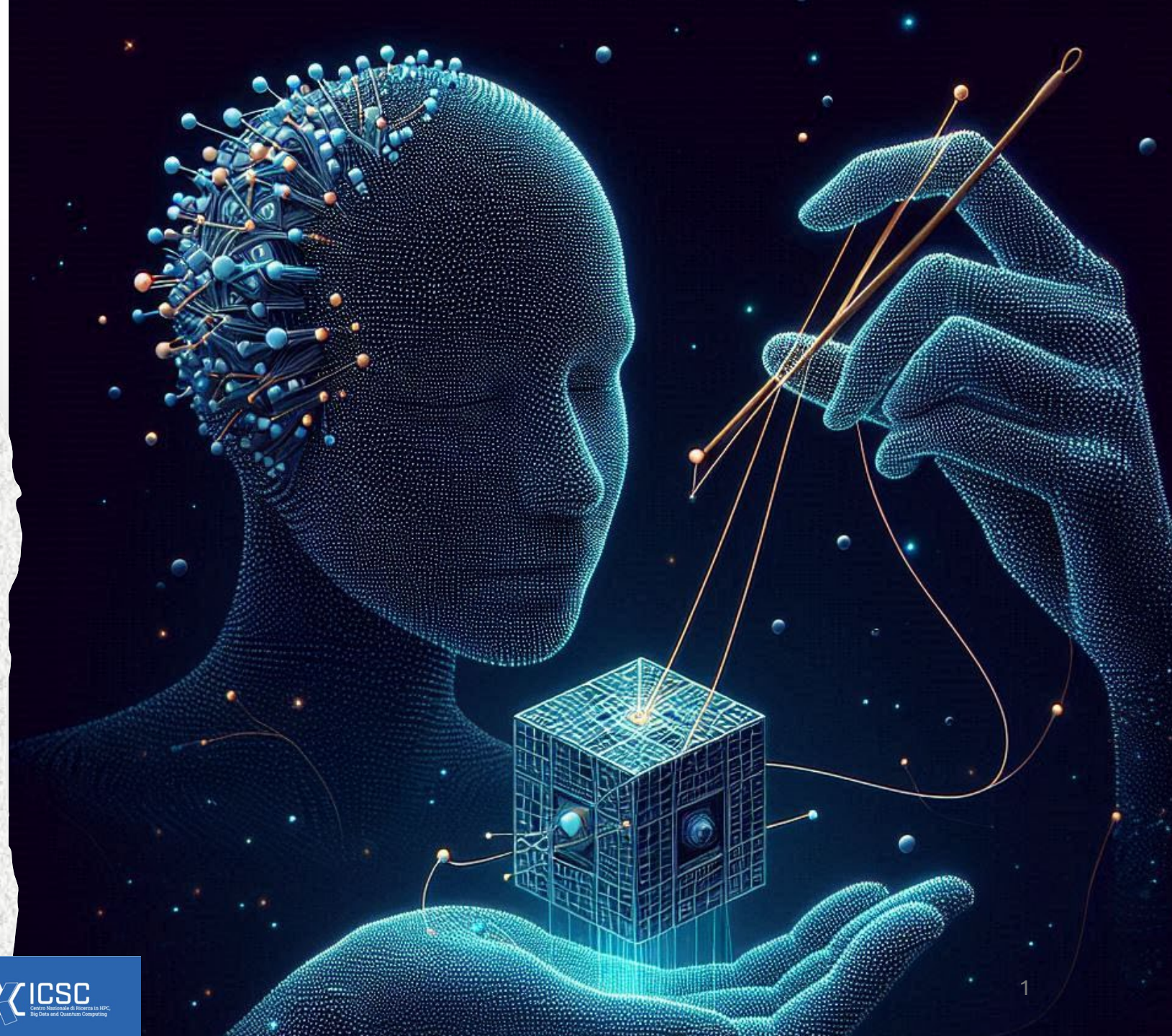
Workshop sul calcolo nell'INFN

Advanced Tracking Analysis in Space Experiments with Graph Neural Networks

F. Cuna, M.Bossa, F. Gargano, N. M. Mazziotta



INFN
Istituto Nazionale di Fisica Nucleare

Finanziato dall'Unione europea NextGenerationEU
Ministero dell'Università e della Ricerca
Italiadomani PIANO NAZIONALE DI RIPRESA E RESILIENZA
ICSC Centro Nazionale di Ricerca in HPC, Big Data and Quantum Computing

OUTLINE

Our work aims to develop robust AI- driven algorithms for space experiments capable of handling real-world experimental data.

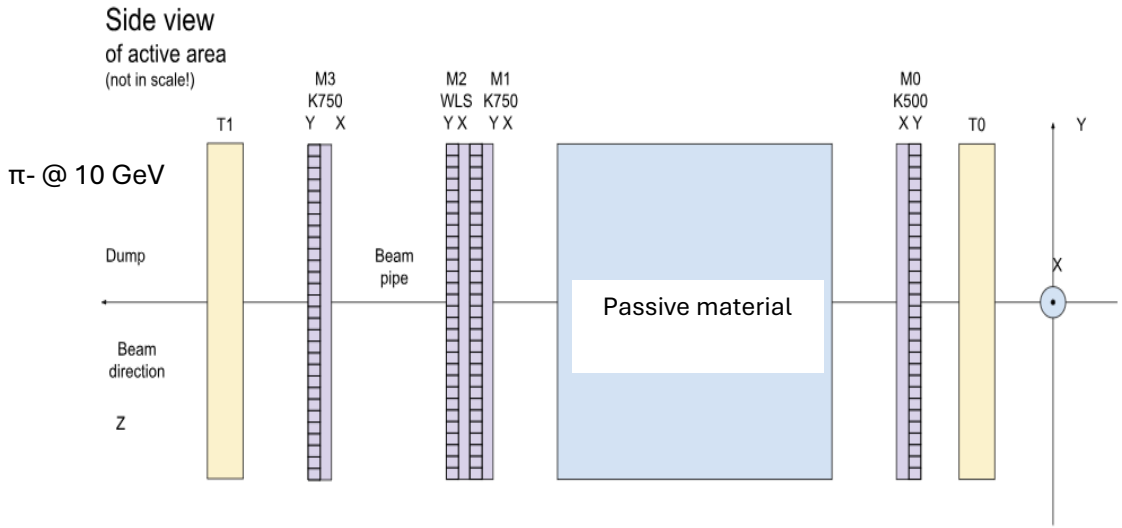
- Test beam dataset as test bench for the AI tracking algorithm
- The Graph Neural Network (GNN)
- The application to the HERD experiment

Beam test setup for developing AI tracking algorithm

A range of models inspired by computer vision applications were investigated, which operated on data from tracking detectors in a format resembling images.
→ Promising but limited by the high dimensionality and sparsity of the data

Tracking data are naturally represented as graph by identifying hits as nodes and tracks segments as (in general) directed edges.
→ Geometric deep learning approach.

We implemented an algorithm which exploits the potentials of the Graph Neural Networks (GNN), a subset of GDL algorithm, for the task of track reconstruction in a model of space experiment.



Beam test set up at CERN T10

The set up has been simulated by using Geant4 toolkit.
Zirettino has not been included for now.
A beam of π- of 10 GeV/c with inclined tracks of 0.5 deg has been simulated.
M0,M1,M2,M3 are fiber tracking layers:

- Fibers are 10 cm long with a radius of 0,25 mm.
- Strip pitch read-out: 0.25 mm

M2 consists of WLS with a 100mm x 100mm x 3mm LYSO crystal in between.

The simulation includes the LYSO crystal but considers the fiber as the scintillating ones.
Random noise hits have been added to simulate properly the electronic noise, spurious hits related to low-energy particles in orbit, backscattering hits…

Brief review of the GNN architecture

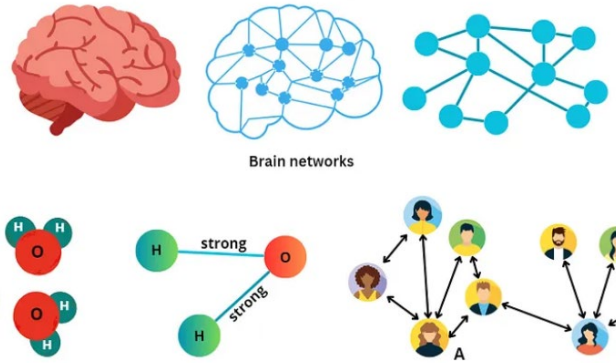
A graph represents the relations (edges or links) between a collection of entities (nodes).
Graph Neural Networks (GNNs) are a class of deep learning models that are designed to operate on graph-structured data.
They have shown remarkable success in tasks such as node classification, link prediction, and graph classification.

