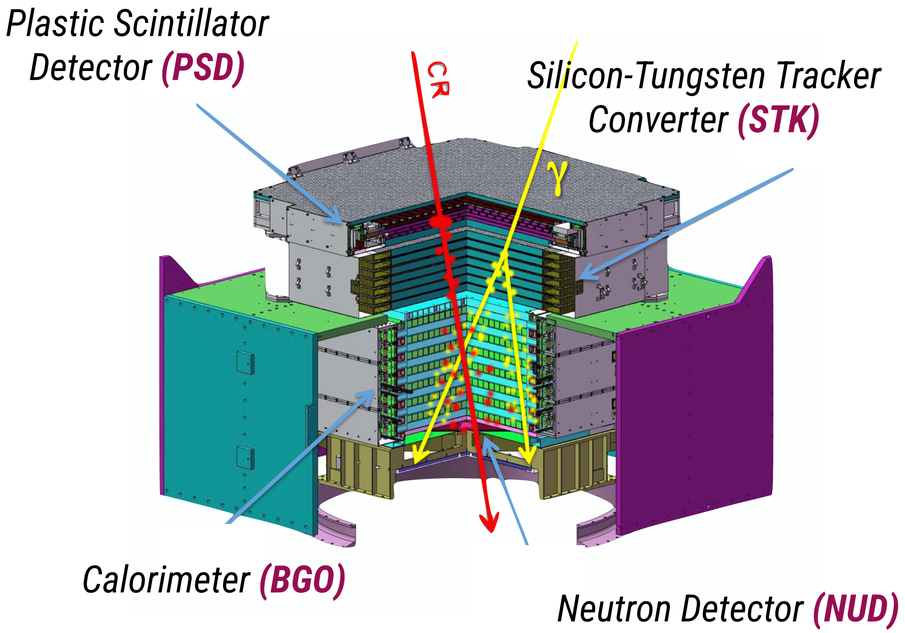


Example of a typical cosmic ray (let particle) gamma ray (right particle) interaction in DAMPE



Joined project of China, Switzerland and Italy (launched in 2015)

DArk Matter Particle Explorer (DAMPE)



PSD

- Z identification up to *Ni* (Z=28)
- γ anti-coincidence signal

STK

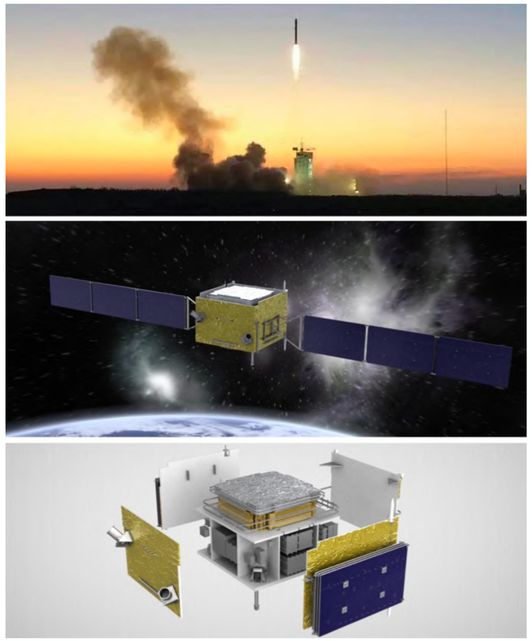
- Position solution ***~50 micron***
- γ-ray angular resolution ***0.5°—0.1°***
- Absolute Charge (Z) identification

BGO

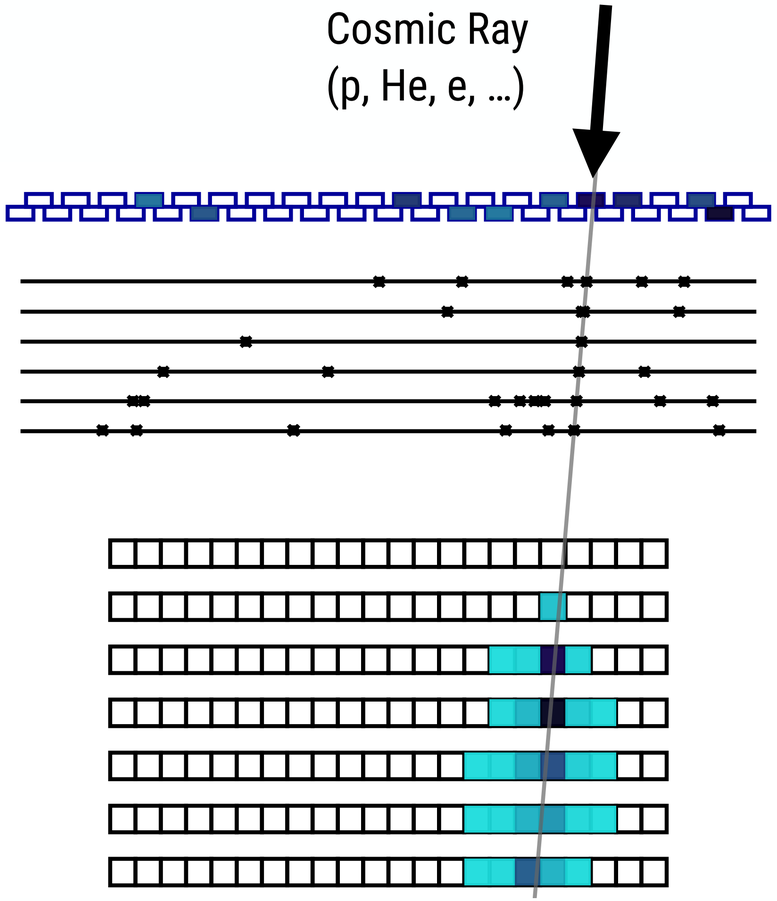
- 31 X_0 — thickest in space
- e/γ detection **GeV — 10 TeV**
- *p*/ions: **50 GeV — PeV**

NUD

- Additional e/p rejection power

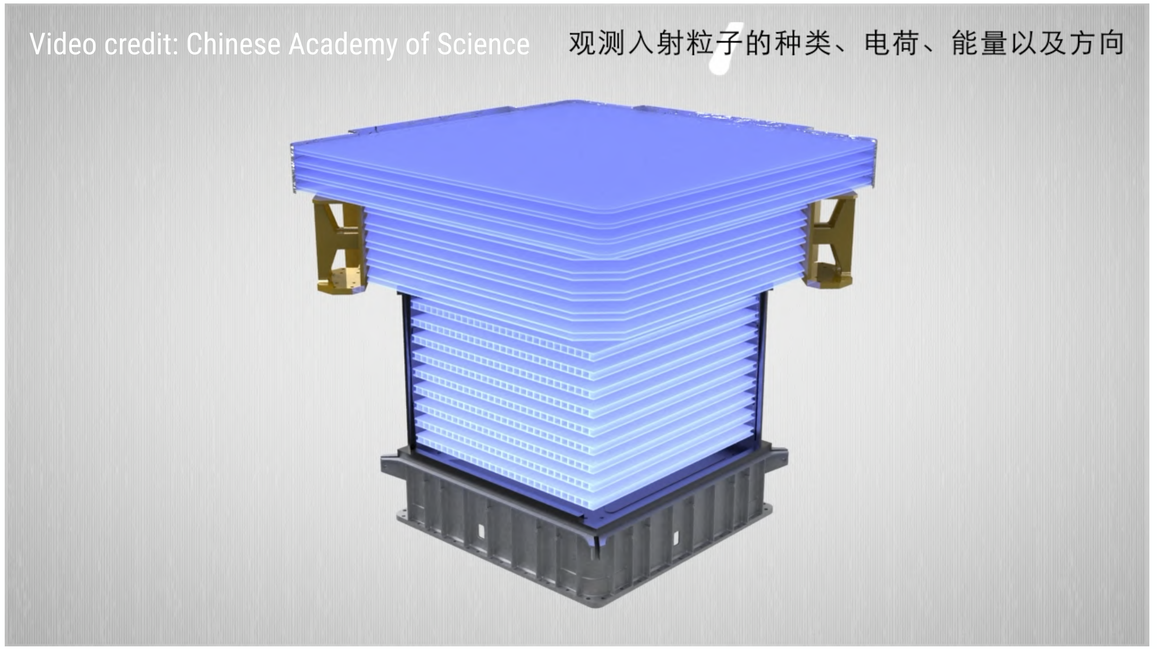


DArk Matter Particle Explorer (DAMPE)

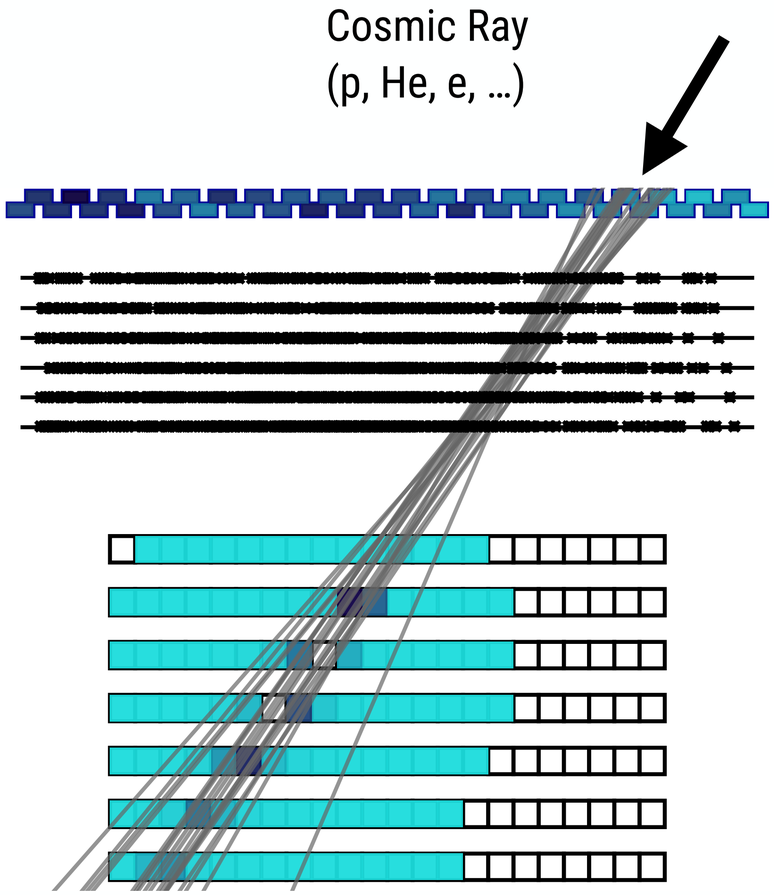


Example of a typical interaction in DAMPE (one projection) for a ~ 1 TeV cosmic ray

As a baseline for data analysis, classical well-understood techniques are use: Kalman filter for reconstructing particle track, linear regression for calorimeter image processing, etc.



DArk Matter Particle Explorer (DAMPE)



Example of a typical interaction in DAMPE (one projection) for a ~ 100 TeV cosmic ray

Analysis becomes challenging at > ~ 100 TeV due to back-scattering of secondaries particle from calorimeter.

Identifying a track of a Cosmic Ray in a vast noise of secondary tracks is a **search for a needle in a haystack!**

Q: Why it is essential to precisely identify a track of a cosmic ray?

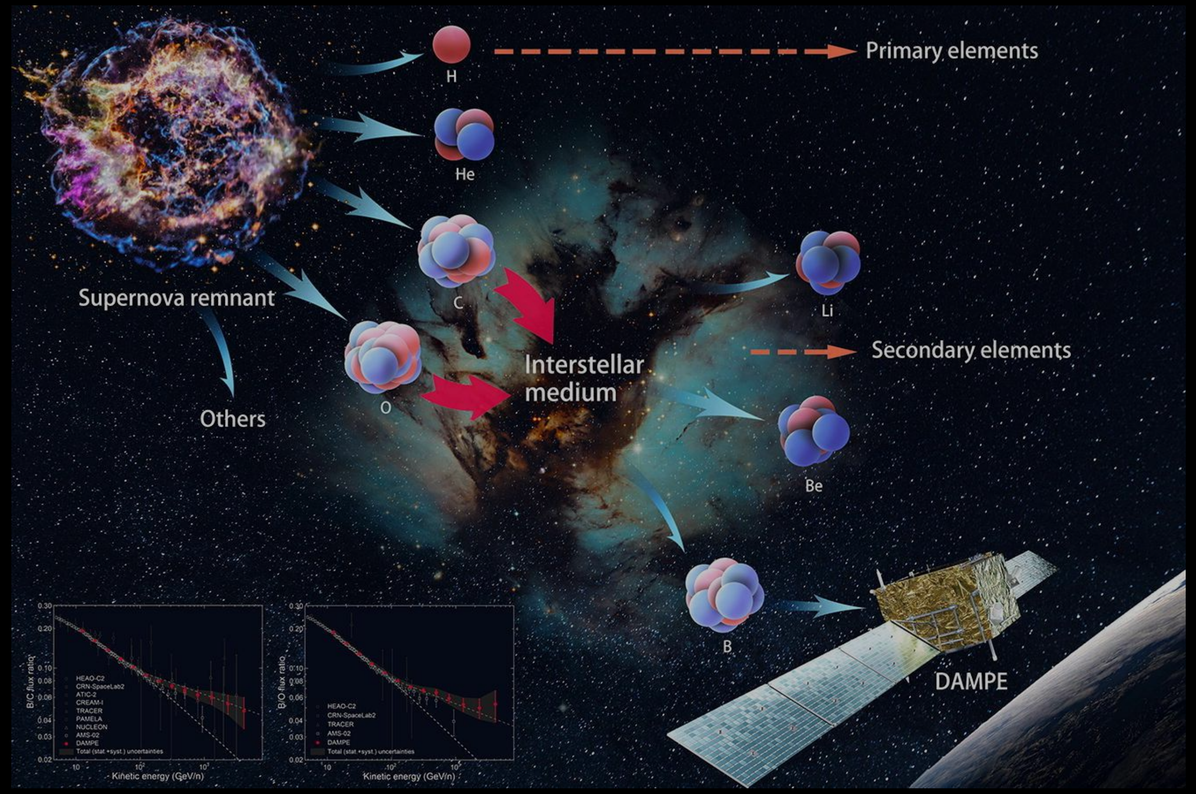
A: To extract information about the particle type (if it is p, He or other ion)!

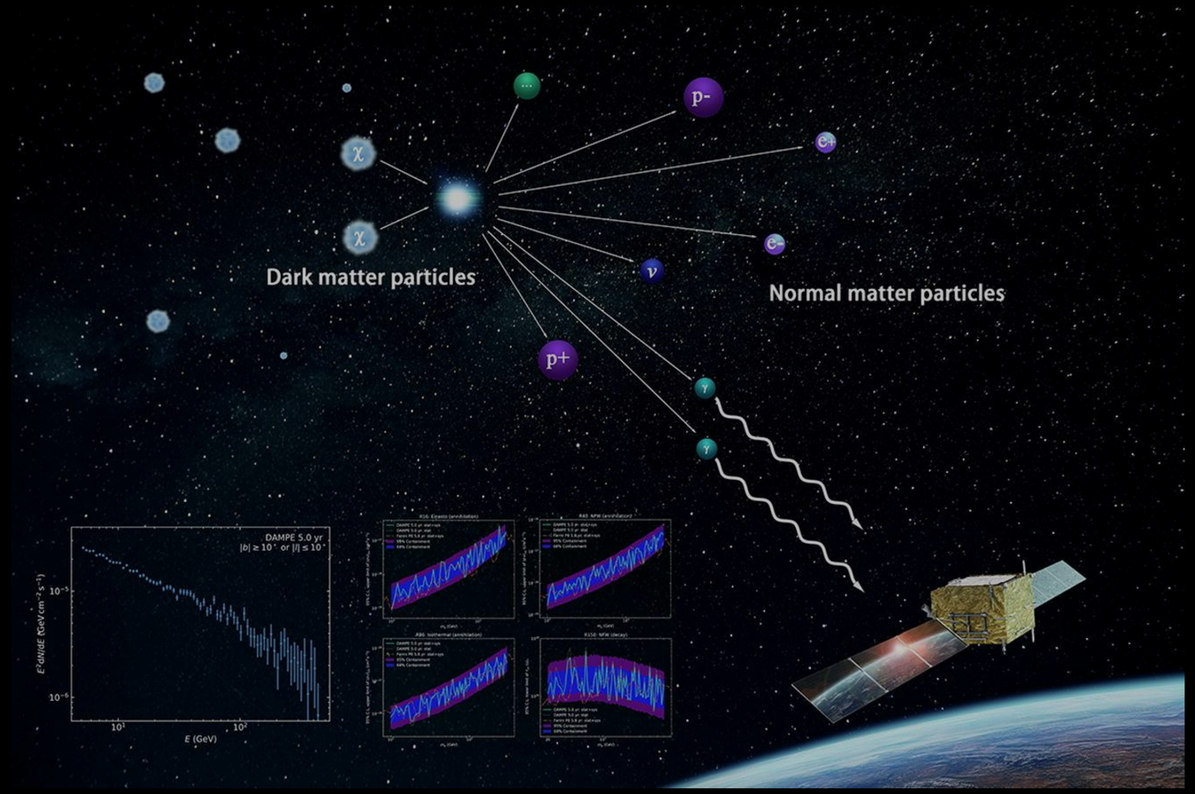
$$\left\langle -\frac{dE}{dx} \right\rangle \sim K z^2 \frac{Z}{A} \frac{1}{\beta^2}$$

(Remember Bethe-Bloch formula for ionization losses)

DAMPE on orbit

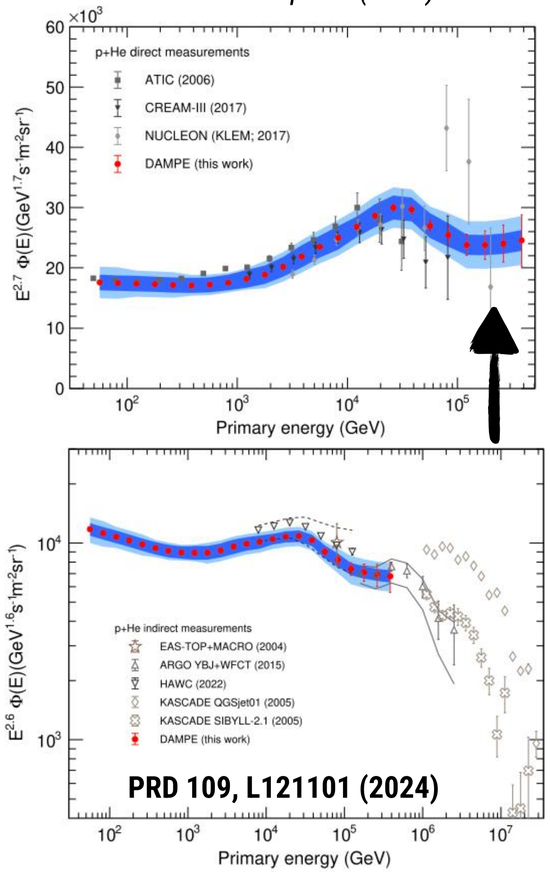
- Many exciting results published since 2015: electrons, protons, helium, B/C and B/O, γ-rays, solar physics
- More in progress (C,O, Ne-Mg-Si, Fe)



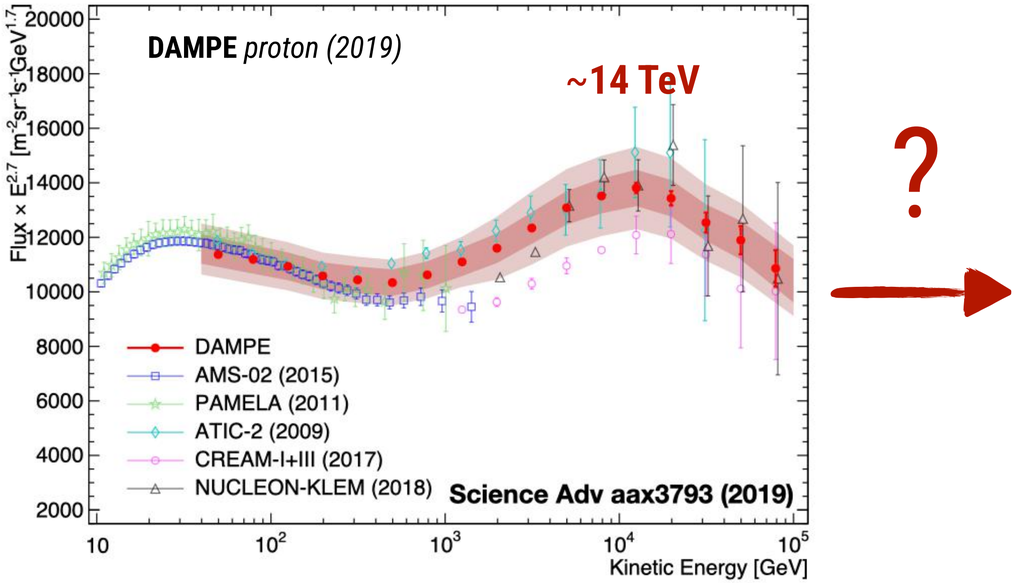


Motivation — exploring TeV-PeV energies in Space

DAMPE *p+HE (2024)*



DAMPE's previous proton-spectrum result is limited to 100 TeV, and it's not only due to statistics!

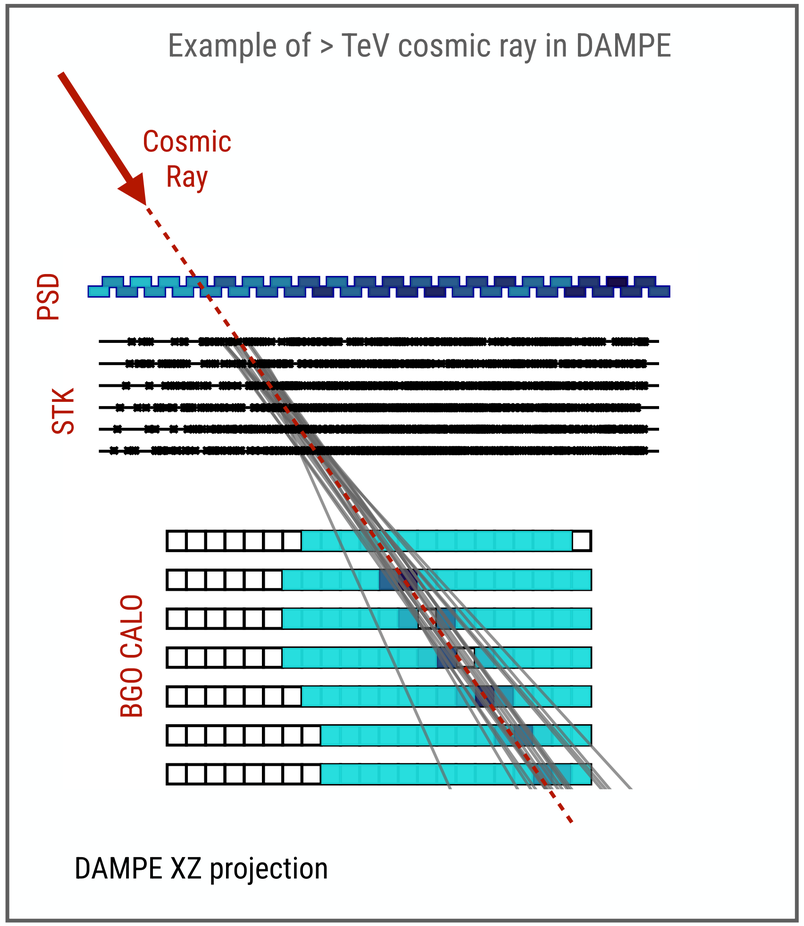


?

