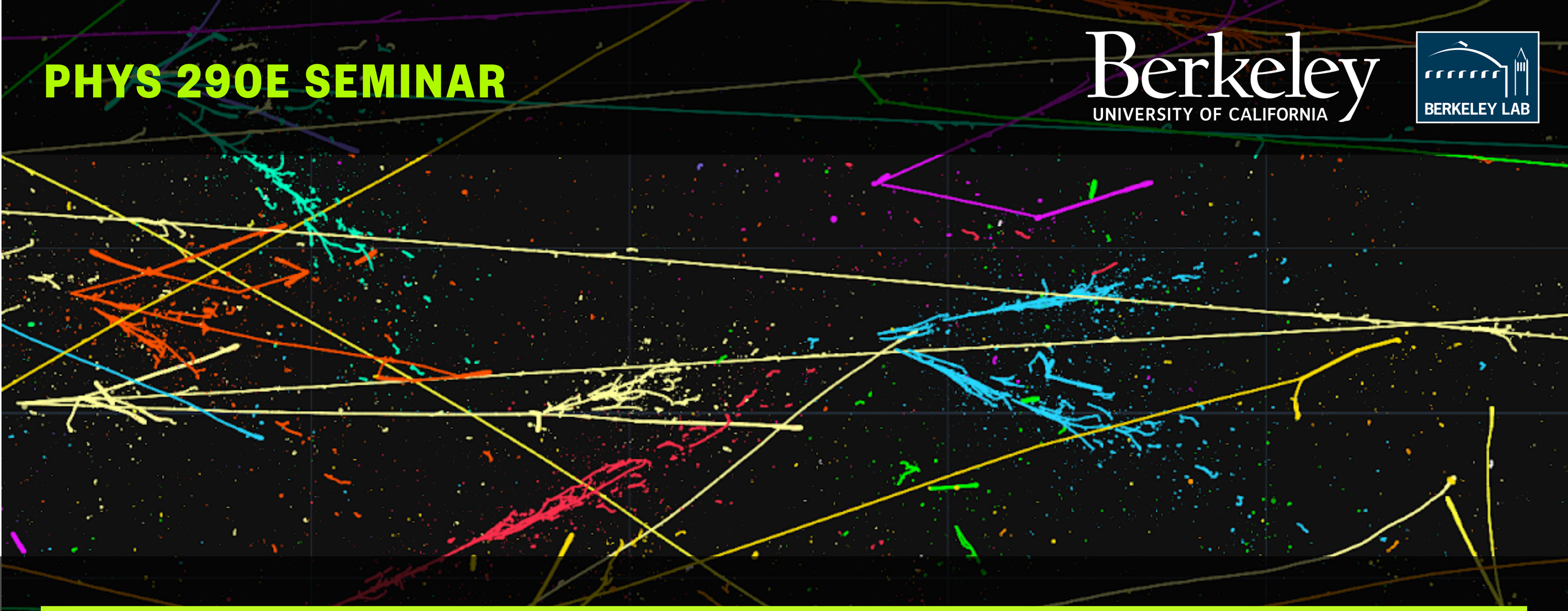
PHYS 290E SEMINAR

Berkeley
UNIVERSITY OF CALIFORNIA

BERKELEY LAB



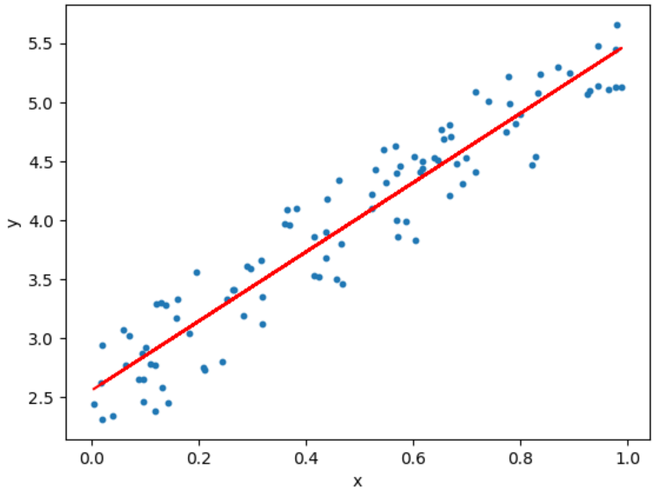
Machine learning and artificial intelligence at the intensity frontier

STEFANO ROBERTO SOLETI
LAWRENCE BERKELEY NATIONAL LABORATORY

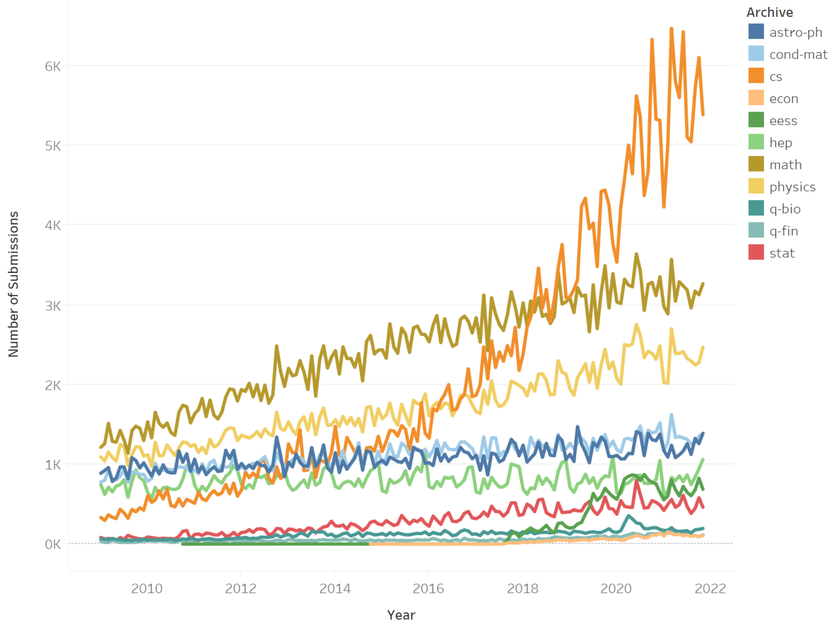
16 MARCH 2022

Why do we need machine learning?

- Explosion of machine learning and artificial intelligence in the last decade.
- Particle physics not immune: **machine learning algorithms used in every analysis step** (trigger, particle identification, calibration, analysis, etc.).
- But what exactly is machine learning? *Any algorithm that can improve automatically through experience and by the use of data*.

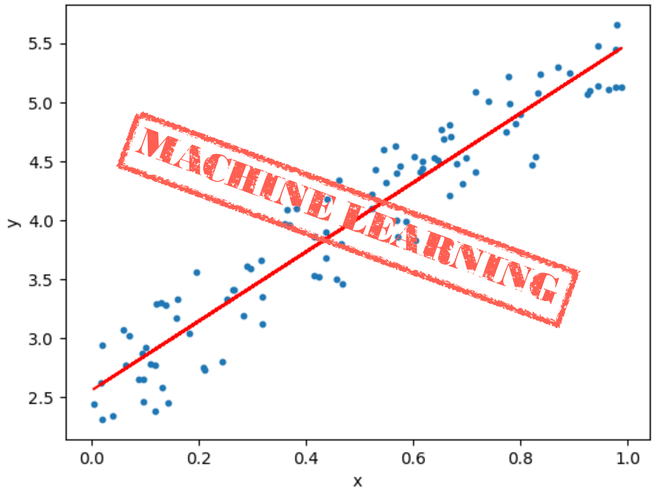


arXiv submissions by category

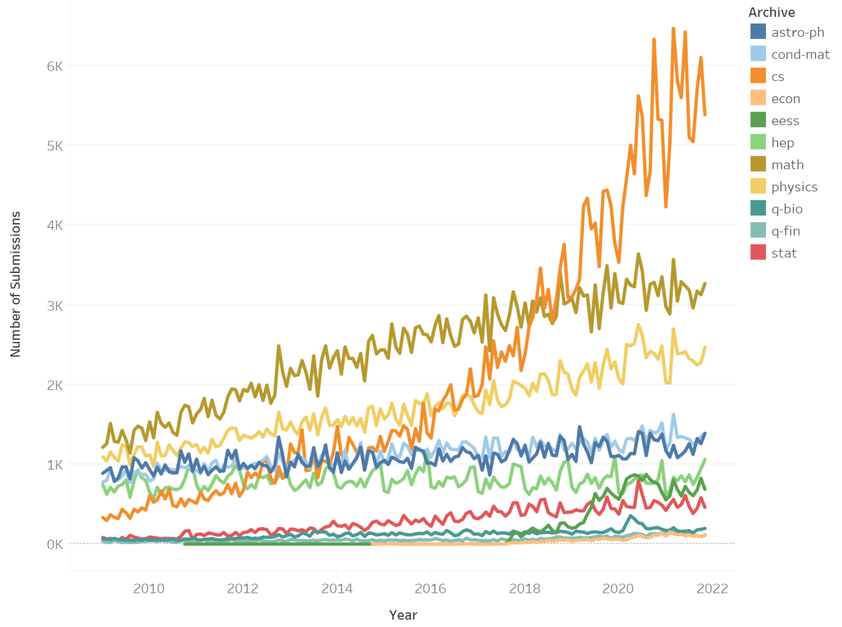


Why do we need machine learning?

- Explosion of machine learning and artificial intelligence in the last decade.
- Particle physics not immune: **machine learning algorithms used in every analysis step** (trigger, particle identification, calibration, analysis, etc.).
- But what exactly is machine learning? *Any algorithm that can improve automatically through experience and by the use of data*.



arXiv submissions by category

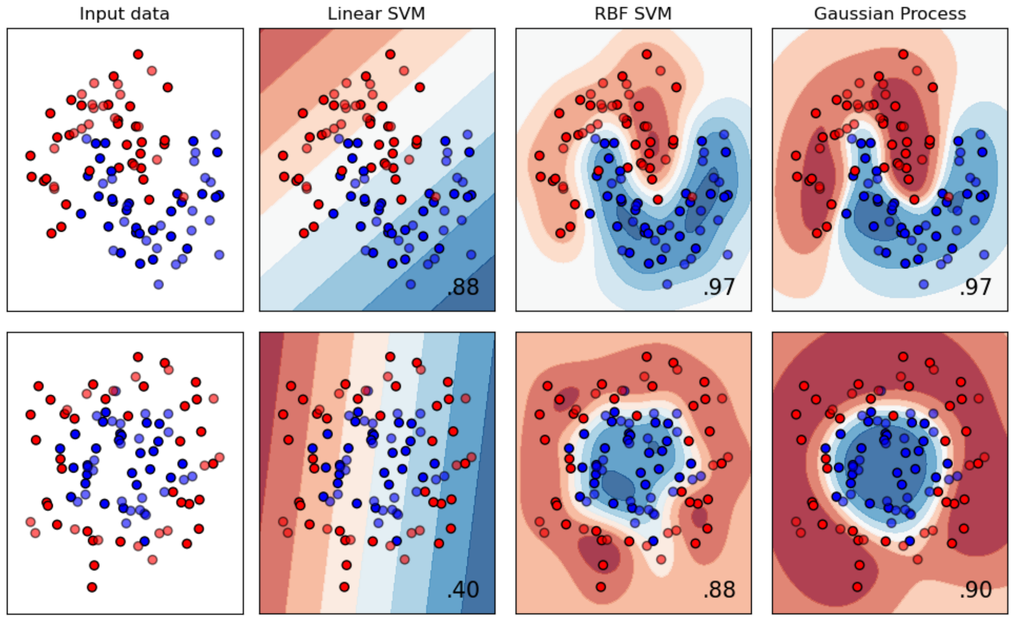


Supervised learning

Classification

predicting a discrete quantity

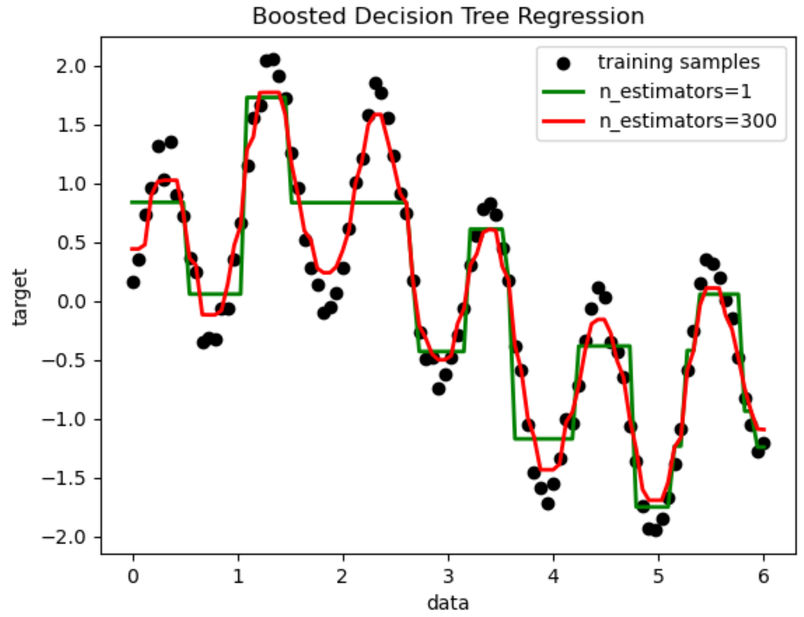
1. Train on signal and background simulated samples
2. Learn the separation between S and B distributions
3. Apply on test sample
4. Apply on data



Regression

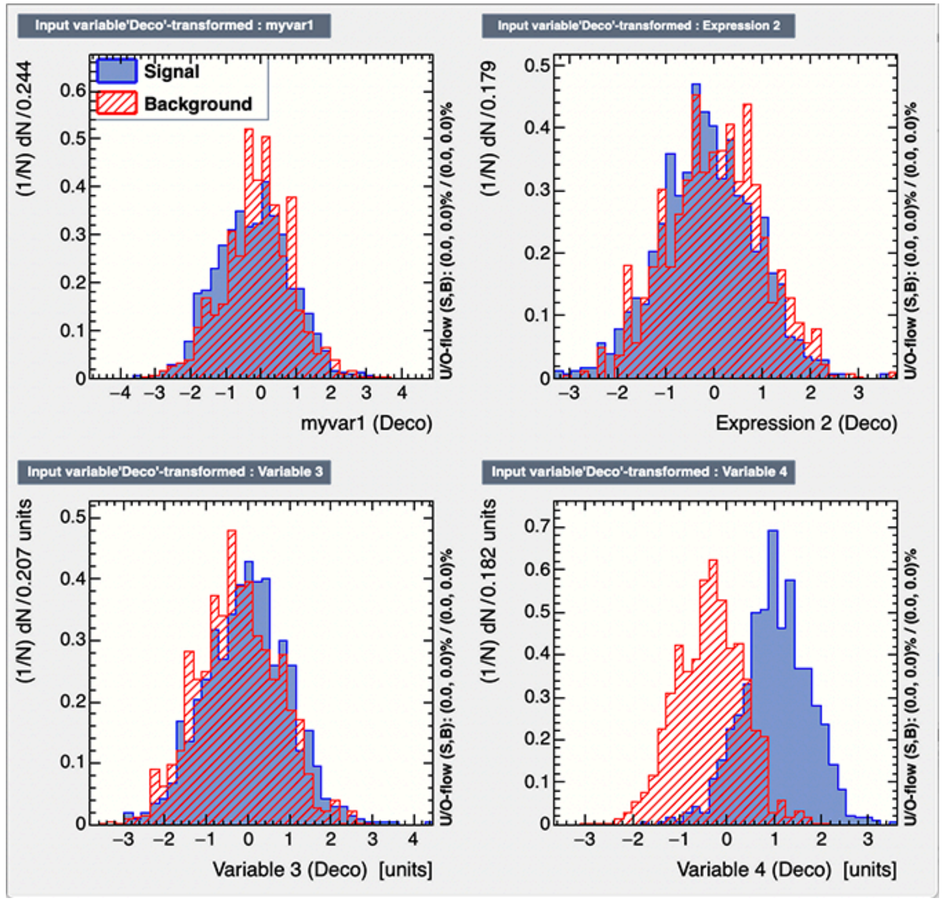
predicting a continuous quantity

1. Train on simulated samples
2. Learn relationship between a dependent variable (outcome) and one or more independent variables (features)
3. Apply on test sample
4. Apply on data



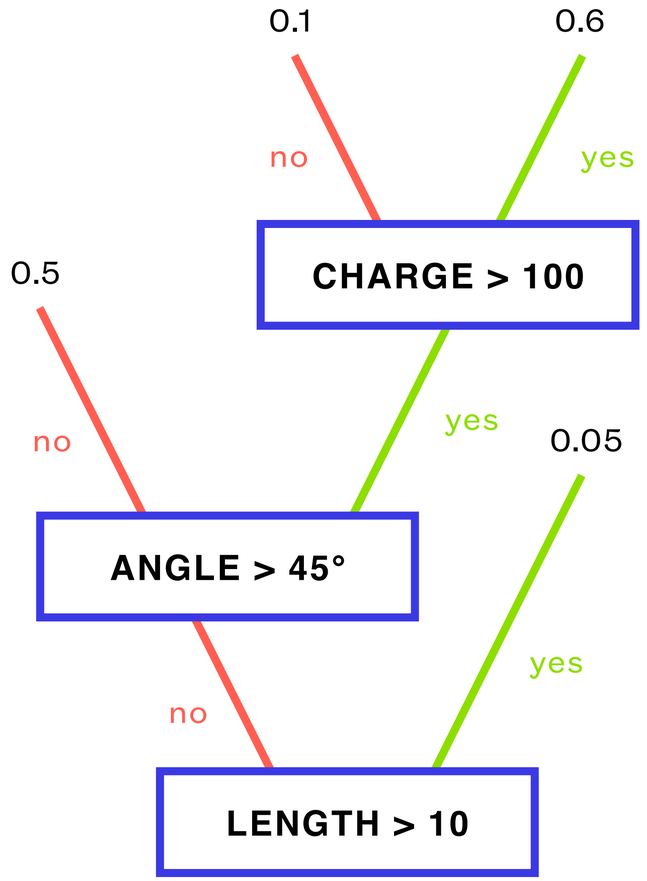
Multivariate analysis

- First systematic use of machine learning methods in particle physics: *multivariate analysis* using TMVA
  - Combine **several, high-level, carefully crafted physics variables** (e.g. number of hits in a detector, reconstructed energy, reconstructed track length) for classification (distinguish between signal and background) or regression (predict the value of an output variable, given one or more input variables)
- The classification or regression is performed through Boosted Decision Tree, Support Vector Machines, ***shallow* neural networks** (few layers), etc.