The key idea behind GNNs is to learn representations for nodes and edges in a graph by aggregating information from their local neighborhood.

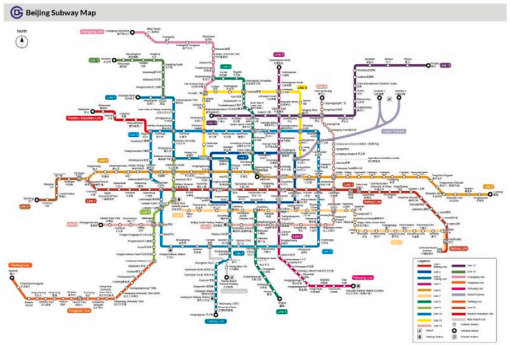
A GNN consists of a number of layers, each of which updates the representation of each node based on its local neighborhood.

The representation of each node is typically a low-dimensional vector that encodes the node's properties and its relationships with other nodes.

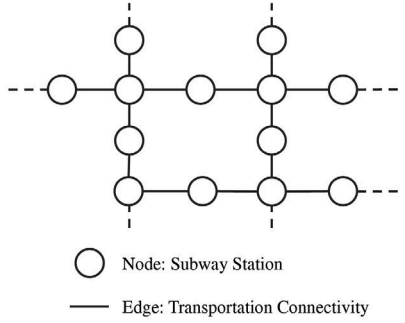
The key component of a GNN layer is the aggregation function, which takes as input the representations of a node's neighbors and produces a new representation for the node.



- *A Gentle Introduction to Graph Neural Networks, https://distill.pub/2021/gnn-intro/*
- Hands-On Graph Neural Networks Using Python, M.Labonne, Packt Publishing Ltd.

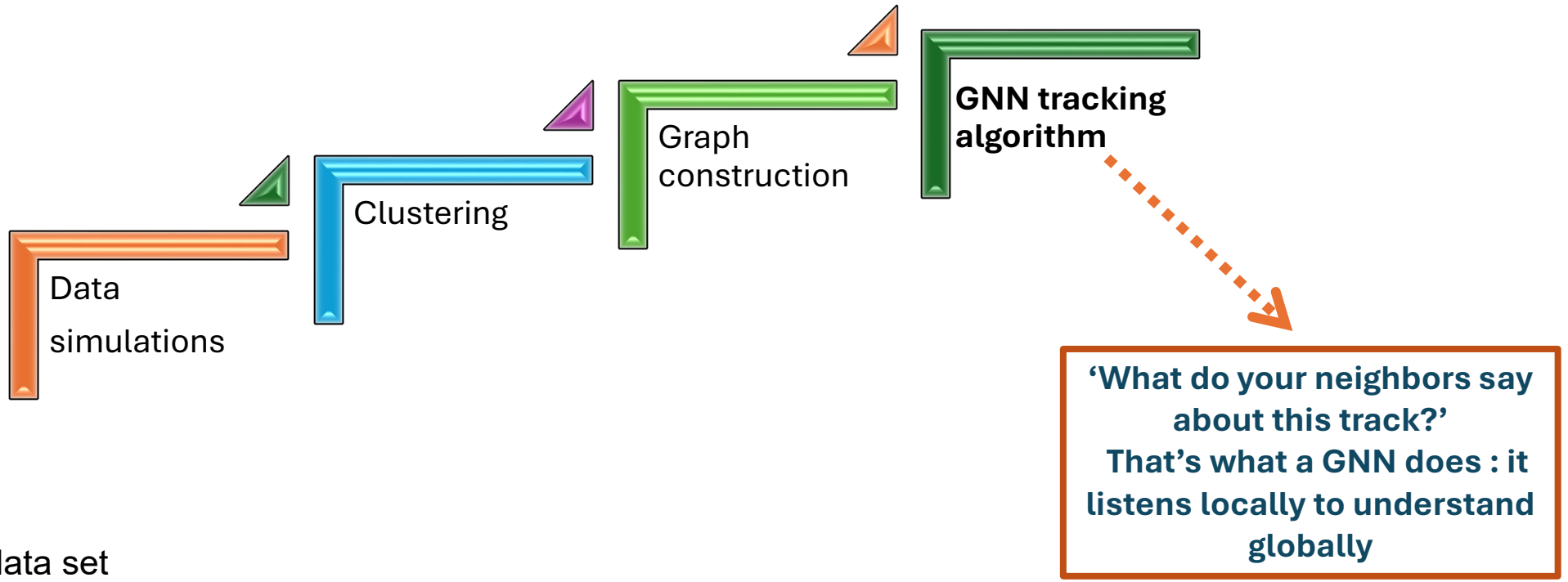


(a)



(b)

Graph Neural Network for tracks reconstruction
The journey to the algorithm



- Simulation of the data set
- Clustering
- Graph construction
- GNN-based tracking algorithm, which consists of a GNN classification of noise clusters from signal clusters and a final linear fit to retrieve the track parameter (angular coefficient and intercept)

Preprocessing phase: from row data to a graph structure

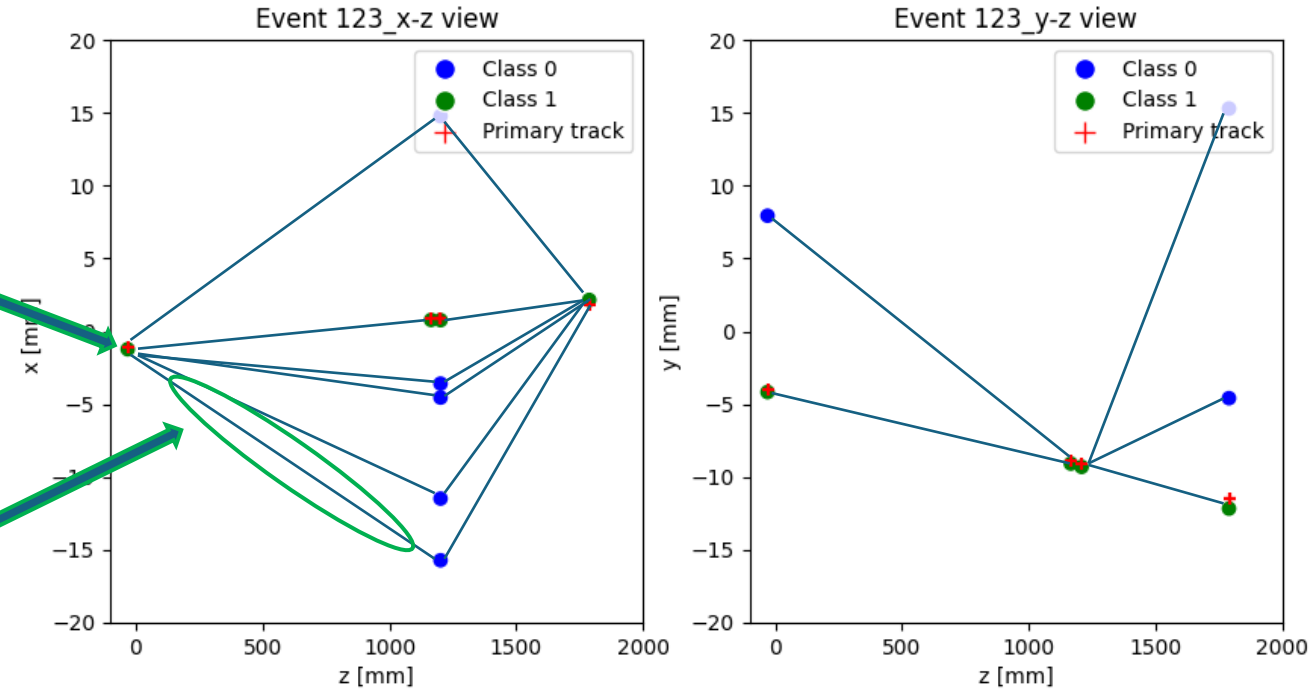
The hits inside tracking layers are clustered by applying a traditional clustering algorithm, where neighboring silicon strips with activated signals are grouped together and the barycenter of charge is calculated.