- Detection of new feature in light cosmic rays (p+He) at **~150 TeV**

  → What is the physics behind it? We need individual p and He measurements towards PeV ... but here comes the problem (see next slides)

Challenge: CR reconstruction at > 100 TeV



Conventional track reconstruction:

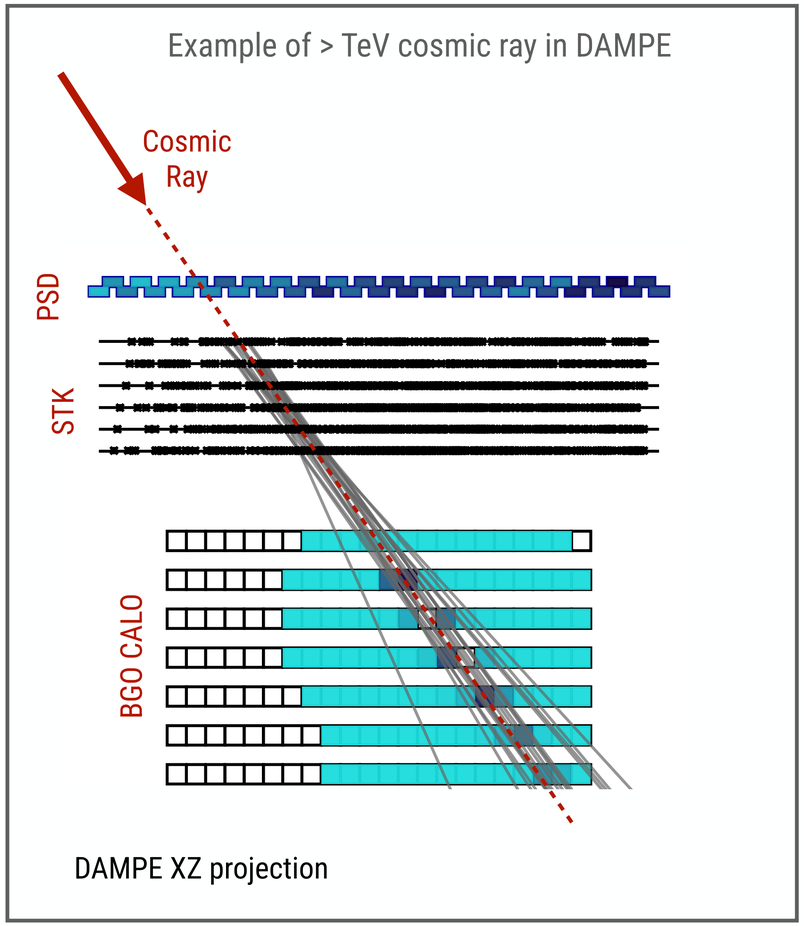
- Shower axis from CALO as a seed
- Kalman fitting
- Combinatorial track finding
- XZ and YZ fitted separately,
- … then combined in 3D tracks

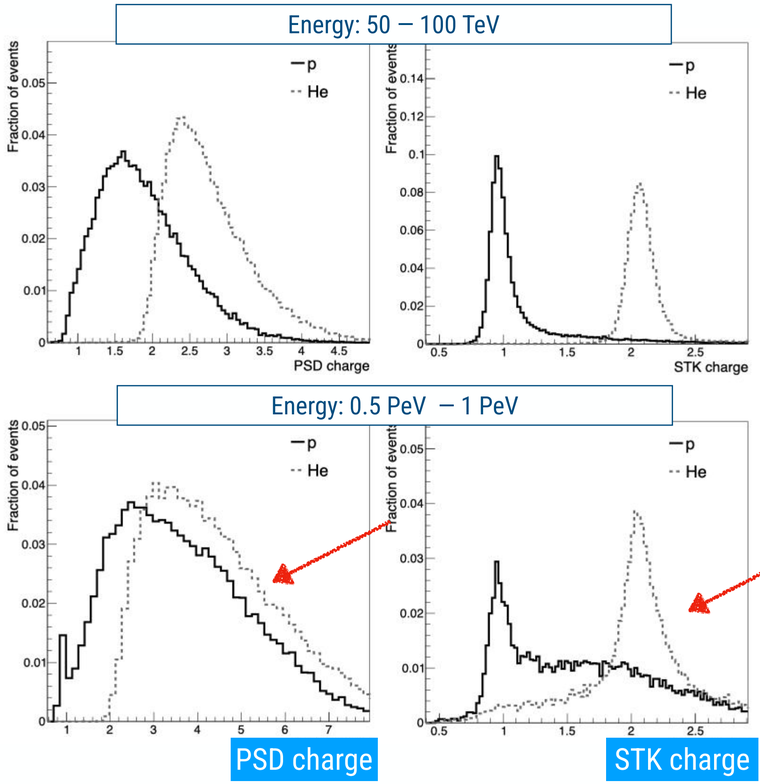
Problems:

- Selection needed to find **the ONLY track**
- Efficiency drops at high hit multiplicity

At TeV– PeV hit multiplicity increases dramatically →
Track reconstruction & identification is a key challenge!

Challenge: CR reconstruction at > 100 TeV





Still OK...

Not sustainable for reliable analysis

Charge identification in PSD – track used as a pointer:

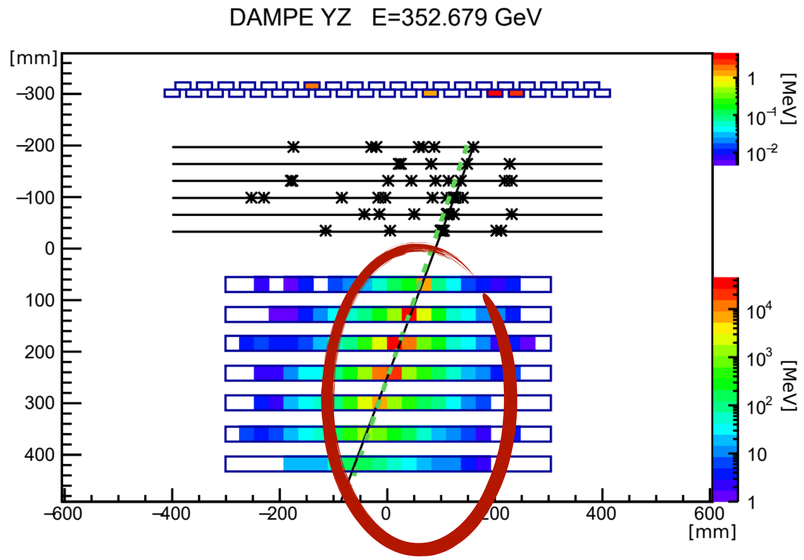
- Tolerant to track mis-identification
- However, **p and He peaks "washed out" at high energies!**

<u>We need a new algorithm to reconstruct particle track with ultimately better precision!</u> (regression type of problem)

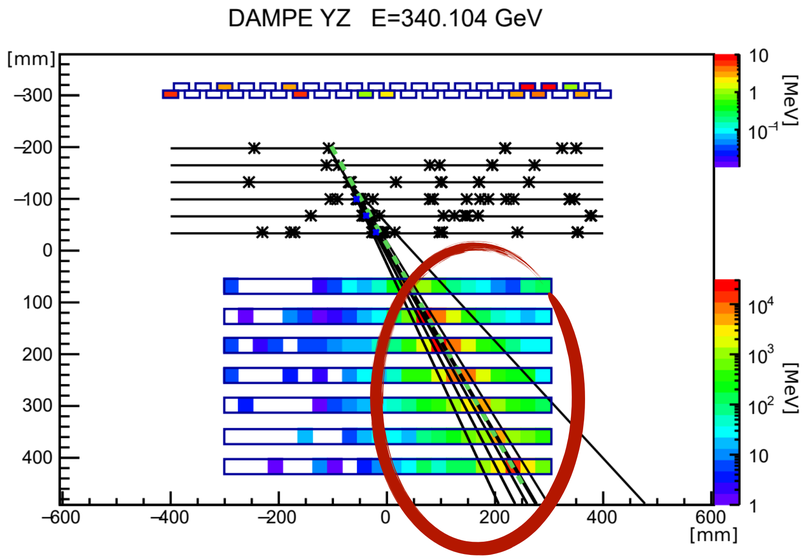
Challenge: CR electron/proton discrimination

Protons are 10^4—10^5 times more than electrons at > TeV energies. While electromagnetic showers are prominently different from hadronic ones, at such high rates one can get proton showers that look exactly like electrons (for example due to >90% pi0 content). As we move to higher energies, our e/p discrimination methods have to "look" for more-and-more subtle details that tell electron from proton

Typical **electron** interaction in DAMPE

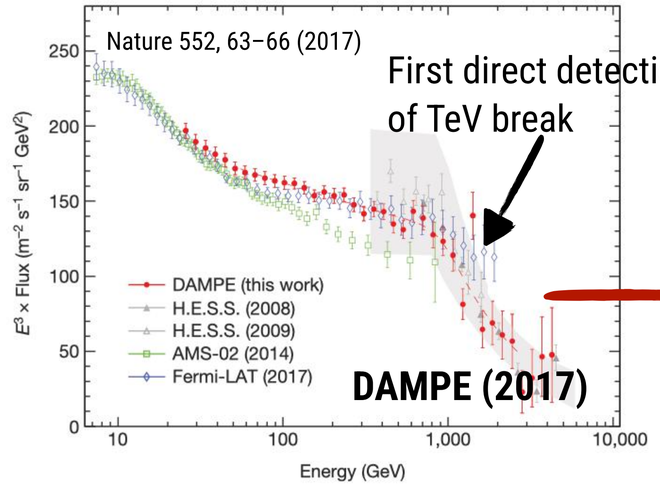


Typical **proton** interaction in DAMPE

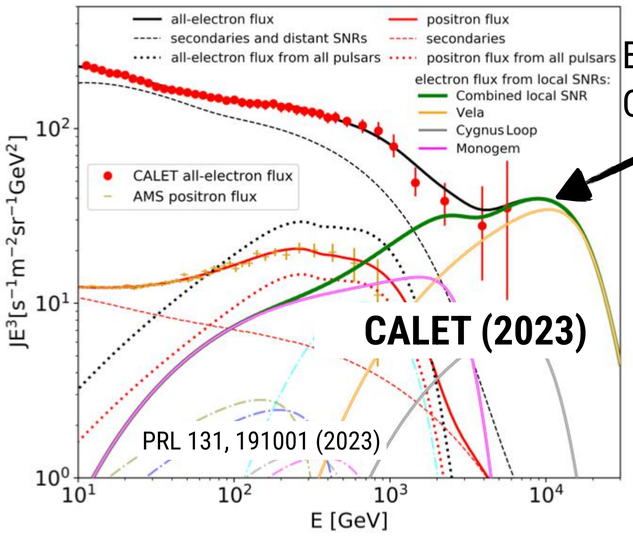


<u>We need very powerful e/p separation technique!</u>
(classification type of the problem)!

Challenge: CR electron/proton discrimination

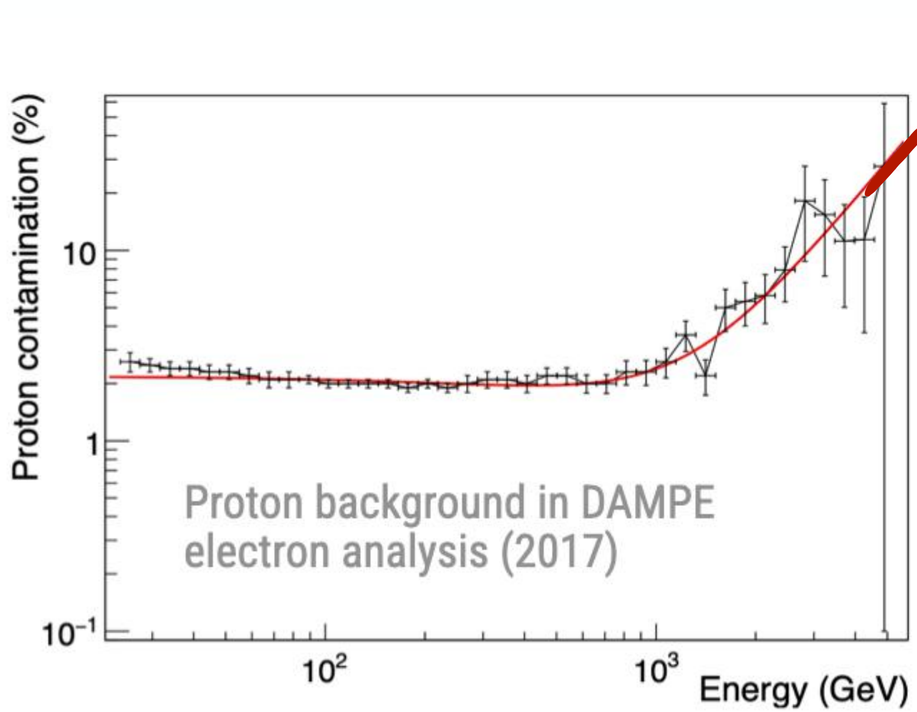


?



Evidence of local CR accelerator?

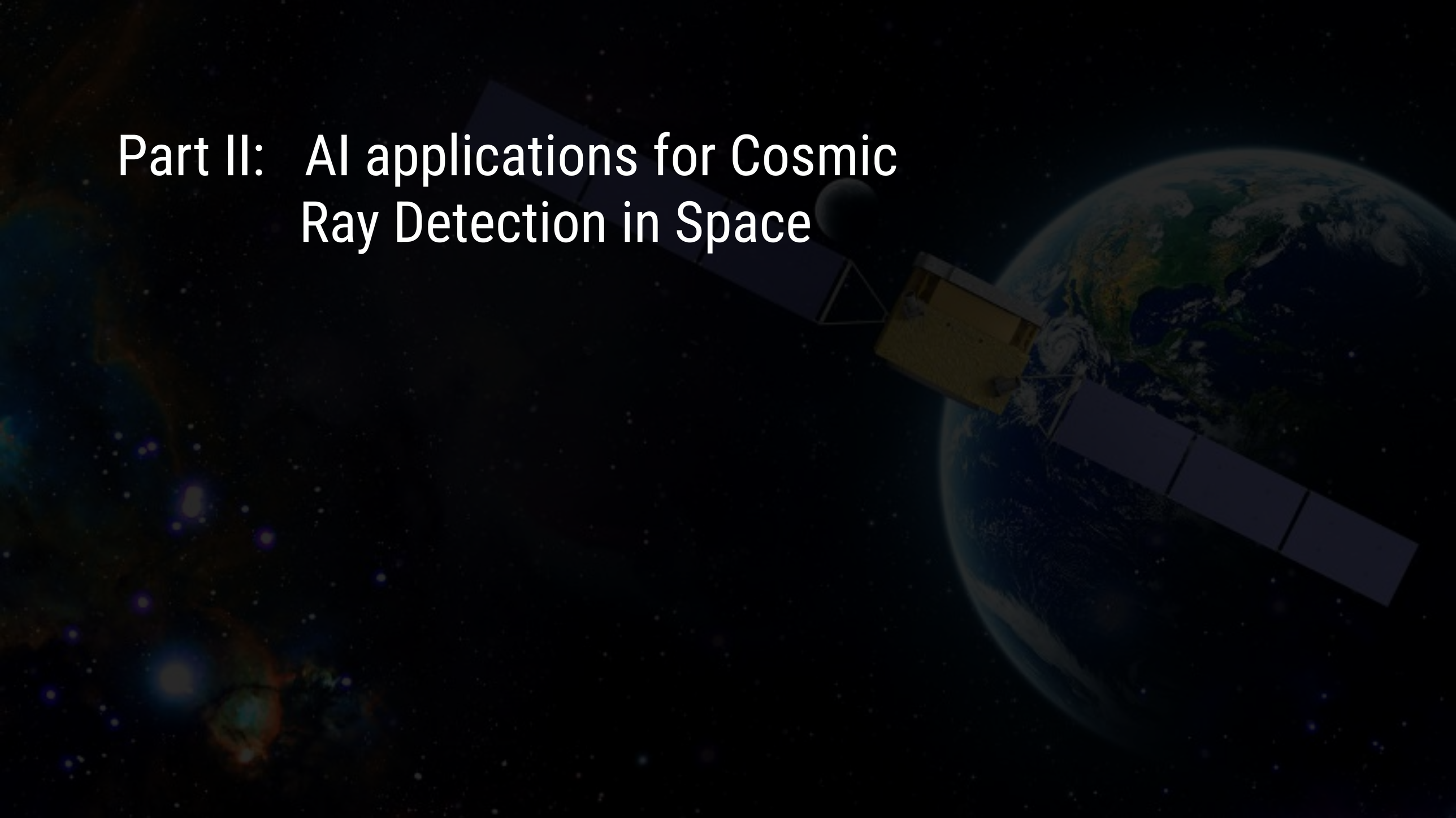
Arguably, we are on a brink of detection of the first CR accelerator directly in CR spectra: electrons at > 10 TeV



Challenge!

We need very powerful e/p separation technique!
(classification type of the problem)!

Part II: AI applications for Cosmic Ray Detection in Space



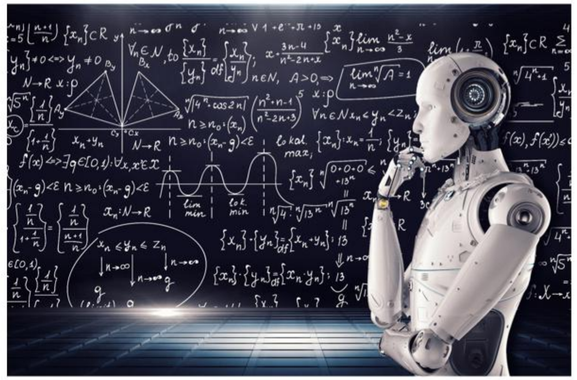
AI terminology

Artificial Intelligence

Human Intelligence
Exhibited by Machines

Machine Learning

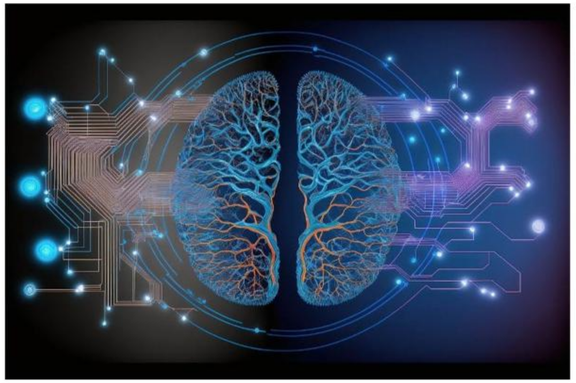
An Approach to Achieve
Artificial Intelligence



We focus mostly on deep learning applications in this talk

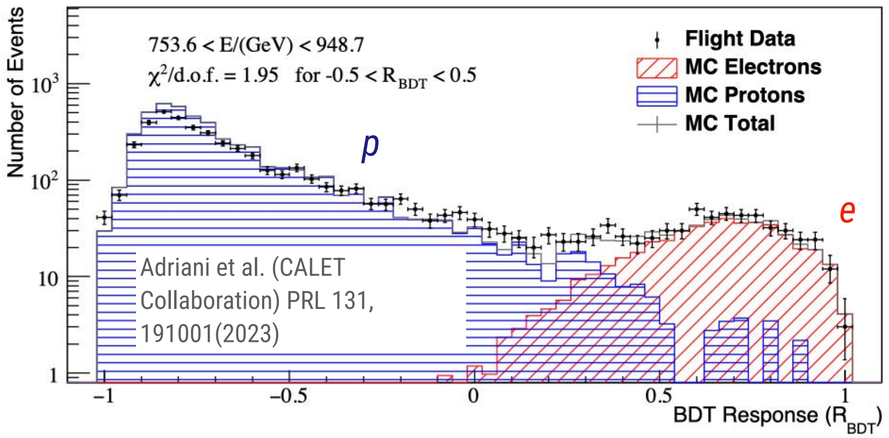
Deep Learning

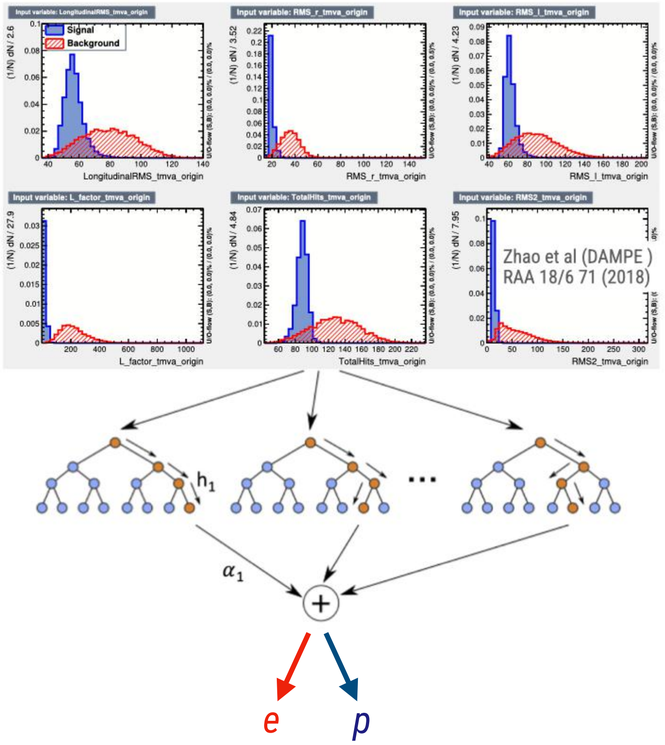
A Technique for
Implementing Machine
Learning

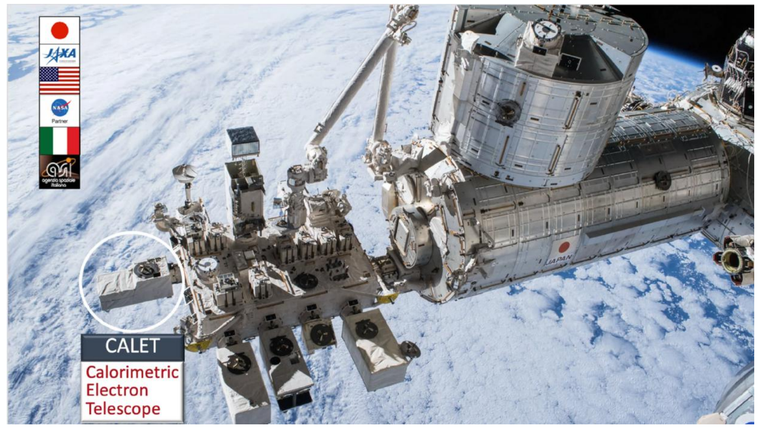


Boosted Decision Trees (BDTs) [... not deep learning]

- Powerful technique for classification (and in certain case regression)
- Well-known/studied since pre-LHC era
- Commonly used in space experiments (AMS-02, CALET, DAMPE)



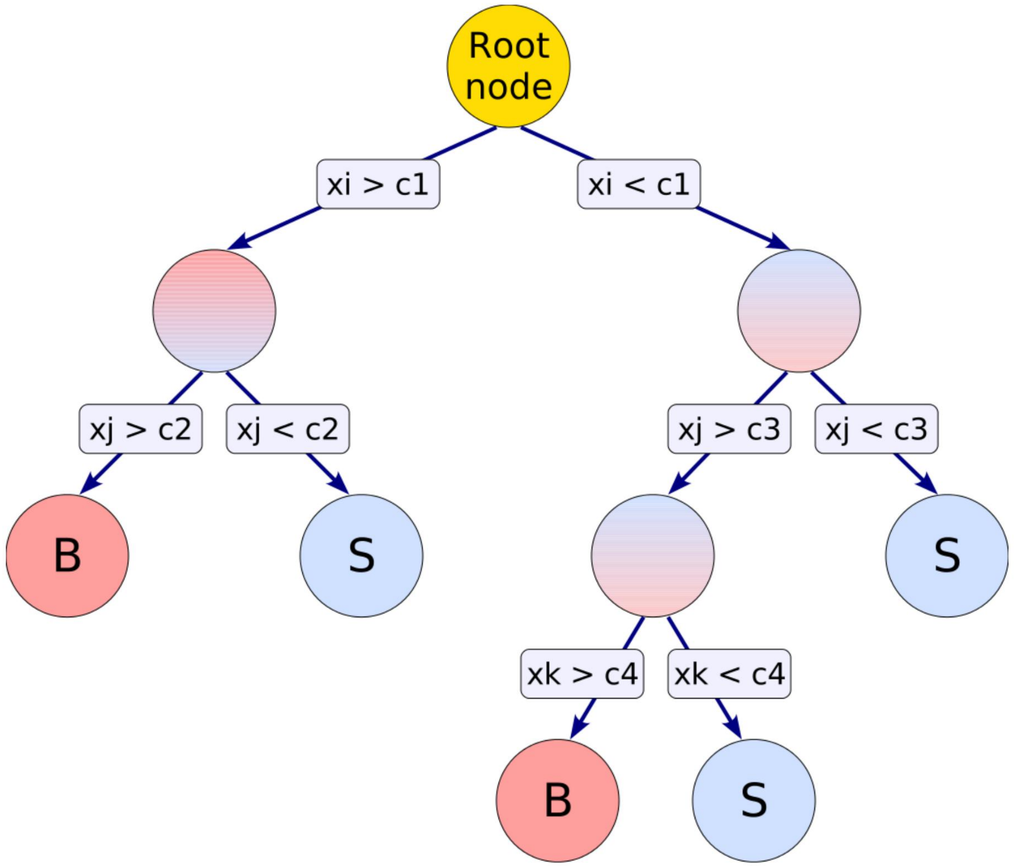




Classical use case: combination of correlated high-level variables each having some classification power (number of hits in the detector, energy-per-layer etc.)

Boosted Decision Trees (BDTs) [... not deep learning]

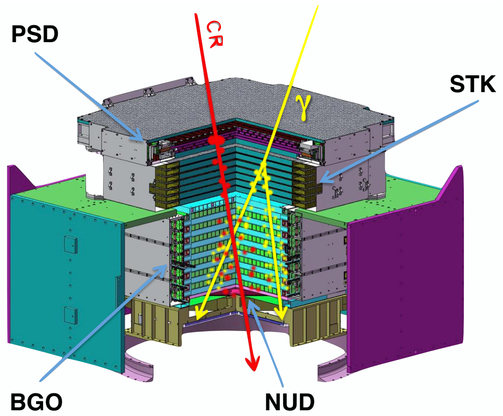
- We all saw this canonical image, but what is BDT and how it is trained?