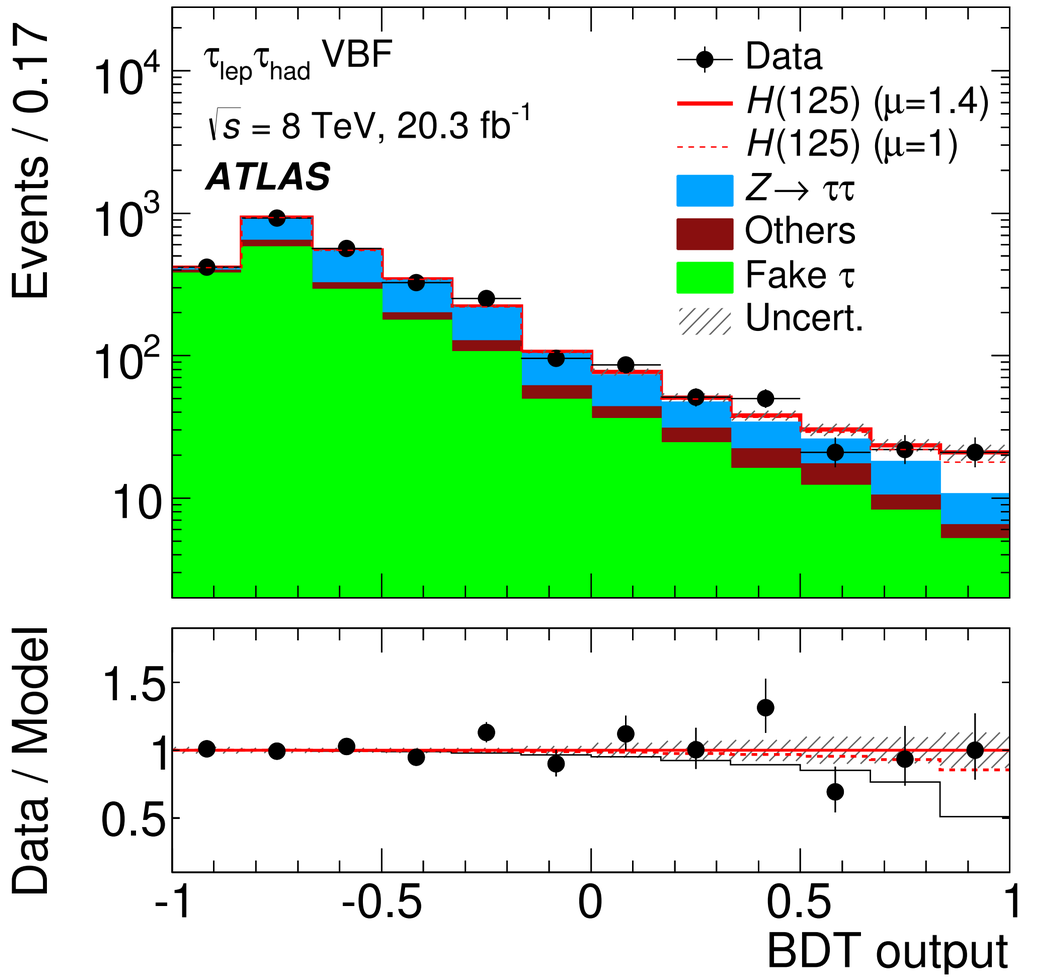
Boosted decision trees

- **Boosted Decision Trees** (BDTs) are one of most common used machine learning algorithm in particle physics, not easy to outperform with neural networks.
  - Each split at a node is chosen to maximize information gain or minimize entropy
  - The splits are created recursively until a stop condition is met (e.g. depth of the tree)
- **Boosting**: several trees are created, the tree output is assigned a weight relative to its accuracy. Misclassified events are assigned a larger weight: next iteration will pay more attention



Real-world example: $H \rightarrow \tau^+\tau^-$

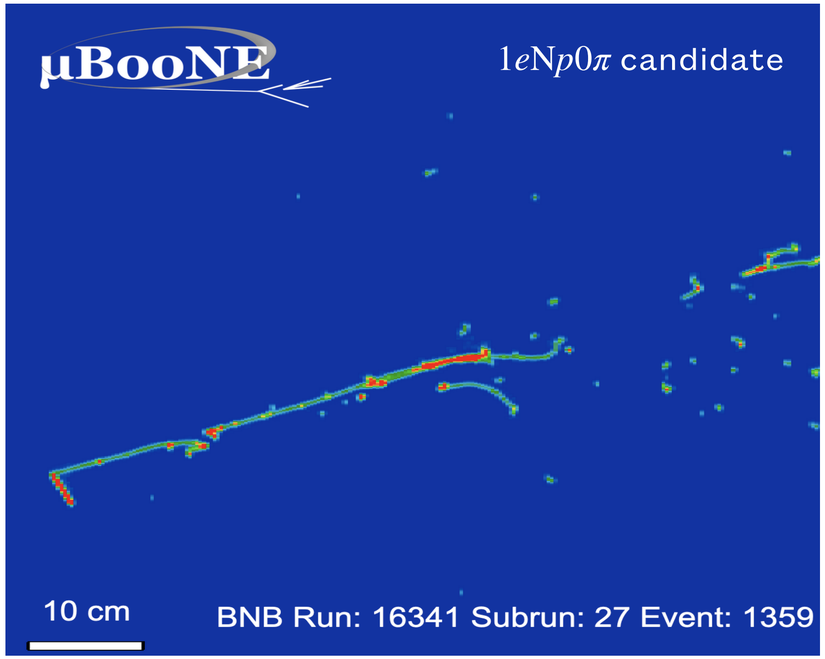
- ATLAS search for the decay of the Higgs boson into a $\tau\tau$ pair
  - If you have well-understood background and signal models you can train a machine learning algorithm to distinguish between the two
  - This ATLAS analysis **combined 12 weakly discriminating input variables** to increase the discriminating power between signal and background.
  - Evidence of Higgs coupling to tau leptons with 40% improved efficiency



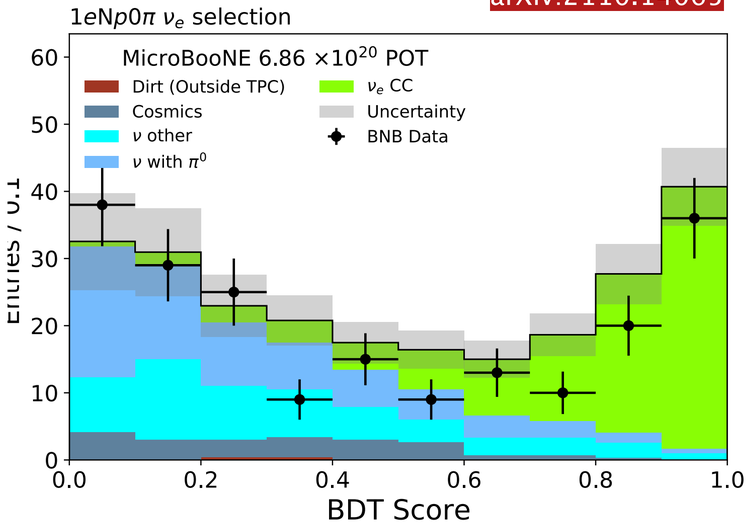
arXiv:1501.04943

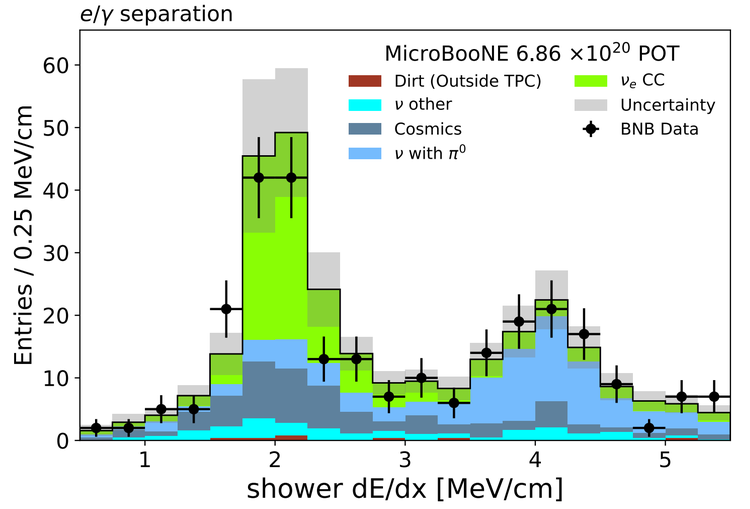
$1eNp0\pi$ channel at MicroBooNE

- The goal of the **MicroBooNE** experiment is to clarify the nature of the low-energy excess of electron events observed by MiniBooNE, which could hint to the presence of a fourth, sterile, neutrino.
- The $1eNp0\pi$ is one of the most promising channel to look for this excess: it consist of **one electromagnetic shower** (the electron), plus **one or more short ionization tracks** (the protons).
- It is **necessary to reject events with photons** (which could look like electrons) **and muons** (which could look like protons). This is **achieved with two BDTs trained with XGBoost** using carefully constructed, high-level variables, (dE/dx, track length, etc.)



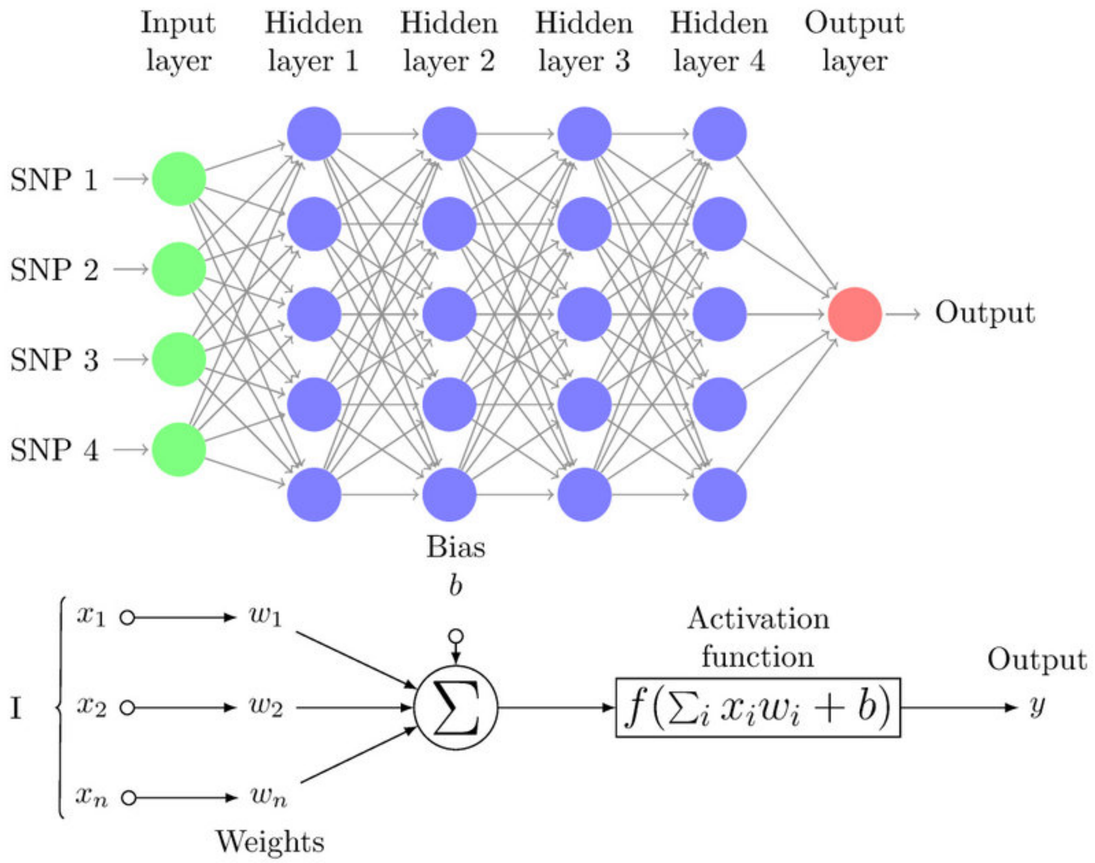
arXiv:2110.14065





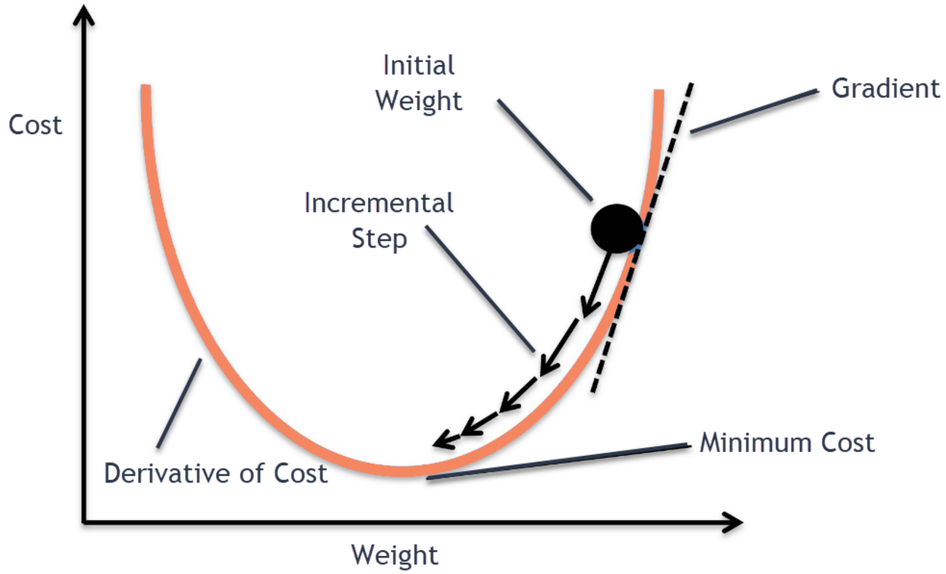
Neural networks

- Loosely inspired by the layout of a human neuron: at the input of each node a weighted sum of inputs is given. It is transformed by the activation function and later send to the output.
- Simplest implementation of a neural network consist in **layers of neurons, fully connected in sequence: multi-layers perceptron**.



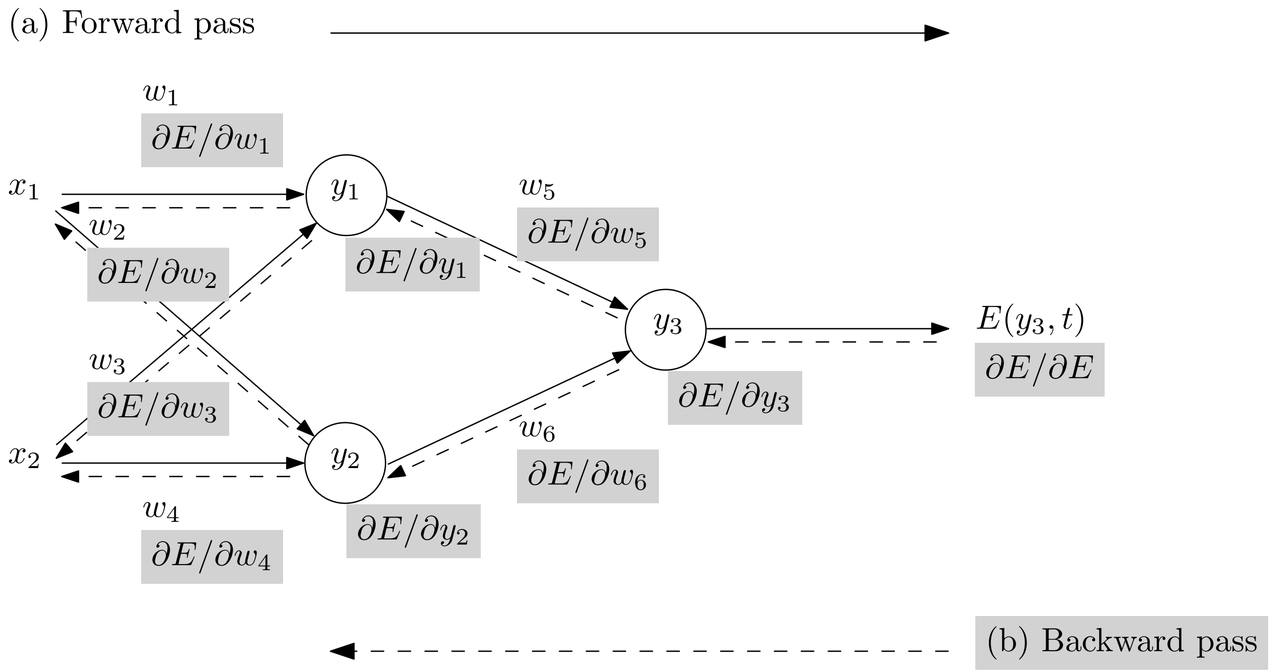
How to train a neural network?

- Once you have several layers made of tens of neurons, **how do you train the network**? Namely, how do you adapt the weights in the hidden layers in order to minimize your loss function? ***Stochastic Gradient Descent with backpropagation***.
- Training steps:
  1. Calculating the network output Y for some input X
  2. Calculating the corresponding value of the loss function
  3. Calculating the loss gradient wrt to the weights
  4. Finding the value of the weights which minimizes the loss gradient.



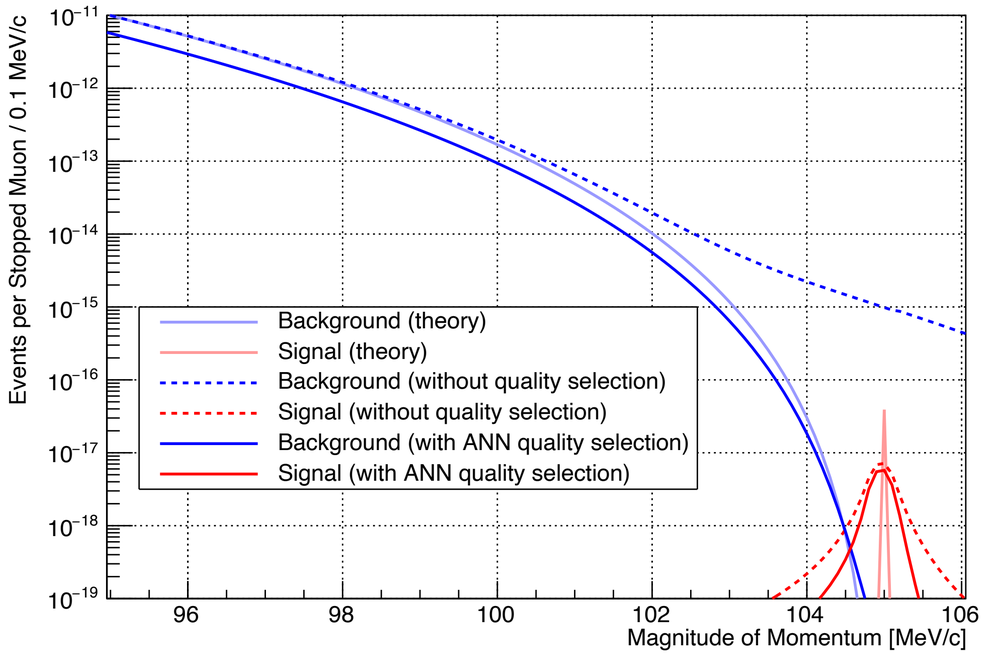
Backpropagation

- **Backpropagation** is the key algorithm that **makes training deep models computationally tractable.**
- It essentially consists in applying the **derivative chain rule backwards** to find the dependence of the output of the network on the weights.
- **Highly efficient**: at each node the derivatives depend only on the derivatives already calculated for the parent nodes and on the node values calculated during the forward pass.