Starting by the clusters, data have been organized in a graph format, where nodes are represented by the clusters position and links by the inter layer connections between clusters.

Class 0: noise clusters
Class 1: signal clusters

Node

Link



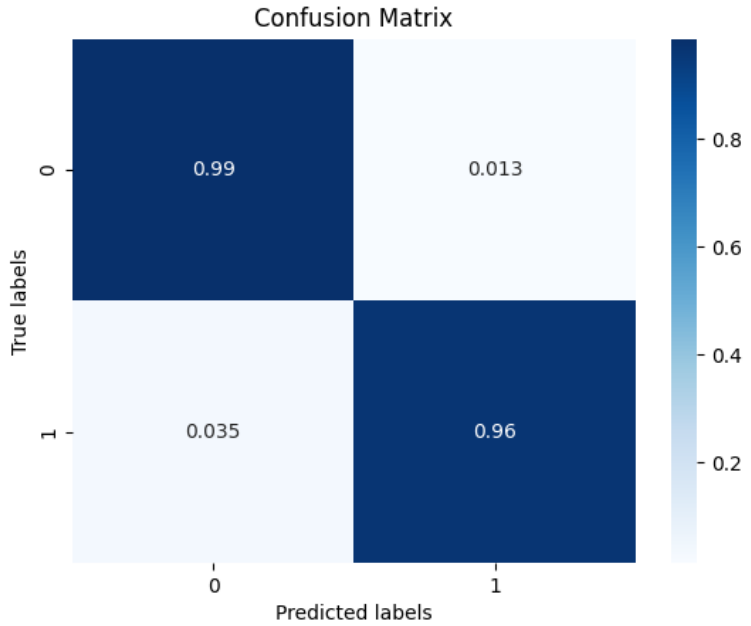
SageConv algorithm: the GNN architecture

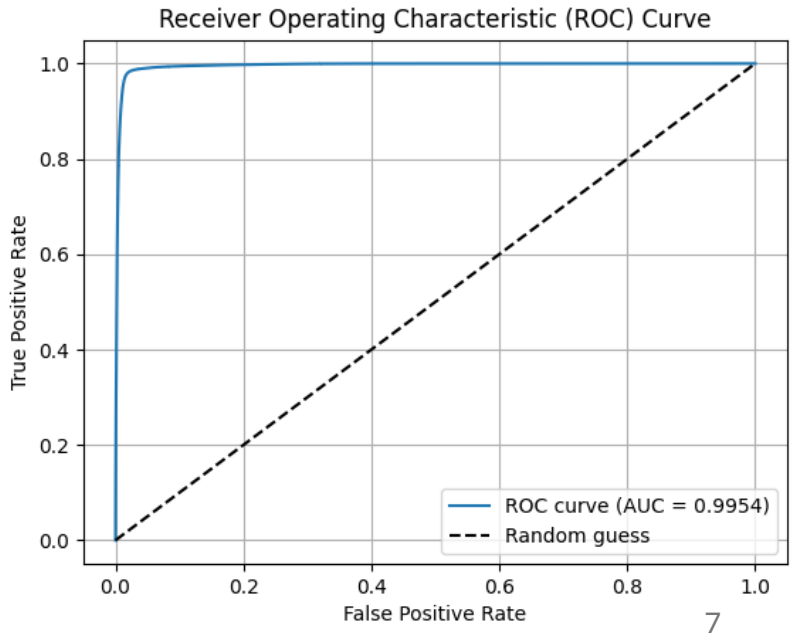
The SageConv architecture is a variant of GNN architecture.
The aggregation function takes into account the degrees of the nodes in the neighborhood.
SageConv uses the average of the representations of the neighbors, normalized by the degree of each neighbor, as the aggregate representation.
This allows it to capture more fine-grained information about the structure of the graph.

SageConv algorithm:

- 4 million event,
- 550 epochs,
- lr =0.0001,
- 7 GNN layer
- Mean aggregation function

CV 5 fold	accuracy	recall	precision
train data	0.972±0.007	0.968 ± 0.009	0.9902 ± 0.0014
validation data	0.972±0.007	0.968 ± 0.009	0.9862±0.0024

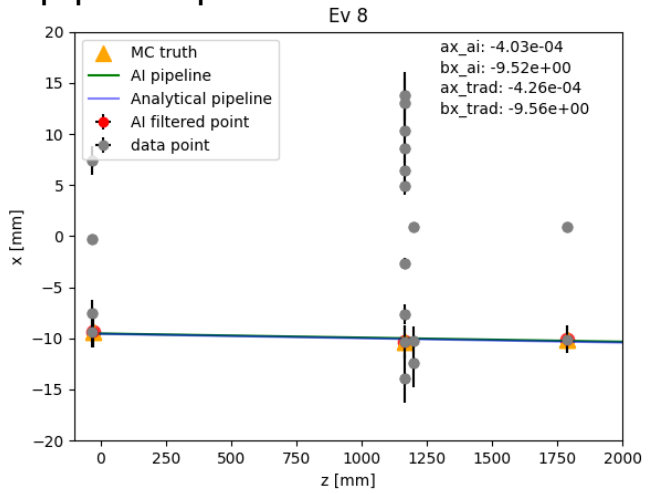


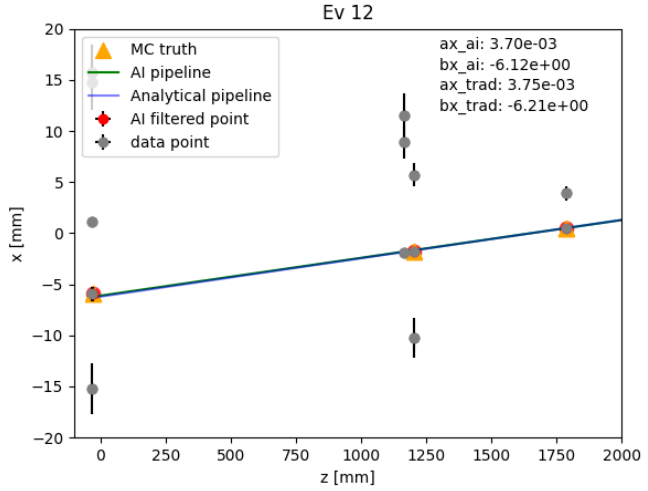


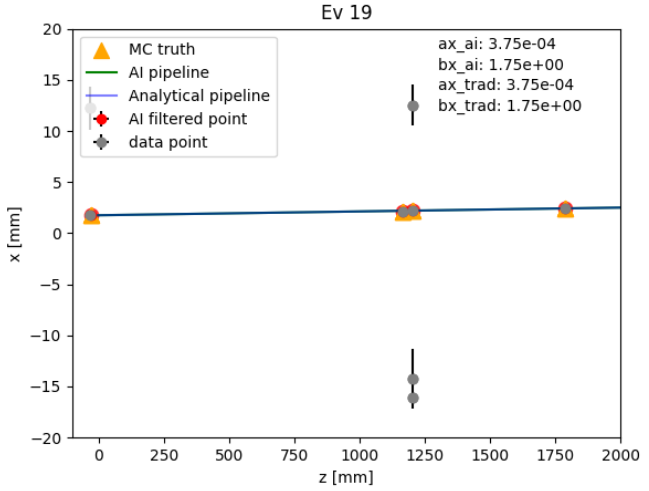
Event display: the tracks identification

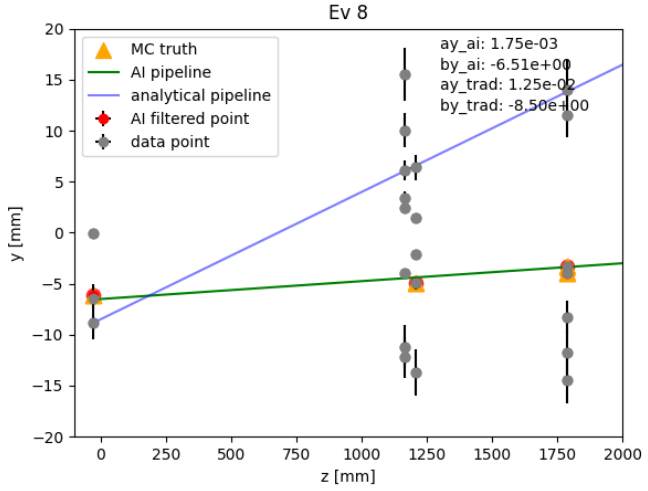
The traditional tracking pipeline minimizes the chi-square between the clusters inside each events, then fit the track with a linear function.