Boosted Decision Trees (BDTs) [... not deep learning]

Let's take an example of DAMPE detector:

- Say, we have one million simulate particle interaction in DAMPE, denoted as **0 for proton, 1 for electron**.
- Each interaction is characterized, say, by 4 **features** (E_{TOT}, ϕ, θ, X_{MAX}) (it is very ovesimplified)



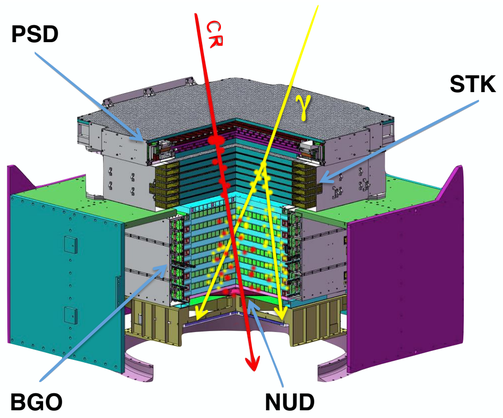
E_TOT	phi	theta	X_max	particle
[7.75681502e+01,	1.53397721e+00,	1.44904093e-01,	4.88372628e-01]	[0]
[9.44332801e+01,	4.99172621e-01,	1.01597742e+00,	4.05432853e-01]	[0]
[8.07924576e+01,	1.09393488e+00,	1.70475418e+00,	8.92284561e-02]	[1]
[9.07662861e+01,	2.94882994e-01,	1.43258714e+00,	6.00182604e-01]	[0]
[2.56174396e+01,	1.60203322e-01,	2.53668119e+00,	9.24928506e-02]	[0]
[5.11821004e+01,	1.55394742e+00,	1.68910805e+00,	4.05057996e-01]	[0]
[5.52805087e+01,	1.49059293e+00,	9.15734664e-01,	2.89532404e-01]	[1]
...				...
...				...
...				...
[6.02792632e+00,	3.36060004e-01,	1.13356232e+00,	2.46623555e-01]	[1]
[9.32034716e+01,	1.38081000e+00,	5.86855693e+00,	6.46444145e-01]	[0]
[1.95756751e+01,	7.29298459e-01,	3.65878315e+00,	3.08917693e-01]	[0]

1 Million simulated particle interactions in DAMPE

Boosted Decision Trees (BDTs) [... not deep learning]

Let's take an example of DAMPE detector:

- Say, we have one million simulate particle interaction in DAMPE, denoted as **0 for proton, 1 for electron**.
- Each interaction is characterized, say, by 4 **features** (E_{TOT}, ϕ, θ, X_{MAX}) (it is very ovesimplified)

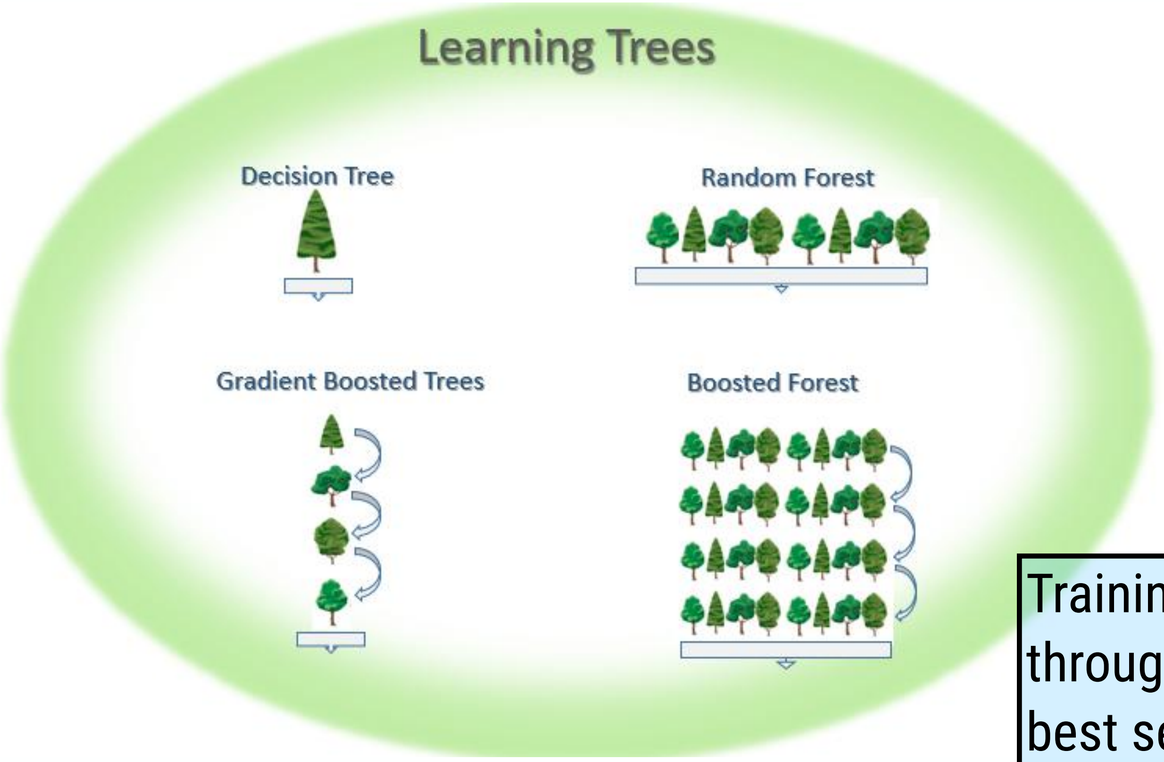



```
E_TOT                phi                 theta               X_max                 particle
[7.75681502e+01, 1.58397721e+00, 1.44904093e-01, 4.88372628e-01]  [0]
[9.44332801e+01, 4.99172621e-01, 1.01597742e+00, 4.05432853e-01]  [0]
[8.07924576e+01, 1.09393488e+00, 1.70475418e+00, 8.92284561e-02]  [1]
[9.07662861e+01, 2.94882994e-01, 1.43258714e+00, 6.00182604e-01]  [0]
[2.56174396e+01, 1.60203322e-01, 2.53668119e+00, 9.24928506e-02]  [0]
[5.11821004e+01, 1.55394742e+00, 1.68910805e+00, 4.05057996e-01]  [0]
[5.52805087e+01, 1.49059293e+00, 9.15734664e-01, 2.89532404e-01]  [1]
...                                                               ...
...                                                               ...
...                                                               ...
[6.02792632e+00, 3.36060004e-01, 1.13356232e+00, 2.46623555e-01]  [1]
[9.32034716e+01, 1.38081000e+00, 5.86855693e+00, 6.46444145e-01]  [0]
[1.95756751e+01, 7.29298459e-01, 3.65878315e+00, 3.08917693e-01]  [0]
```

1 Million simulated particle interactions in DAMPE

- Training of the first "tree" (stump):
  - ➡Look for the best feature (one out of 4) and its corresponding value (1 out of Million)
  - ➡Set the above as a threshold: if the feature is < threshold - classify event as "0" (in reality it's a continuous number), otherwise "1"
- Train the second tree to correct for the error of the previous one, by selecting new feature-value pair (threshold)
- Repeat many times each time creating a new tree, until reaching a desired level of prediction accuracy

Boosted Decision Trees (BDTs) [... not deep learning]



Training/learning of BDT is a **discretized** process of looping through features and their values, finding the one giving the best separation at each step. This is fundamentally different from training Neural Networks, where parameters of the model are updated **continuously** through gradient descent method

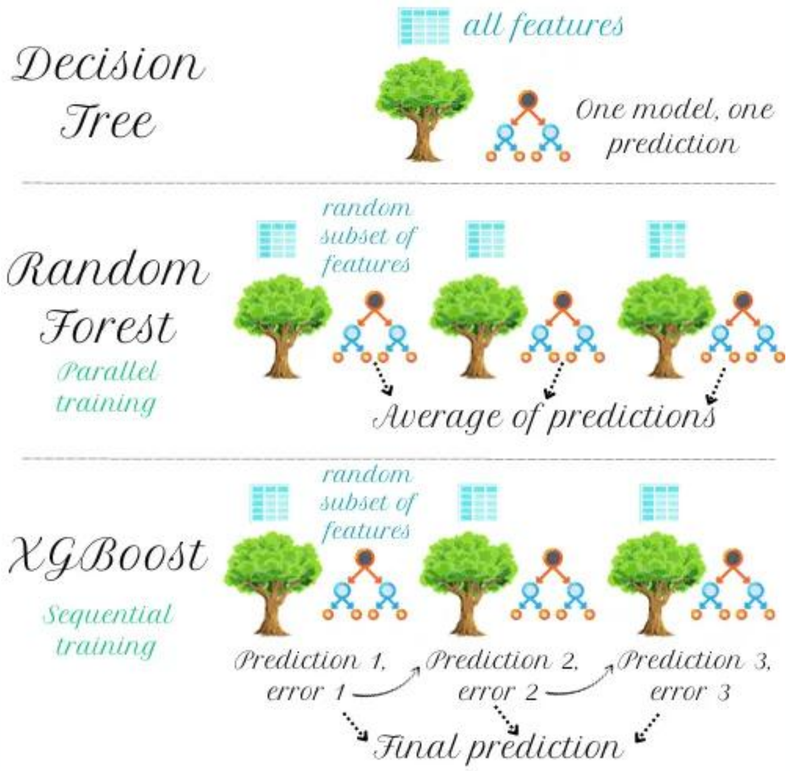
Boosted Decision Trees (BDTs) [... not deep learning]

BDTs are still considered well suitable for many applications:

- Easier to interpret (e.g. compared to "black-box" Neural Networks)
- Easier to train (computation-wise)
- Considered more reliable (e.g. to outliers)
- Work well with small datasets

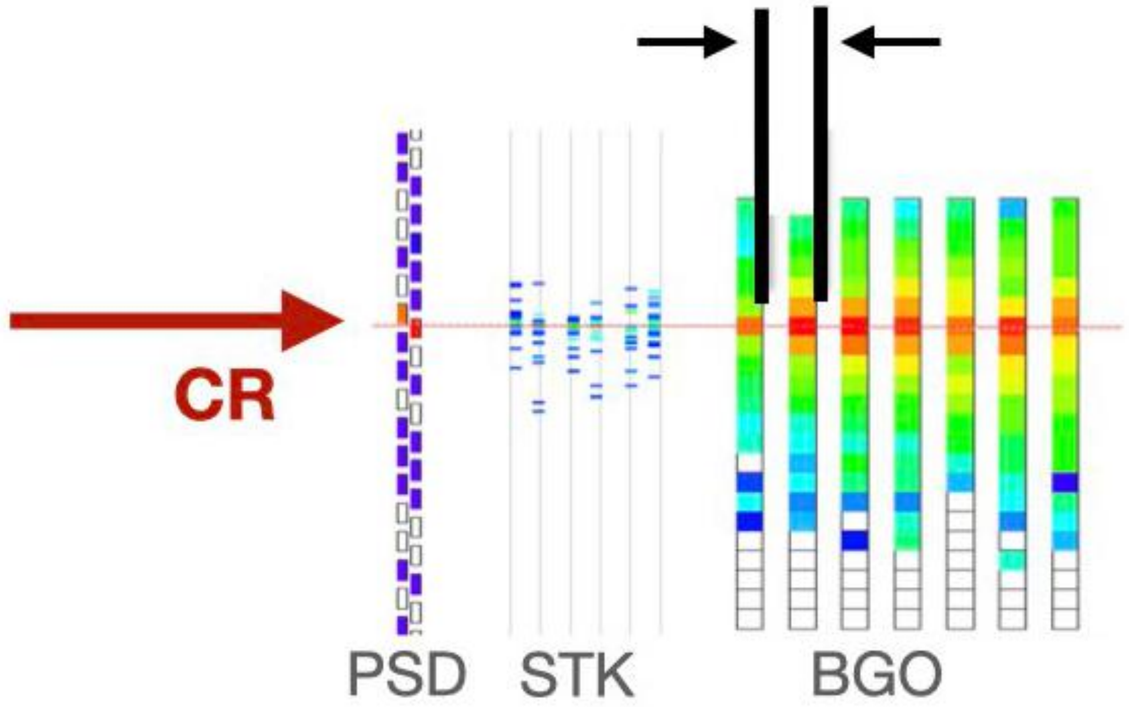
Disadvantages of BDTs

- Limited performance on high-dimensional unstructured data
- Prior feature extraction is important
- Limited architecture flexibility
- **Mostly classification tasks (regression is also possible, however BDTs by construction do not predict continues values, but rather a "grid" of values defined by the training sample)**



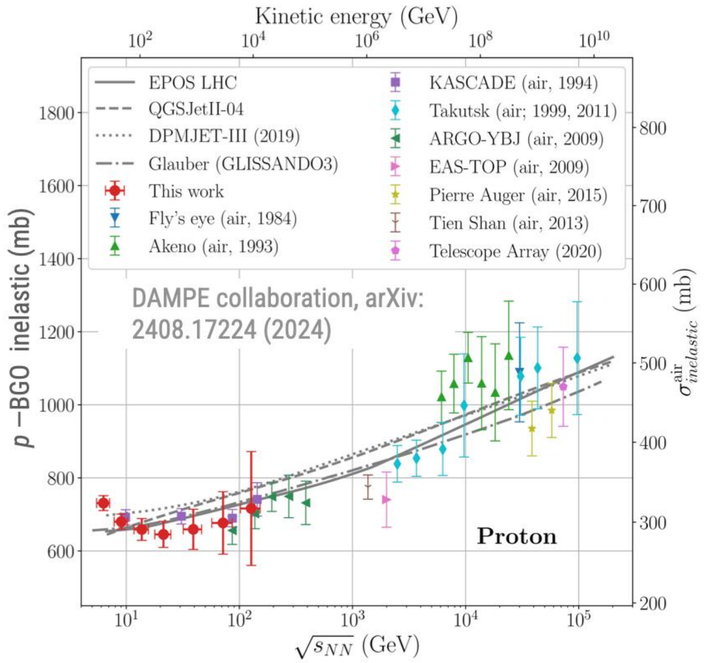
BDTs @ DAMPE - particle physics in space!

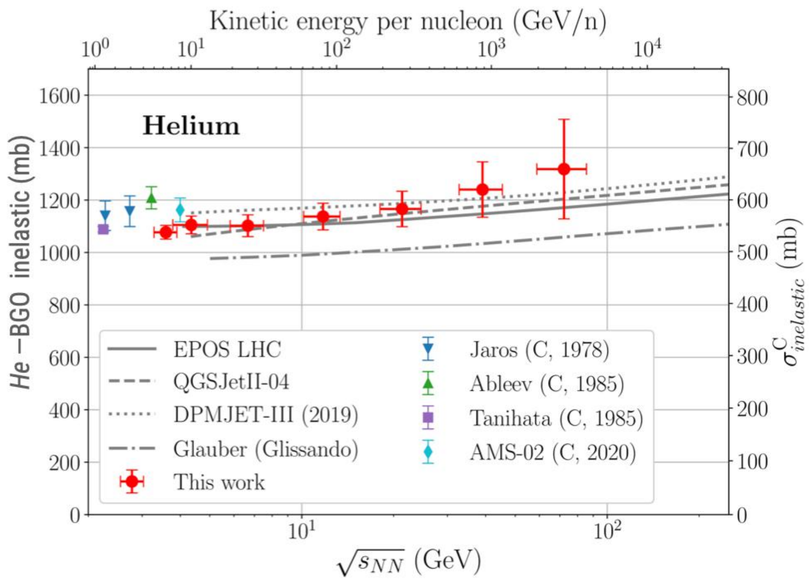
We found BDTs quite efficient for identifying particle interaction point in the detector:



BDTs @ DAMPE - particle physics in space!

We found BDTs quite efficient for identifying particle interaction point in the detector:

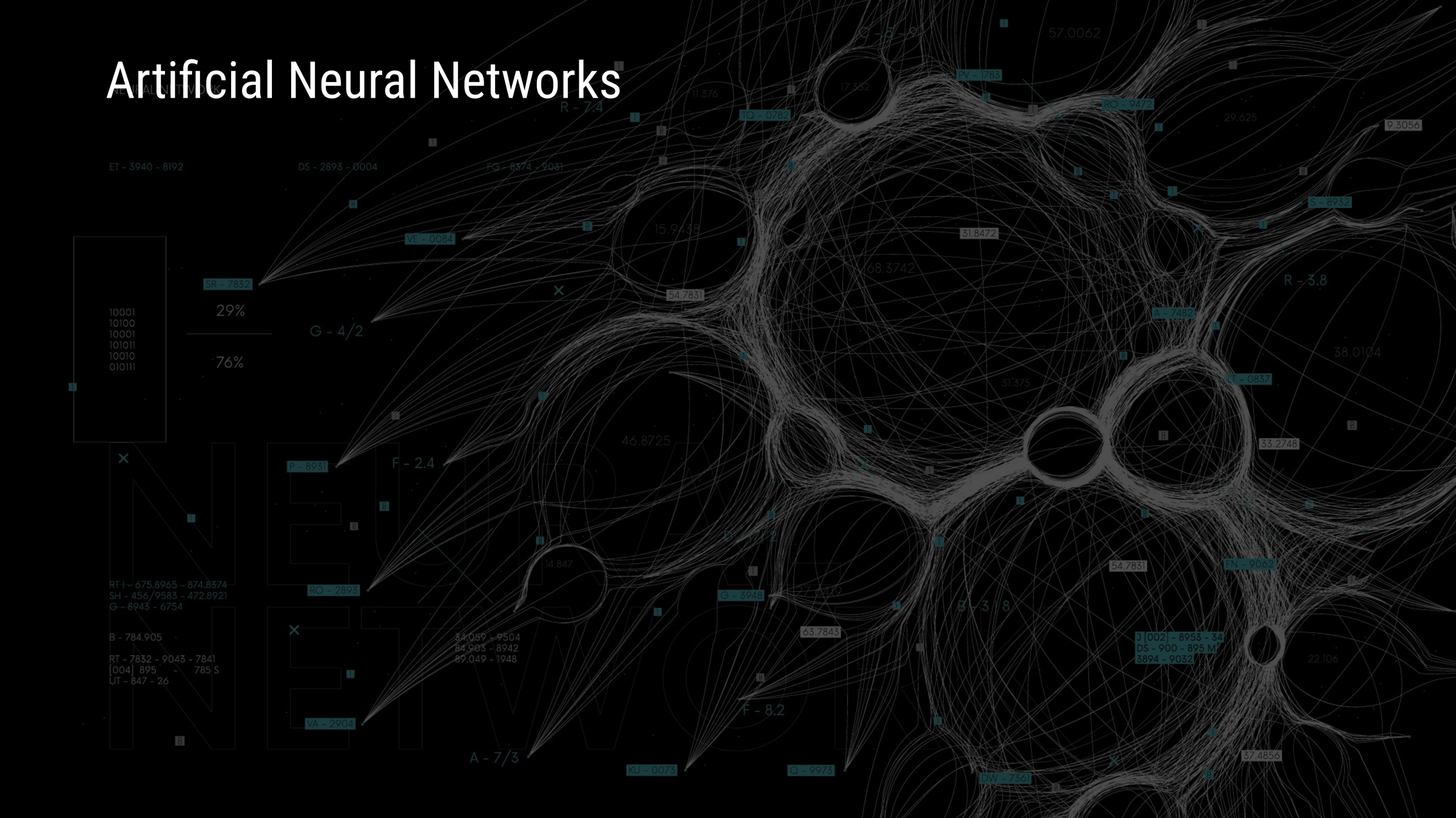




Phys. Rev. D 111, 012002 (2025); arxiv:2408.17224.

This allowed us to measure proton and helium inelastic cross sections on heavy target in the energy regions not probed by accelerators or EAS detectors!

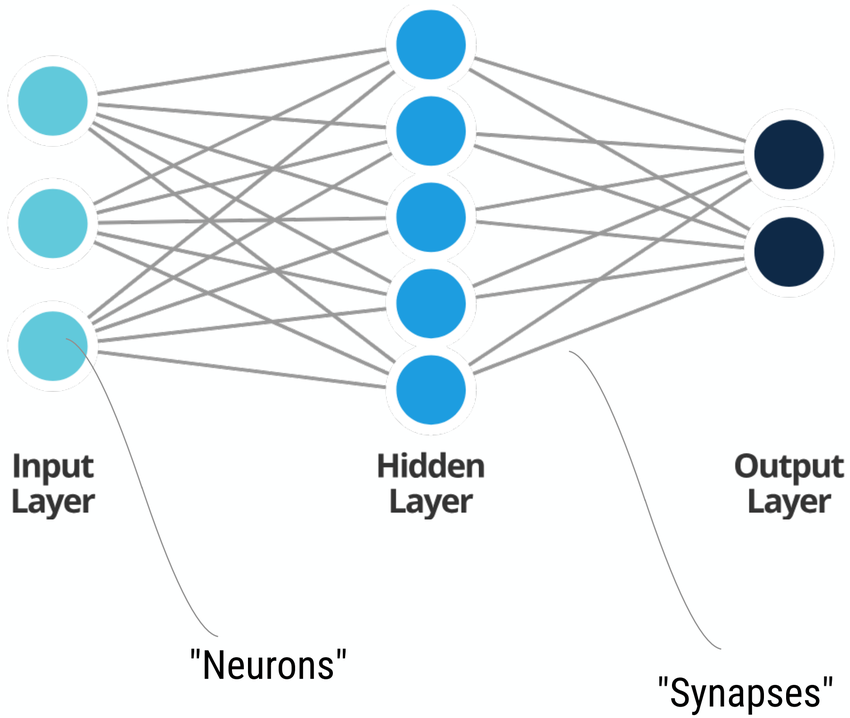
Artificial Neural Networks

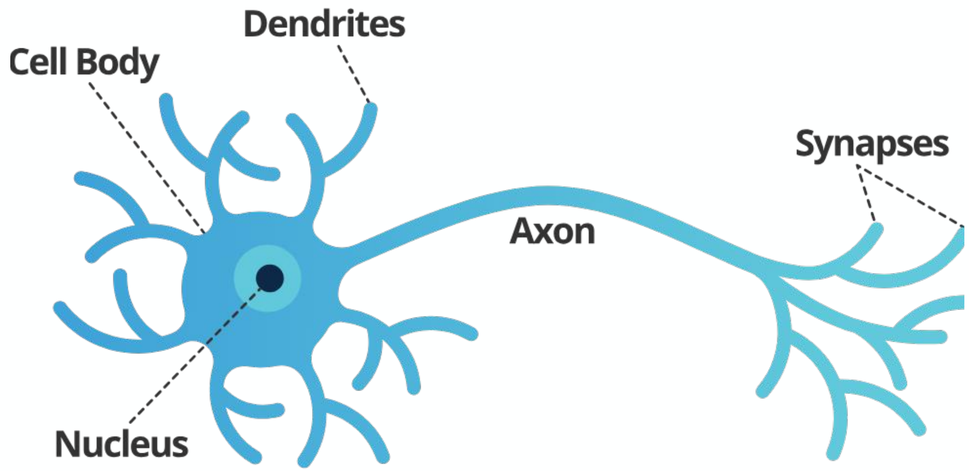


Neural Networks (NNs)

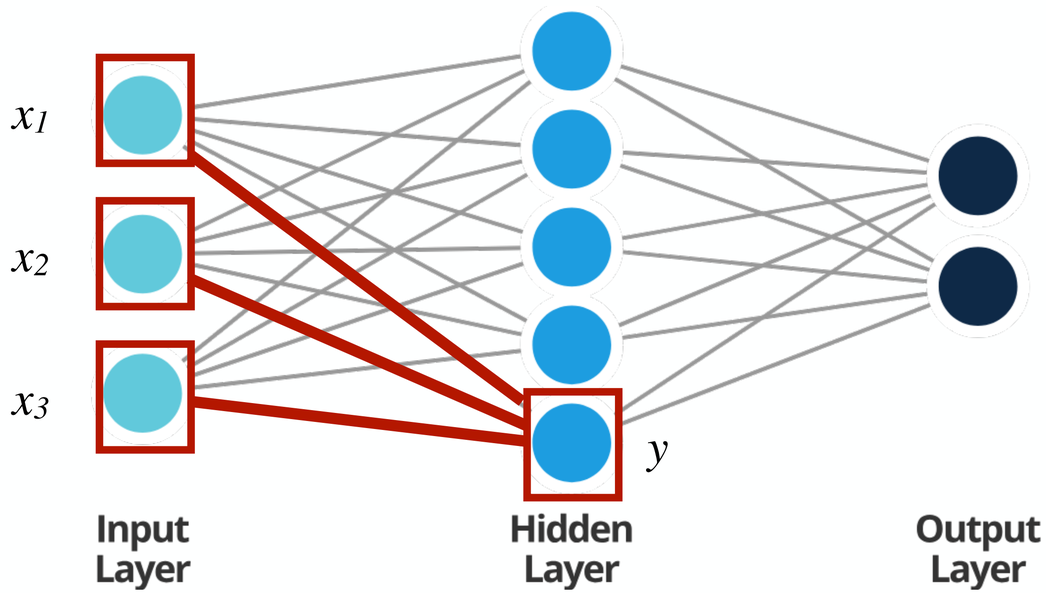
Artificial Neural Networks are inspired by human brain's signal transmission and processing

In a fully-connected NN, each neuron takes a linear combination of signals in previous layer, adds a bias/offset and apply a non-linear activation





Neural Networks (NNs)

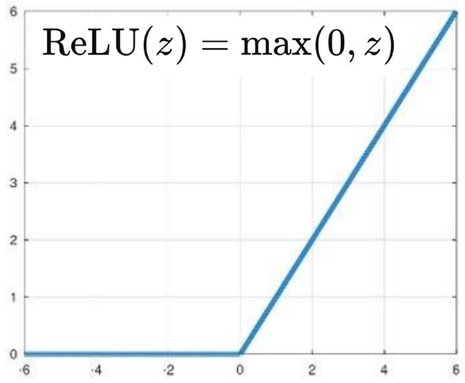


$$y = f(w_1x_1 + w_2x_2 + w_3x_3 + b)$$

wights

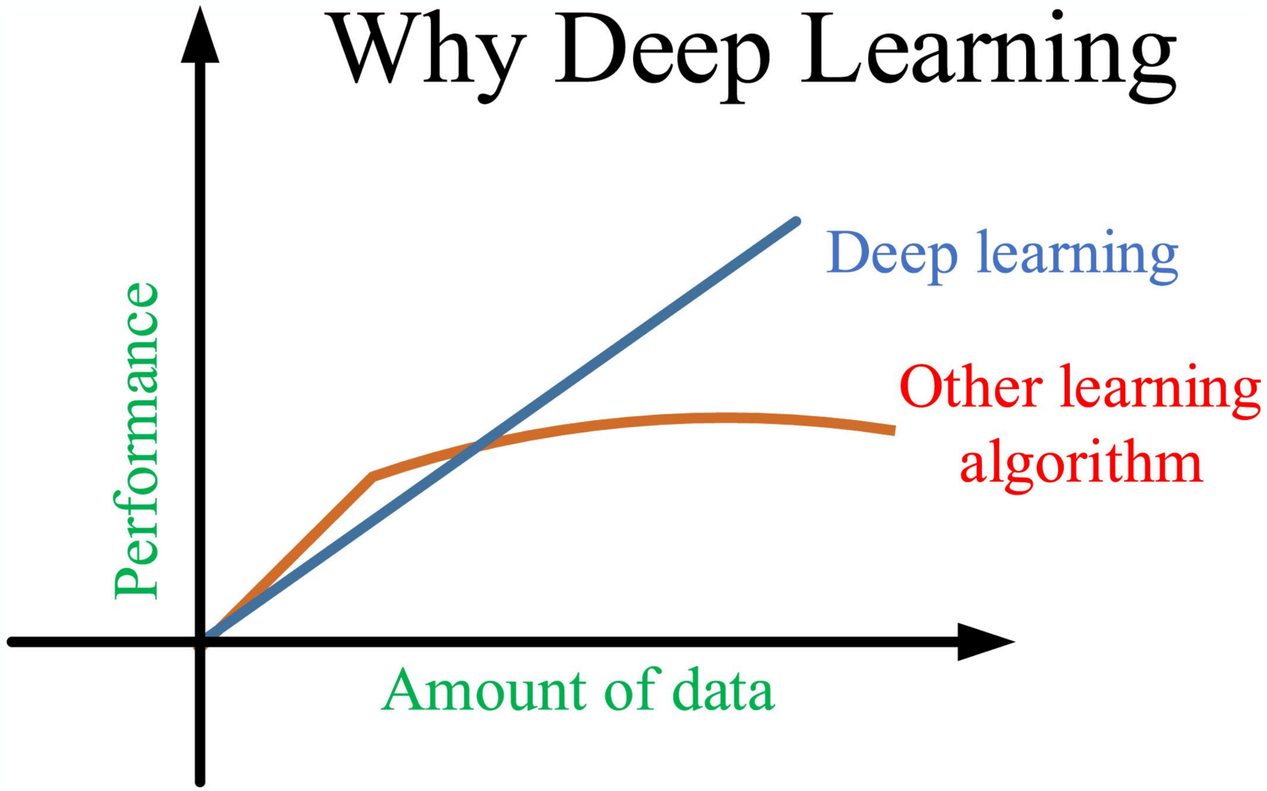
bias

f - activation function:
ReLU, Sigmoid, etc.