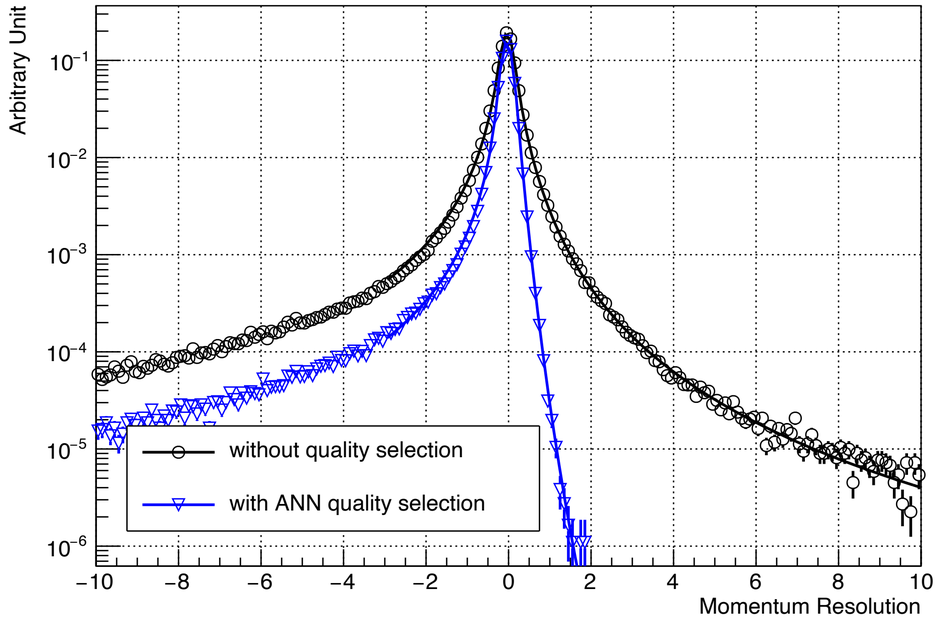
Shallow NN applications: track selection

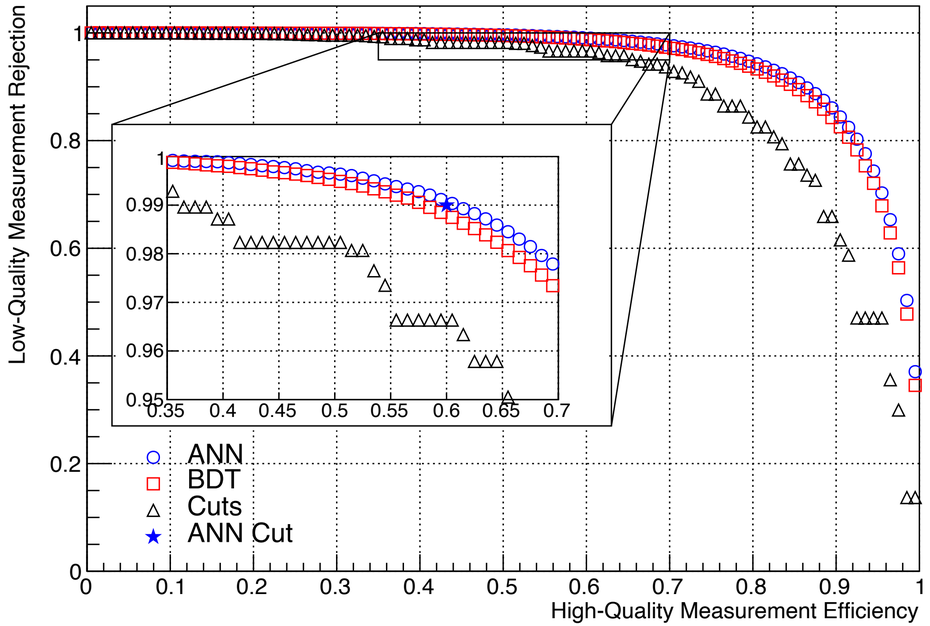
- The goal of the Mu2e experiment is to look for the conversion of a muon into an electron in the field of a nucleus.
- The **signature is a mono-energetic electron** around the muon mass: however badly reconstructed tracks and limited experimental resolution can make the background distribution wider and suppress the experimental sensitivity.



Shallow NN applications: track selection

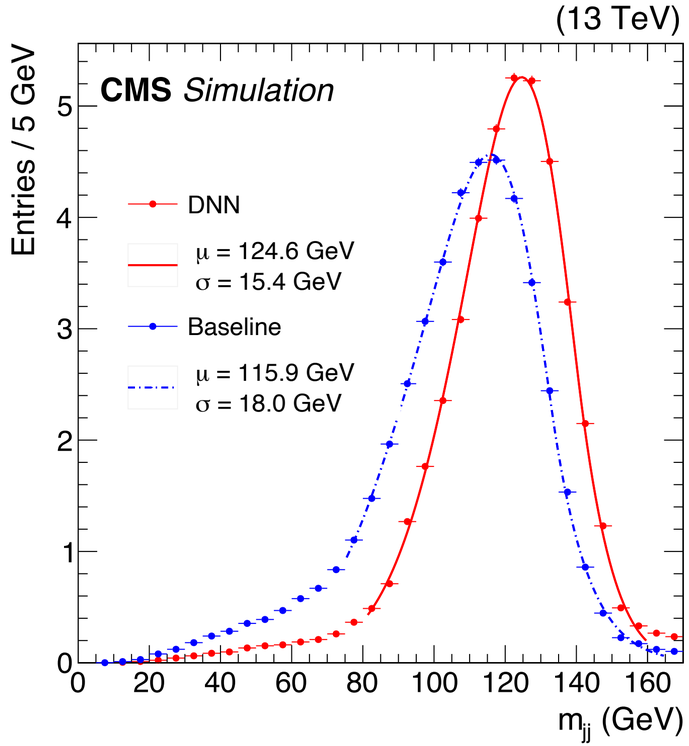
- First **extract meaningful, high-level variables** from the reconstructed tracks: e.g. fit momentum error, number of hits, etc.
- Train a **multi-layer perceptron network** with two layers and O(10) neurons each to distinguish between “good” tracks ($|p_{reco} - p_{true}| < 250\ \mathrm{keV}/c$) and “bad” tracks ($p_{reco} - p_{true} > 700\ \mathrm{keV}/c$).
- The ROC curve shows a significant better performance than just applying boxed cuts on the meaningful variables and it’s also slightly better than a BDT.





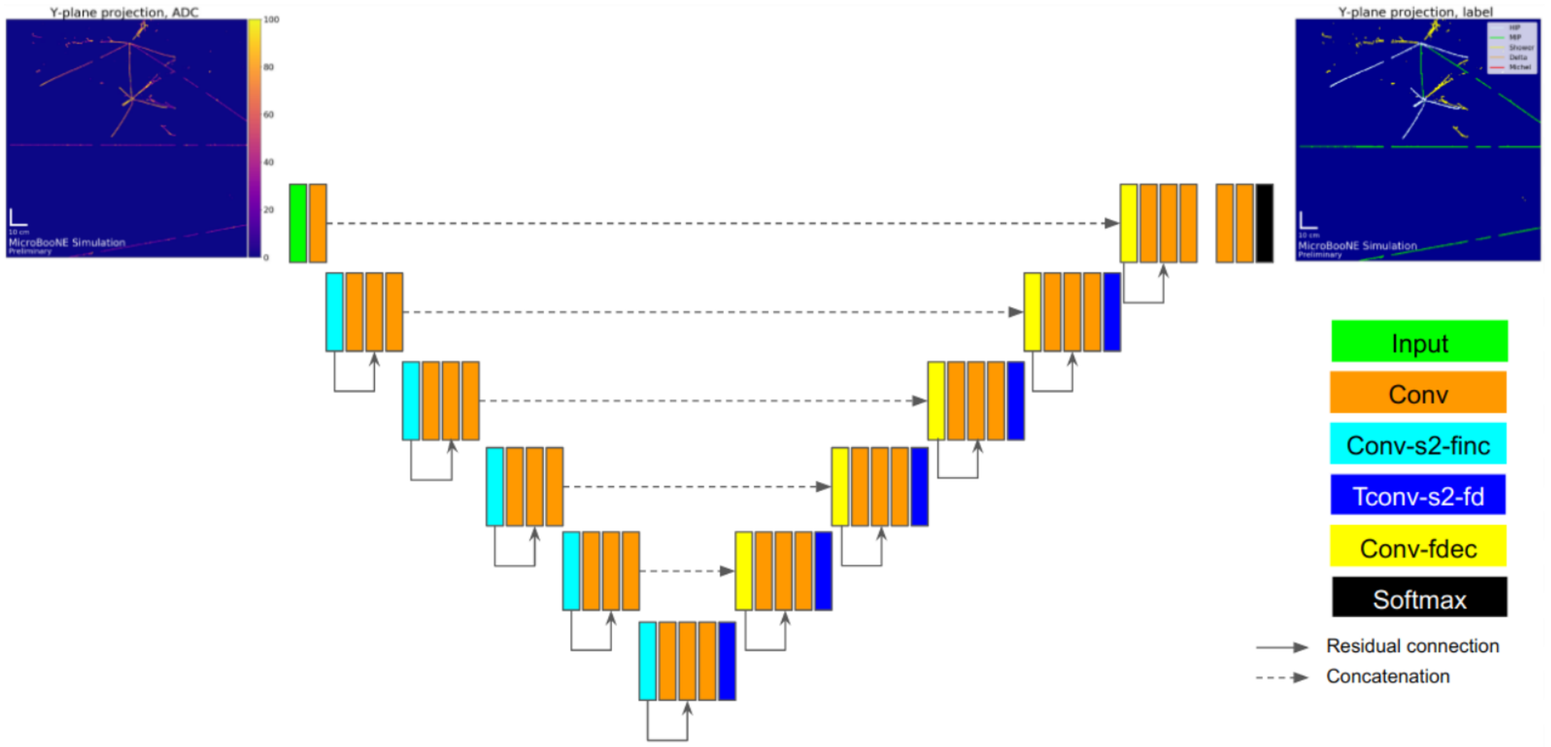
From classification to regression

- Instead of classifying tracks as “well reconstructed” or “bad reconstructed”, one could train a NN that tries to predict the value of p_{true}, given p_{reco} and the other auxiliary variables (fit momentum error, number of hits, etc.): *regression*.
- This has the advantage of **avoiding arbitrary thresholds** between “good and “bad” tracks and provides a more natural way to cut tracks in the tail of the distribution.
- Most importantly, **it can improve the experimental resolution** by accounting for the energy lost, thus impacting the sensitivity of the experiment (especially important for low-counting experiments such as Mu2e).
- Technique already adopted by CMS and ATLAS.



From machine learning to *deep* learning

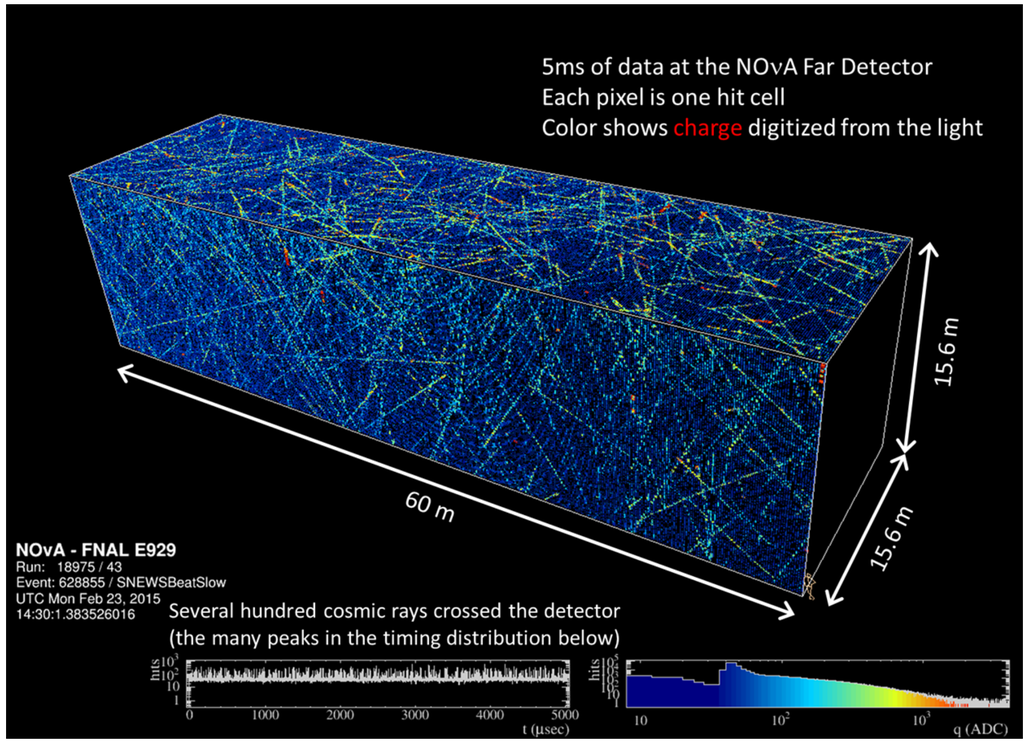
- What if, instead of using few layers with few neurons, we start building network with tens of layers and hundreds of neurons?
- This is usually what we call **deep learning**: there is no hard threshold, but when networks start to have more than a couple of layers then they are called *deep neural networks*.