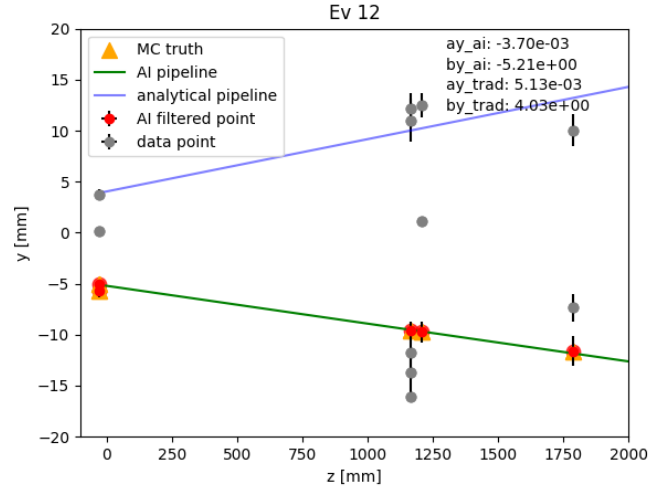
The AI pipeline performs the selection of good hits and fit the track with the same linear function.

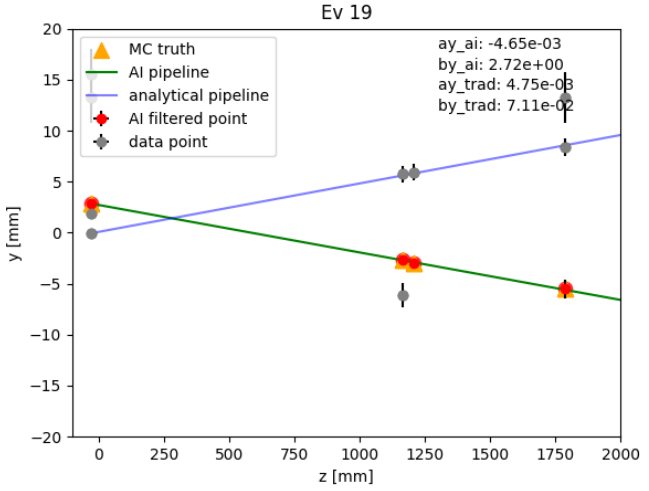




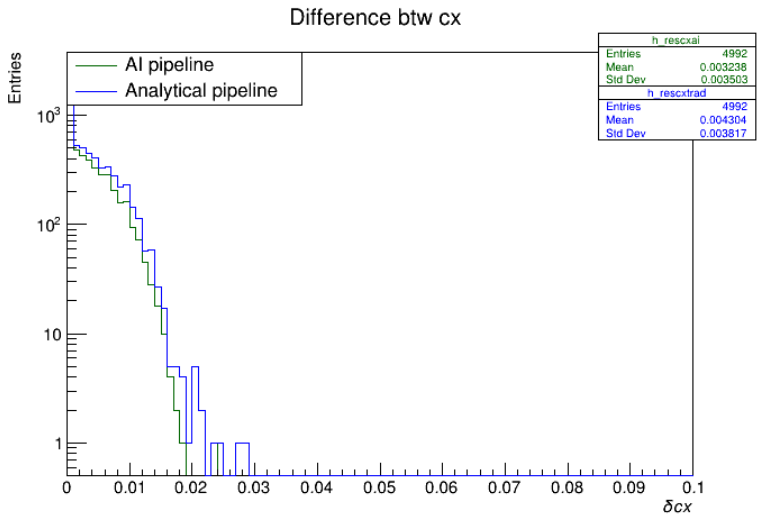


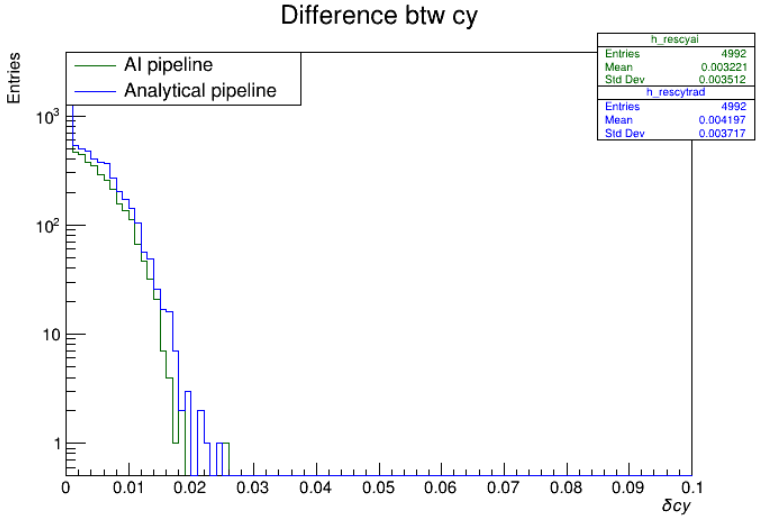


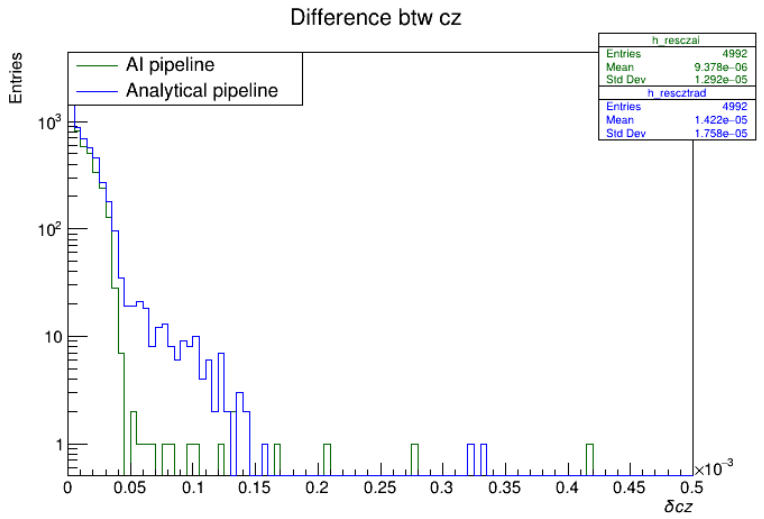




The SageConv algorithm: comparison with traditional pipeline of the director cosines







To process 5000 tracks the analytical pipeline takes 102 min, the AI pipeline takes 240 ms!

Spartan project with Nuclear Instrument

We are developing an “on-board tracker” by implementing the trained network on low consumption GPUs.

The AI-tracker will work as a filter which:

- Reduces the amount of data
- Performs a first rough tracking

HERD EXPERIMENT: more complex use case

To test our algorithm on more complex data, we decided to analyze HERD simulated data.
The **HERD** (High Energy cosmic-Radiation Detection) experiment is a space mission designed to directly detect cosmic rays, and it is set to be installed on the **Chinese Space Station (CSS)** in 2028.
The main goals of the mission:

- enhance our understanding of high-energy cosmic rays,
- search for indirect signals of dark matter,
- probe sources of high-energy particles such as protons, electrons, and photons.

Fiber Tracker (FIT):

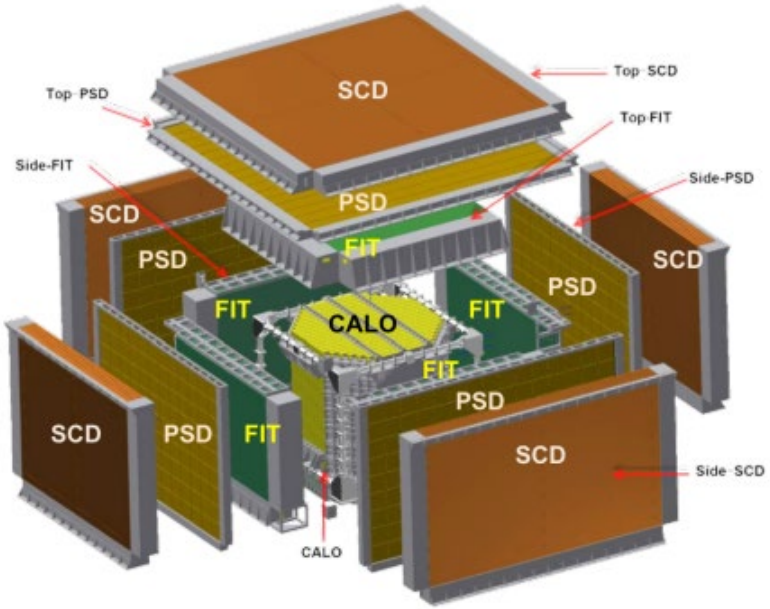
- Surrounds CALO on top and sides.
- Particle tracking and charge measurements
- 5 sectors, each with 7 X-Y scintillating fiber layers.
- Provides 7 precise position measurements.