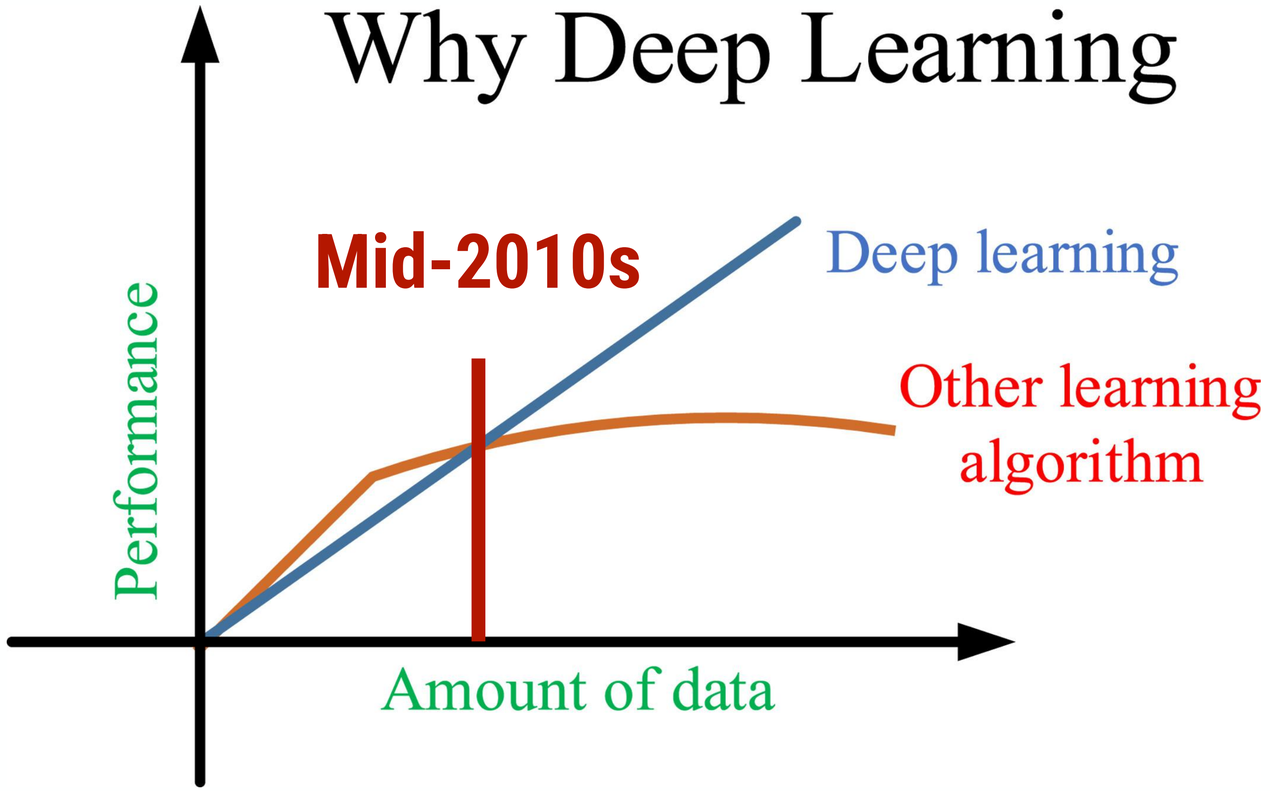


1943: theoretical concept by Warren McCulloch and Walter Pitts: "A Logical Calculus of the Ideas Immanent in Nervous Activity"
1958: Frank Rosenblatt developed the Perceptron, the first trainable neural network (on IBM 704)

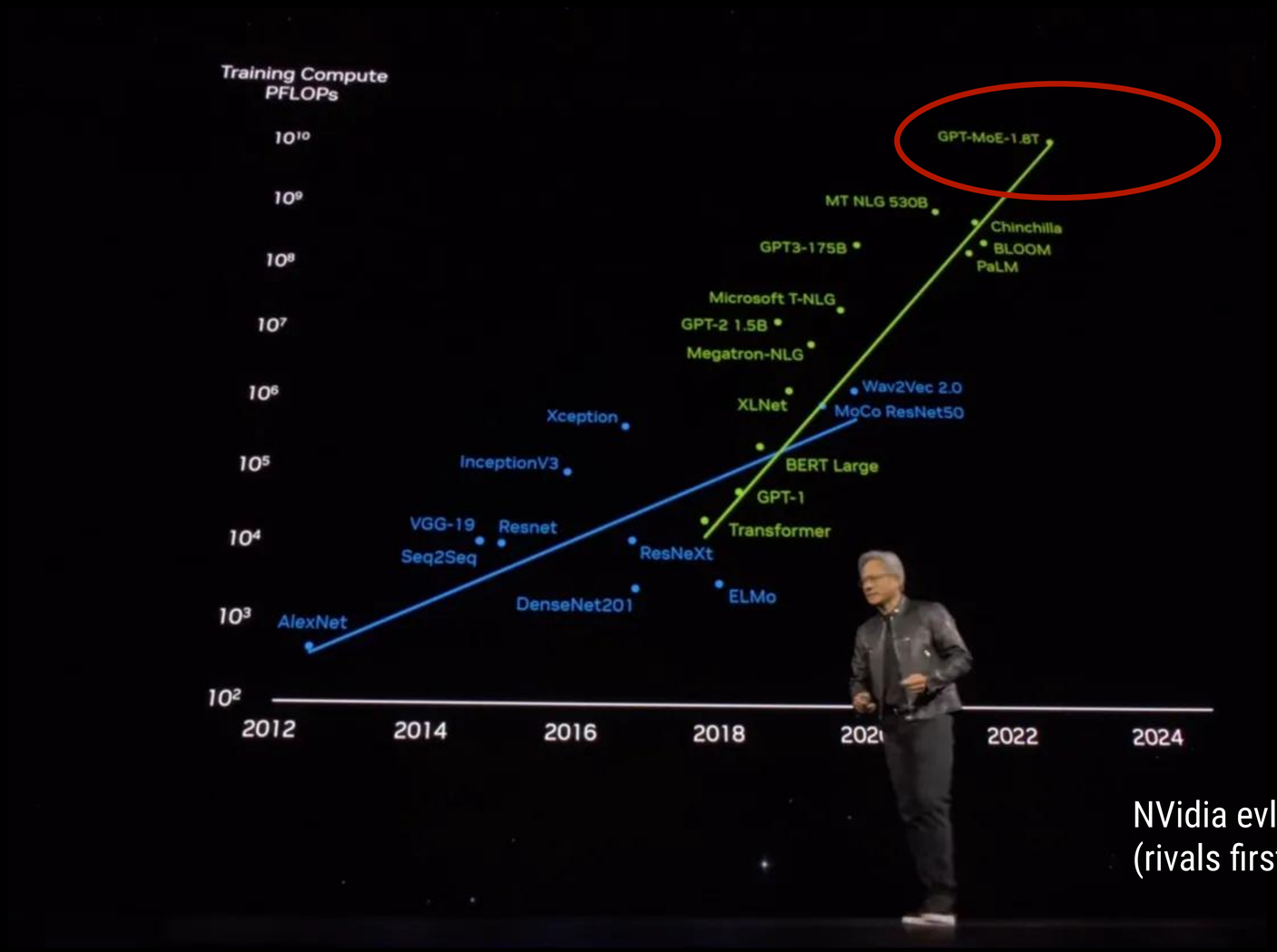
Why Deep Learning



Why Deep Learning



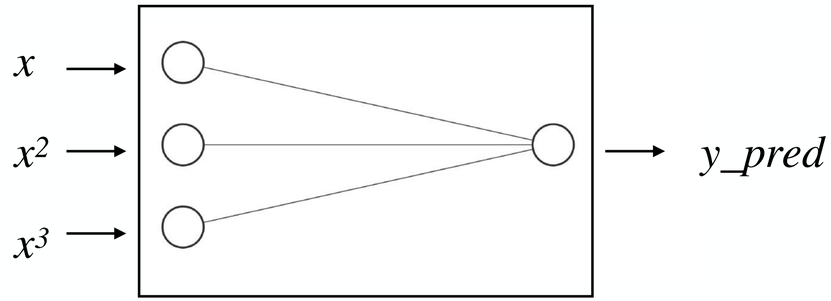
Most advanced NNs, e.g. GPT4-turbo ~ (arguably) 1-10 Trillion parameters
Human brain ~ 100-1000 Trillion synapses



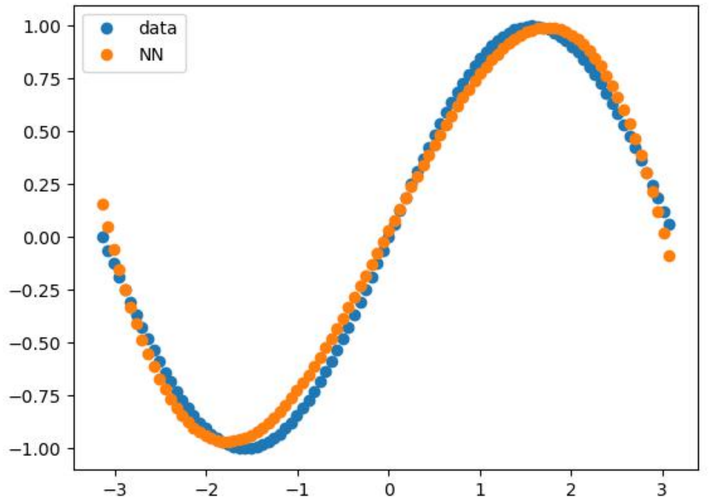
NVidia evlaution as of 2025 ~ 3.5 Trillion USD
(rivals first place with Microsoft and Apple)

NNs: do-it-yourself (no Tensorflow, no PyTorch)


```
import numpy as np
import math

# Create random input and output data
x = np.linspace(-math.pi, math.pi, 2000)
y = np.sin(x)

# Randomly initialize weights (4 numbers)
a,b,c,d = np.random.randn(4)

learning_rate = 1e-6
for t in range(100):
    # Forward pass: compute predicted y
    # y = a + b x + c x^2 + d x^3
    y_pred = a + b * x + c * x ** 2 + d * x ** 3

    # Compute and print loss
    loss = np.square(y_pred - y).sum()
    print(t, loss)

    # Backprop to compute gradients of a, b, c, d with respect to loss
    grad_y_pred = 2.0 * (y_pred - y)
    grad_a = grad_y_pred.sum()
    grad_b = (grad_y_pred * x).sum()
    grad_c = (grad_y_pred * x ** 2).sum()
    grad_d = (grad_y_pred * x ** 3).sum()

    # Update weights
    a -= learning_rate * grad_a
    b -= learning_rate * grad_b
    c -= learning_rate * grad_c
    d -= learning_rate * grad_d
```



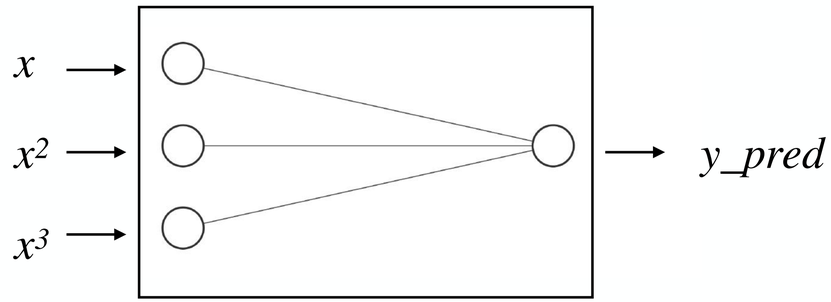
One-neuron NN



Code from: https://docs.pytorch.org/tutorials/beginner/pytorch_with_examples.html

NNs: do-it-yourself (no Tensorflow, no PyTorch)


```
import numpy as np
import math

# Create random input and output data
x = np.linspace(-math.pi, math.pi, 2000)
y = np.sin(x)

# Randomly initialize weights (4 numbers)
a,b,c,d = np.random.randn(4)

learning_rate = 1e-6
for t in range(100):
    # Forward pass: compute predicted y
    # y = a + b x + c x^2 + d x^3
    y_pred = a + b * x + c * x ** 2 + d * x ** 3

    # Compute and print loss
    loss = np.square(y_pred - y).sum()
    print(t, loss)

    # Backprop to compute gradients of
    grad_y_pred = 2.0 * (y_pred - y)
    grad_a = grad_y_pred.sum()
    grad_b = (grad_y_pred * x).sum()
    grad_c = (grad_y_pred * x ** 2).sum()
    grad_d = (grad_y_pred * x ** 3).sum()

    # Update weights
    a -= learning_rate * grad_a
    b -= learning_rate * grad_b
    c -= learning_rate * grad_c
    d -= learning_rate * grad_d
```



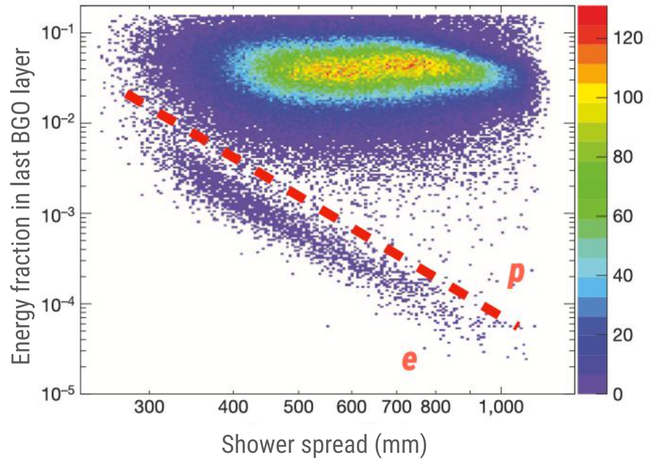
One-neuron NN

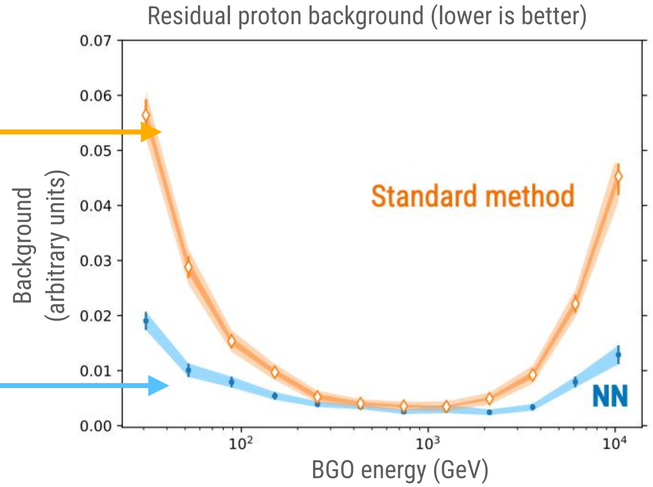
Training/learning of NN done through gradient descent: iteratively compute gradients of loss function and update parameters (weights, biases) of neurons...

NN application in DAMPE

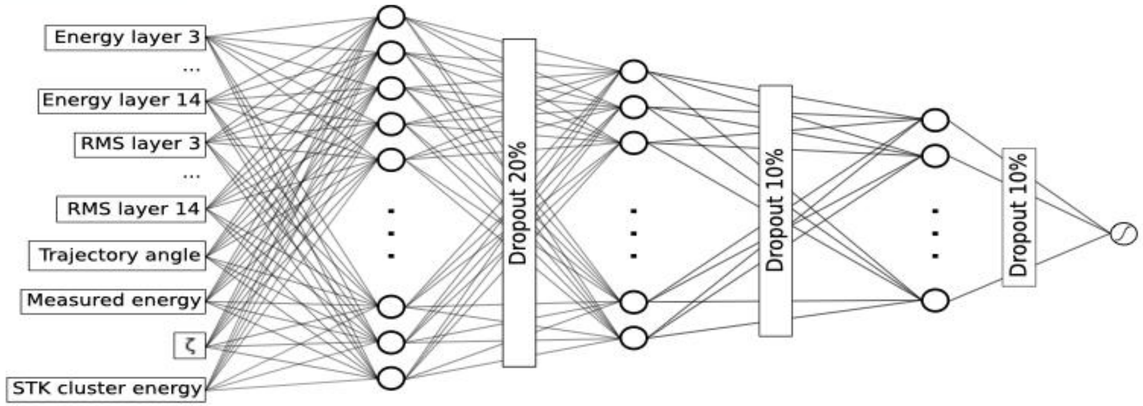
Particle identification (classification): *e* / *p*

Standard method (shower-shape):





Neural Network classifier:

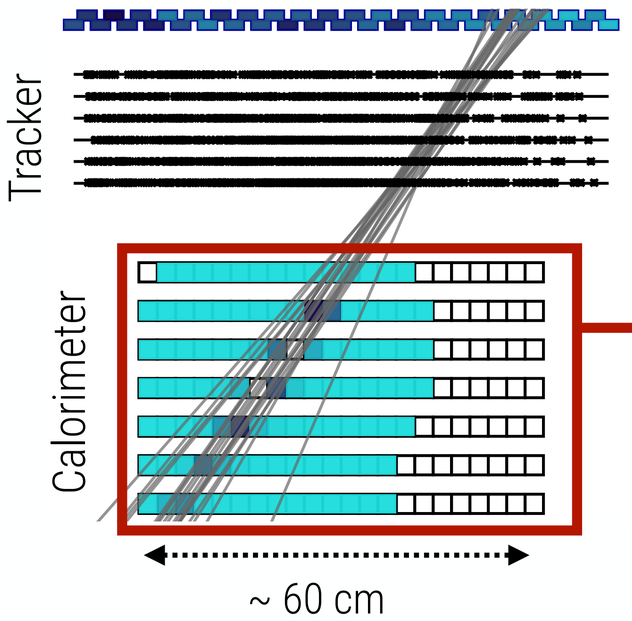


x4 better proton background rejection at > 10 TeV compared to classical methods!

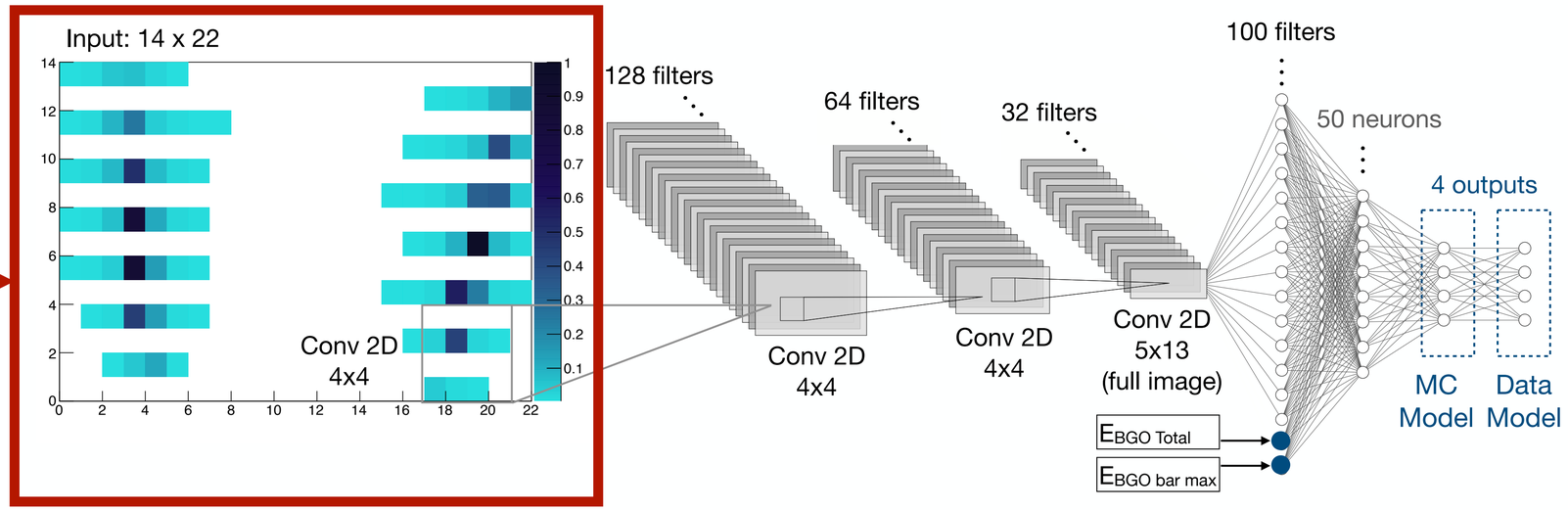
Droz et al. JINST 16 P07036 (2021)

Regression: reconstructing particle trajectory

We start with the analysis of calorimeter data to get a first “rough” prediction of particle trajectory:



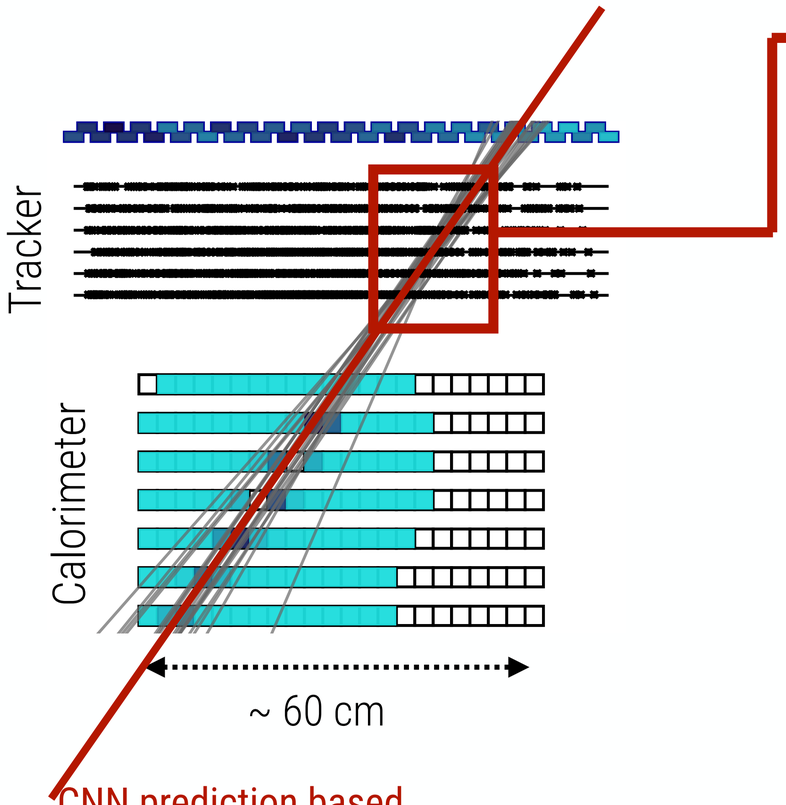
<u>Convolutional Neural Networks (CNNs)</u> have proven to be extremely efficient for this kind of tasks:



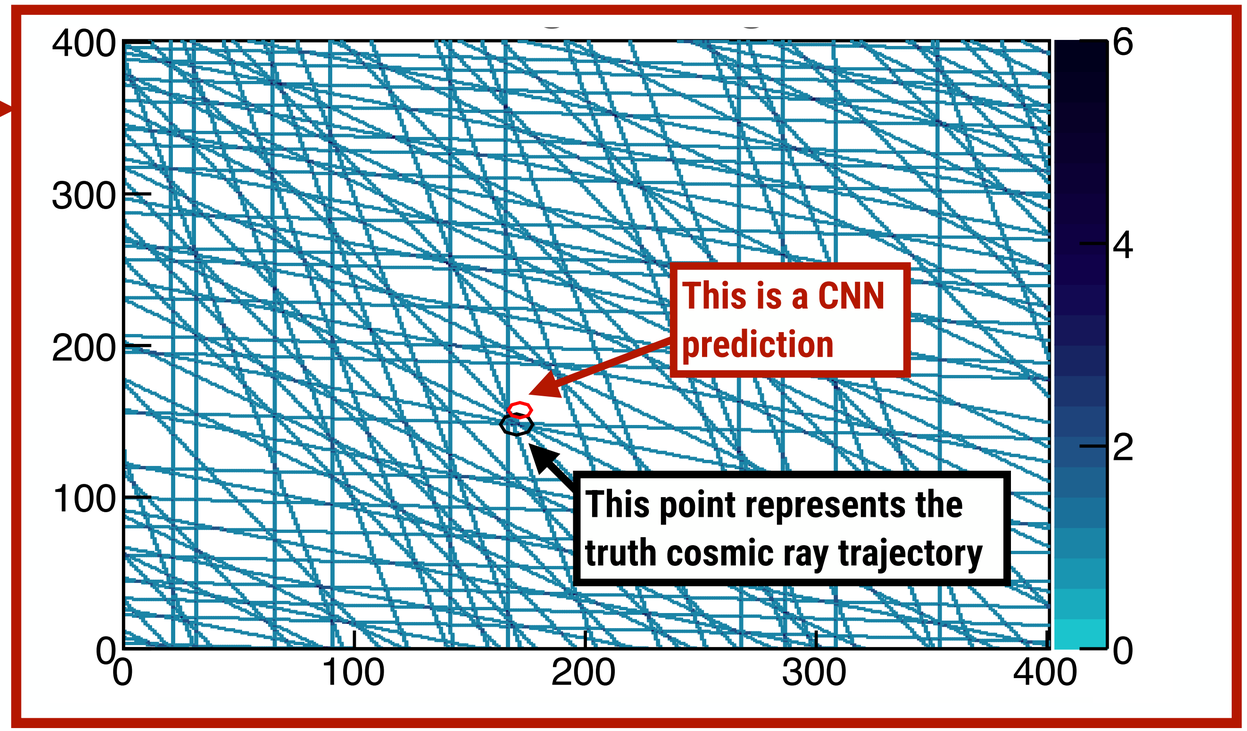
We use combined calorimeter image (two projections merged in one image) which are processed by a CNN to predict a particle direction (2 variables — intercept and slope per projection — 4 output variables in total)

Regression: reconstructing particle trajectory

The calorimeter-based CNN prediction particle trajectory with a **millimeter precision**

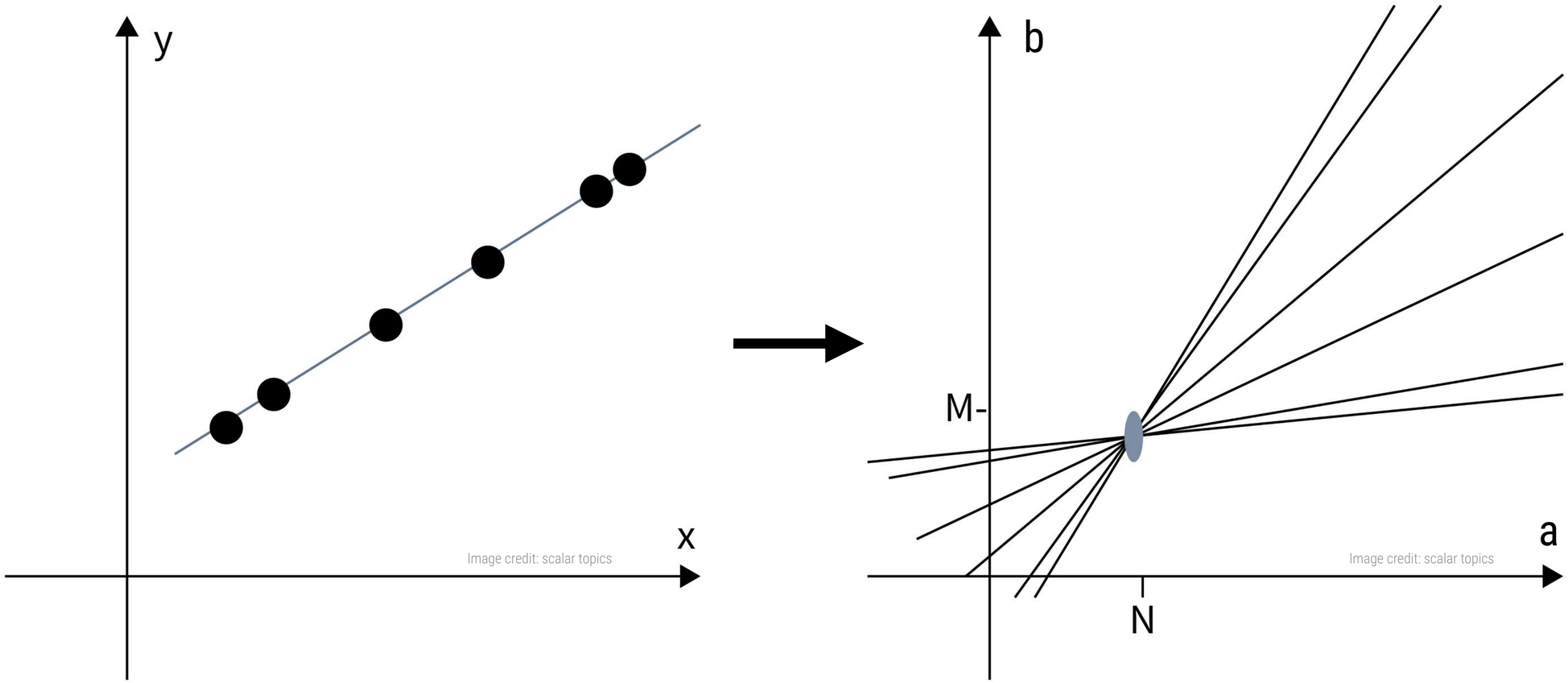


Next step is to analyze tracker data, given the few millimeters “region-of-interest” provided by the previous CNN



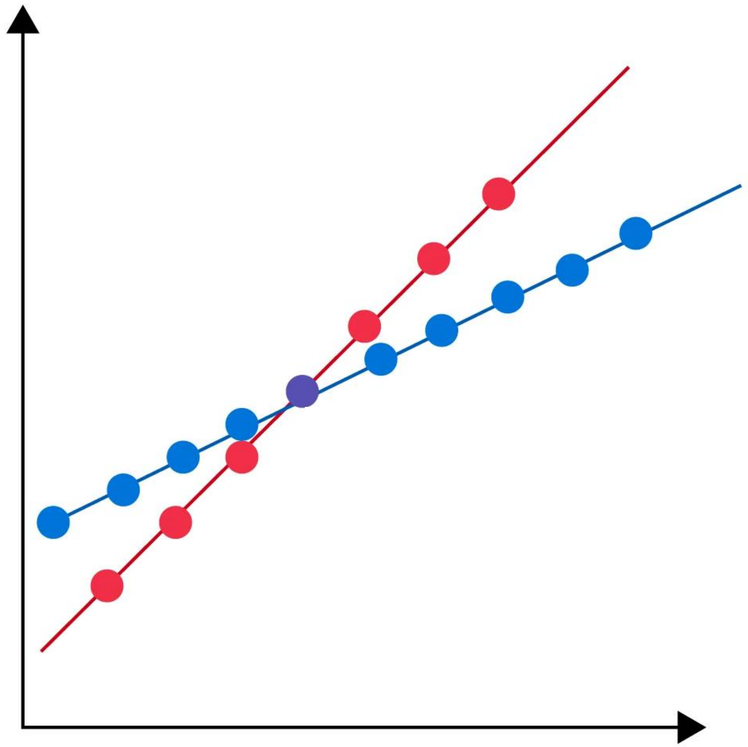
Before applying an ML algorithm to the tracker data, we convert tracker signals (points on the left image into lines using the Hough transformation!)

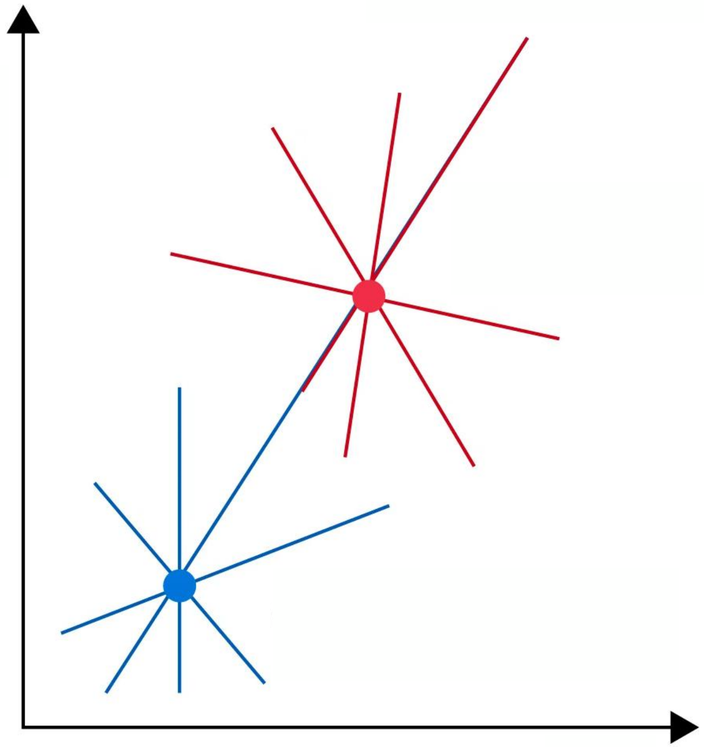
On the Hough transform (not necessarily used with CNNs)



Hough transform: every point (left image) converted into a line in the parameter space of a line questions **(y = ax + b)**

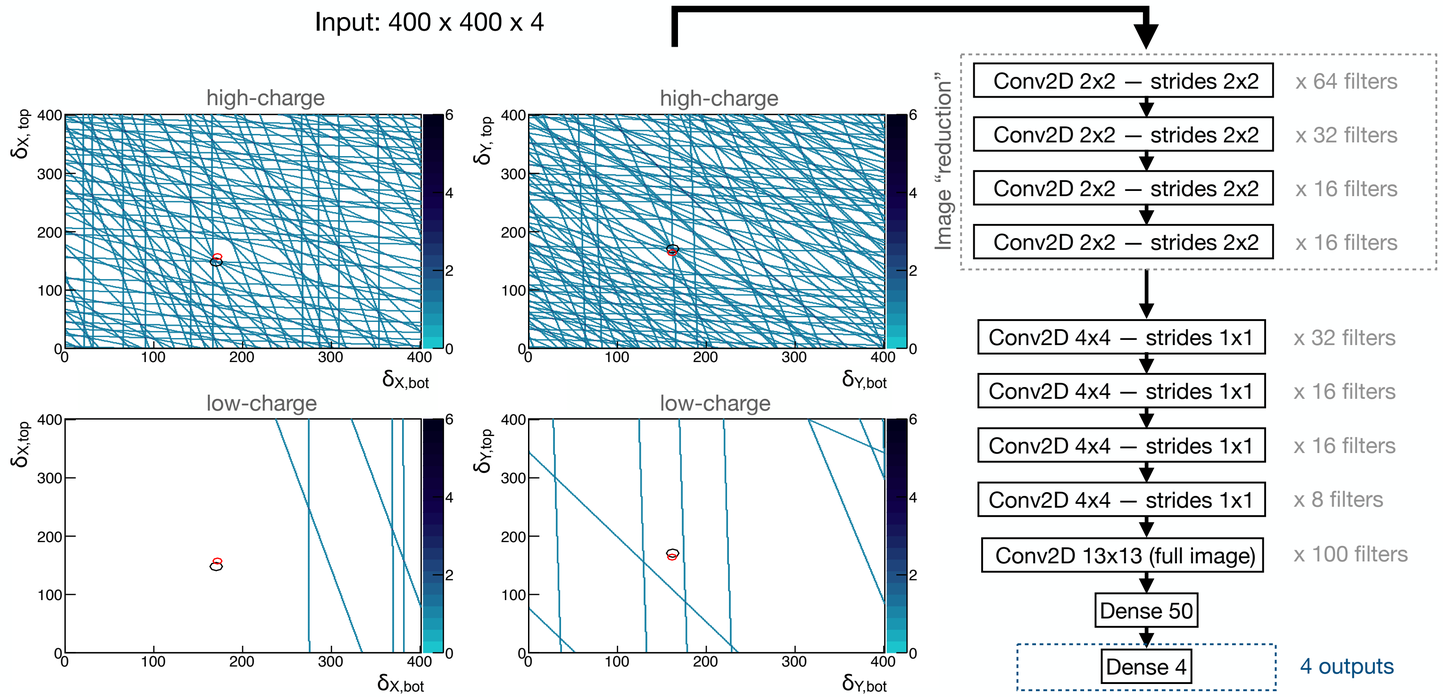
On the Hough transform (not necessarily used with CNNs)





CNNs & Hough-transformed images

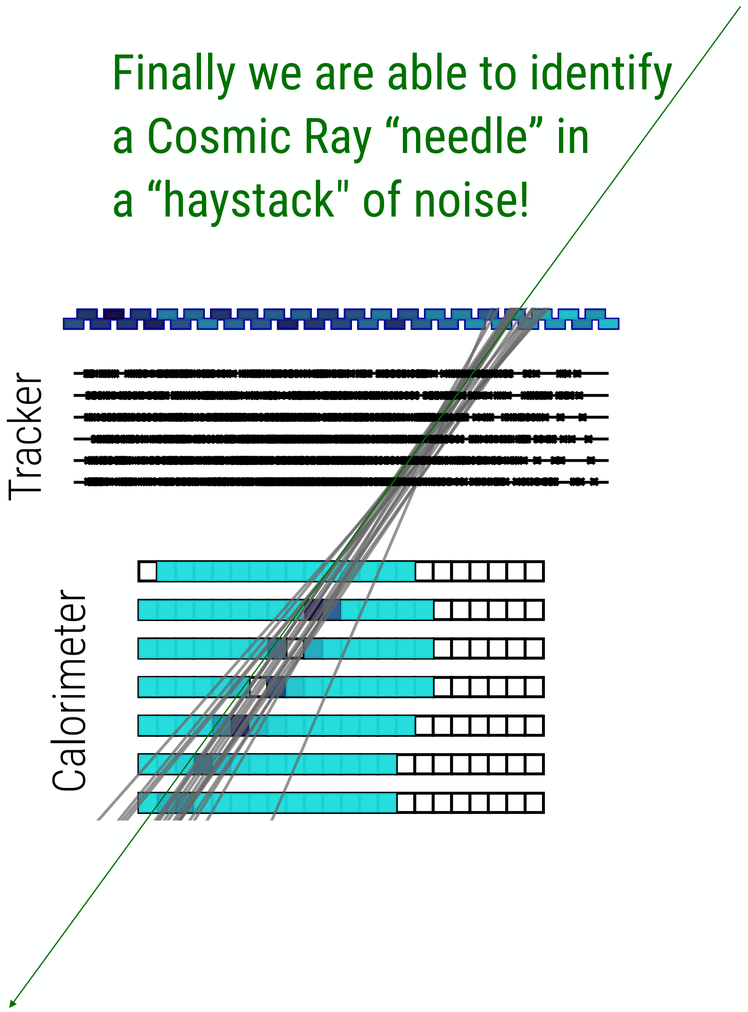
From the Region-of-Interest inferred by a calorimeter CNN, we create a Hough image (two images in two projections, we also artificially divide signals into two groups “low” and “high” - resulting in 4 images total)

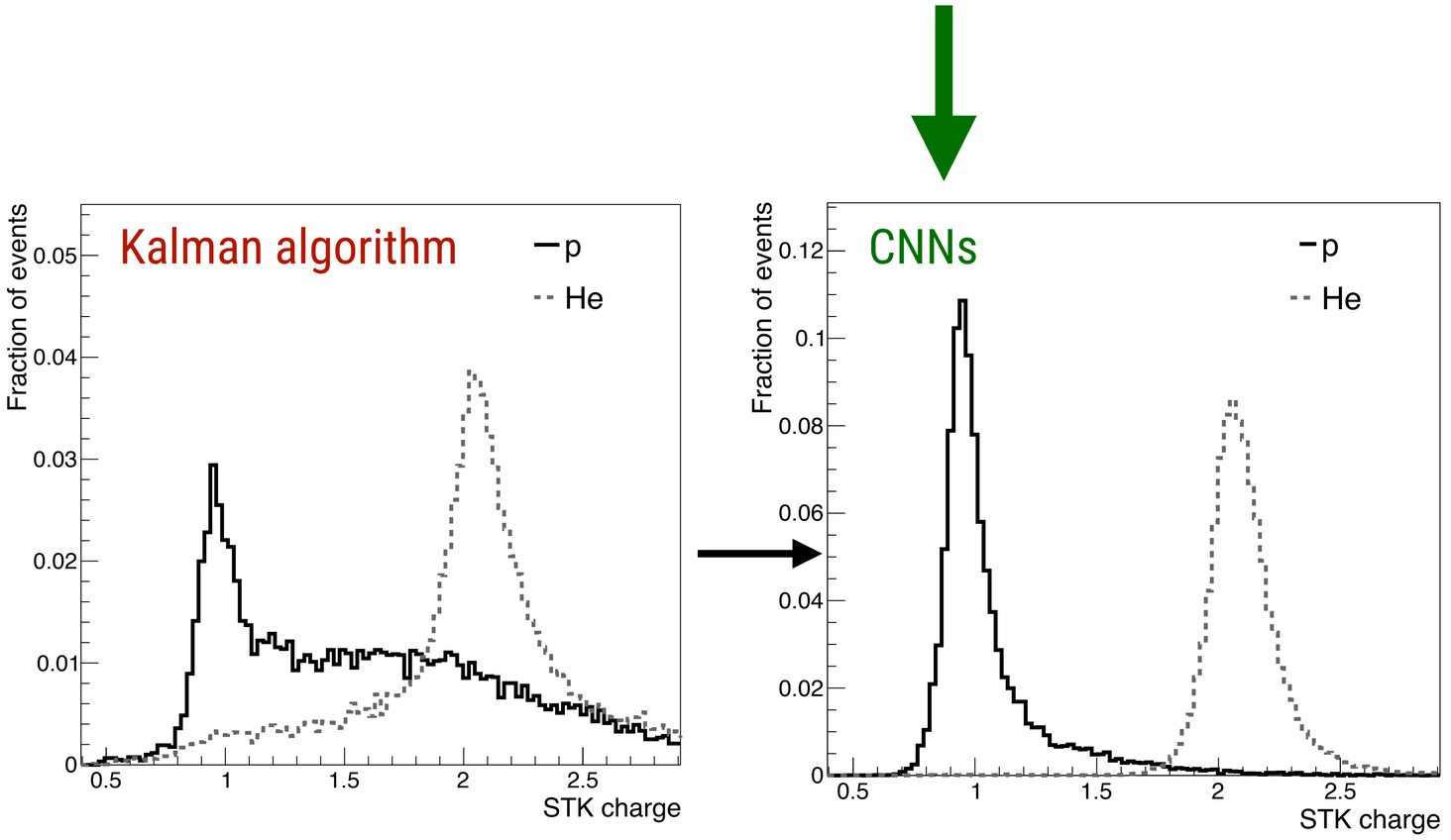


The tracker-based neural network infers particle position with **100 micron precision**!

CNNs & Hough-transformed images

Finally we are able to identify
a Cosmic Ray “needle” in
a “haystack" of noise!



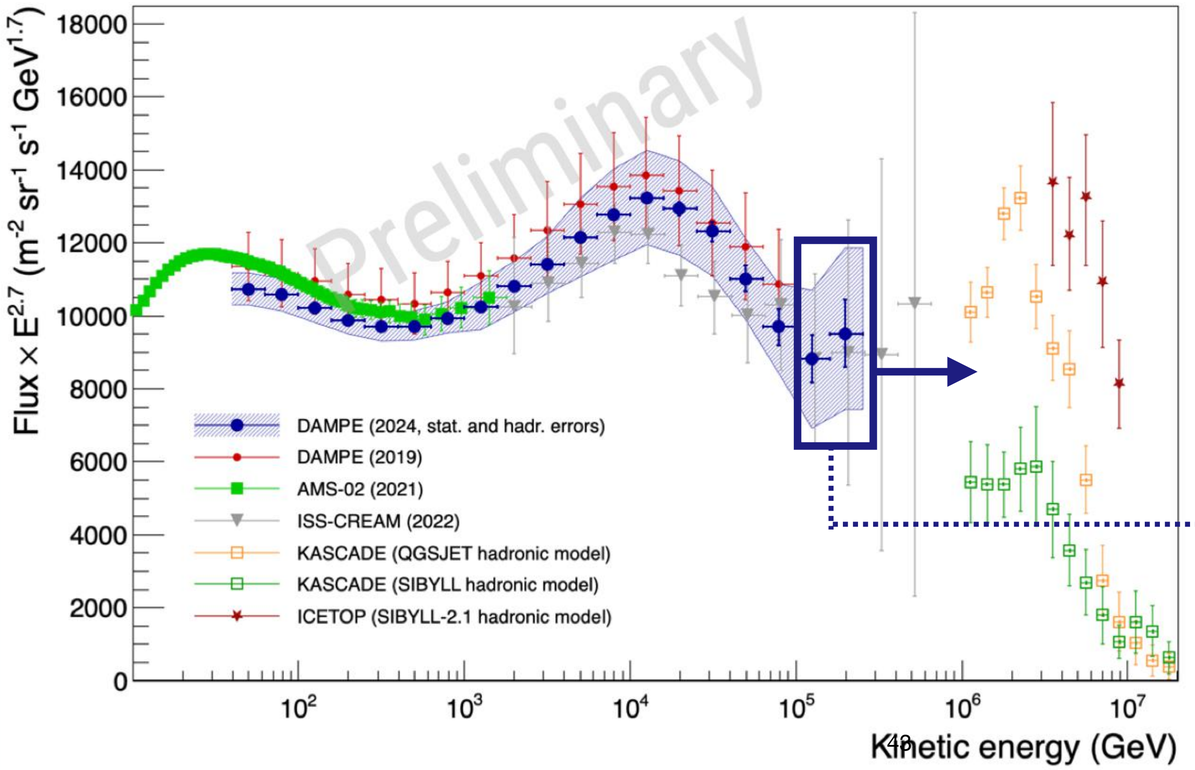


Comparison of Cosmic Ray signal (charge) obtained by analyzing the synthetic DAMPE data with two different algorithms, classical (Kalman) and ML (CNNs):

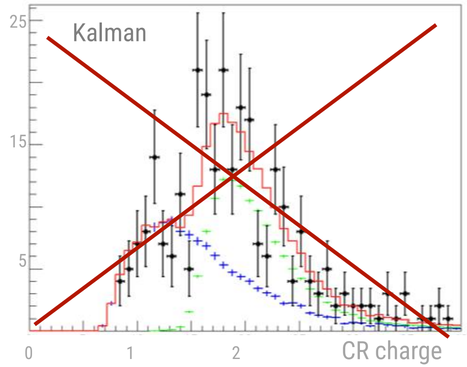
Power of CNNs

Why this is important/exciting for physics?

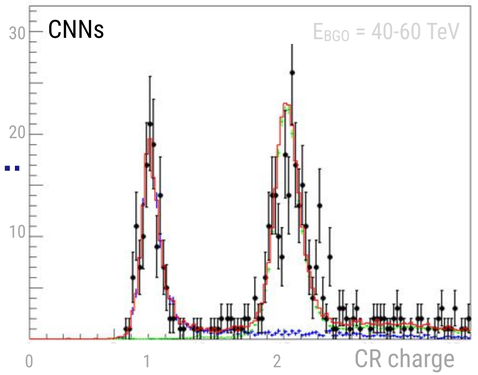
Enables the most precise extension of the DAMPE proton flux measurement beyond 100 TeV!