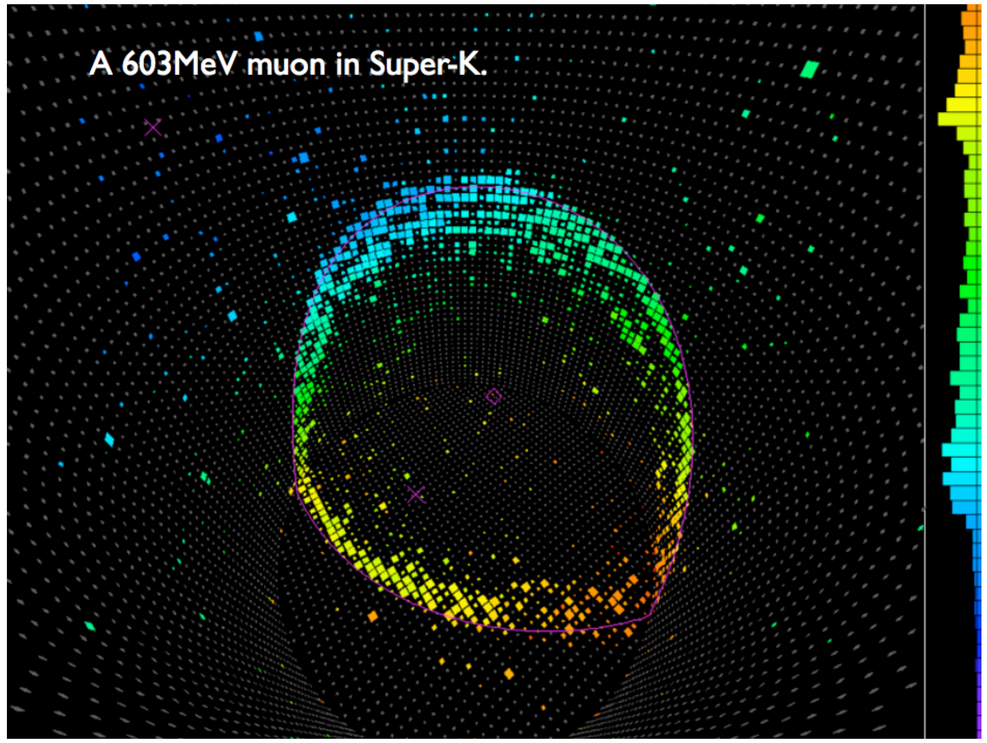


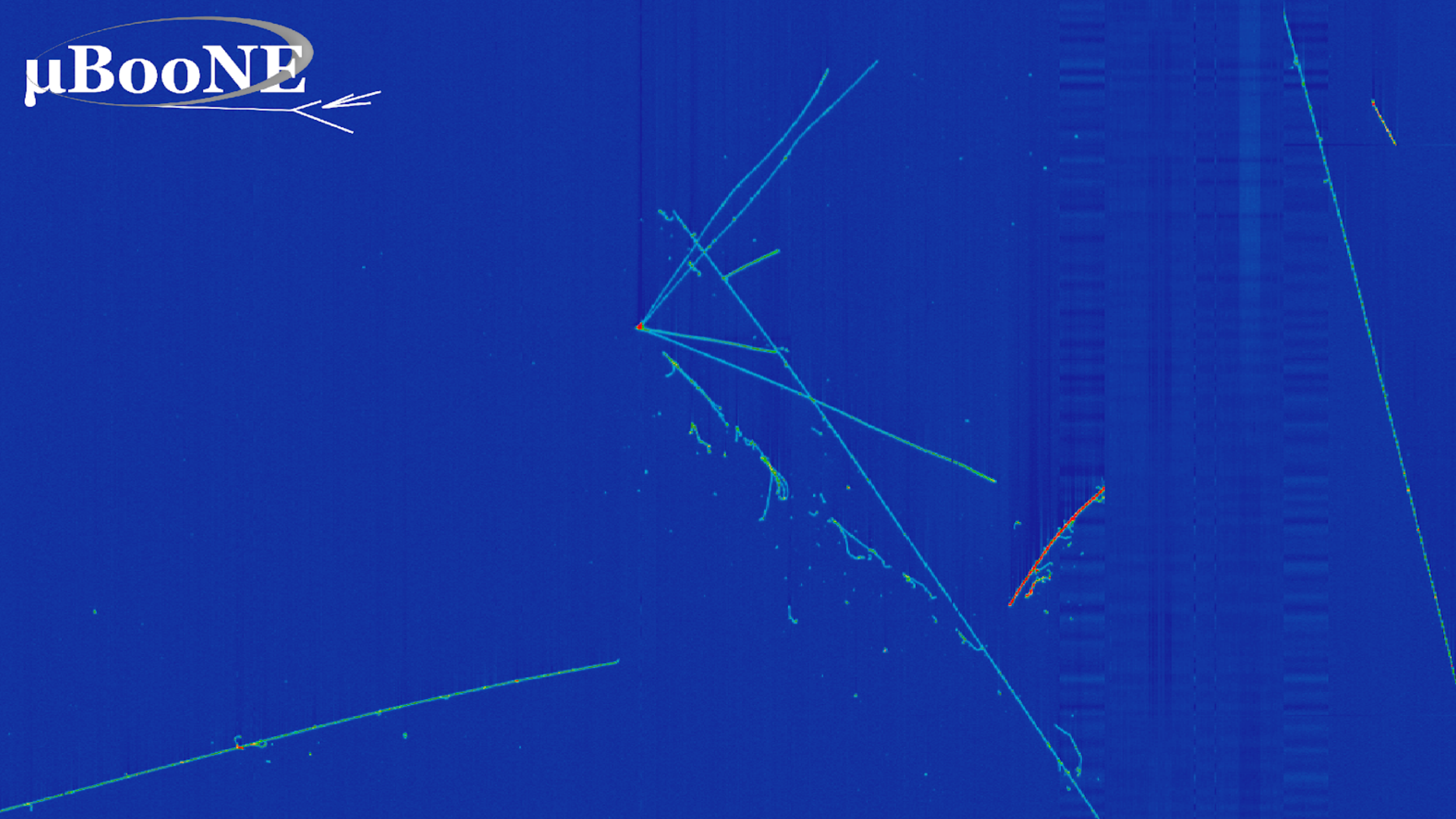
arXiv:2012.08513

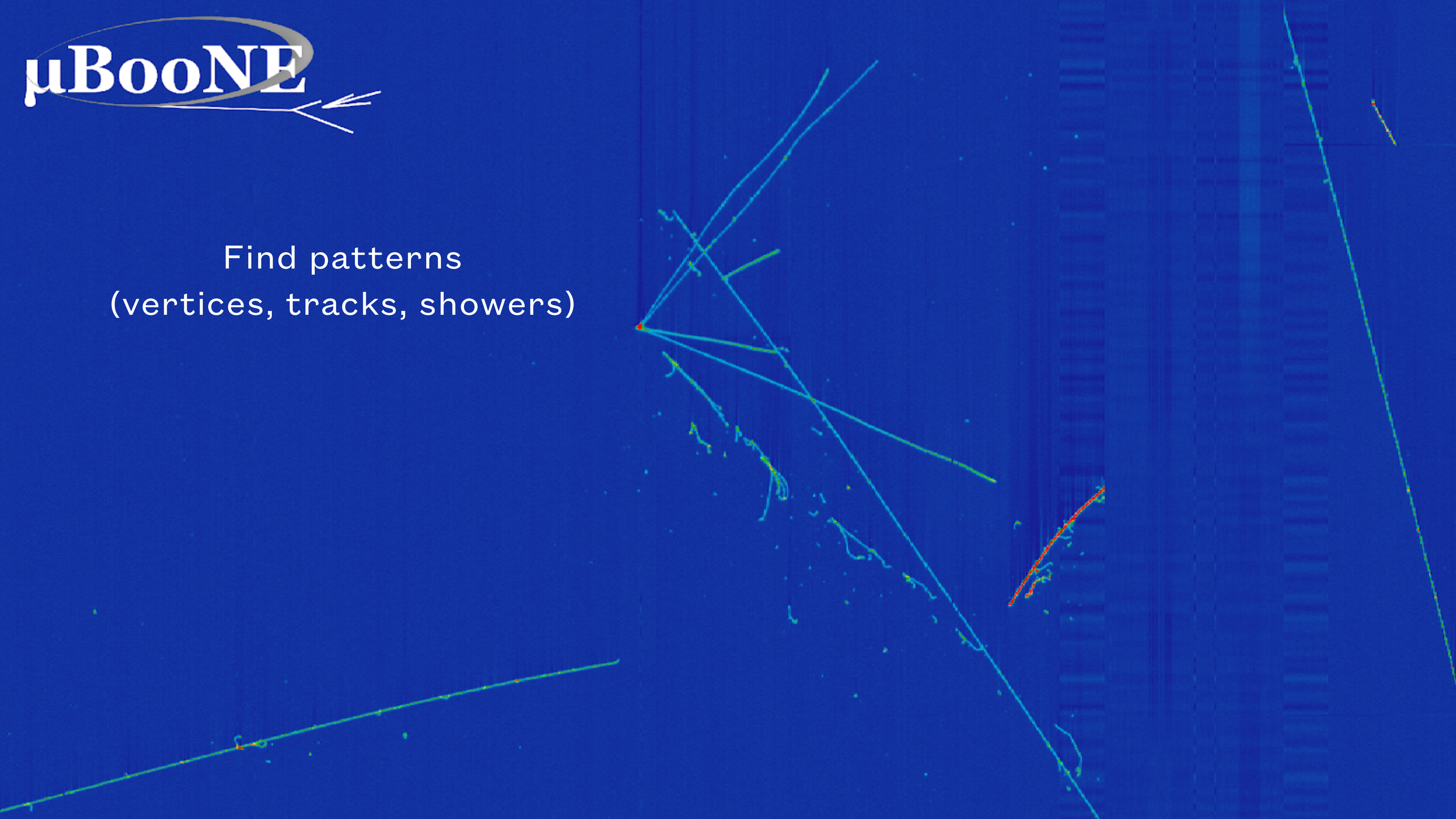
Deep learning for neutrino experiments

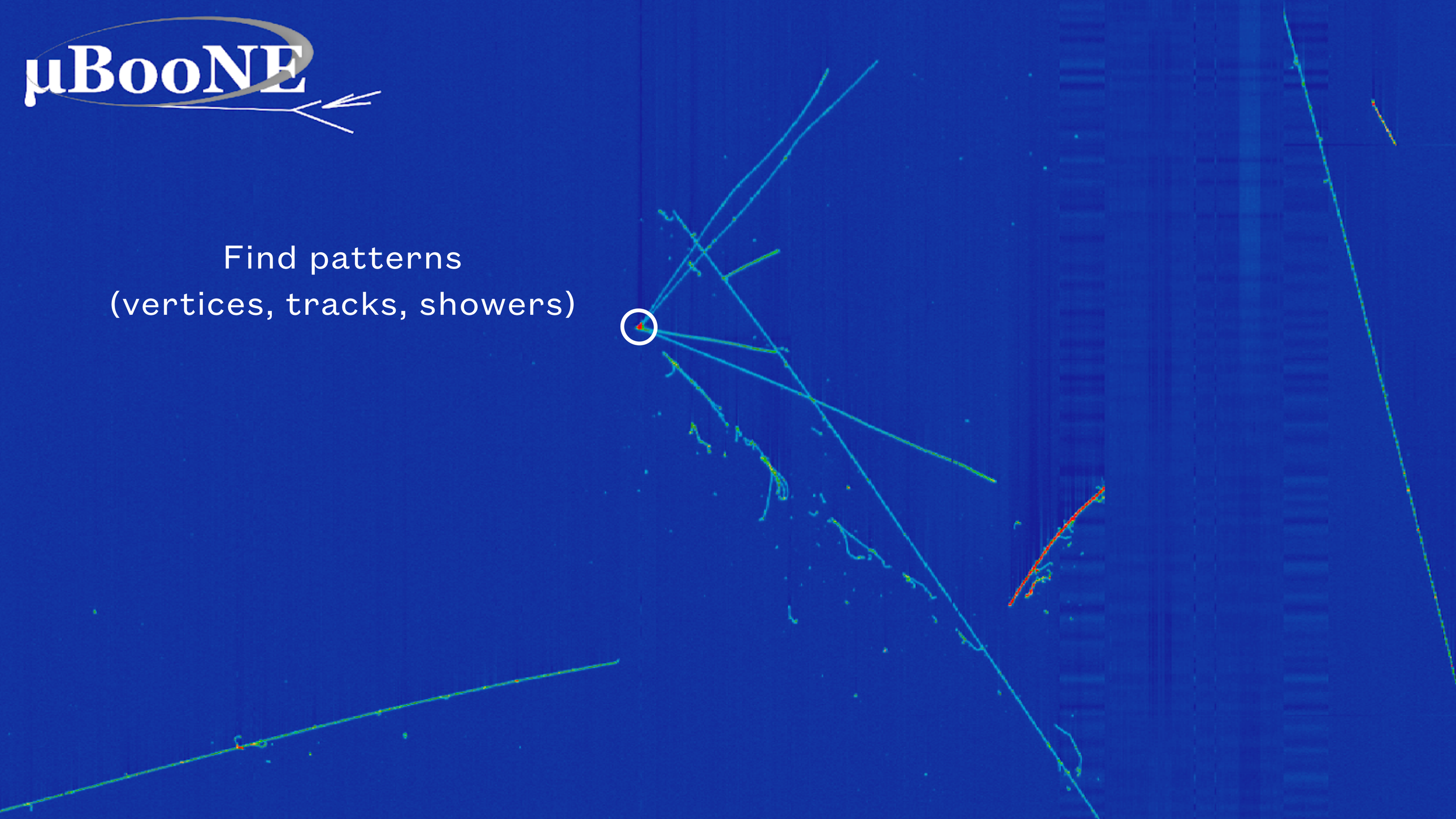
- **Neutrino experiments**, in theory, represent the **ideal candidate to port image-recognition concepts and techniques to particle physics**.
- They produce event display where a trained human is usually able to distinguish the features: electron neutrino interaction, muon neutrino interaction, cosmic ray, etc.

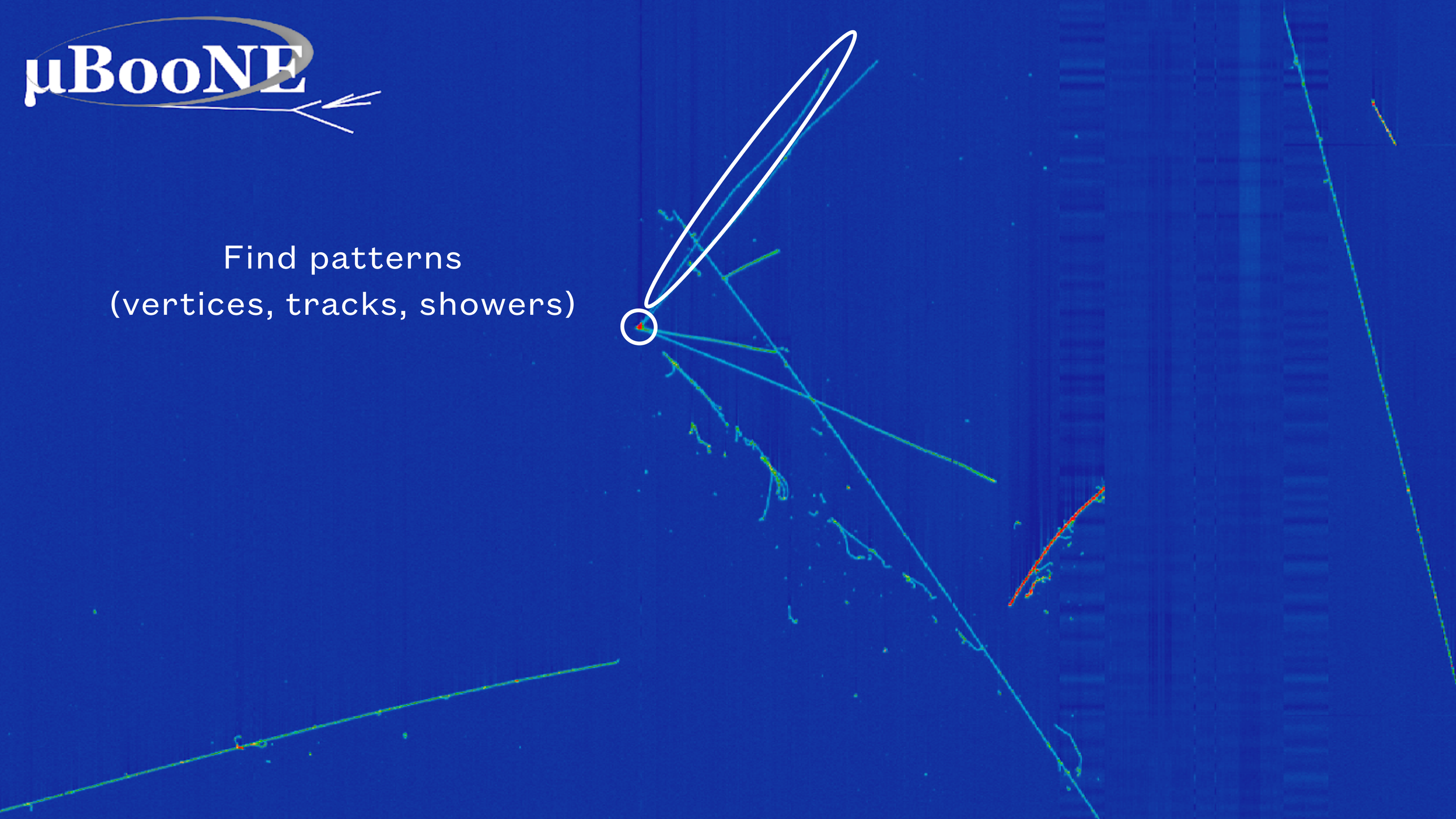


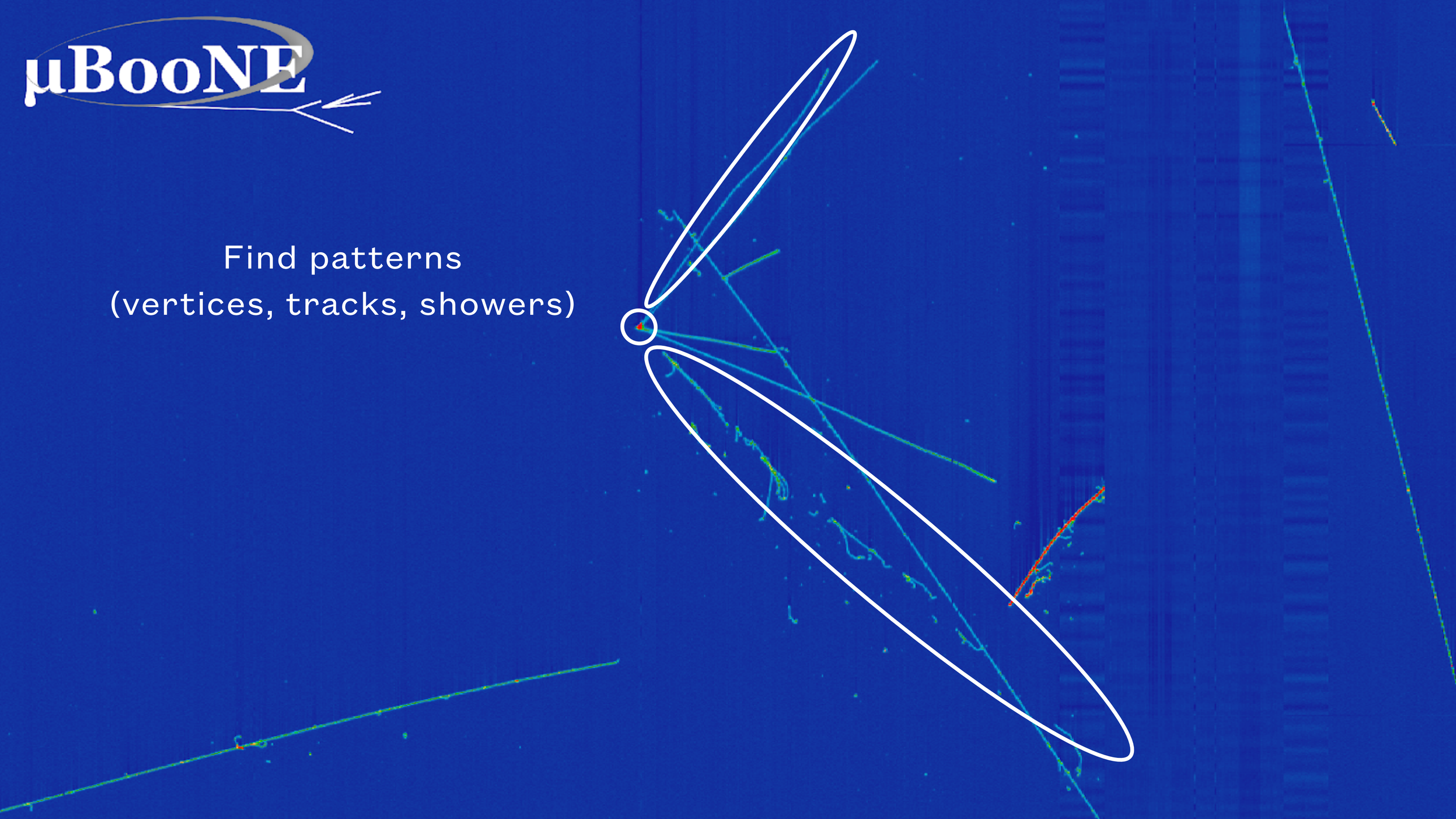


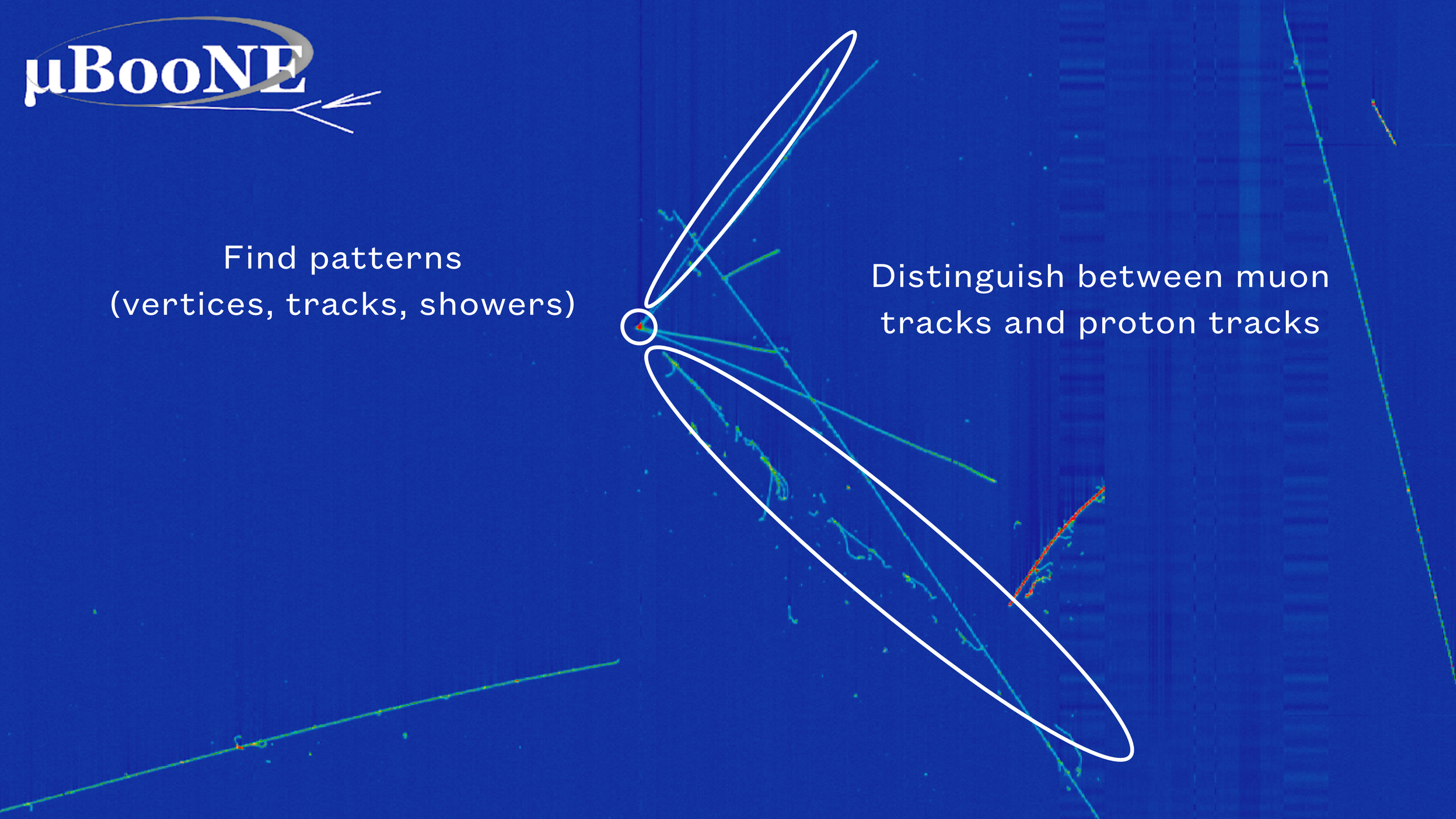

μBooNE

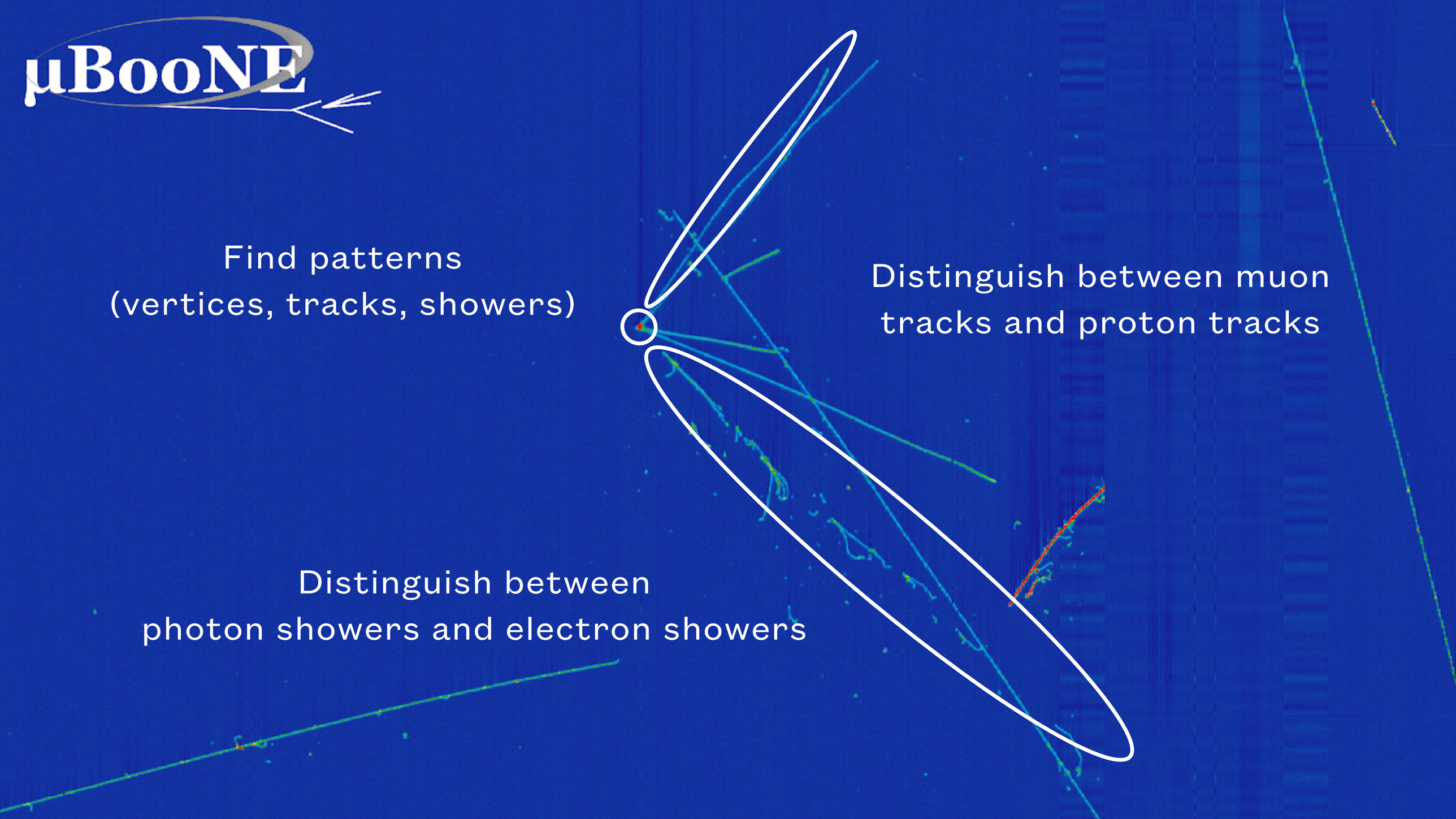

μBooNE
Find patterns
(vertices, tracks, showers)