The HERD simulation dataset

The full dataset, generated using the custom HerdSoftware simulation framework, consists of 4,300,000 events, equally divided into **2,150,000 electron** events and 2,150,000 proton events.

Both event sets are simulated within a power-law energy spectrum E^{-1}, spanning an energy range from 100 GeV to 1 TeV, and are distributed within a spherical region surrounding the HERD detector.

Electrons set was used for the tracking algorithm.

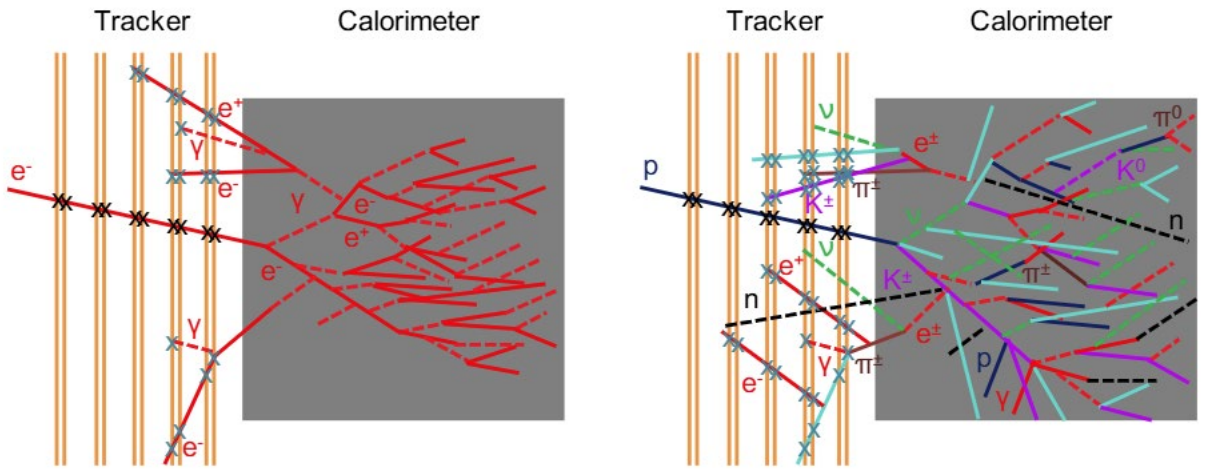


Preprocessing phase: prepare data for the GNN

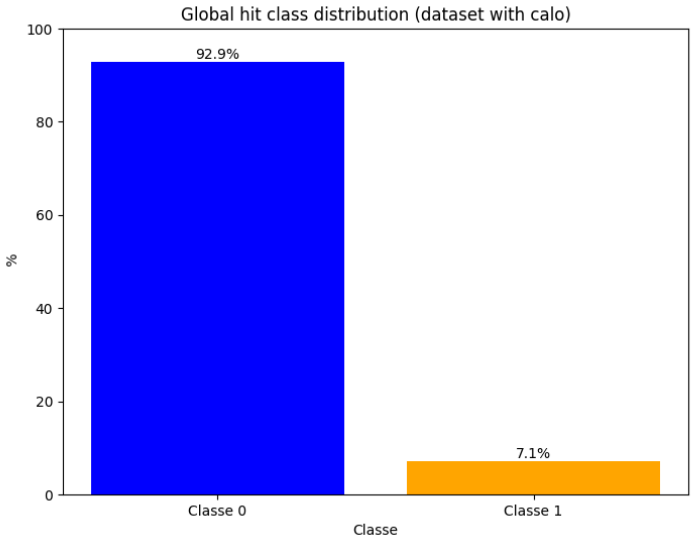
Data are highly imbalanced, since there are many bacskattering tracks originating from the calorimeter, which interfere with the correct identification of the primary particle's trajectory .

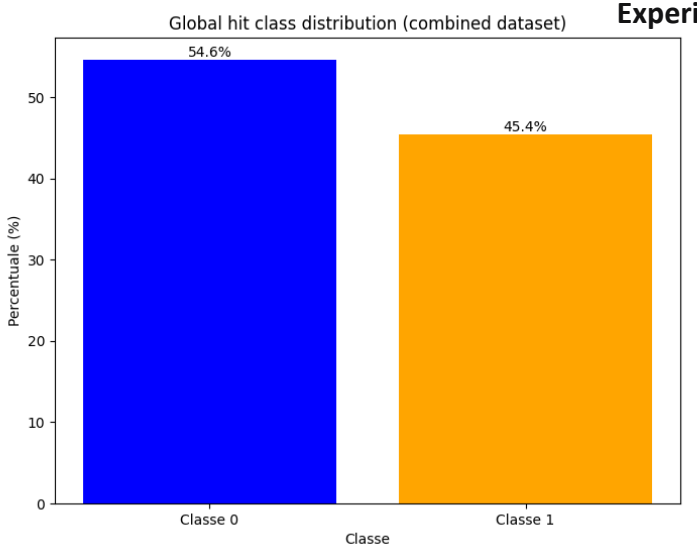
To mitigate the imbalance:

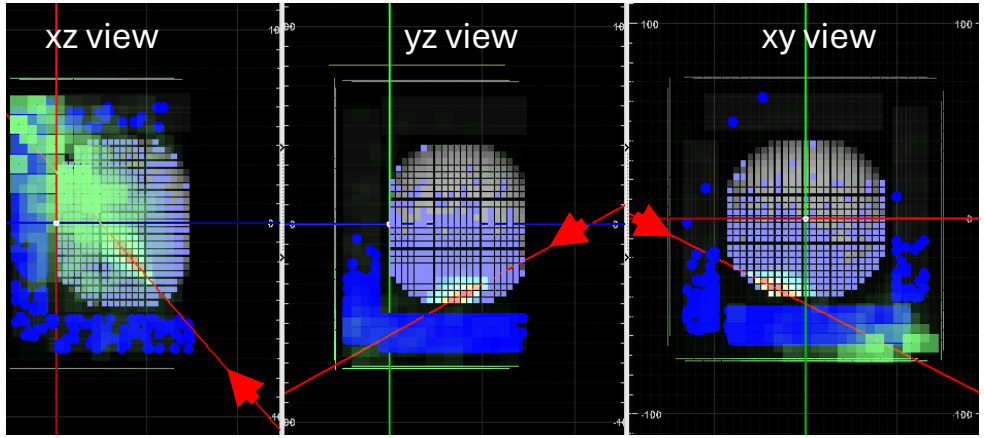
1. Clustering algorithm
2. Cut of noise clusters “far away” from the primary track
3. Adding simulated events without calorimeter



Advantages and Requirements in Time Resolving Tracking for Astroparticle Experiments in Space, M.Duranti et al.







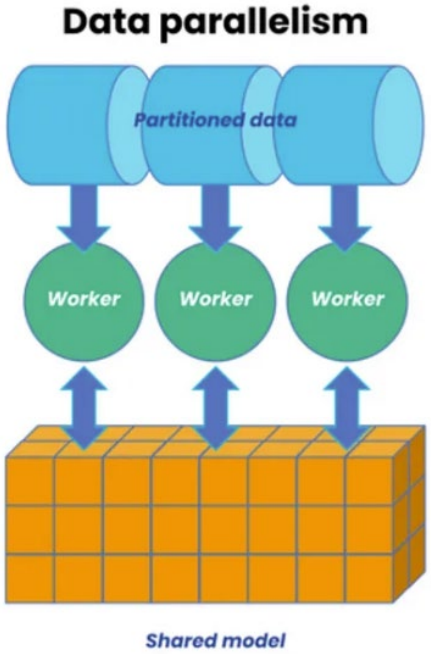
The GNN algorithm: distributed training

The training for these networks requires a lot of time, since graphs are quite dense, this requires new strategy to enhance time consuming!