Before:

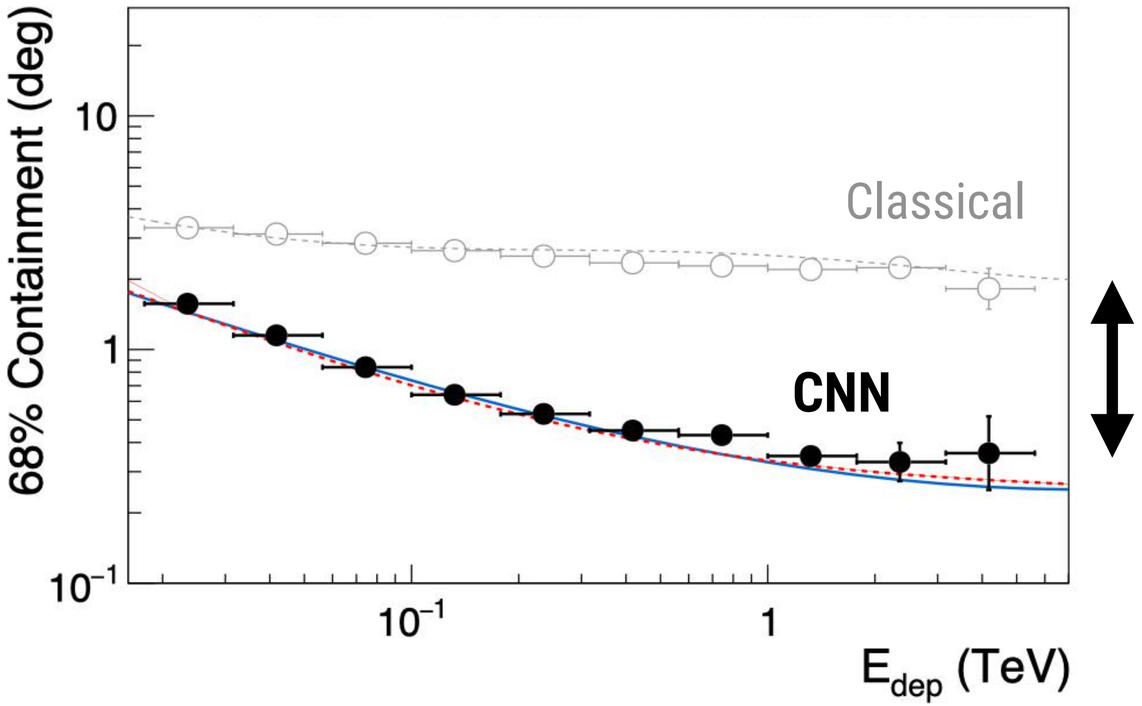


Now:

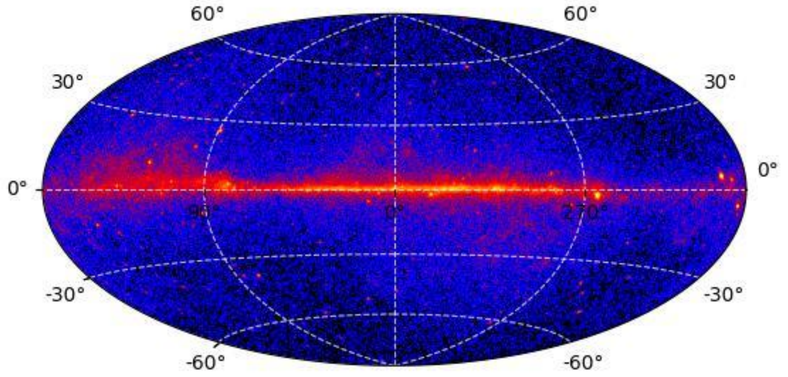


Power of CNNs

Why this is important/exciting for physics?



Our calorimeter CNN model achives ~6 times better PSF compared to classical methods!
(Key implications for gamma-ray detection)



Power of CNNs

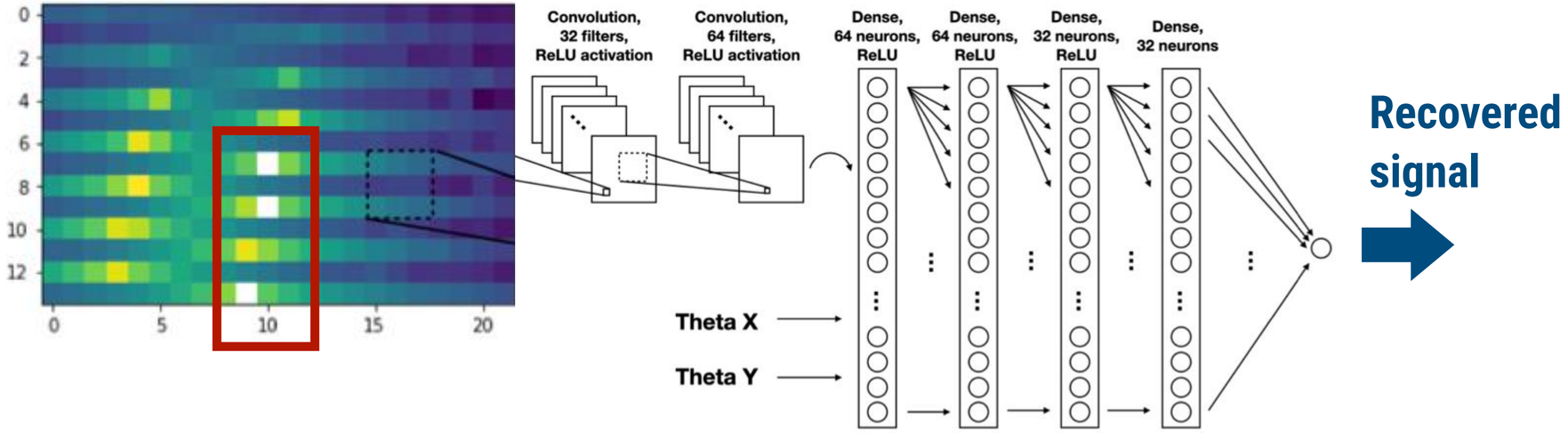
Important instrumentation implications ...

PROBLEM: Lost signal: happens at very high energies (> 100 TeV) due to the detector saturation

➡ Up to 50% of information lost in such events...

SOLUTION: We employ CNNs to recover the missing "pixels" using the image of the calorimeter

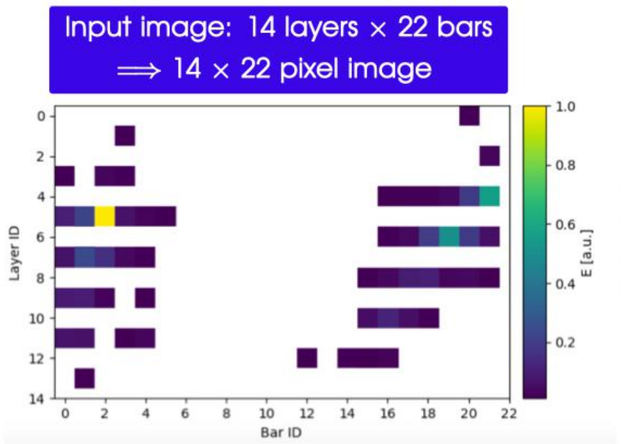
➡ recovered profiting of the information of surrounding pixels

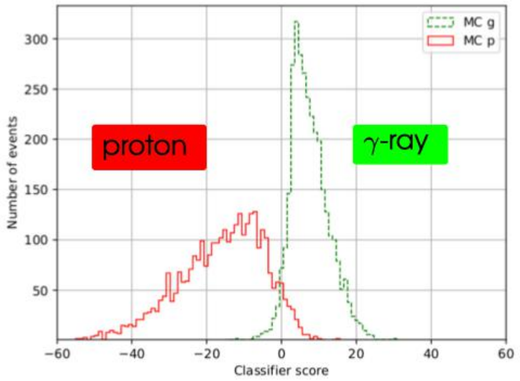


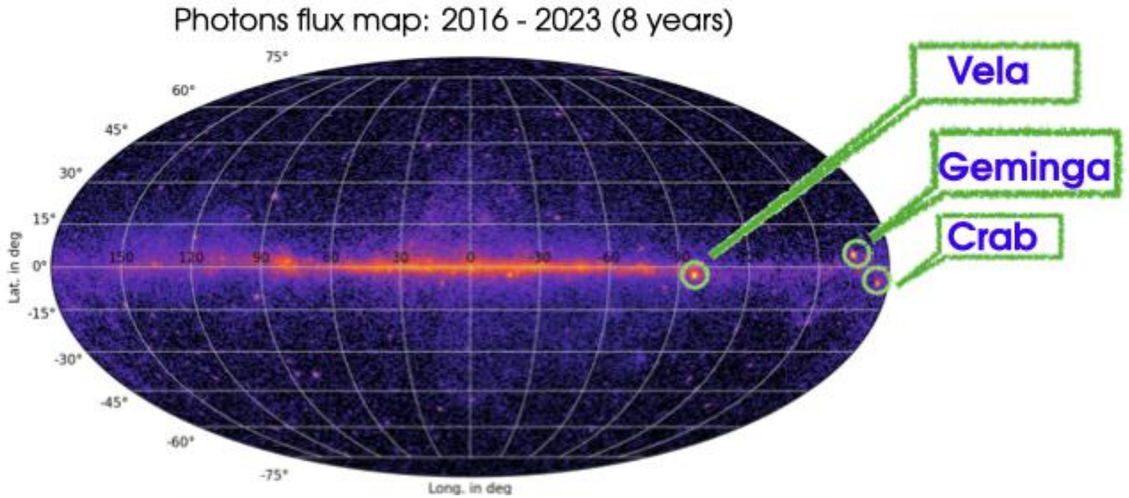
Power of CNNs

CNNs as a “working horse” of ML in DAMPE (also used in classification tasks)

- Gamma ray astronomy is part of DAMPE science
- Digging out gamma rays from overwhelming proton background is a challenge
- We use similar CNN architecture based on calorimeter images as an input for gamma-ray VS proton classification

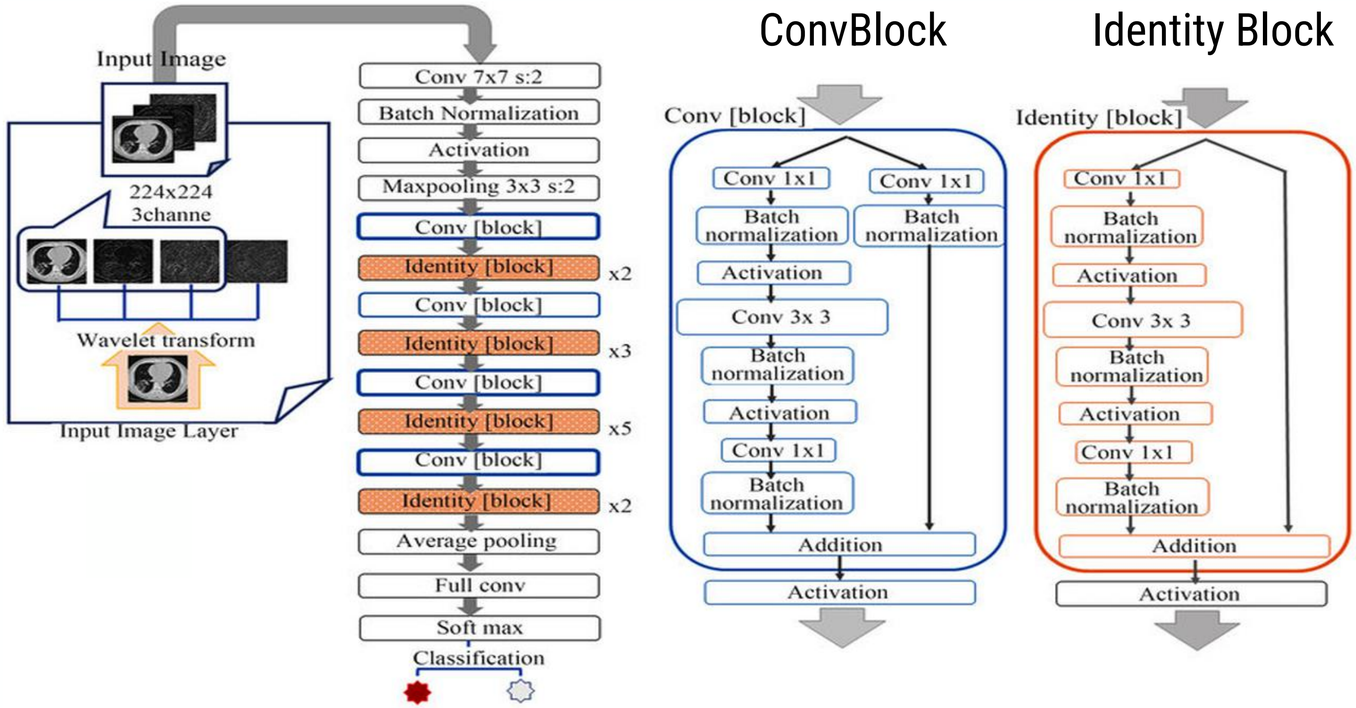






ResNet - a deep CNN

ResNet — a modern CNN architecture



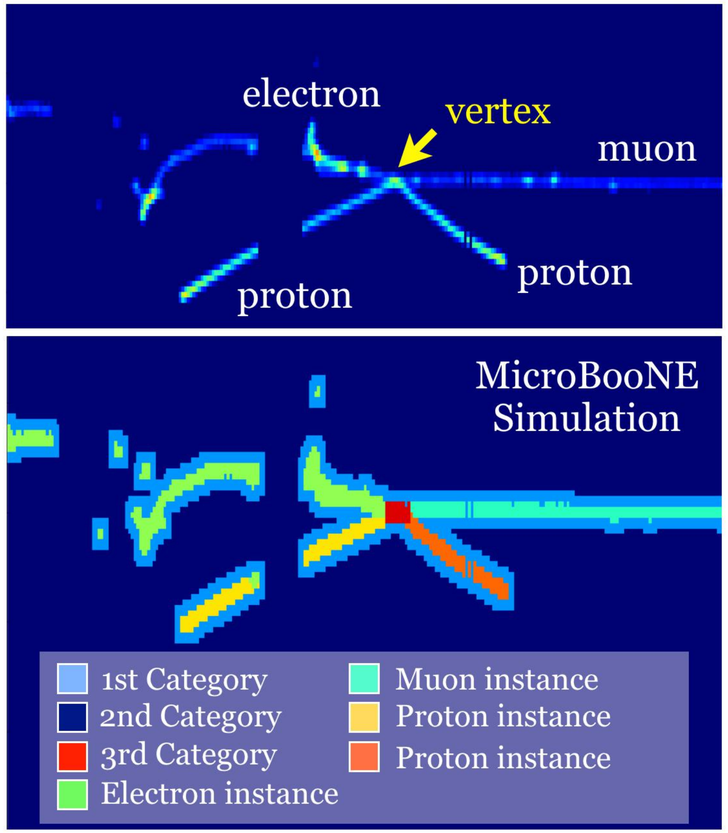
ResNet is a popular type of CNN architecture where the convolutional-processed data is added to the original image.

- The dimension either remains unchanged (Identity block) or is reduced (Conv Block).
- The ResNet allows to train very deep networks avoiding the vanishing gradient problem.
- The ResNet model represents a stack of repeating Conv and Identity blocks (e.g. 50 blocks)

https://wisdomml.in/category/computer-vision/cv-tutorial/

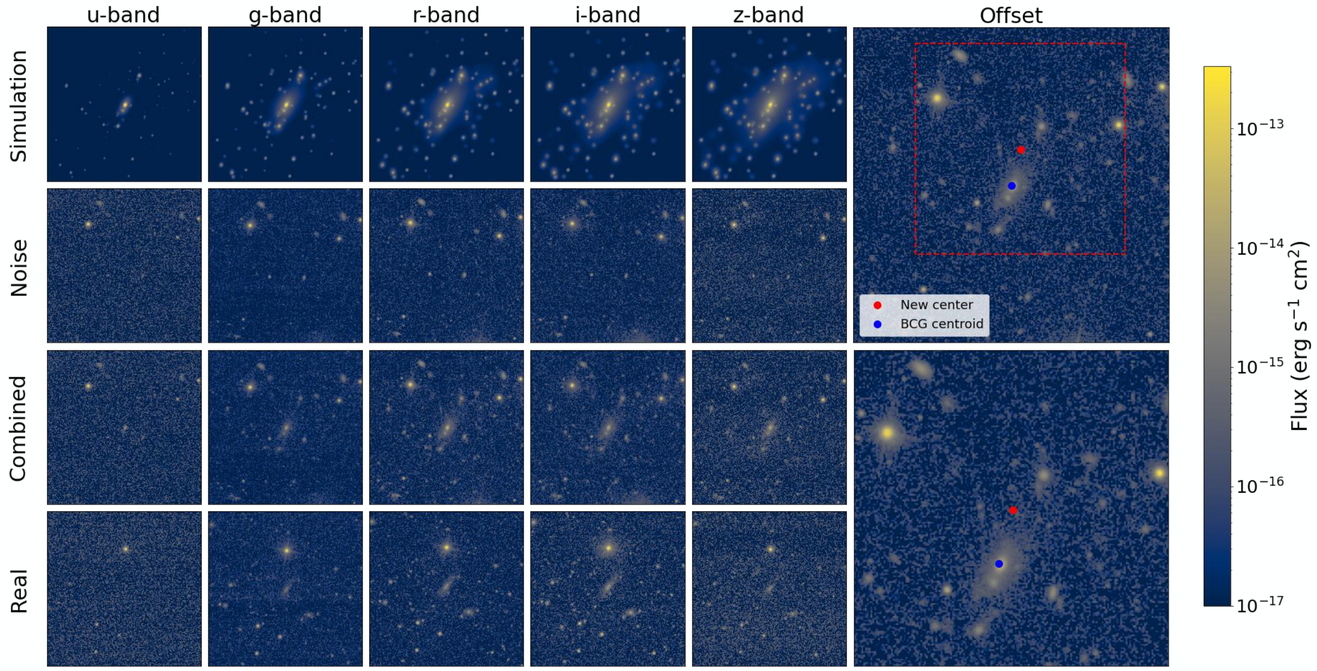
ResNet - a deep CNN

Pixel-level particle identification
in MicroBooNE Liquid Argon
Time Projection Chamber



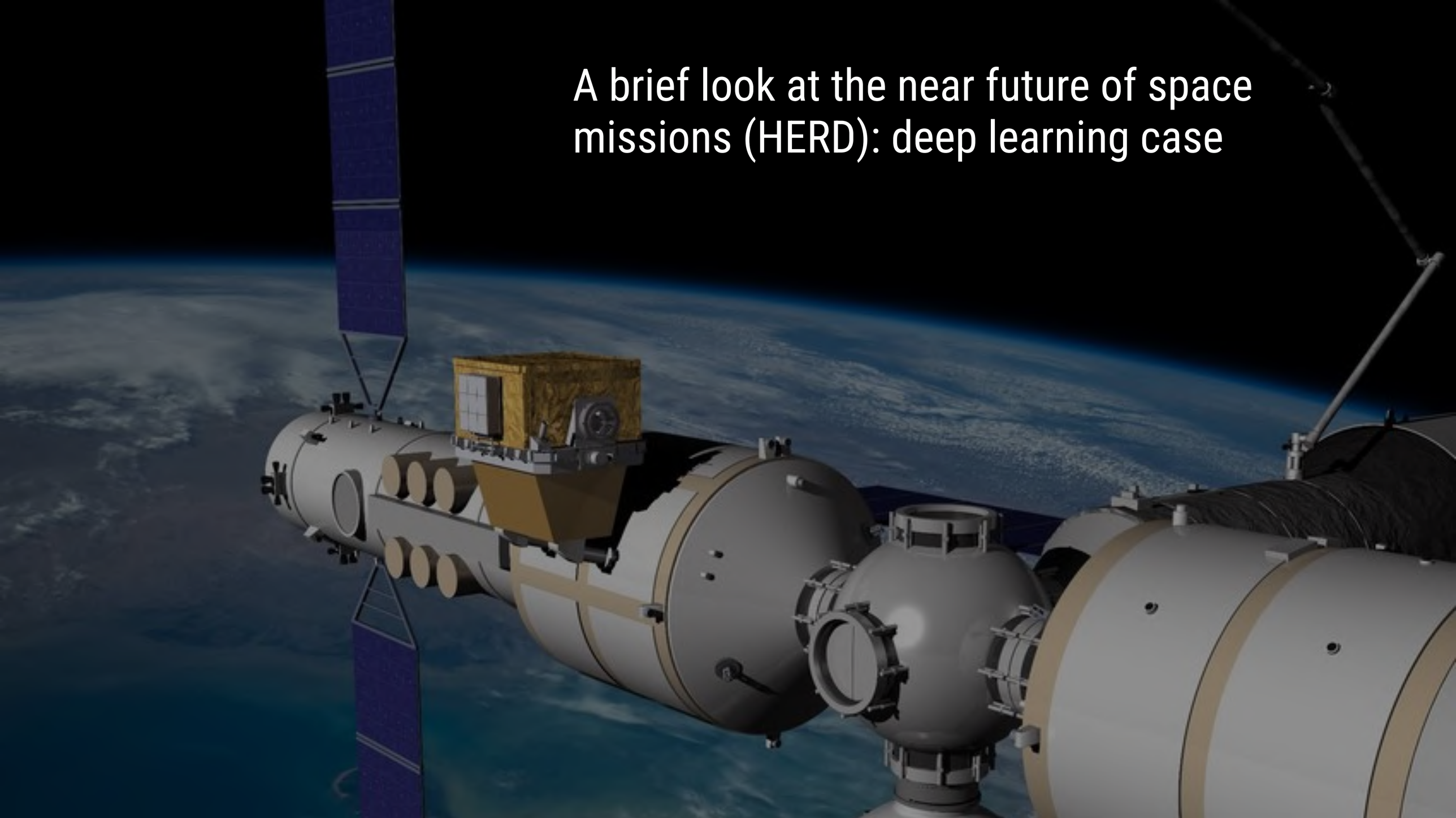
PhysRevD.99.092001,
arXiv.1808.07269

Identification of Brightest Cluster
Galaxies in Large Surveys



arXiv:2502.00104

A brief look at the near future of space missions (HERD): deep learning case

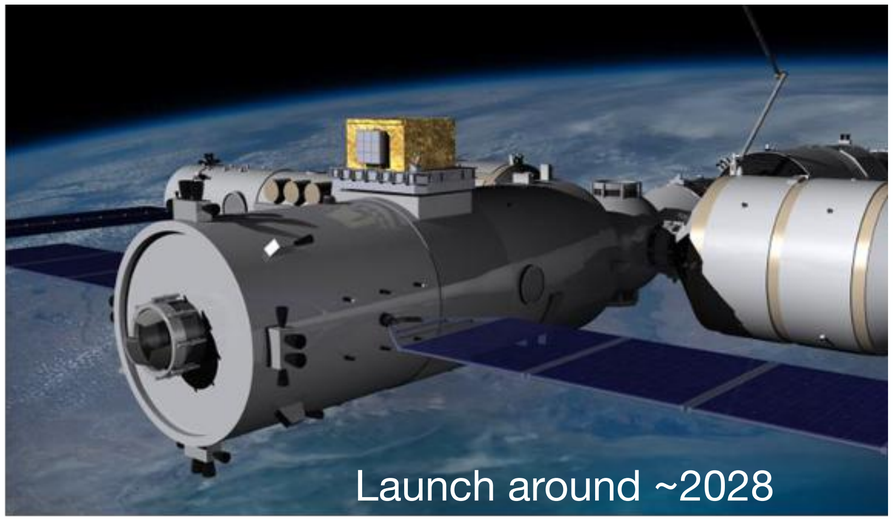


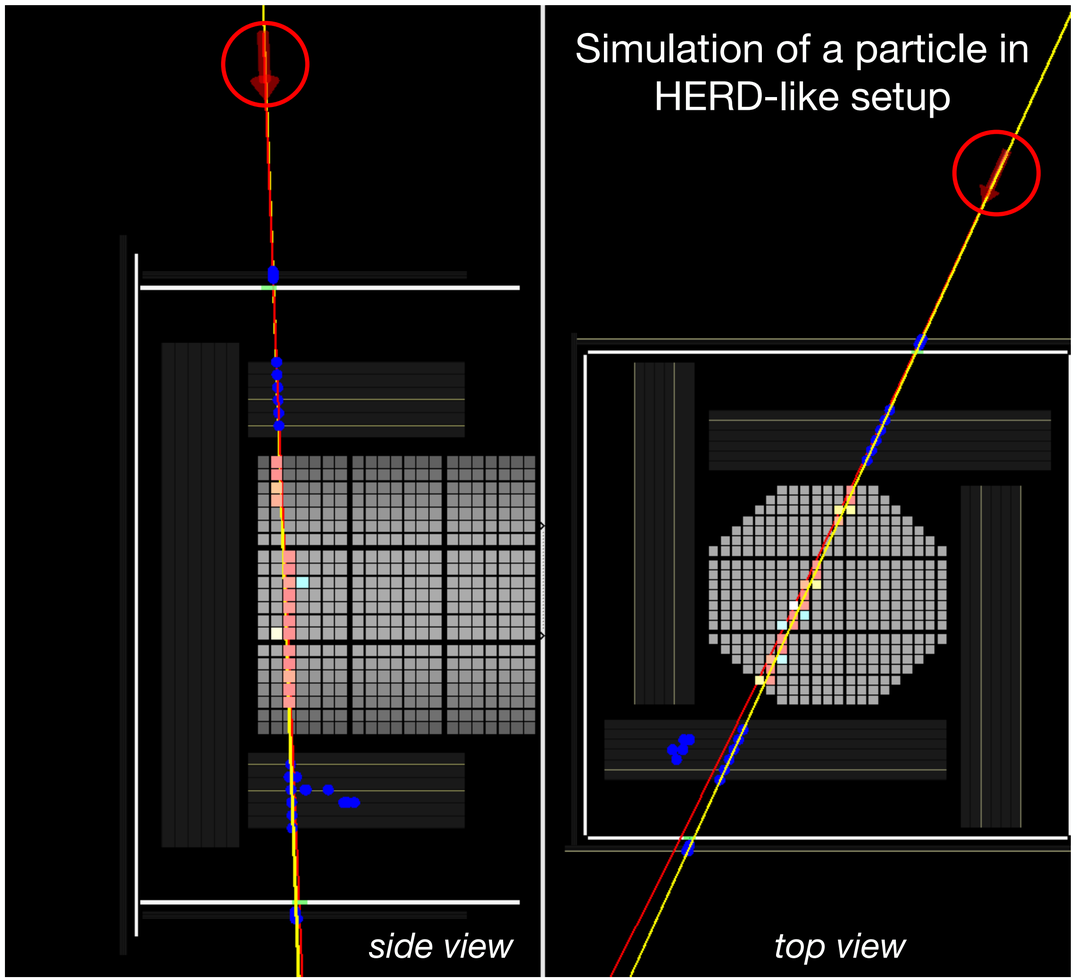
HERD (High Energy cosmic Radiation Detector)

3D calorimeter of 55X_0 (3Λ) + 5-side tracker

- CR electrons up to 100 TeV
- CR p/ions up to PeVs
- x 10 acceptance compared to DAMPE

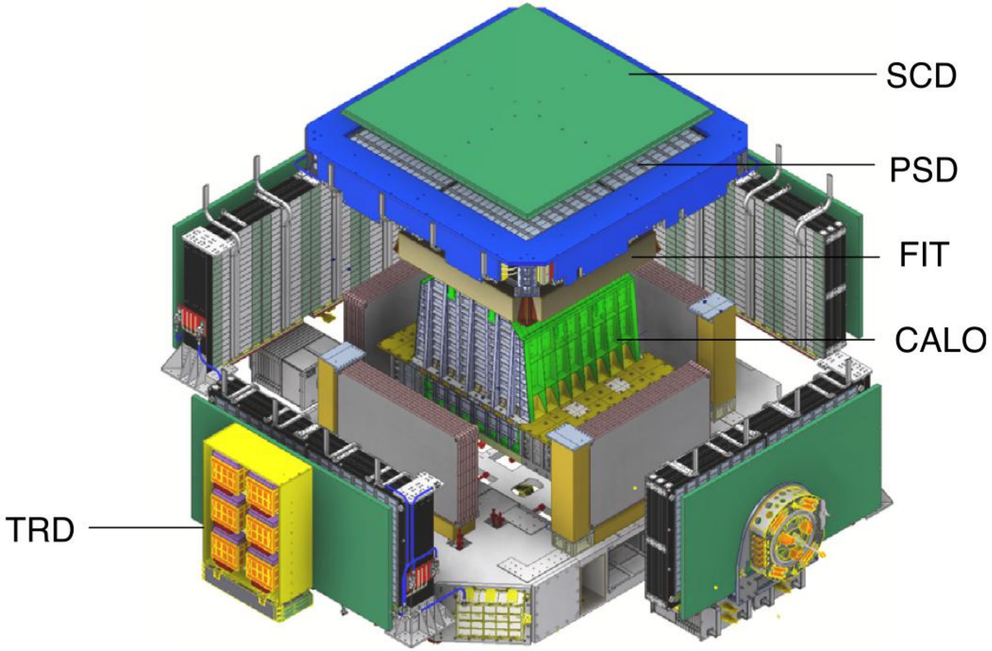
→ hundreds of PeV cosmic rays / year

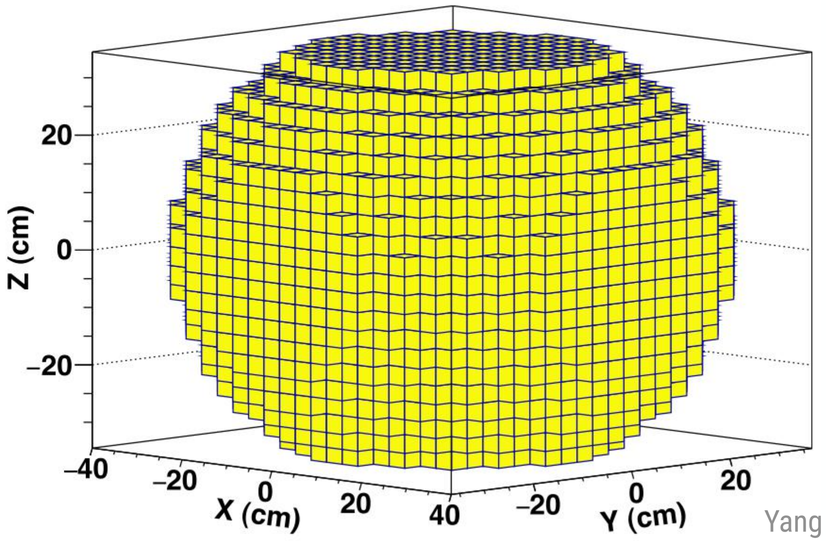




HERD - the first 3D calorimeter in space

One **order of magnitude higher complexity** of the detector (compared to DAMPE) — ideal "playground" for deep learning ...



