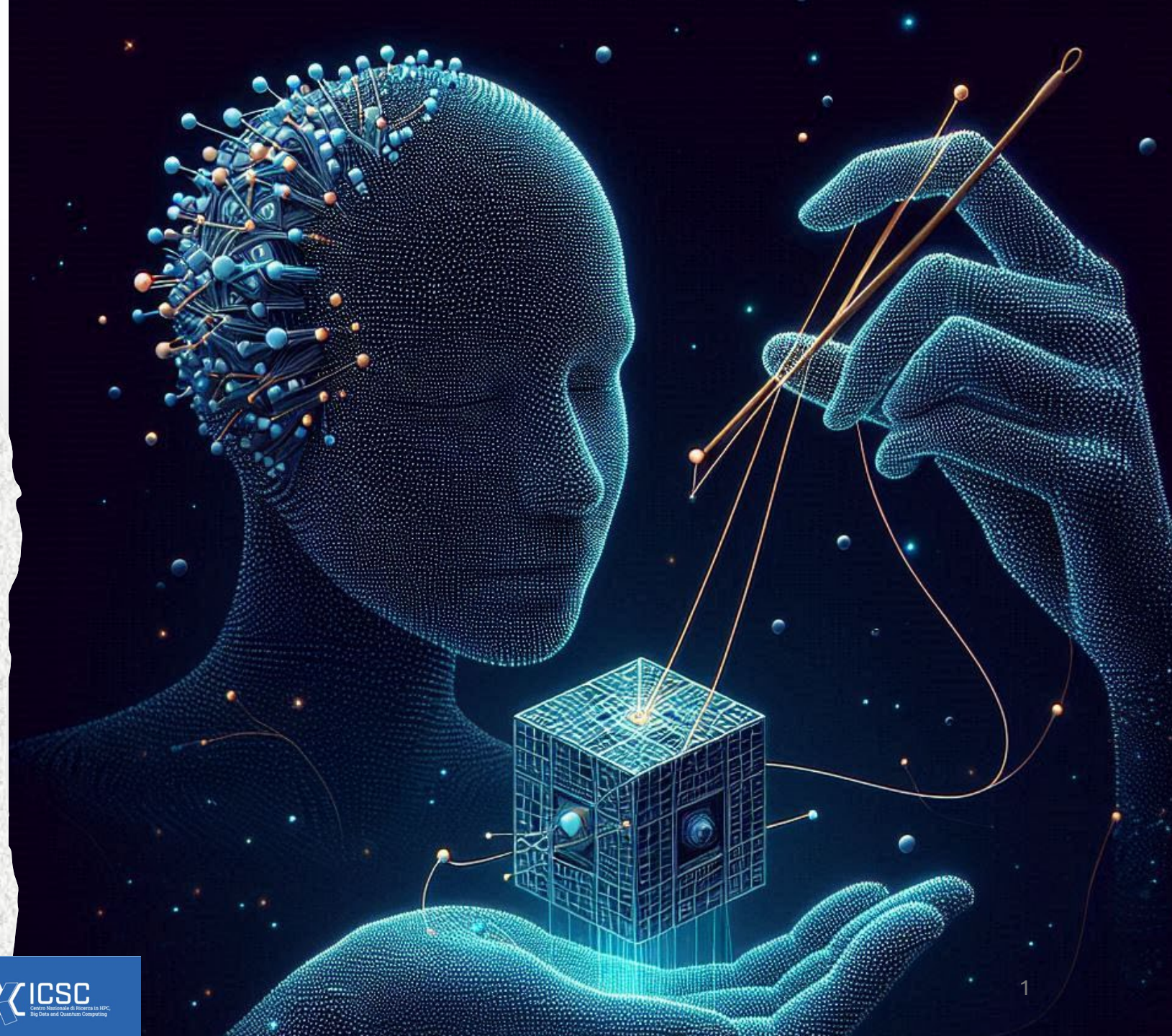
ADAPT Collaboration meeting 4-7 November 2025

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.