Solution: distributed training!

We can't afford to wait 10 days to find out the network isn't working correctly!

For **data parallelism**, the **DistributedDataParallel** module wraps any PyTorch model and handles partitioning of data across devices as well as gradient aggregation automatically.



The same model replica handles a different mini-batch of input data on each device.
Their parameter updates are aggregated.

The GNN algorithm: distributed training and hyperparameter tuning

Hyperparameter tuning is the secret sauce that turns a decent model into a production powerhouse.
But...it is a time-consuming, computationally heavy task.

Solution: Ray Tune

- it conducts large-scale hyperparameter searches efficiently, saving time and computational resources.
- it comes equipped with advanced search algorithms (like Bayesian optimization and Population-Based Training) and schedulers designed for scalability.

How it works

1) Setup the model
2) Creating the Training Function
3) Setting Up Ray Tune Search Space for Hyperparameters
4) Creating the Ray Tune-Compatible Training Function
5) Choosing and Implementing a Scheduler
6) Using Search Algorithms for Improved Efficiency
7) Configuring Ray Tune for a Distributed Setup

```
# Configure for multi-node with distributed Ray setup
ray.init(address="auto") # Use the head node's address here

# Define Ray Tune setup for multi-GPU use
tuner = tune.Tuner(
train_model,
param_space=config,
tune_config=tune.TuneConfig(
resources_per_trial={"cpu": 4, "gpu": 3}, # 3 GPUs per trial
num_samples=100
)
)
tuner.fit()
```

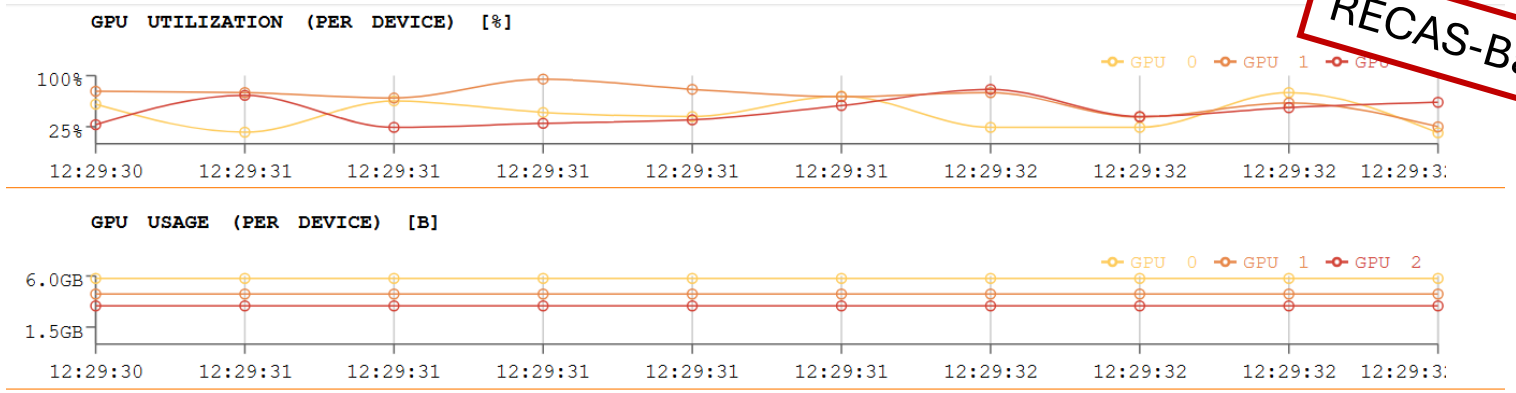
Tools for GNN training algorithm

- SageConv architecture
- 18 layers
- Mean aggregation function
- 128 hidden size
- Adam optimizer
- Binary cross entropy loss function

4,5 million simulated track data
75% train-15% validation-10% test

To enhance the time consuming we performed the distributing training by using:

- the JupyterLab instance with 3 A100 NVIDIA GPUs
- the Leonardo Hub instance with 4 A100 NVIDIA GPUs



RECAS-Bari

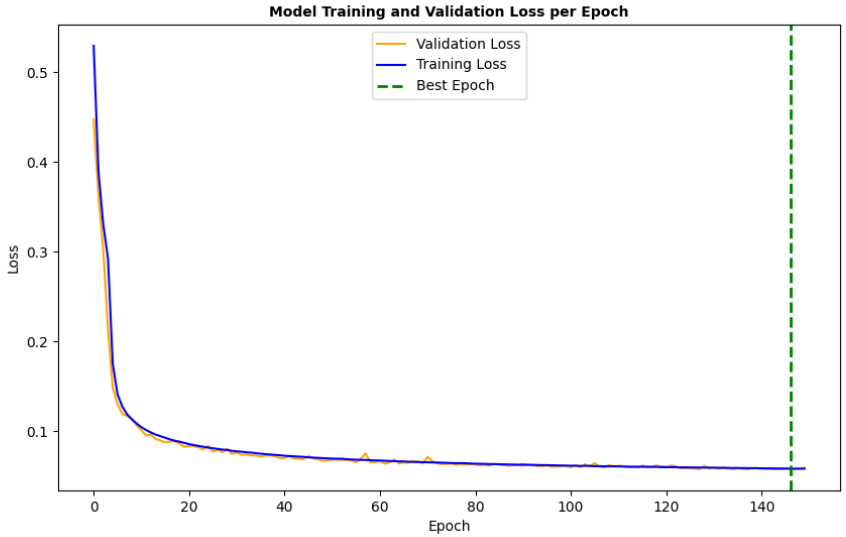
Leonardo-HUB
G. Bianchini's talk

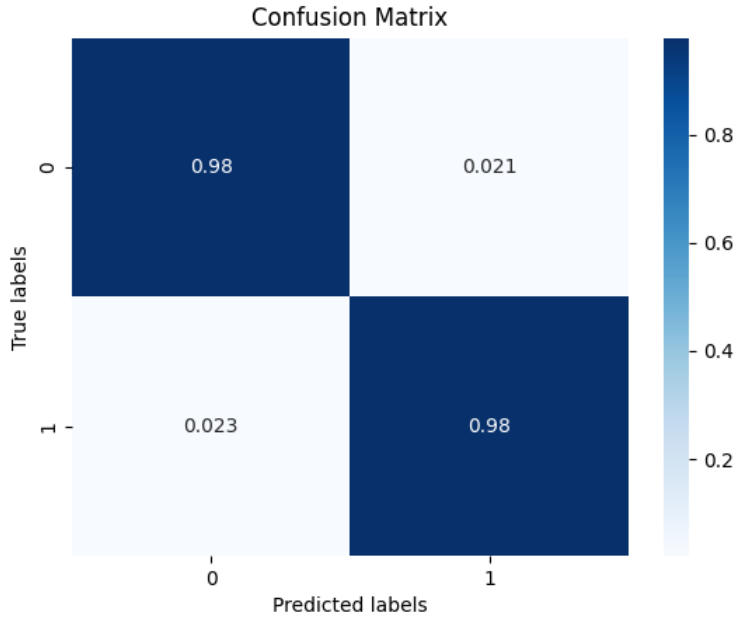
```
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 530.30.02              Driver Version: 530.30.02    CUDA Version: 12.1 |
|-------------------------------+----------------------+----------------------+
| GPU  Name                 Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf           Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                                        |                      |               MIG M. |
|===============================+======================+======================|
|   0  NVIDIA A100-SXM-64GB           On | 00000000:1D:00.0 Off |                    0 |
| N/A   44C    P0              83W / 459W|   7771MiB / 65536MiB |      1%      Default |
|                                        |                      |             Disabled |
+-------------------------------+----------------------+----------------------+
|   1  NVIDIA A100-SXM-64GB           On | 00000000:56:00.0 Off |                    0 |
| N/A   45C    P0              86W / 461W|   3999MiB / 65536MiB |     70%      Default |
|                                        |                      |             Disabled |
+-------------------------------+----------------------+----------------------+
|   2  NVIDIA A100-SXM-64GB           On | 00000000:8F:00.0 Off |                    0 |
| N/A   45C    P0              88W / 458W|   3957MiB / 65536MiB |     86%      Default |
|                                        |                      |             Disabled |
+-------------------------------+----------------------+----------------------+
|   3  NVIDIA A100-SXM-64GB           On | 00000000:C8:00.0 Off |                    0 |
| N/A   44C    P0              82W / 457W|   4103MiB / 65536MiB |     10%      Default |
|                                        |                      |             Disabled |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
+-----------------------------------------------------------------------------+
```