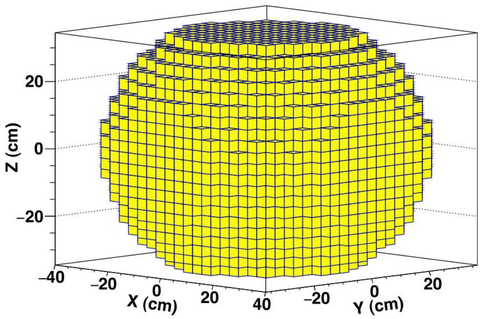


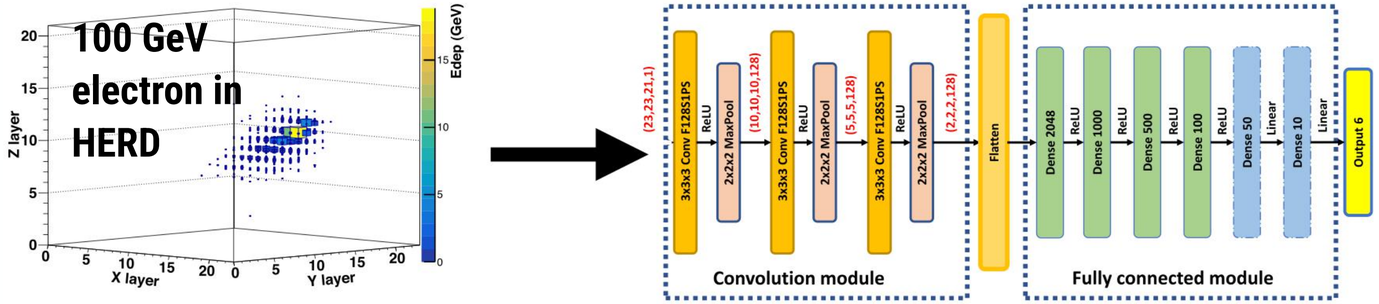
Yang et al. NIM A 1066 (2024) 169571

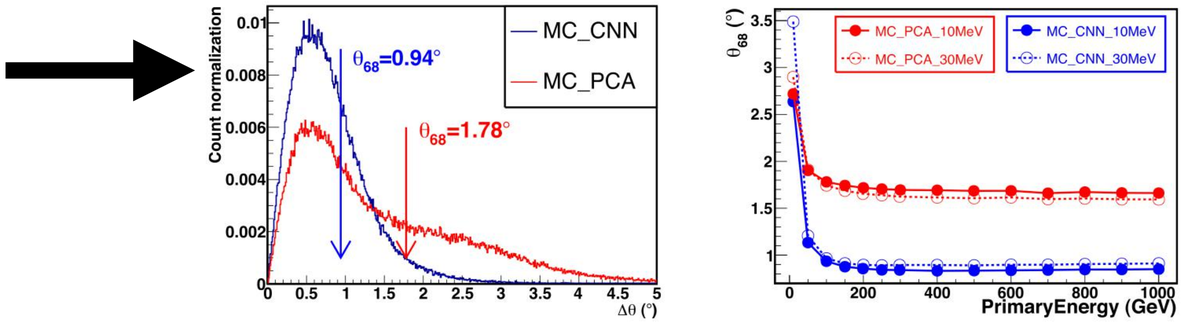
3D CNN for HERD

Unlike DAMPE, HERD features is a fully 3D calorimeter

- Inspired by the DAMPE example, a **3D CNN developed**
- CNN shows twice better PSF at 100 GeV compared to classical methods
- ... surprisingly less prominent improvement compared to DAMPE case)

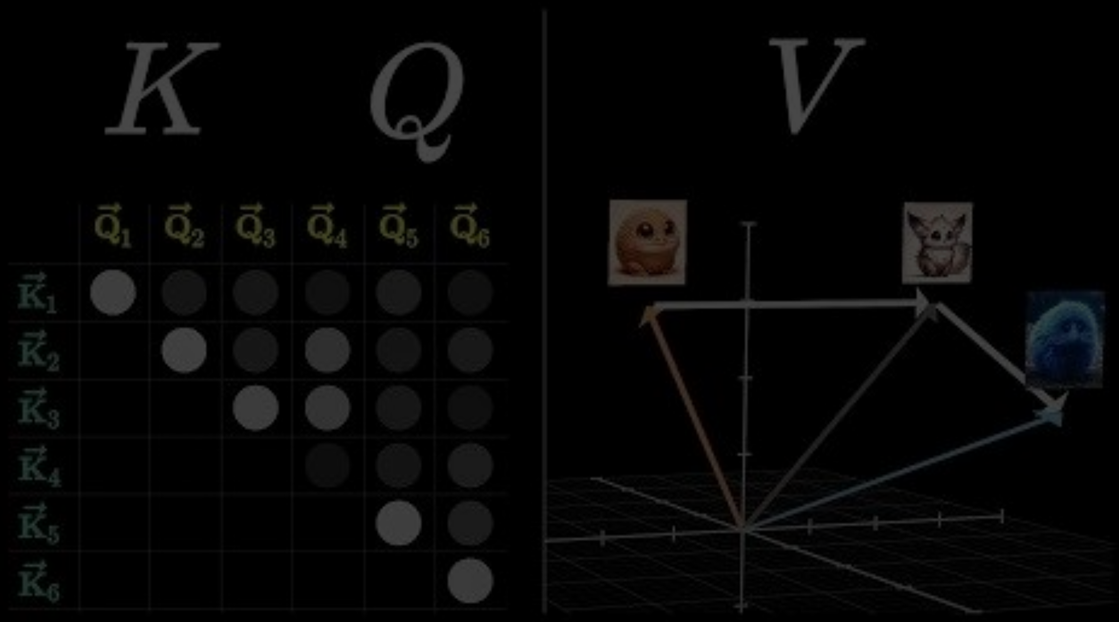






Yang et al. NIM A 1066 (2024) 169571

Beyond CNNs
(Transformers, GNNs, Normalizing flows) ...



Transformers @ AMS

- Astro-particle community is also exploring modern deep learning methods
  - AI enthusiasts at AMS claim that **Transformers** outperform CNNs (including ResNet) by about one order of magnitude and generalize better when trained / applied across different energies (still to be confirmed with real data)

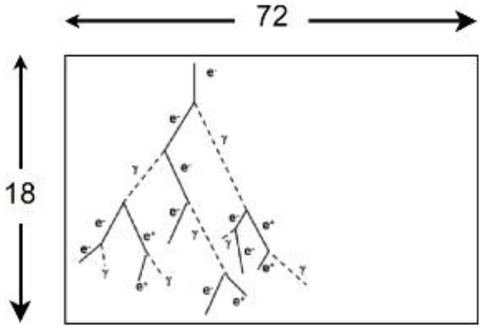
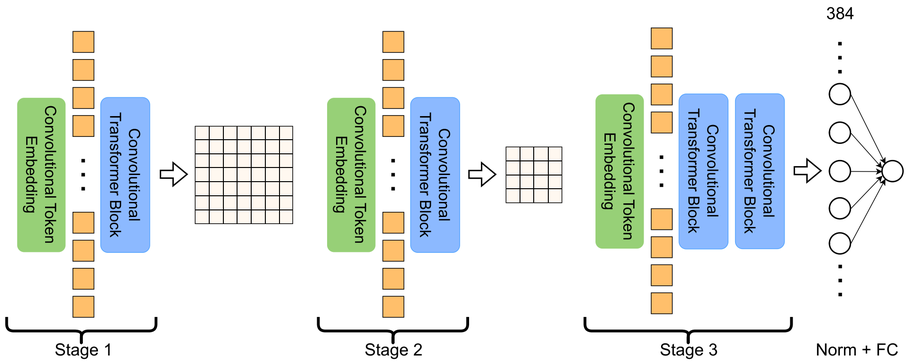
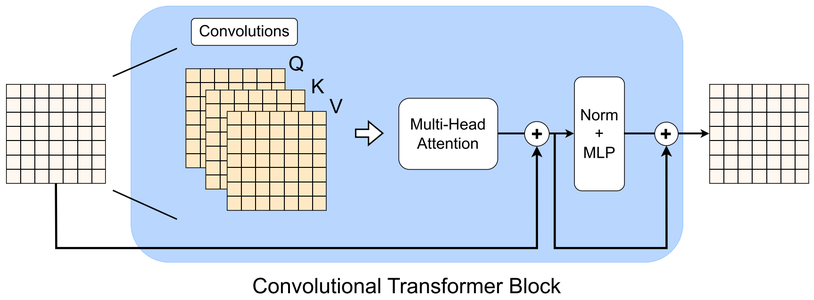


Image of particle shower in AMS-02 calorimeter is taken as an input (similar to DAMPE and HERD cases)



(a)

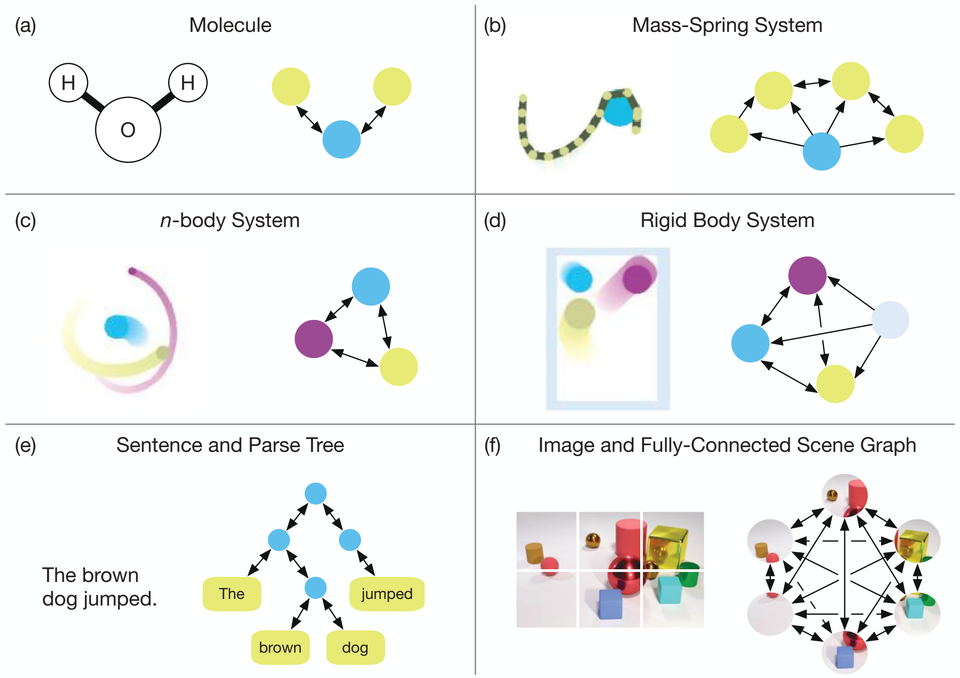


Convolutional Transformer Block

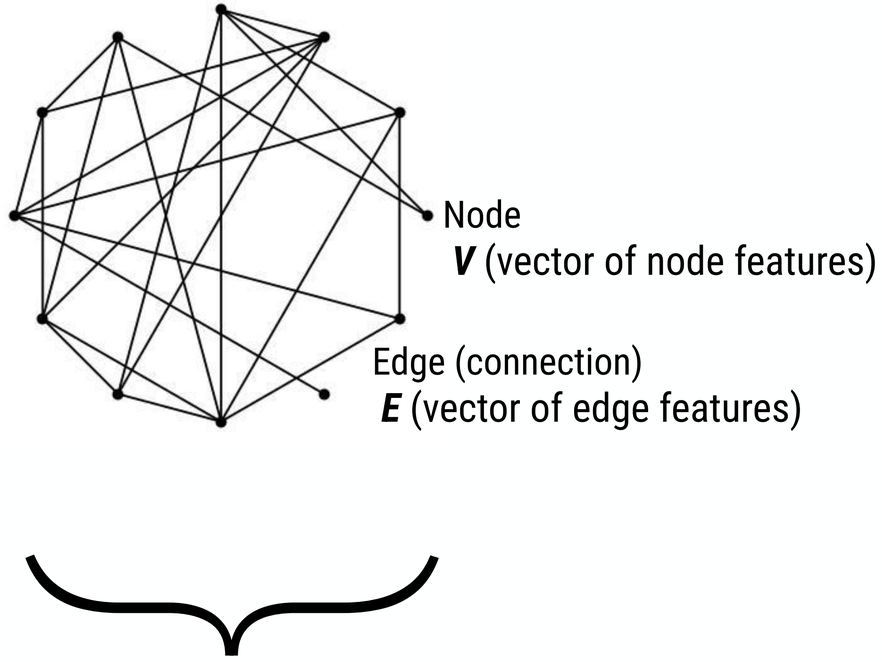
Validation of ML models with real data is probably the most challenging part of applying AI in space astroparticle physics!

Graph Neural Networks (GNN)

- Not always the data can be represented in a grid-like form compatible with image (hence CNNs)
- A more generic/universal representation of data is graph - anything can be presented as a graph



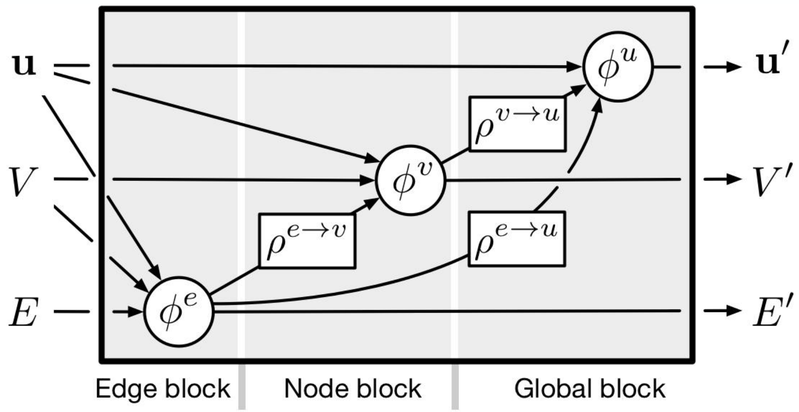
arXiv:1806.01261v3



Graph Neural Networks (GNN)

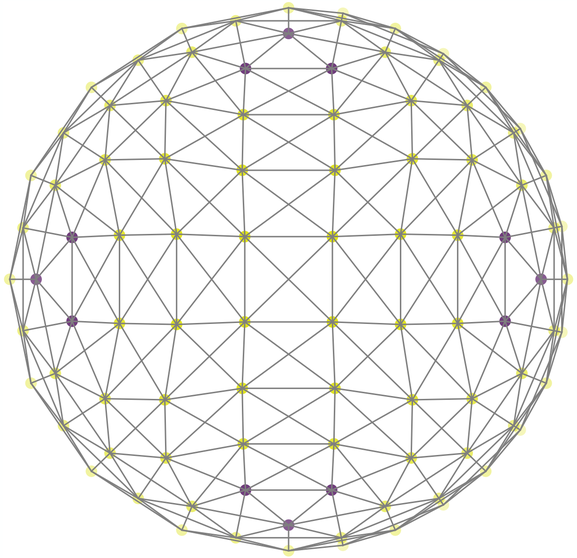

```
# Generated with chat-GPT
# GNN code pseudo-code generated by GPT
# Inputs:
# - V: node features (list of v_i)
# - E: edge features (list of (e_k, sender, receiver))
# - u: global feature

# Output:
# - Updated V, E, u

def GN_layer(V, E, u):
    # Edge update
    for each edge k:  # e_k connects nodes i -> j
        sender, receiver = E[k].sender, E[k].receiver
        e_k = E[k].features
        E[k].features = phi_e(e_k, V[sender], V[receiver], u)

    # Node update
    for each node i:
        incoming_edges = [e_k for e_k in E if e_k.receiver == i]
        aggregated_edge_info = aggregate_e([e.features for e in incoming_edges])
        V[i] = phi_v(V[i], aggregated_edge_info, u)

    # Global update
    aggregated_nodes = aggregate_v(V)
    aggregated_edges = aggregate_e([e.features for e in E])
    u = phi_u(u, aggregated_nodes, aggregated_edges)

    return V, E, u
```

A GNN is essentially a set of layers, where each layer performs a transformation of node, edge, and global feature vectors, updating each vector based on information from related nodes and edges



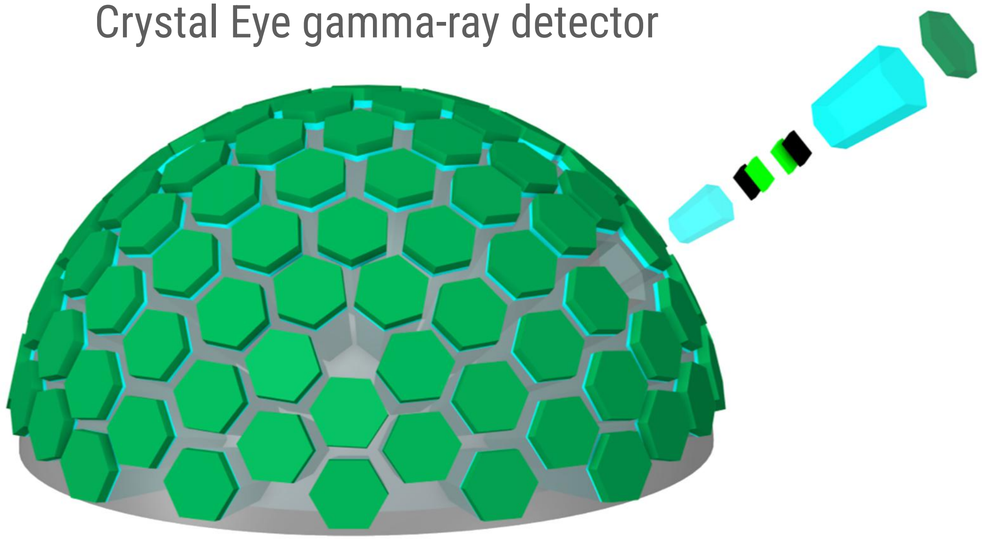
GNNs ands spherical images (Crystal Eye ?)

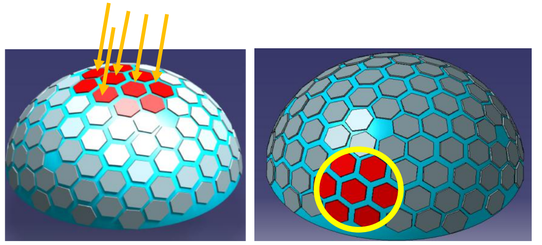
- A good example of a possible GNN application is the Crystal Eye gamma ray detector
  - ➡ sphere mesh of (scintillator) hexagons - difficult to represent as a flat image without distortions - a possible way to circumvent the mesh as a graph:



The spherical CNN is constructed by representing the sphere as a graph.
arXiv:1810.12186v2

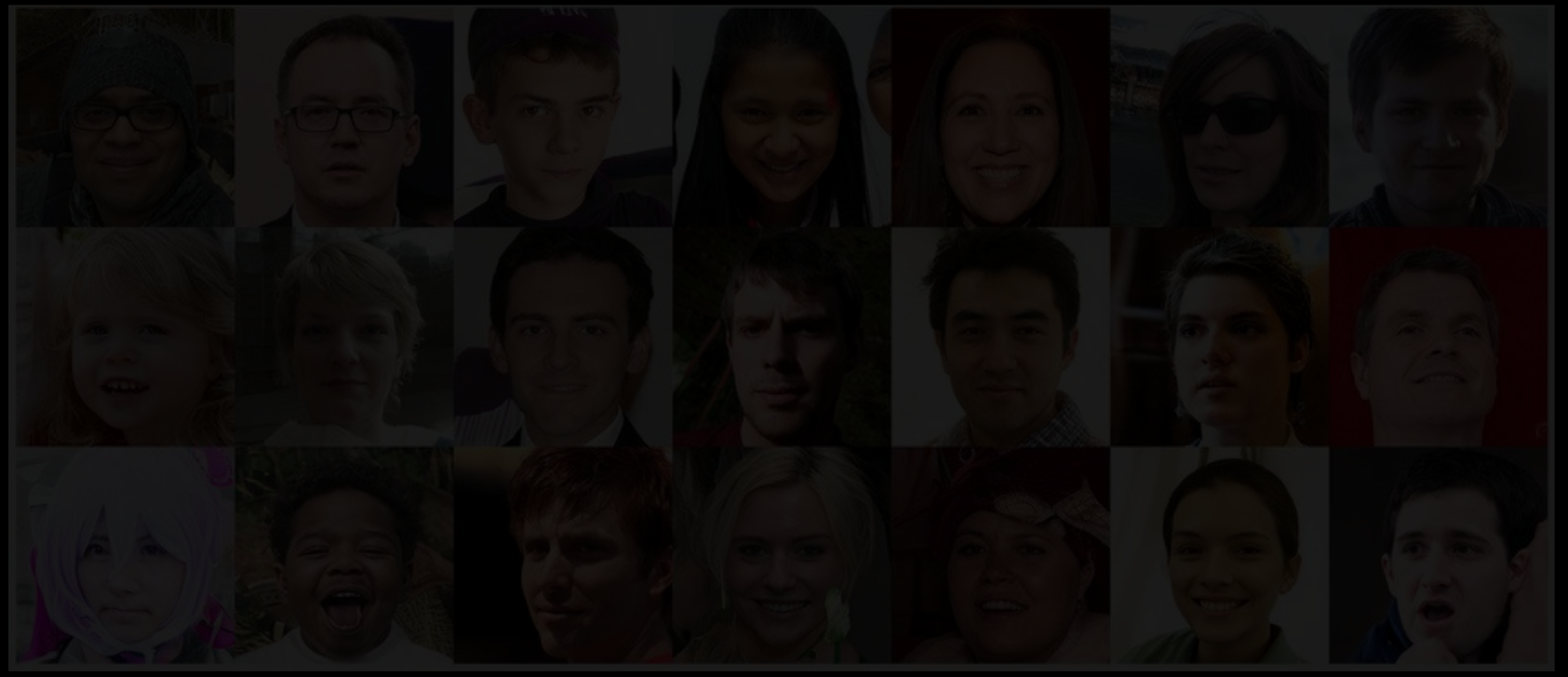
Crystal Eye gamma-ray detector





https://indico.gssi.it/event/227/contributions/559/attachments/261/383/Crystal_Eye.pdf