μBooNE
Find patterns
(vertices, tracks, showers)


μBooNE
Find patterns
(vertices, tracks, showers)

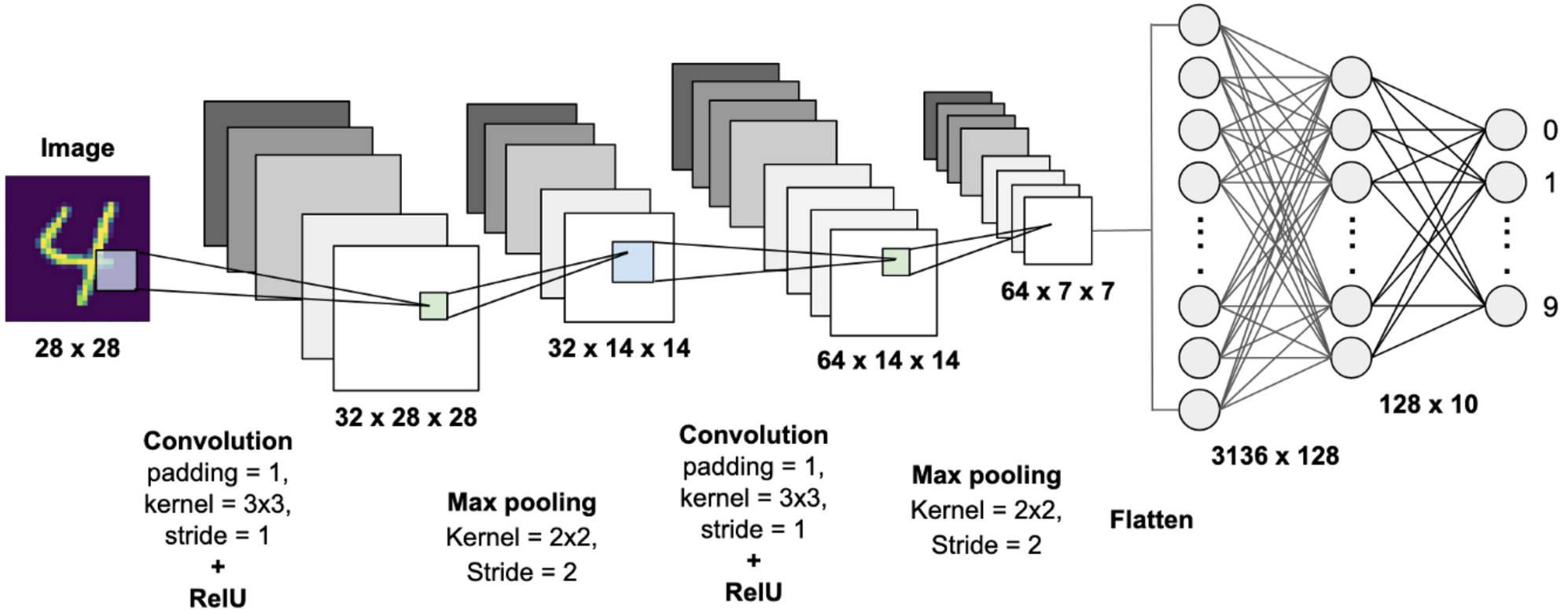

μBooNE
Find patterns
(vertices, tracks, showers)

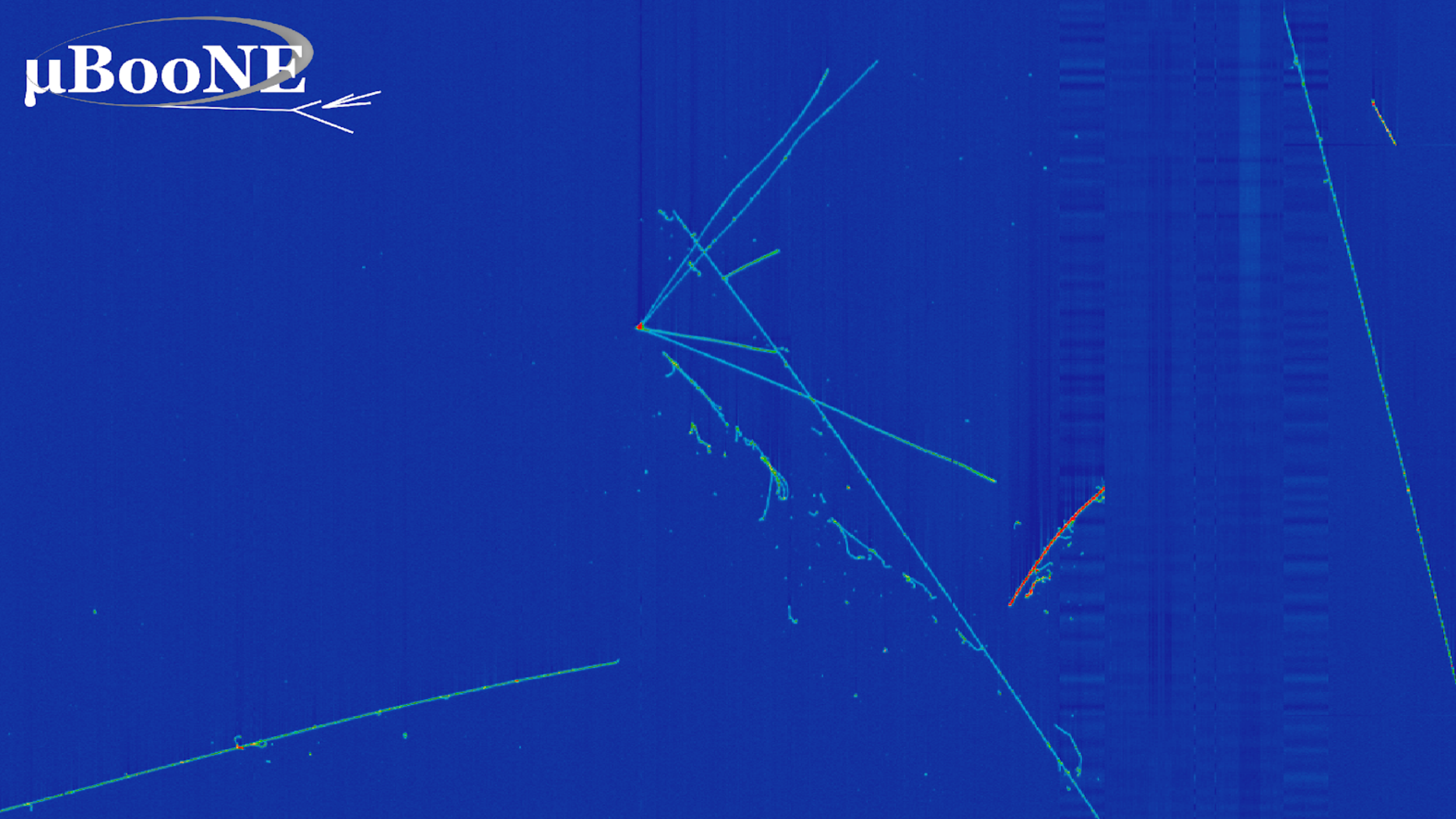

μBooNE
Find patterns
(vertices, tracks, showers)
Distinguish between muon
tracks and proton tracks

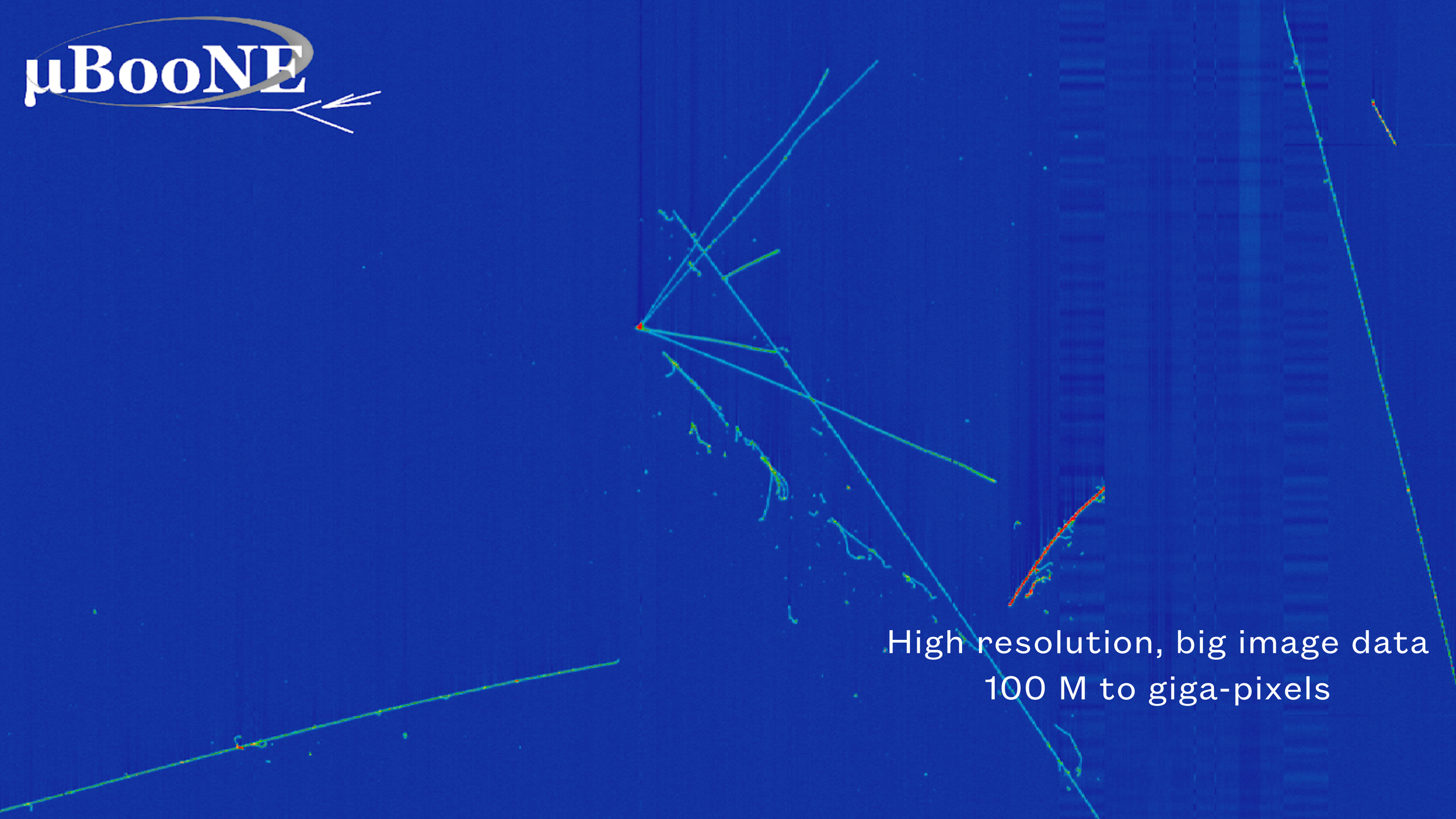

μBooNE
Find patterns
(vertices, tracks, showers)
Distinguish between muon
tracks and proton tracks
Distinguish between
photon showers and electron showers

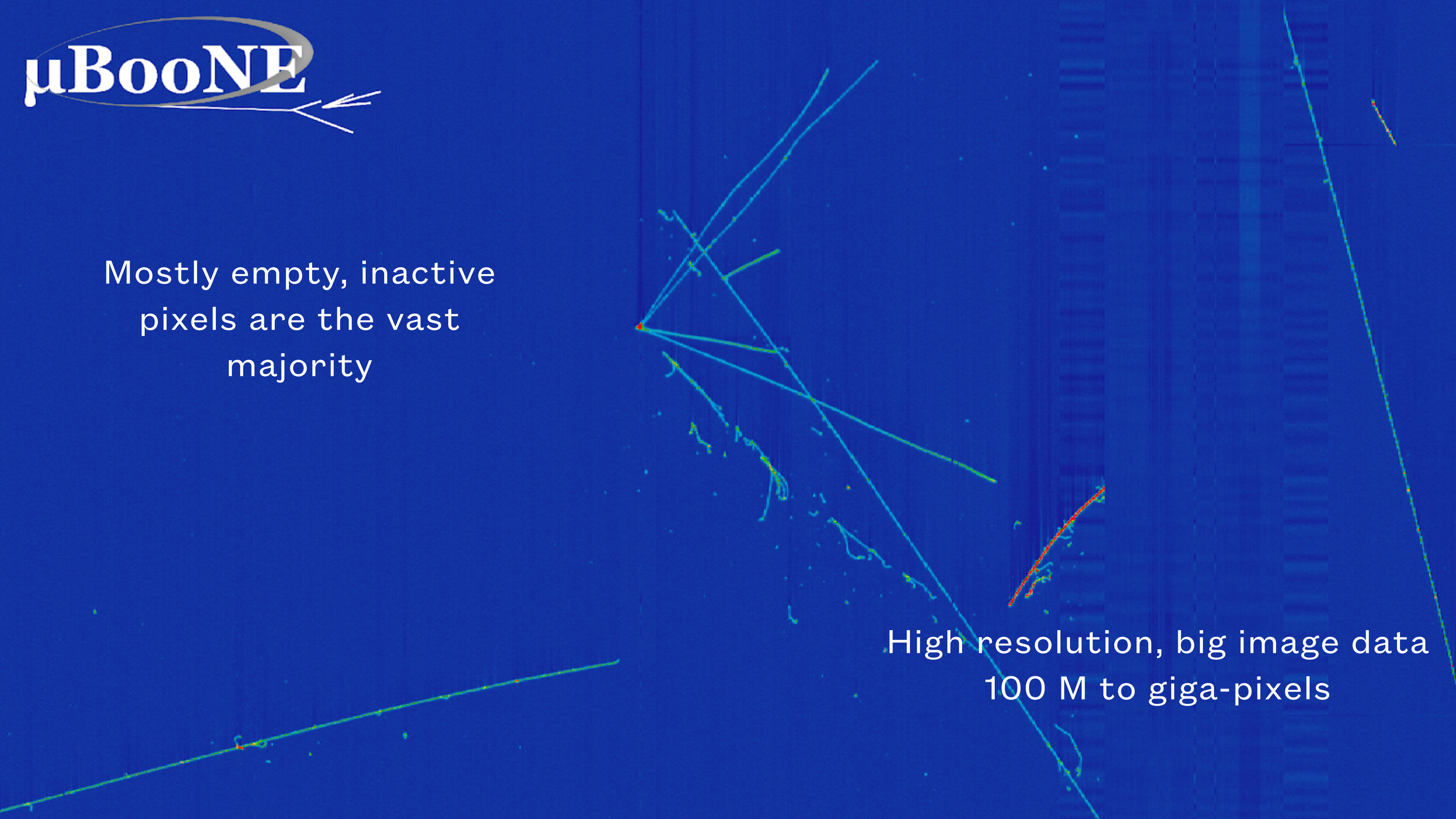
Deep learning for neutrino experiments

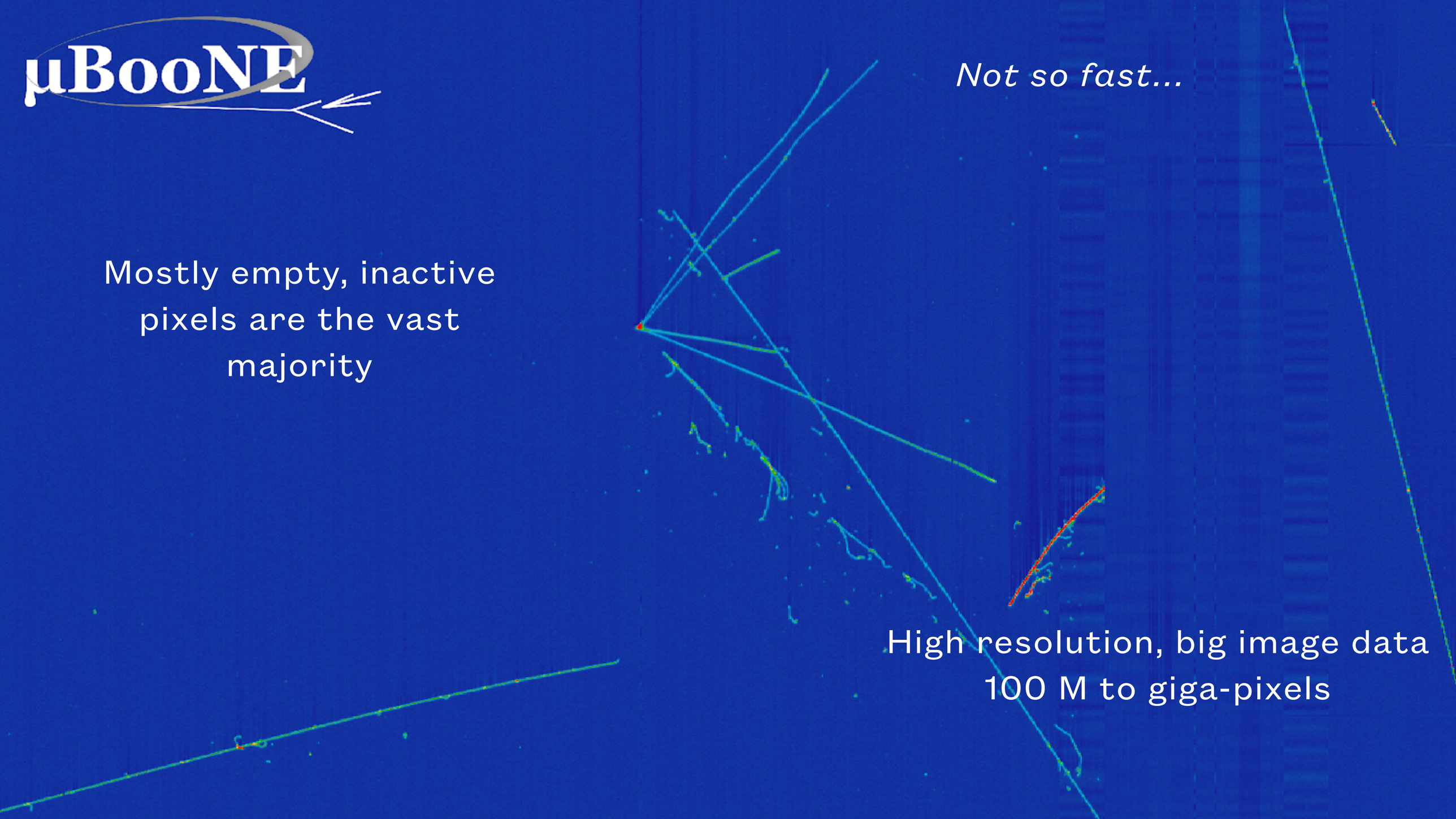
- **Convolutional Neural Networks** are usually used for image recognition tasks
- Let's just the same CNN we used to do our first deep learning exercise: recognize MNIST handwritten digits!