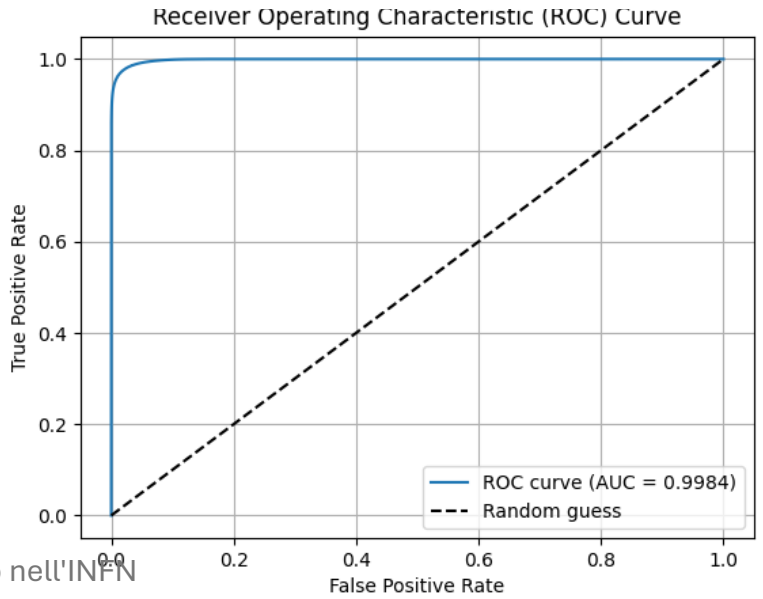
Thanks to G. Vino, G. Donvito and all Recas people for their support !
Thaks to spoke0/spoke3/datacloud support for Leonardo Hub!

Main Results: GNN algorithm evaluation

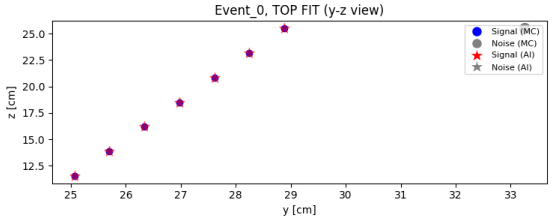


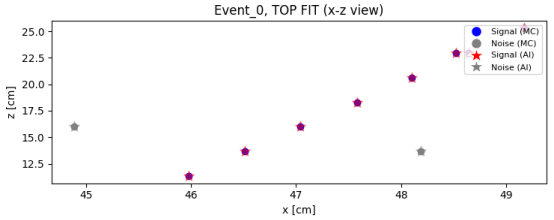


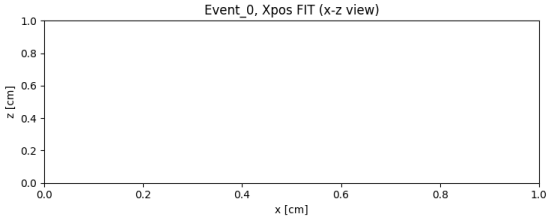
Metrics	Values
Accuracy	97.80%
Recall	97.65%
Precision	97.81%
F1-score	97.73%
ROC AUC	99.84%

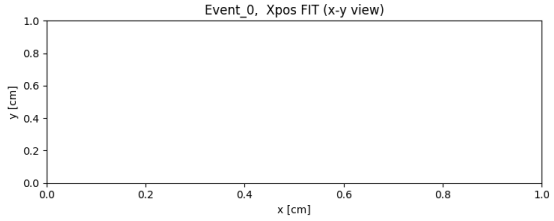


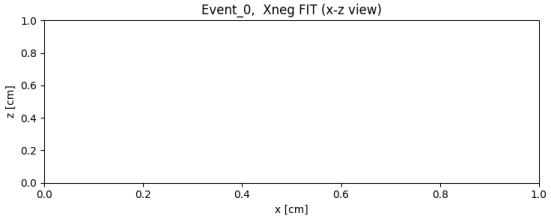
Main Results: events display

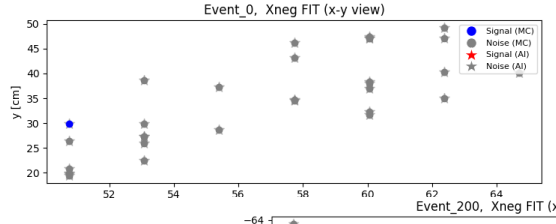


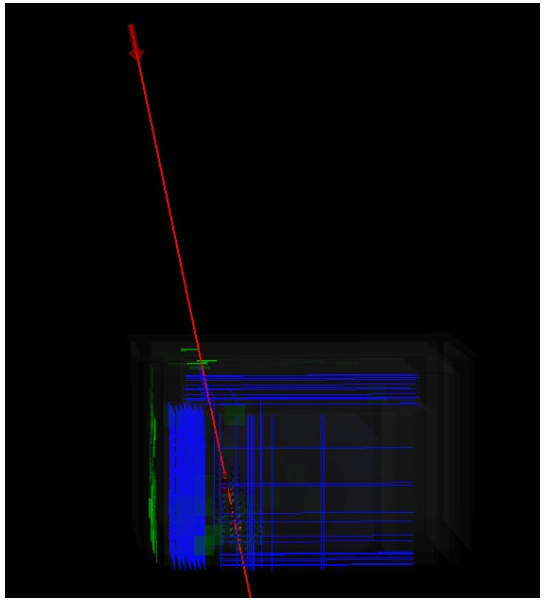




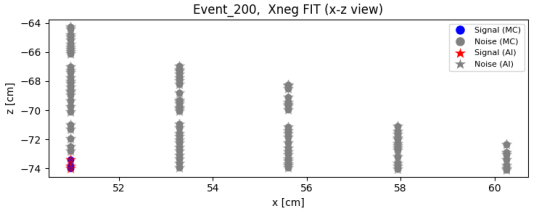


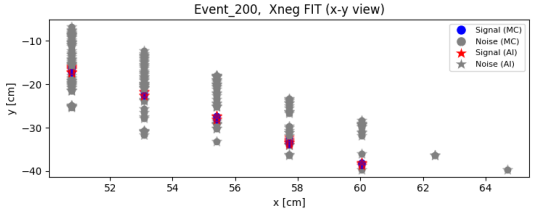


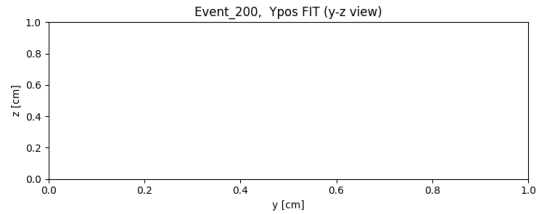


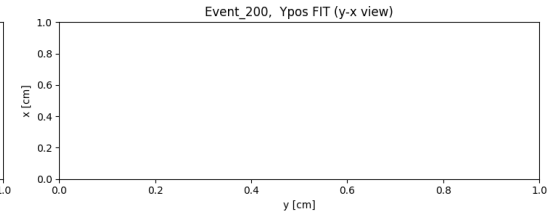


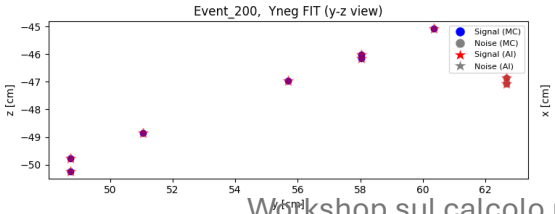
Paper submitted at PDP Conference!!!
https://ieeexplore.ieee.org/xpl/conhome/10974743/proceeding

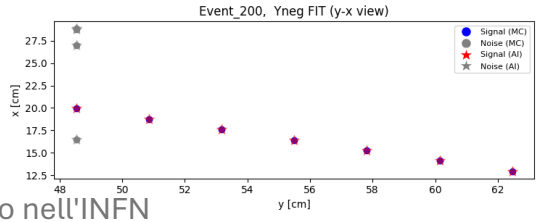


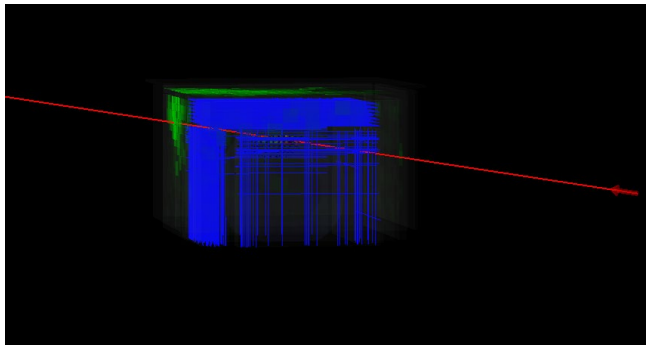












Conclusions and next steps

The GNN shows promising performance for tracking tasks compared to traditional analytical approaches, both in terms of accuracy and time efficiency.