Generative Models



Normalizing flows

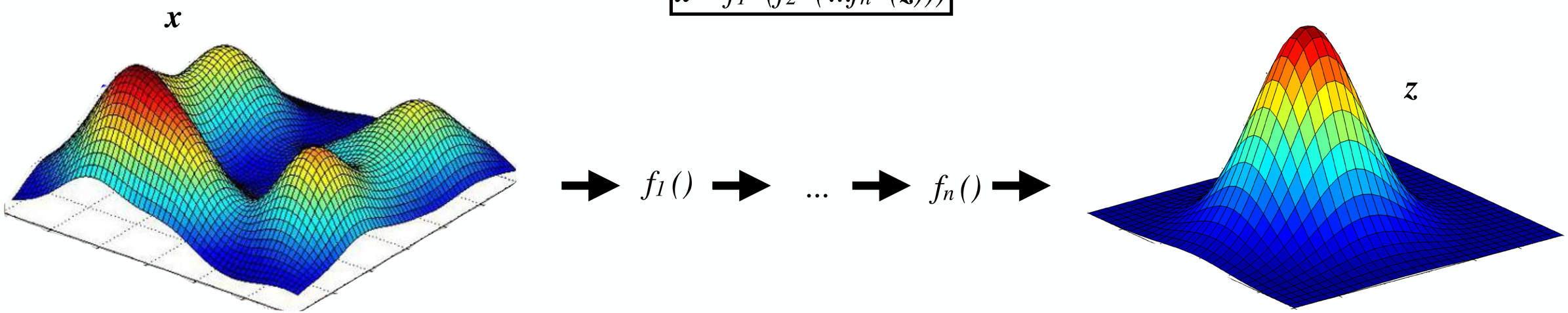
Normalizing flow is a machine learning technique that serves two main purposes:

1. Explicitly learn the probability distribution of complex data
2. Generate new data (given the learned distribution)

This is achieved by converting the original complex data representation (x) into a simple one (z) -usually a multidimensional normal distribution - through a series of **invertible** and **differentiable** transformations

$$z = f_n(f_{n-1}(..f_1(x)))$$

$$x = f_1^{-1}(f_2^{-1}(..f_n^{-1}(z)))$$



Normalizing flows

Normalizing flow usually use specific invertible NN layers - **affine coupling layers**

➡ Can be considered as wrappers around the usual (e.g. fully-connected) NN layers

➡ One of the tricks: split the data vector in two parts - one converted, the other not

Direct transformation:

$$y_1 = x_1$$
$$y_2 = x_2 \odot \exp(s(x_1)) + t(x_1)$$

Inverted transformation:

$$x_1 = y_1$$
$$x_2 = (y_2 - t(y_1)) \odot \exp(-s(y_1))$$

s and ***t*** are usual (fully-connected) NN transformations

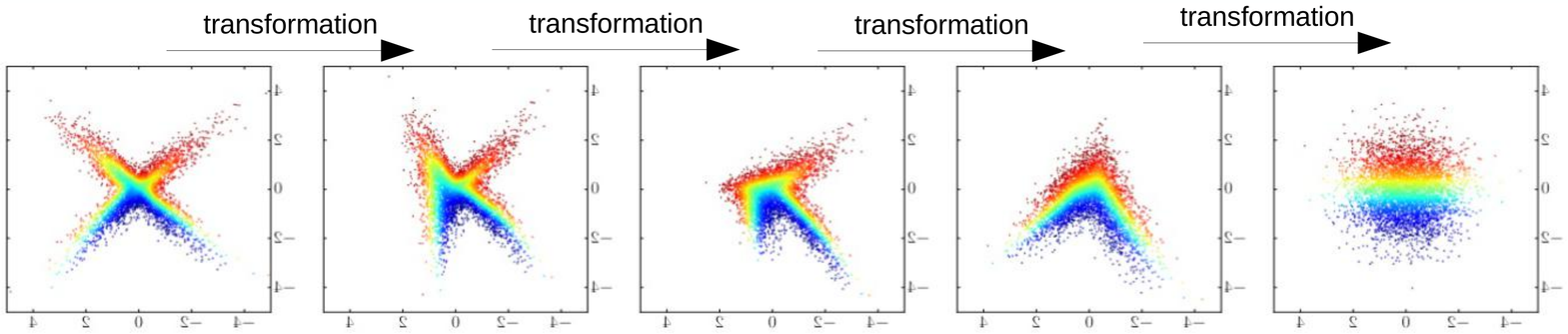


Image credit: Jonas Glombitza

PyTorch normalizing flow tutorial with MNIST dataset (hand-written numbers):
https://uvadlc-notebooks.readthedocs.io/en/latest/tutorial_notebooks/tutorial11/NF_image_m

Normalizing flow training

Training of normalizing flow network is done by maximizing the log likelihood $p_x(x)$:

$$p_x(x) = p_z(f^{-1}(x)) \left| \det \left(\frac{\partial f^{-1}(x)}{\partial x} \right) \right|$$

$$p_x(x) = p_z(z) \left| \det \left(\frac{\partial f(z)}{\partial z} \right) \right|^{-1}$$

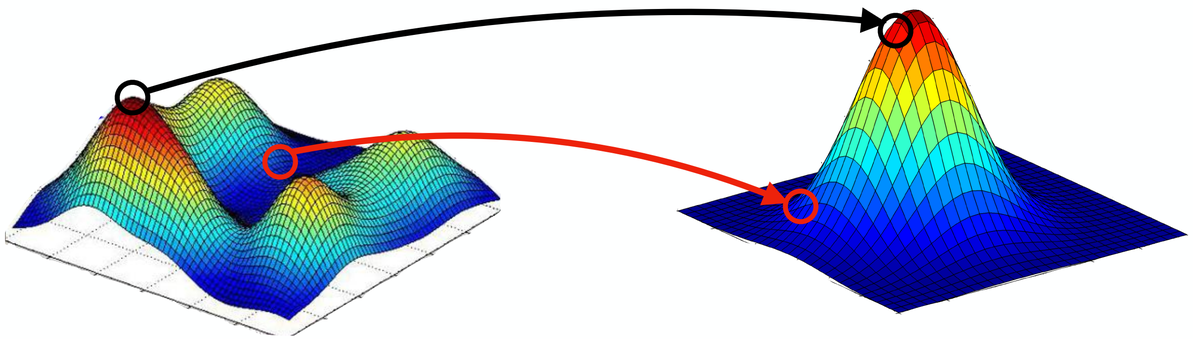
Fixed Trained

$$\sum \left(\log p_z(f^{-1}(x_i)) + \log \left| \det \left(\frac{\partial f^{-1}(x_i)}{\partial x} \right) \right| \right)$$

where:

$p_z(z)$ - fixed (multi-dimensional Gaussian)

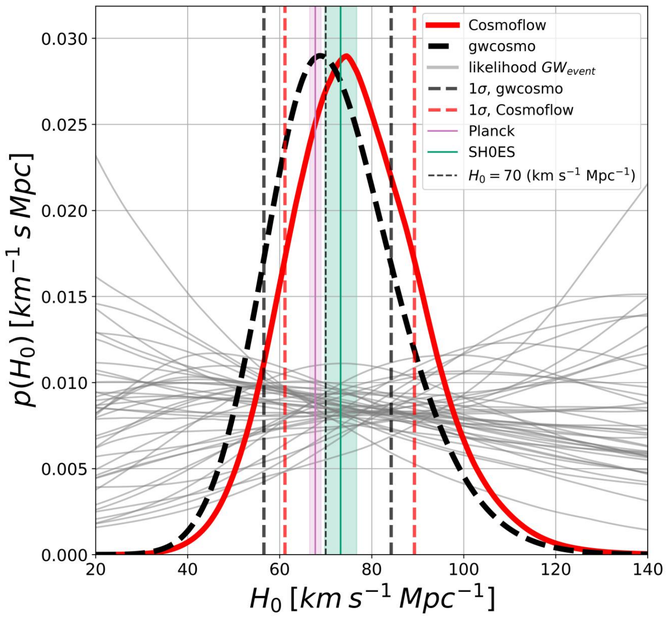
det(...) is the Jacobian of the transformation



In other words, the transformation **f(...)** itself during training is "guided" towards representing Gaussian as close as possible

Normalizing Flows application examples

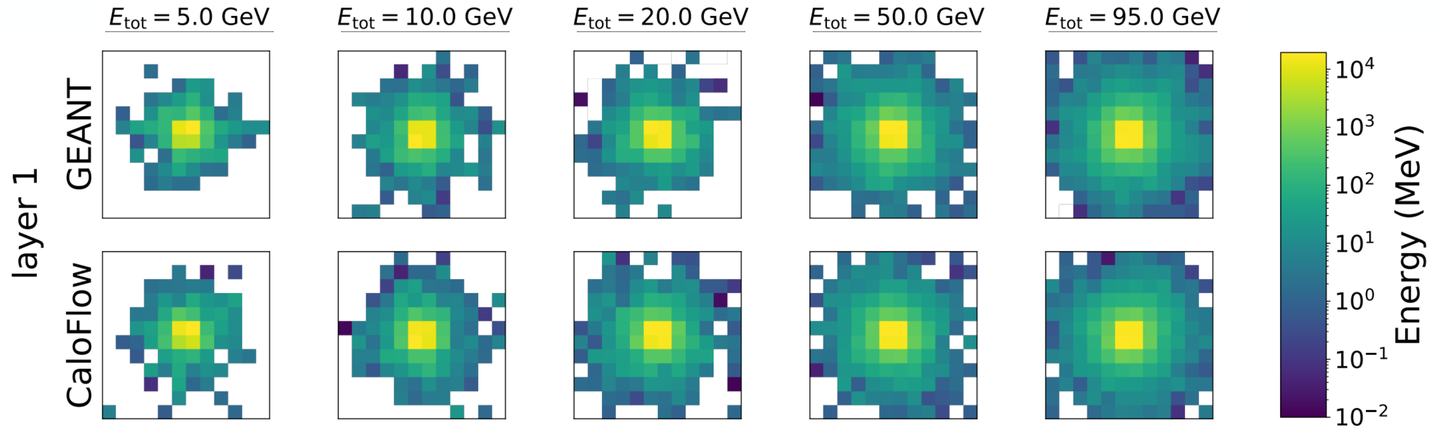
Modeling realistic probability distributions
➡ Inference of cosmological parameters



Example: Hubble constant evaluation from Gravitation Wave detection: arXiv:2310.13405v2

See also: arXiv:2105.12024v1 - modeling probability distributions in cosmology

Generative models
➡ Fast replacement of computationally intense GEANT4 (or perhaps CORSIKA?) simulations

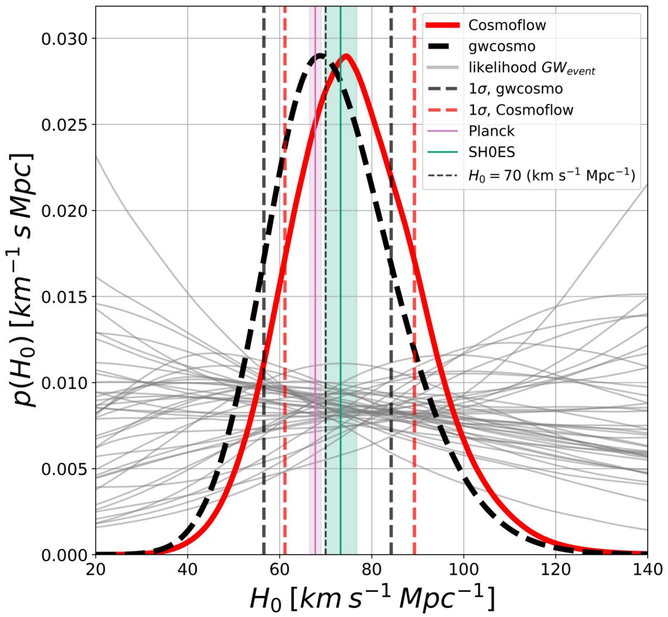


Example: Fast and Accurate Generation of Calorimeter Showers with Normalizing Flows: arXiv:2106.05285v3

Normalizing Flows application examples

Modeling realistic probability distributions

➡ Inference of cosmological parameters

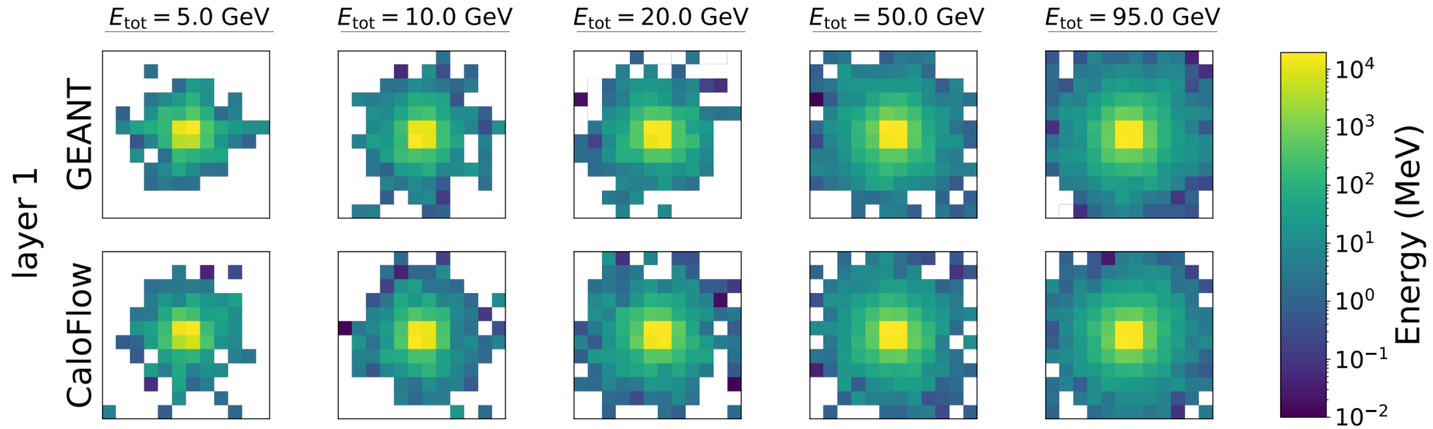


Example: Hubble constant evaluation from Gravitation Wave detection: arXiv:2310.13405v2

See also: arXiv:2105.12024v1 - modeling probability distributions in cosmology

Generative models

➡ Fast replacement of computationally intense GEANT4 (or perhaps CORSIKA?) simulations



Example: Fast and Accurate Generation of Calorimeter Showers with Normalizing Flows: arXiv:2106.05285v3

On Generative models

GAN: minimax the classification error loss.

VAE: maximize ELBO.

Flow-based generative models: minimize the negative log-likelihood

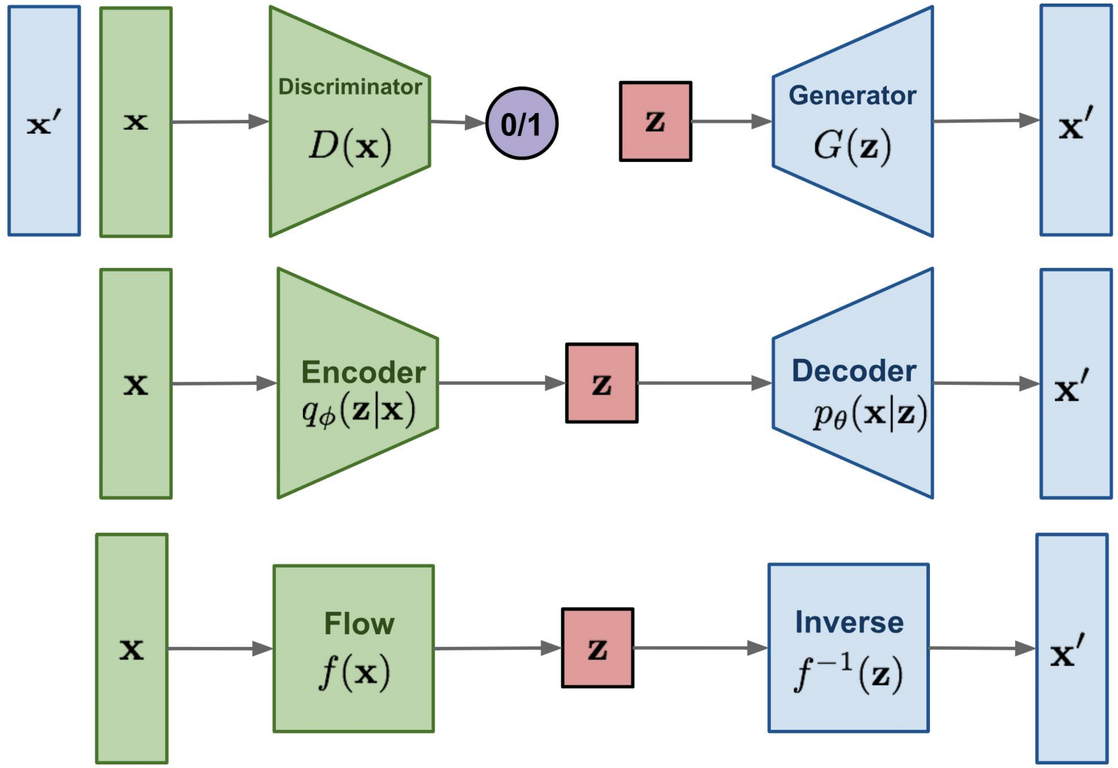
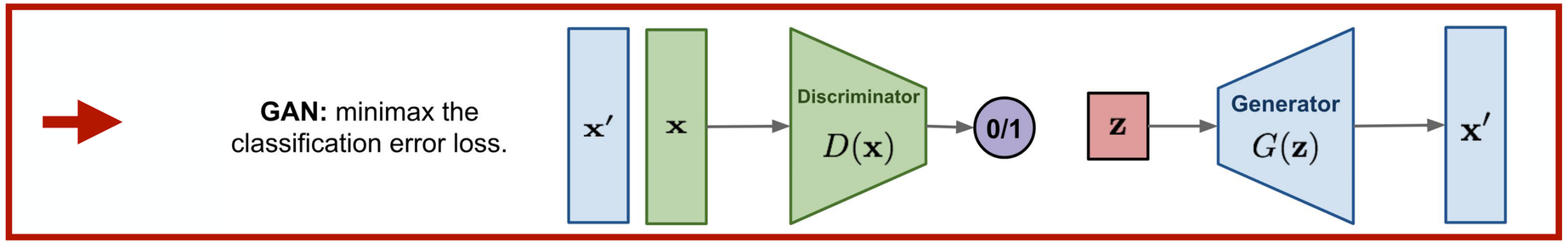


Image credit lilianweng.github.io/posts/2018-10-13-flow-models

Discussed previously

On Generative models: Generative Adversarial Networks (GANs)

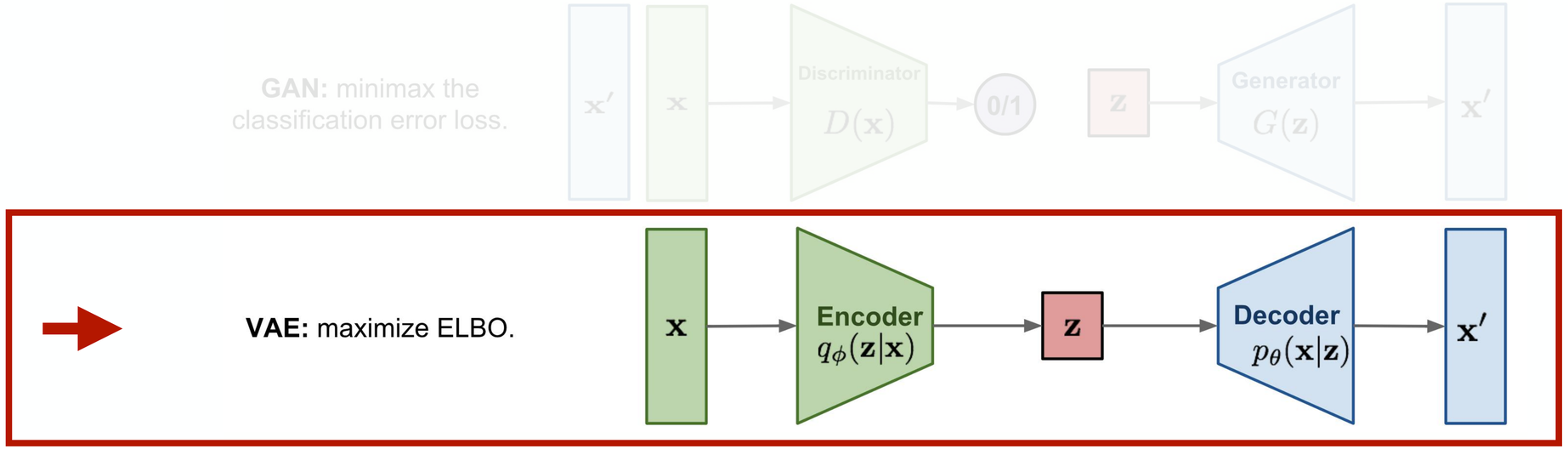


Training: Has two networks (**Discriminator** and **Generator**) trained iteratively one after another:

- Generate samples from random noise in z space, train Discriminator by feeding it fake and real samples (classification)
- Fix the parameters of the Discriminator and train the Generator to fool the Discriminator (classification)
- Repeat the cycle by feeding the discriminator the fake samples generated by the previously-updated Generator etc.

Application (data generation): the trained **Generator** produces generated data from random noise in z space

On Generative models: Variational Auto-Encoders (VAEs)



Training: Has two networks (**Decoder** and **Encoder**) trained simultaneously:

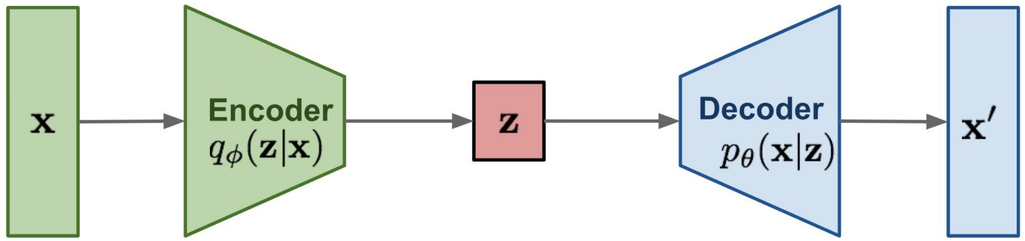
- The so-called ELBO loss (Estimated Lower Bound) is minimized
- The encoder produces a Gaussian guess on x representation in latent space z: μ and σ
- z is sampled based on μ and σ, and from this z the Decoder produced the output (x')

Application (data generation): the Decoder produces generated data from Gaussian noise in z space

On Generative models: Variational Auto-Encoders (VAEs)

GAN: minimax the classification error loss.

VAE: maximize ELBO.



$$\text{ELBO Loss}(x) = L_{\text{recon}}(x, \hat{x}) + L_{\text{KL}}(\mu, \log \sigma^2)$$

$$L_{\text{recon}}(x, \hat{x}) = \sum_{j=1}^{d} (x_j - \hat{x}_j)^2$$

Accuracy of decoder reconstruction

$$L_{\text{KL}}(\mu, \log \sigma^2) = \frac{1}{2} \sum_{j=1}^{D} (\exp(\log \sigma_j^2) + \mu_j^2 - 1 - \log \sigma_j^2)$$

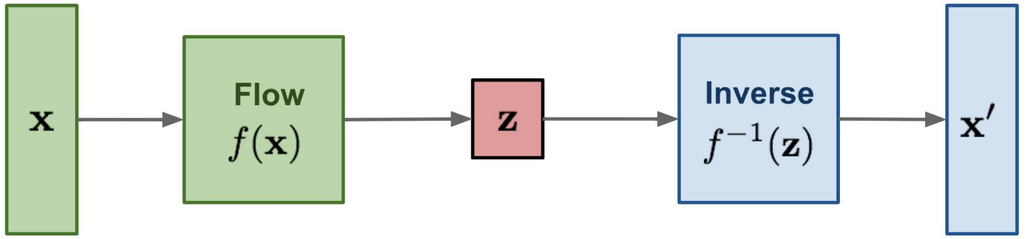
Kullback-Leibler divergence (measures how far is the probability distribution from encoder output is different from Gaussian)

On Generative models: Normalizing Flows

GAN: minimax the classification error loss.

VAE: maximize ELBO.

Flow-based generative models: minimize the negative log-likelihood



Training: one (invertable) flow NN

- Loss: negative log-likelihood which is directly analytically

Application (data generation): sample from latent space (z) using the inverse of flow NN

Summary [before we proceed to coding]

- **Boosted Decision Trees**
  - ➡ not "deep learning" but still powerful for certain applications
- **Neural Networks** - industry standard & economy driver
  - ➡ Excellent for high-dimensionality data, big data revolution
  - ➡ Fully-connected NNs - can be used as standalone architectures
  - ➡ ... or building blocks of more complex architectures
  - ➡ CNNs - very powerful but limited mostly to image processing
  - ➡ GNNs - a way to generalize CNN approach to any data
  - ➡ Transformers - hot topic (ChatGPT etc.) - **[see coding session ...]**
  - ➡ Applications: classification (discrete prediction), regression (continuous prediction), generation, anomaly detection, modeling of probability distributions, ...

Proven to solve real-life problems!

[from speaker's very biased point of view and experience in astro-particle physics]