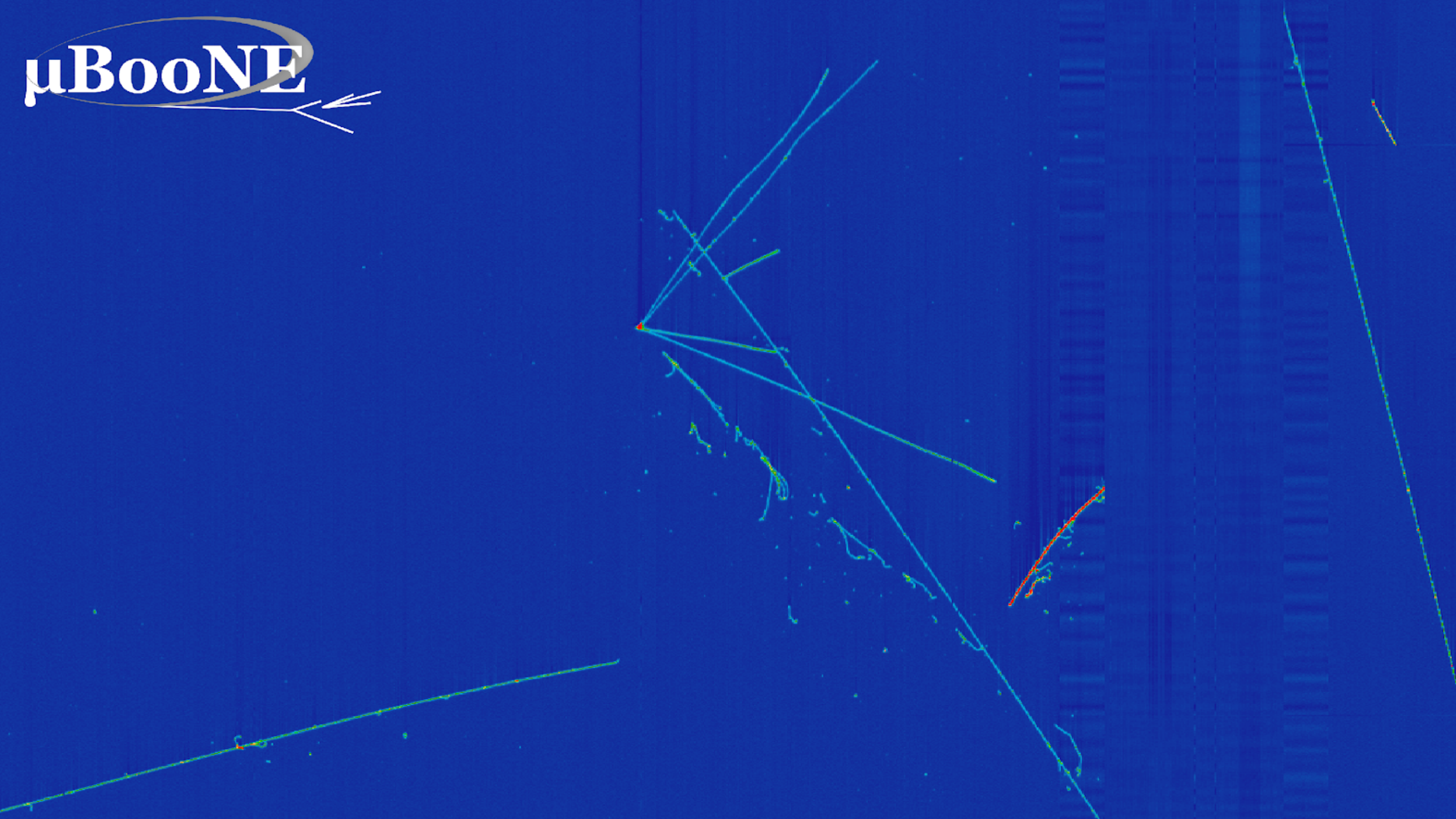



μBooNE


μBooNE
High resolution, big image data
100 M to giga-pixels

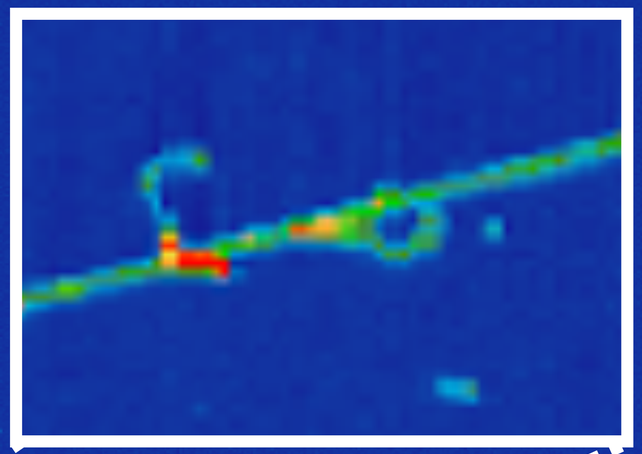

μBooNE
Mostly empty, inactive pixels are the vast majority
High resolution, big image data
100 M to giga-pixels


μBooNE
Not so fast...
Mostly empty, inactive pixels are the vast majority
High resolution, big image data
100 M to giga-pixels


μBooNE

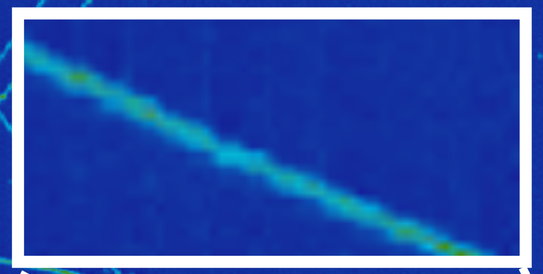
μBooNE

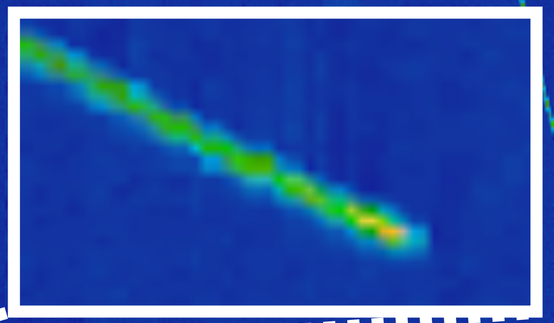
Small things matter
(inform direction and topology)



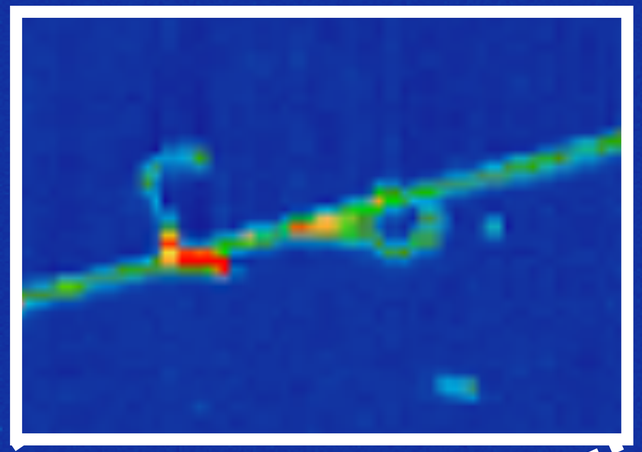
μBooNE

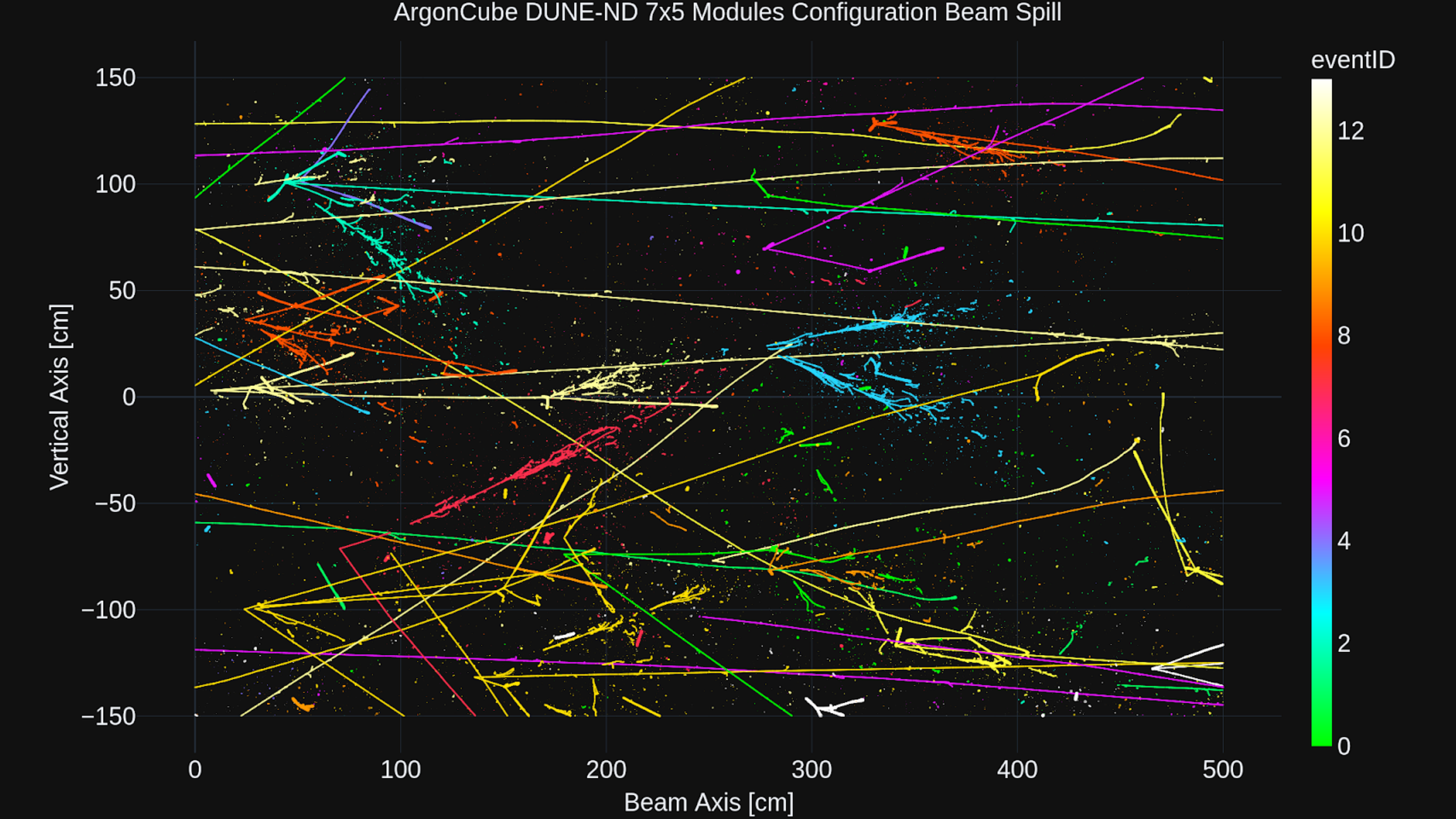
Color is important
(both absolute value and gradient)

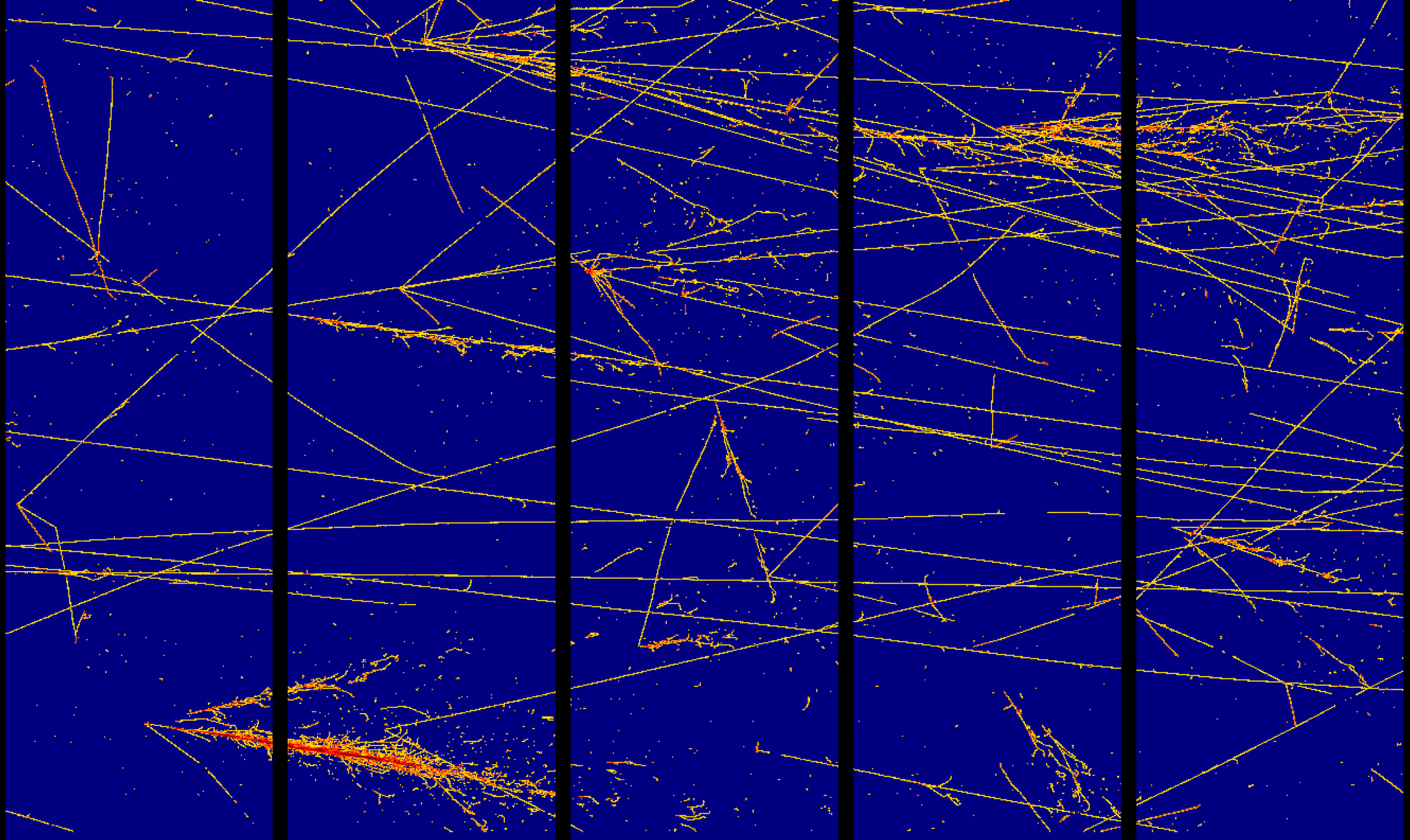


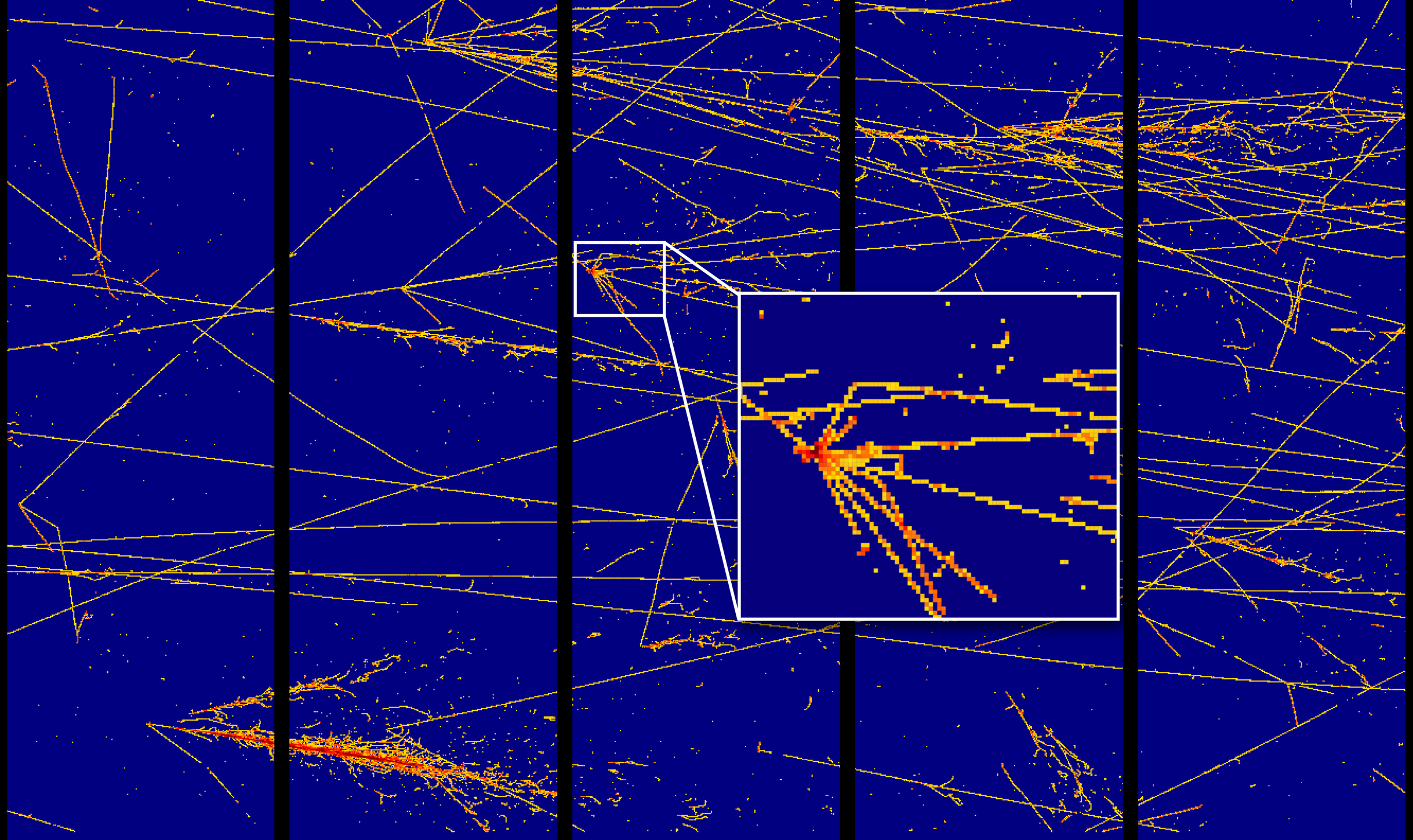


Small things matter
(inform direction and topology)