Spoiler alert: the perfect classifier? It doesn’t exist!

The algorithm's validity is limited beyond a certain energy range!
Between 100 TeV and 1 PeV, the information from the FIT alone is insufficient to distinguish between backscattering and primary tracks.

What can we do???

1) Find the precise validity limits
2) Incorporate additional data from other subdetector (heterogeneous graph neural network and more sophisticated data preprocessing)

No worries, we’ll handle it! Stay tuned!

Progetto ICSC Centro Nazionale di Ricerca in High-Performance Computing, Big Data and Quantum Computing - CN00000013 PNRR Missione 4, Componente 2, Investimento 1.4 - CUP I53C21000340006 . Progetto ICSC Centro Nazionale di Ricerca in High-Performance Computing, Big Data and Quantum Computing - CN00000013 PNRR Missione 4, Componente 2, Investimento 1.4 - CUP I53C21000340006 .

Thank you

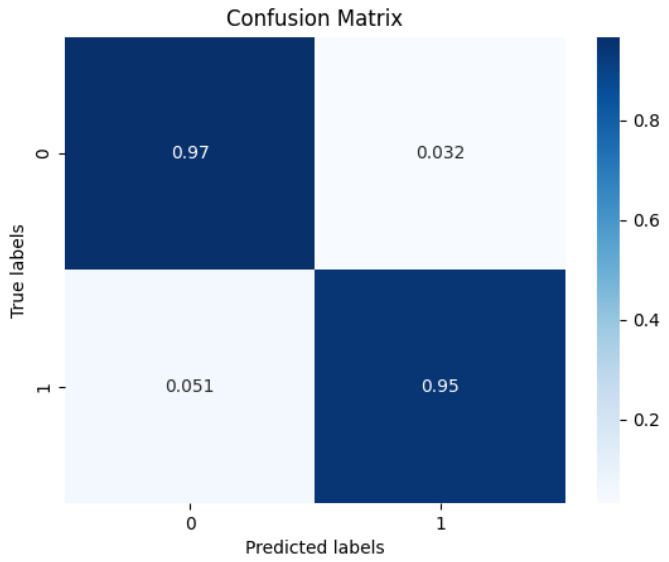
BACKUP

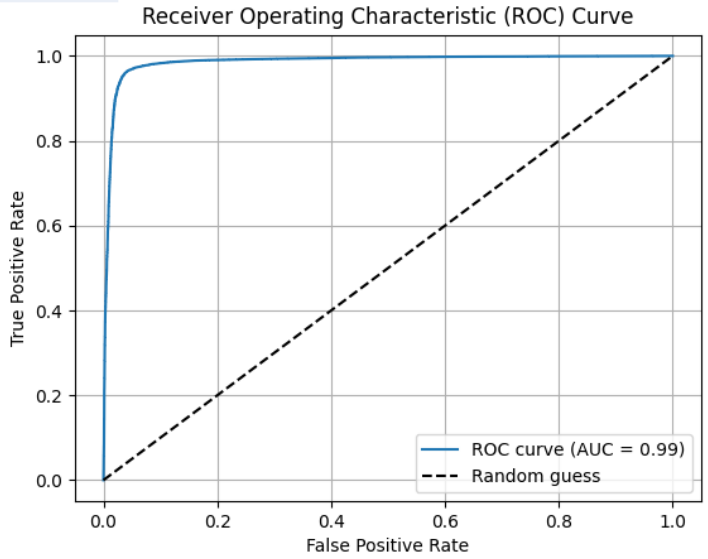
GAT algorithm

Graph Attention Networks (GATs) are a variant of Graph Neural Networks (GNNs) that leverage attention mechanisms for feature learning on graphs.
In standard GNNs, such as Graph Convolutional Networks (GCNs), the feature update of a node is typically the average of the features of its neighbors. This approach does not differentiate between the contributions of different neighbors.
GATs, on the other hand, assign an attention coefficient to each neighbor, indicating the importance of that neighbor's features for the feature update of the node. These coefficients are computed using a shared self-attention mechanism, which calculates an attention score for each pair of nodes. The scores are then normalized across each node's neighborhood using a SoftMax function.

CV 5 fold	accuracy	recall	precision
train data	0.941± 0.024	0.926 ± 0.033	0.9860 ± 0.0038
validation data	0.941 ± 0.024	0.926 ± 0.033	0.9859 ± 0.0038

600 epochs
2,6 million events
learning rate: 1e-4

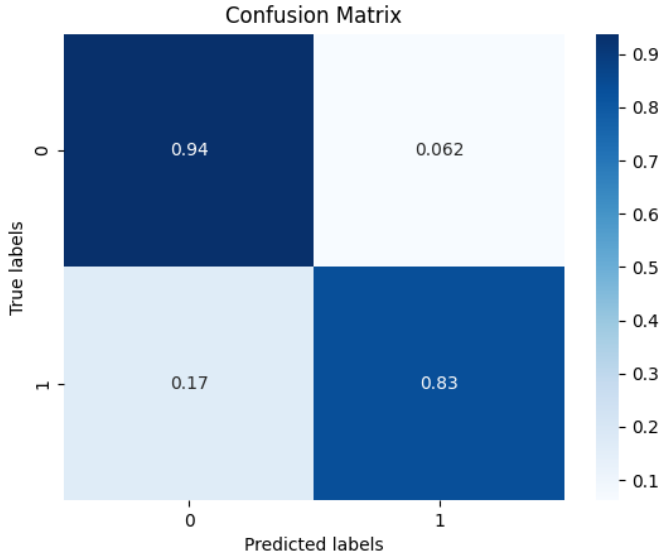


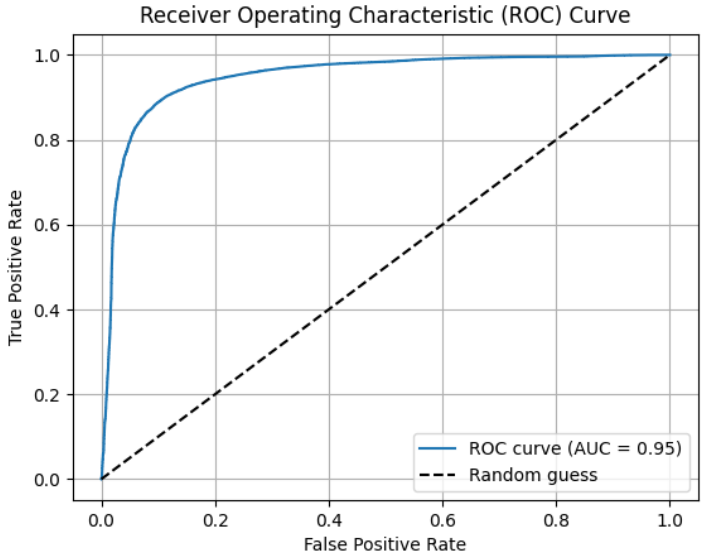


Performances on test data
Accuracy: 0.9557
Recall: 0.9493
Precision: 0.9849

GCN algorithm

The general idea of GCN is to apply convolution over a graph. Instead of having a 2-D array as input as in the classical CNN algorithm, GCN takes a graph as an input





Algorithm performances:
1500 epochs
2 million events
lr 5e-4
Accuracy: 0,8662
Recall: 0,8326
Precision: 0,9663