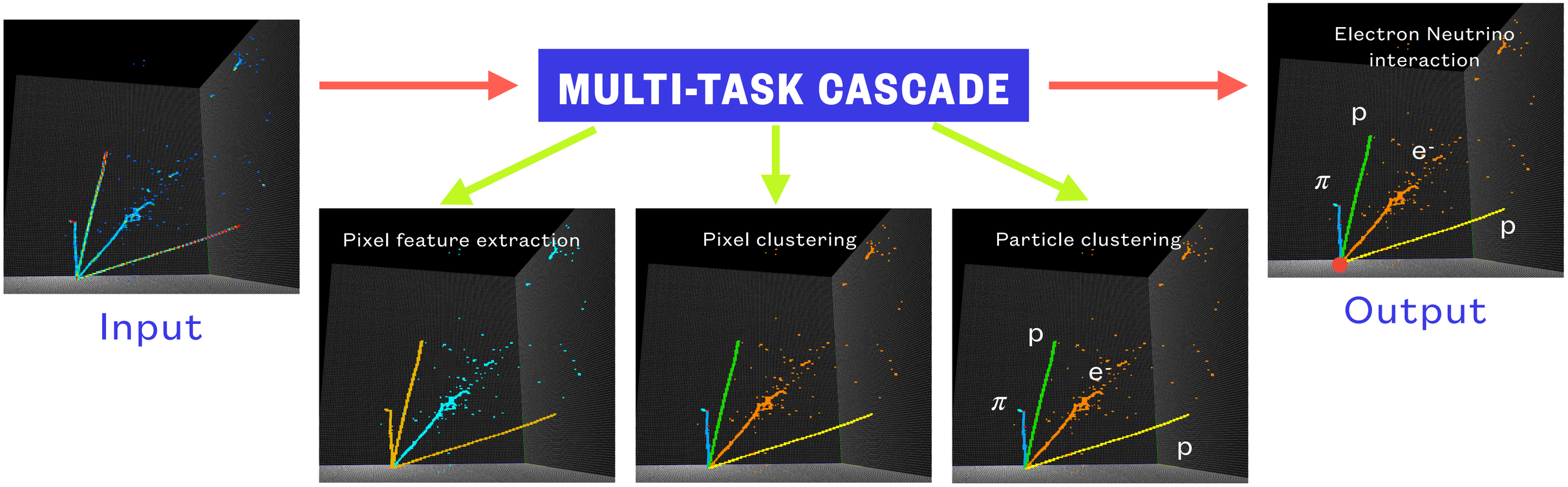

ArgonCube DUNE-ND 7x5 Modules Configuration Beam Spill
eventID
12
10
8
6
4
2
0
150
100
50
0
−50
−100
−150
Vertical Axis [cm]
0
100
200
300
400
500
Beam Axis [cm]





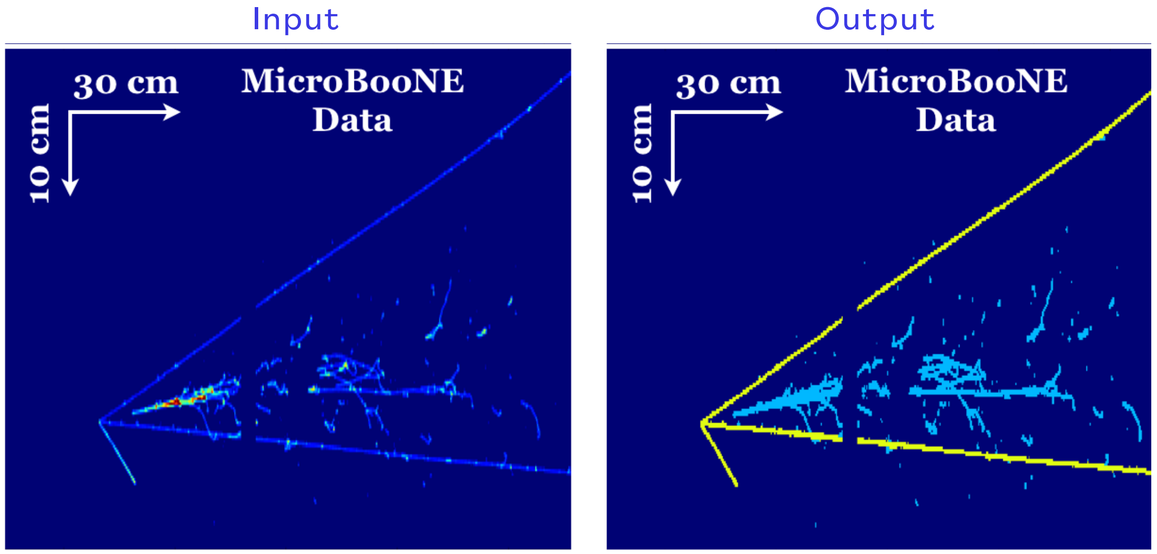
End-to-end reconstruction using ML

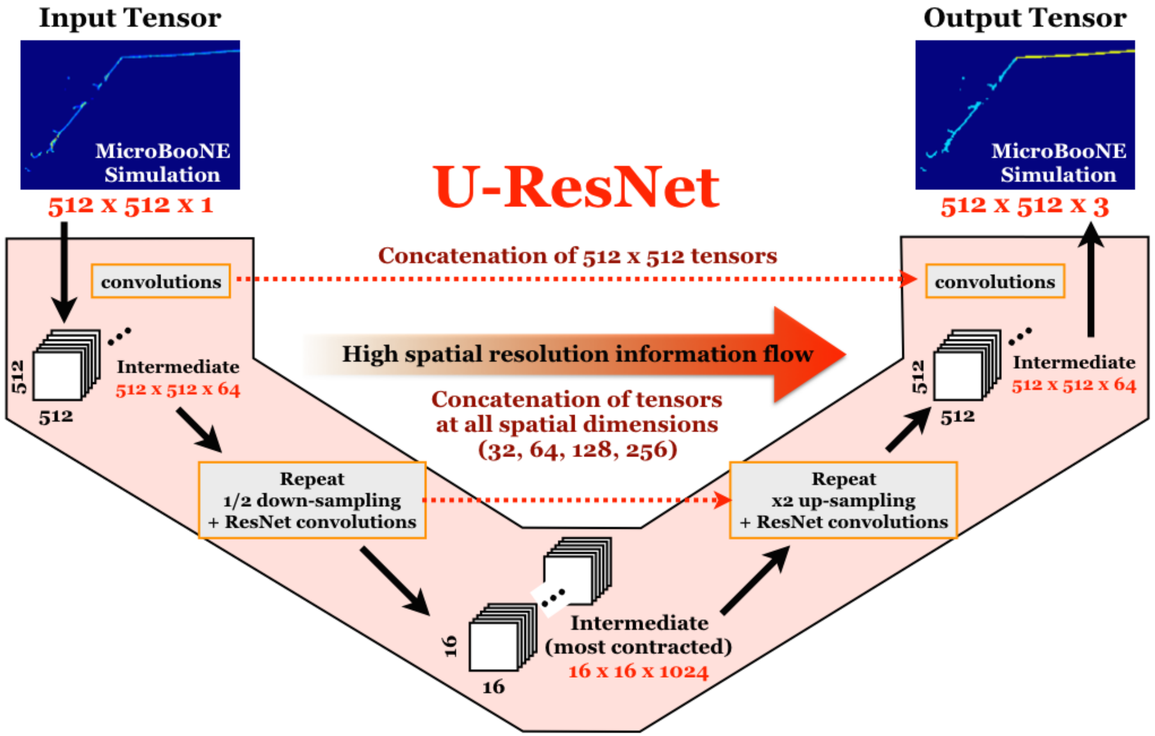
- The goal is to identify neutrino interactions in liquid argon. The reconstruction can be automated using ML by extracting physically meaningful, hierarchical features by chaining multiple ML models designed for each task.
- The input of the chain are 3D hits in the liquid argon, and the output is the full neutrino interaction, with each product being tagged and reconstructed.



Pixel feature extraction

- **Distinguish between different topologies** (e.g. tracks and showers)
- Identify edge points (track start/end point, shower start)
- Supervised classification using simulated samples and a U-ResNet-likde network (*encoder-decoder* structure)

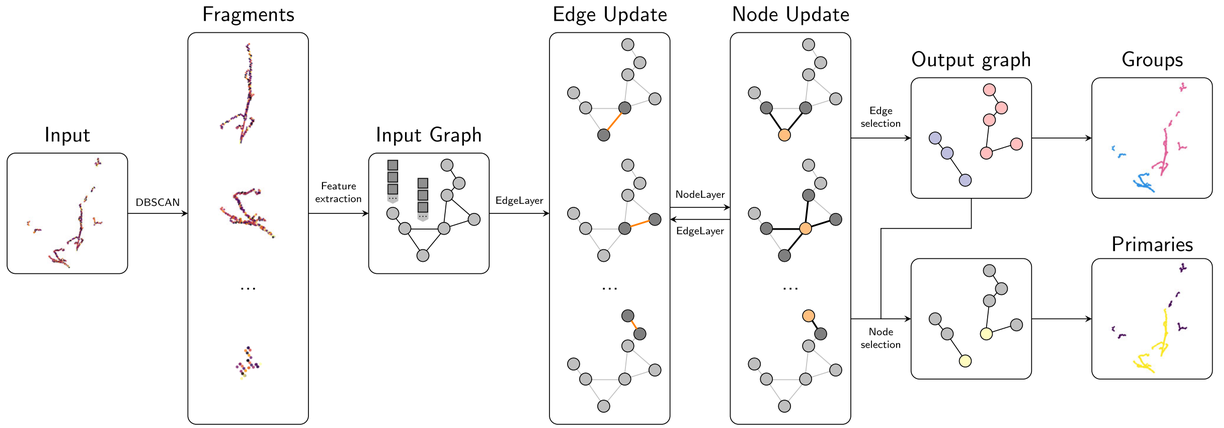




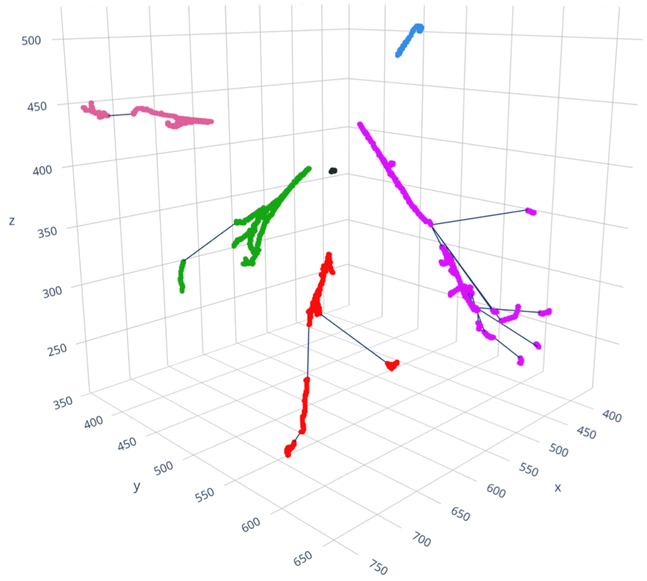
arXiv:1808.07269

Pixel clustering

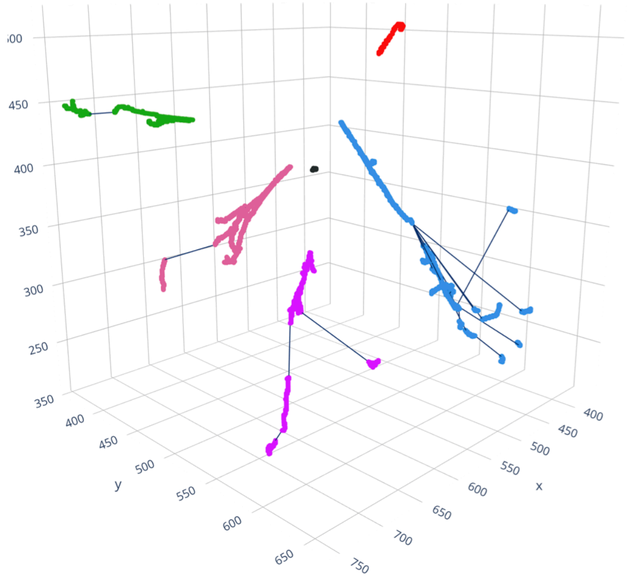
- Graph NN called GrapPA (graph Particle Aggregator) employed to predict the adjacency matrix of EM shower fragments and identify the origin of the showers



Target



Prediction

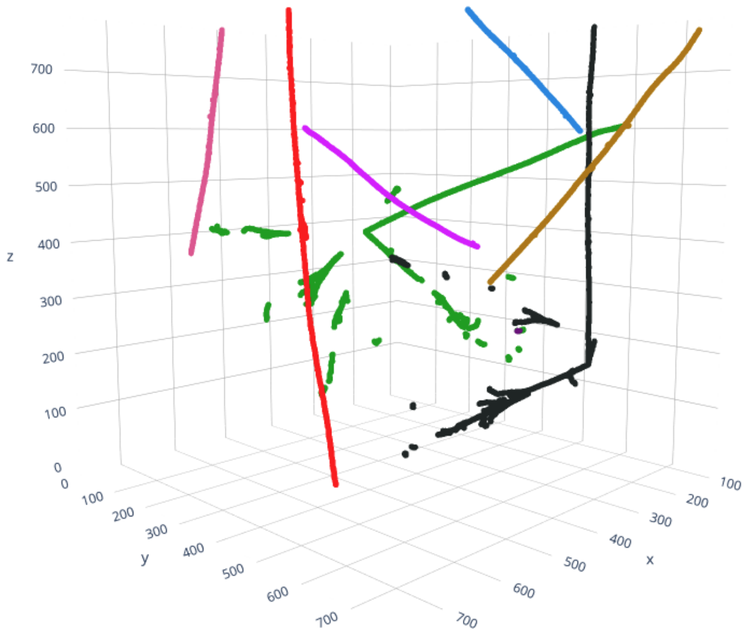


arXiv:2007.01335

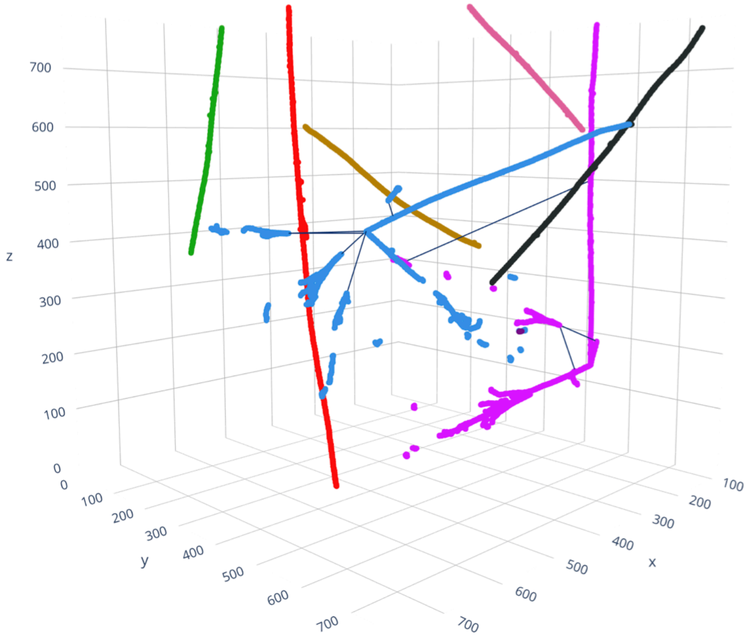
Particle clustering

- Grouping task, we can re-use GrapPA:
  - Interaction = a group of particles that shared the same origin (i.e. neutrino interaction)
  - Edge classification to identify an interaction
  - Node classification for particle type ID

Target



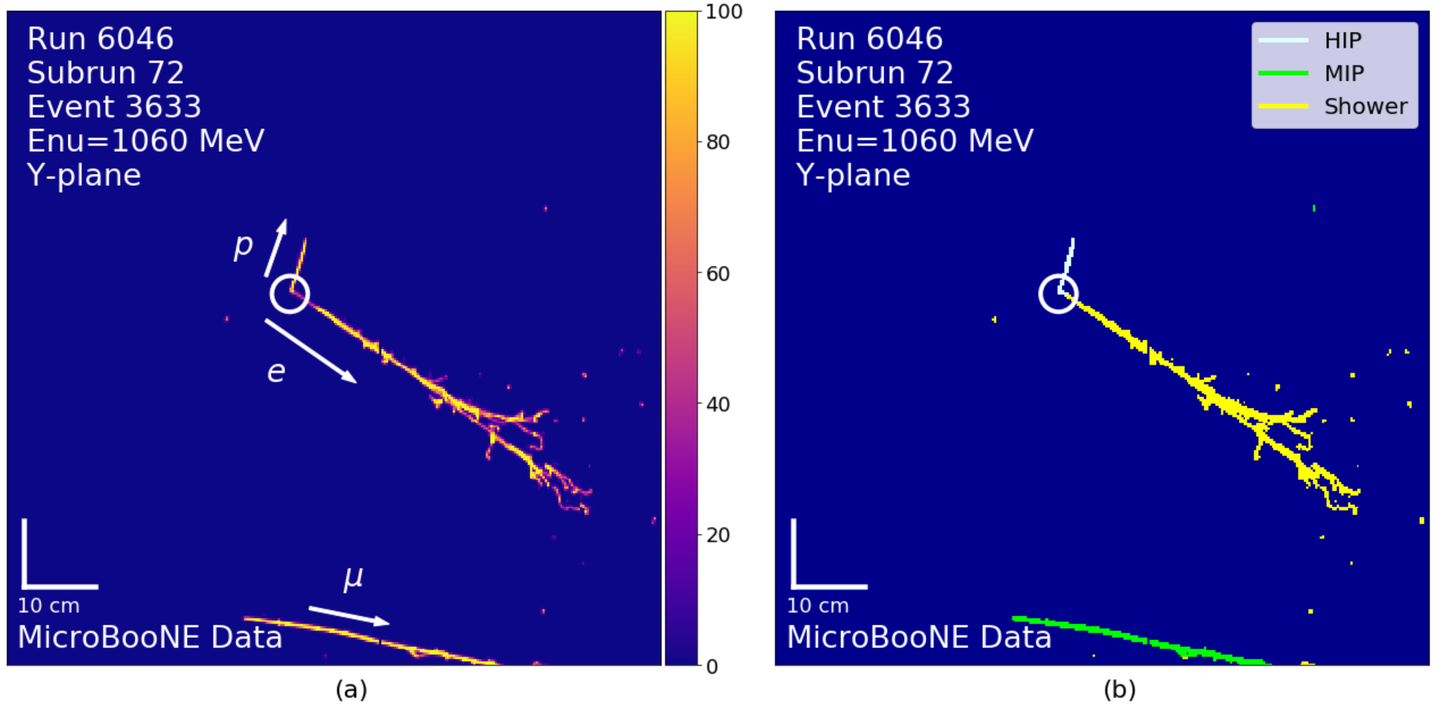
Prediction

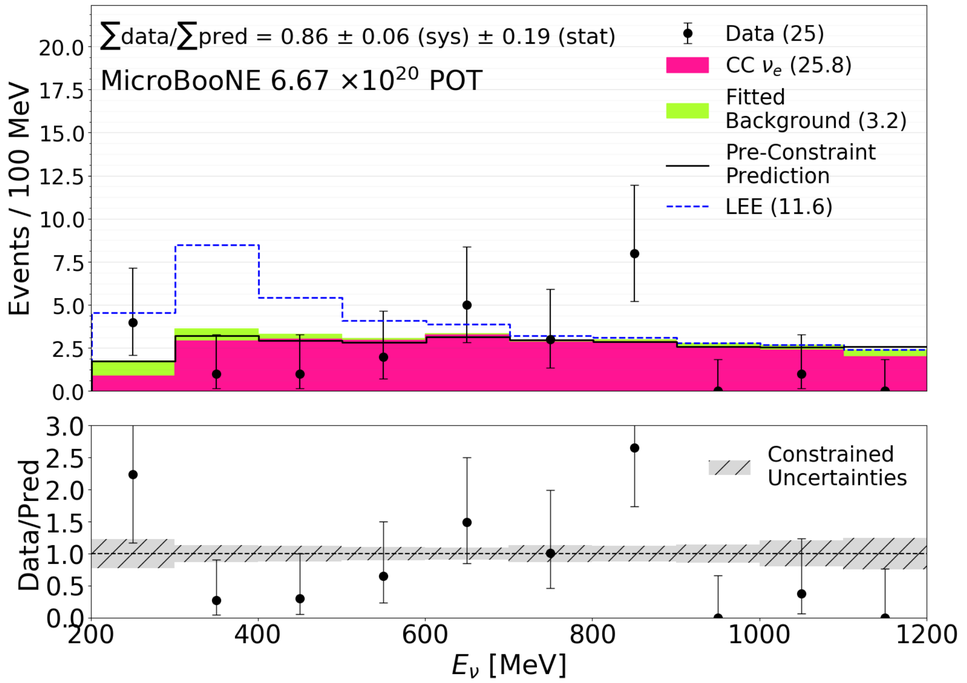


arXiv:2007.01335

$1e1p$ channel at MicroBooNE

- First analysis using deep learning methods in a LArTPC
- Results consistent with $1eNp0\pi$ analysis (more classical approach): “no excess”





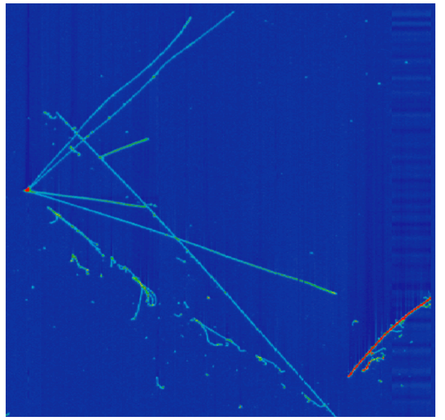
arXiv:2110.14080

Differentiable simulation

- The adoption of GPGPU algorithms can enable the implementation of a ***differentiable* simulator**, which means applying gradient-based methods to learning and control of physical systems.
- This technique requires computing the **model derivatives** at each step: can be easily implemented with popular ML packages such as PyTorch.
- Two main applications
  - automatic inference of the **detector physics model parameters** (*calibration*)
  - automatic inference of the **detector simulation input**

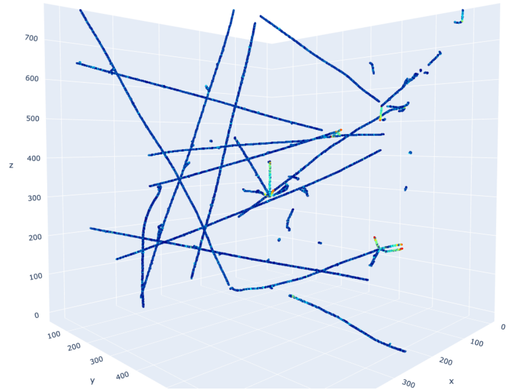
Y_{data}

Define the **data/sim discrepancy as loss**



Minimize the loss by computing the **model derivatives** and **propagating the gradient back** to model parameters

Y_{sim}

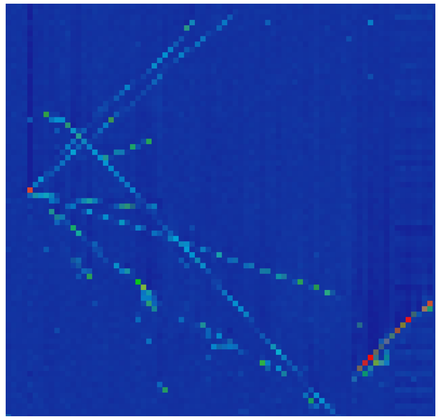


X: input

Courtesy of K. Terao (SLAC)

F(x, θ)

Detector physics modeling



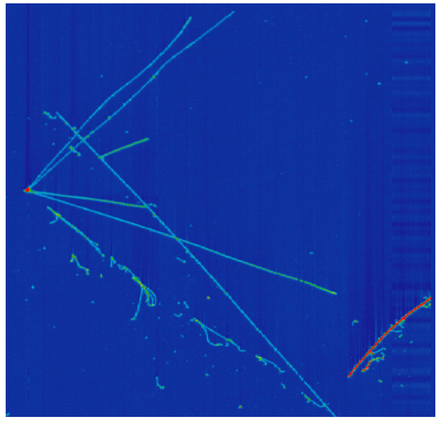
θ: model parameters

Differentiable simulation

- The adoption of GPGPU algorithms can enable the implementation of a ***differentiable* simulator**, which means applying gradient-based methods to learning and control of physical systems.
- This technique requires computing the **model derivatives** at each step: can be easily implemented with popular ML packages such as PyTorch.
- Two main applications
  - automatic inference of the **detector physics model parameters** (*calibration*)
  - automatic inference of the **detector simulation input**

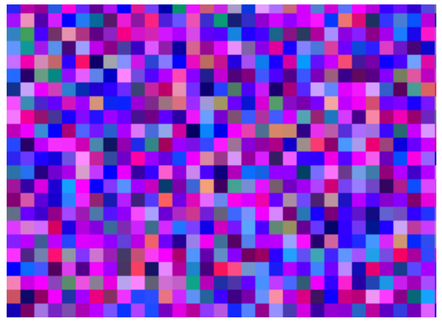
Y_{data}

Define the **data/sim discrepancy as loss**



Minimize the loss by computing the **model derivatives** and **propagating the gradient back** to the input

Initialize the input with random-values



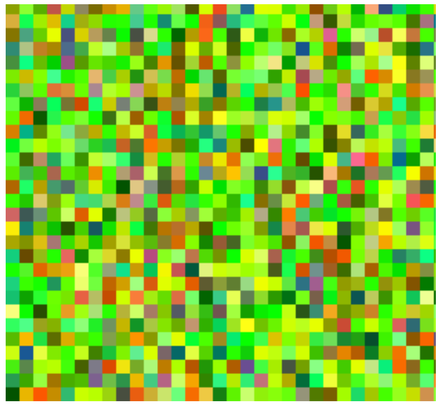
X: input

F(x, θ)

Detector physics modeling

θ: model parameters

Y_{sim}



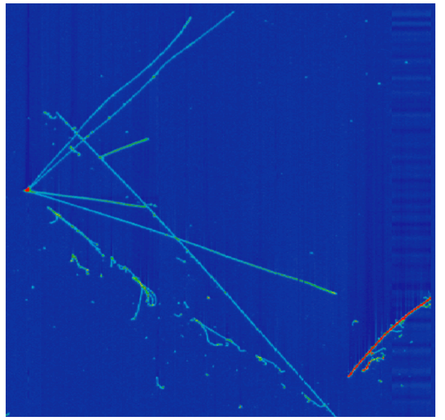
Courtesy of K. Terao (SLAC)

Differentiable simulation

- The adoption of GPGPU algorithms can enable the implementation of a ***differentiable* simulator**, which means applying gradient-based methods to learning and control of physical systems.
- This technique requires computing the **model derivatives** at each step: can be easily implemented with popular ML packages such as PyTorch.
- Two main applications
  - automatic inference of the **detector physics model parameters** (*calibration*)
  - automatic inference of the **detector simulation input**

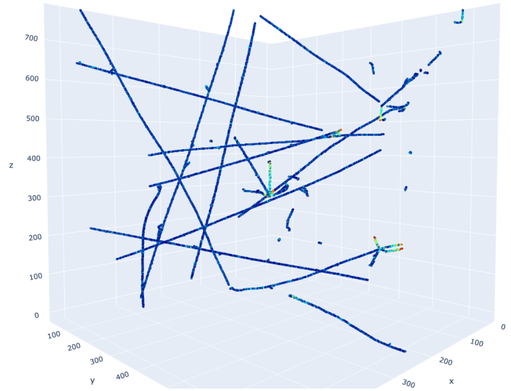
Y_{data}

Define the **data/sim discrepancy as loss**



Minimize the loss by computing the **model derivatives** and **propagating the gradient back** to the input

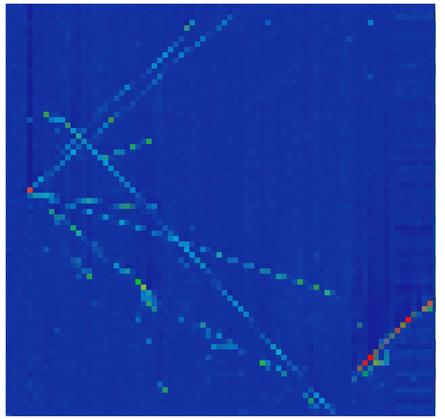
Y_{sim}



X: input

F(x, θ)

Detector physics modeling



θ: model parameters

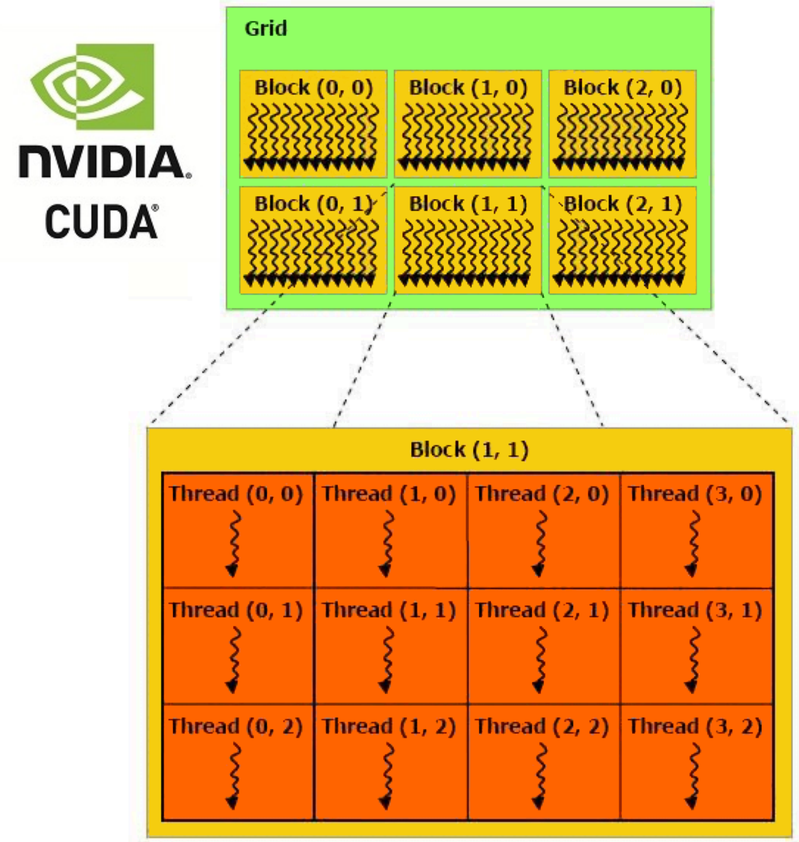
Courtesy of K. Terao (SLAC)

Technical implementation

- The detector simulation software is implemented as a set of GPU algorithms that use **Numba**, a just-in-time compiler that allows to **speed-up pure Python code** both on CPU and on GPU, using CUDA libraries.

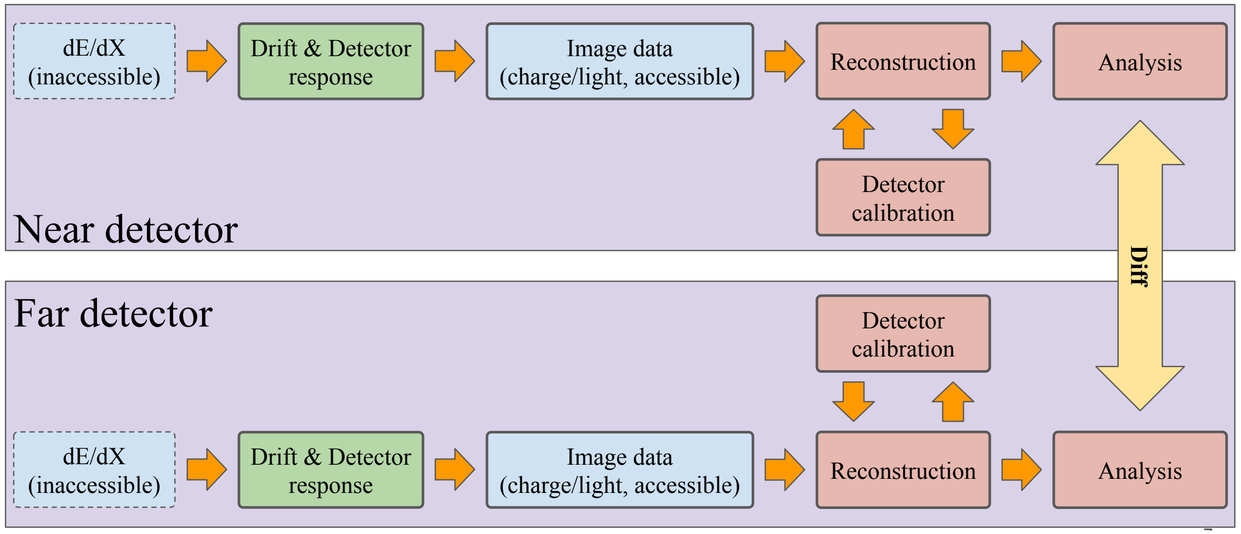
- The **CUDA platform** lets you run your function (the *kernel*) in large number of *threads*, that run in parallel on the GPU and are organized in *blocks.* It comes with a natural C++ extension, but can be used also with other languages (Java, Fortran, Python).

- The advantage is that the CUDA *hides* the specific underlying architecture, and allows to compile the **same code on different GPUs** with automatic scalability.

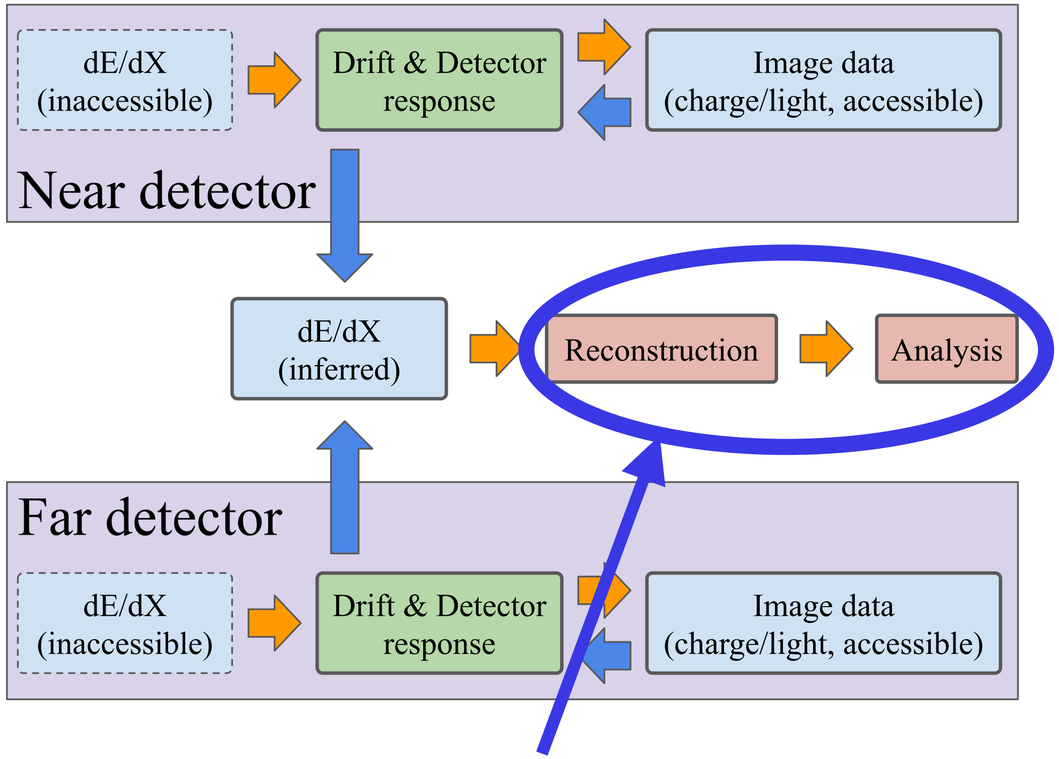


Implications for DUNE

TRADITIONAL APPROACH



DIFFERENTIABLE SIMULATOR APPROACH



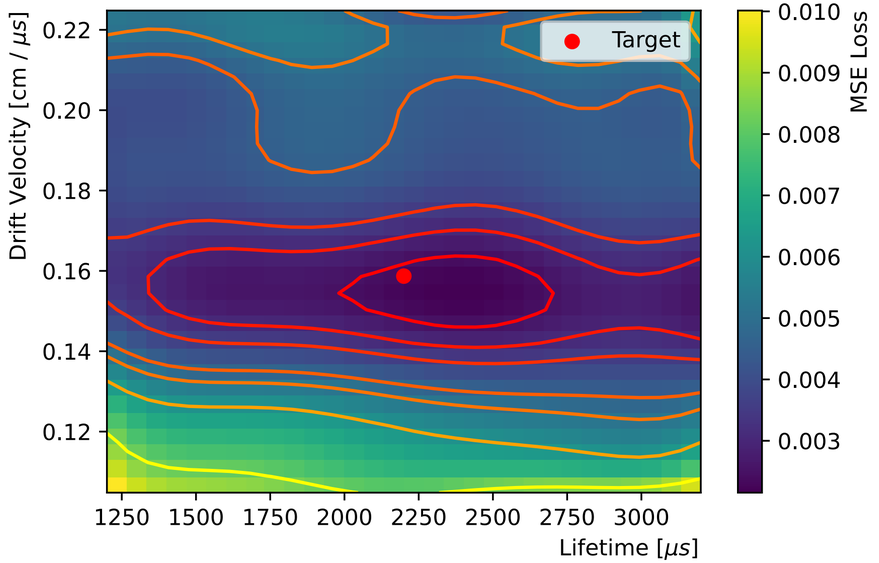
Reconstruction and analysis can be shared across detectors

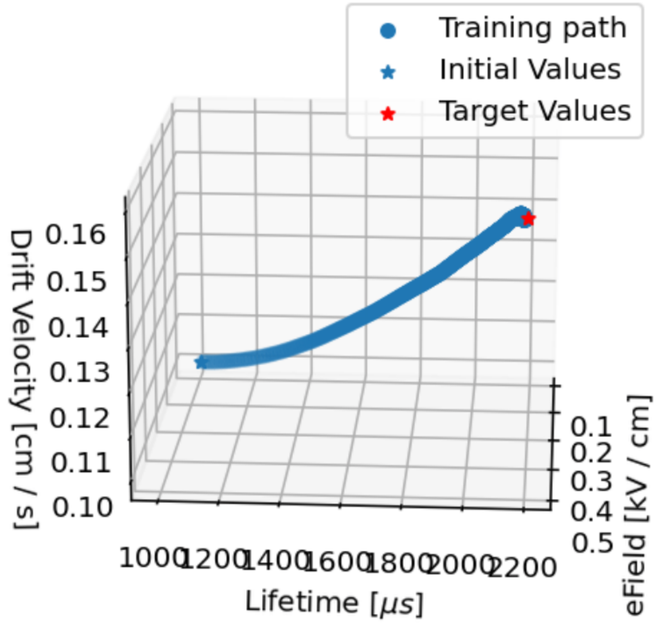
Courtesy of K. Terao (SLAC)

An example: drift velocity and lifetime

- End-to-end differentiable simulator for the DUNE liquid argon near detector has been used to make proof-of-principle studies
- As an example, the **gradient descent algorithm** has been used to find the minimum of the loss surface in the *(drift velocity; lifetime)* plane. The loss is defined as the discrepancy between the pixels.

Loss surface in the drift velocity - lifetime plane
followed by the gradient descent algorithm





Courtesy of K. Terao (SLAC)

Summary and prospects

- Machine learning has gone from a niche field with limited applications to an all-encompassing term in ~10 years.
- Particle physics has **started using shallow learning** (BDTs, low-depth NN trained with high-level variables) for a couple of decades. Virtually every modern analysis has one or more “machine-learning” component.
- **Focus has shifted towards deep learning**: use directly raw features (event displays) with large, complex networks, often O(10) of layers of with O(100) of neurons each.
- Not only classification: deep learning is being used for fast simulations (GANs, VAEs), speed up fitting, anomaly searches (unsupervised learning).
- **Differentiable simulator** gaining traction: exploiting tools developed for deep learning (*autodiff*) to automize parameter estimation and make reconstruction tools detector-agnostic.