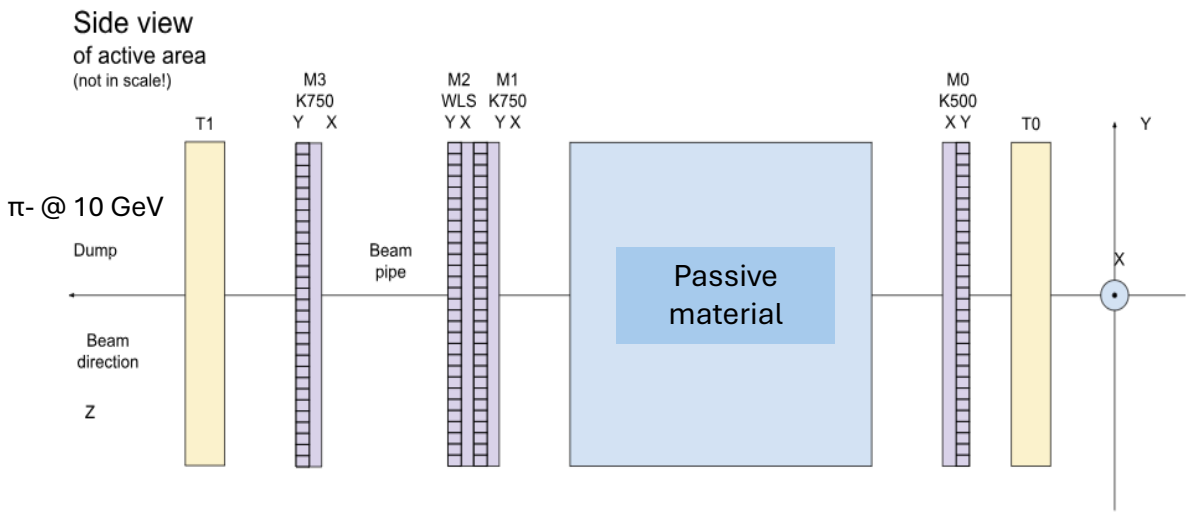
- Test beam dataset as test bench for the AI tracking algorithm
- GNN algorithm and the implementation on FPGA/GPU
- The application to the HERD experiment

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

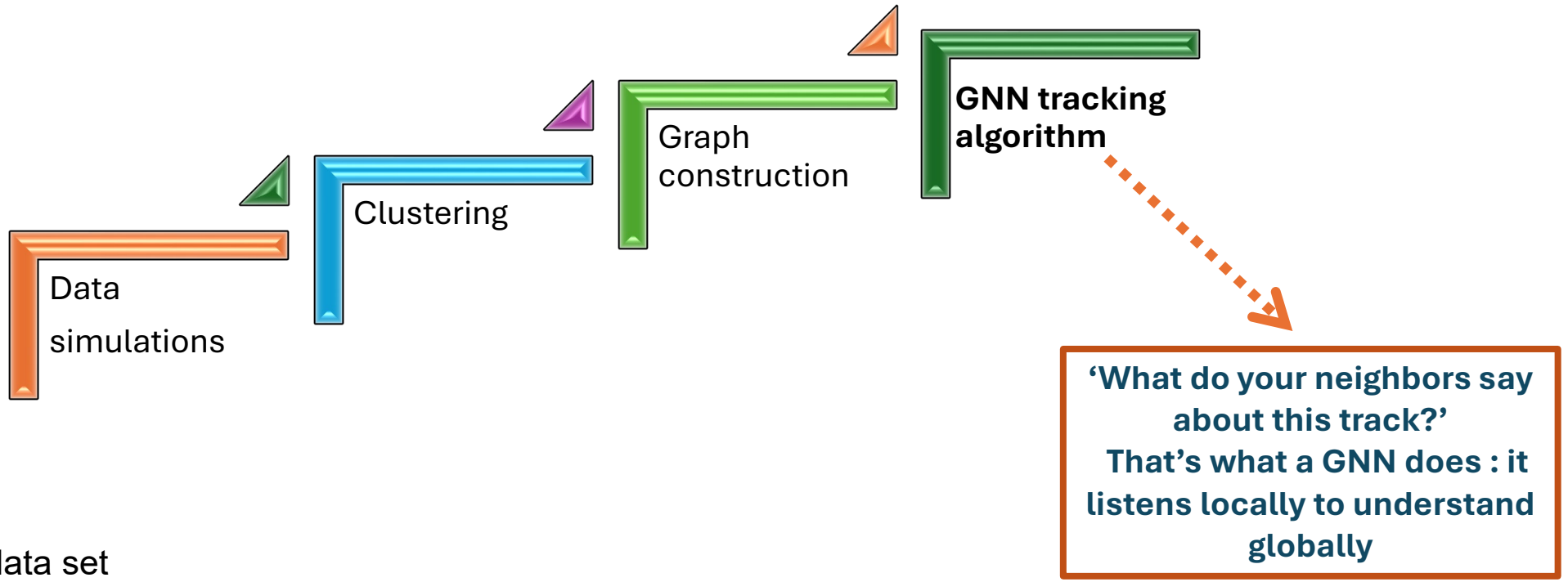
The set up has been simulated by using Geant4 toolkit.
A beam of π- of 10 GeV/c with inclined tracks of 0.5 deg has been simulated.
M0,M1,M2,M3 are fiber tracking layers:

- 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…

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)

Brief review of the GNN architecture

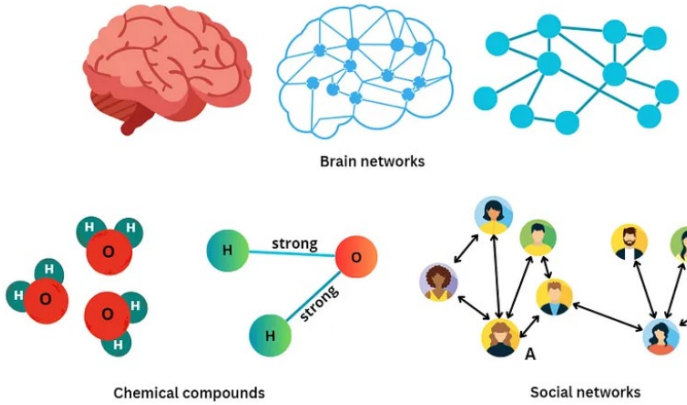
A graph represents the relations (edges or links) between a collection of entities (nodes).
Graph Neural Networks (GNNs) are a class of deep learning models that are designed to operate on graph-structured data. They have shown remarkable success in tasks such as node classification, link prediction,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

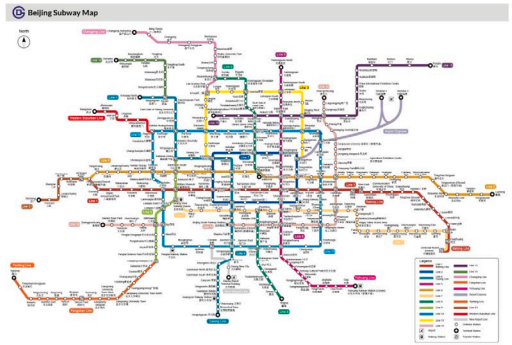
A GNN consists of a number of layers, each of which updates the representation of each node based on its local neighborhood.

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

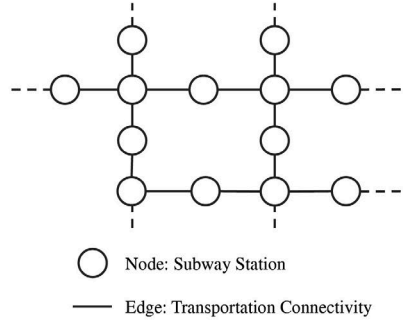
The key component of a GNN layer is the aggregation function, which takes as input the representations of a node's neighbors and produces a new representation for the node.



- *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)

Preprocessing phase: from row data to a graph structure

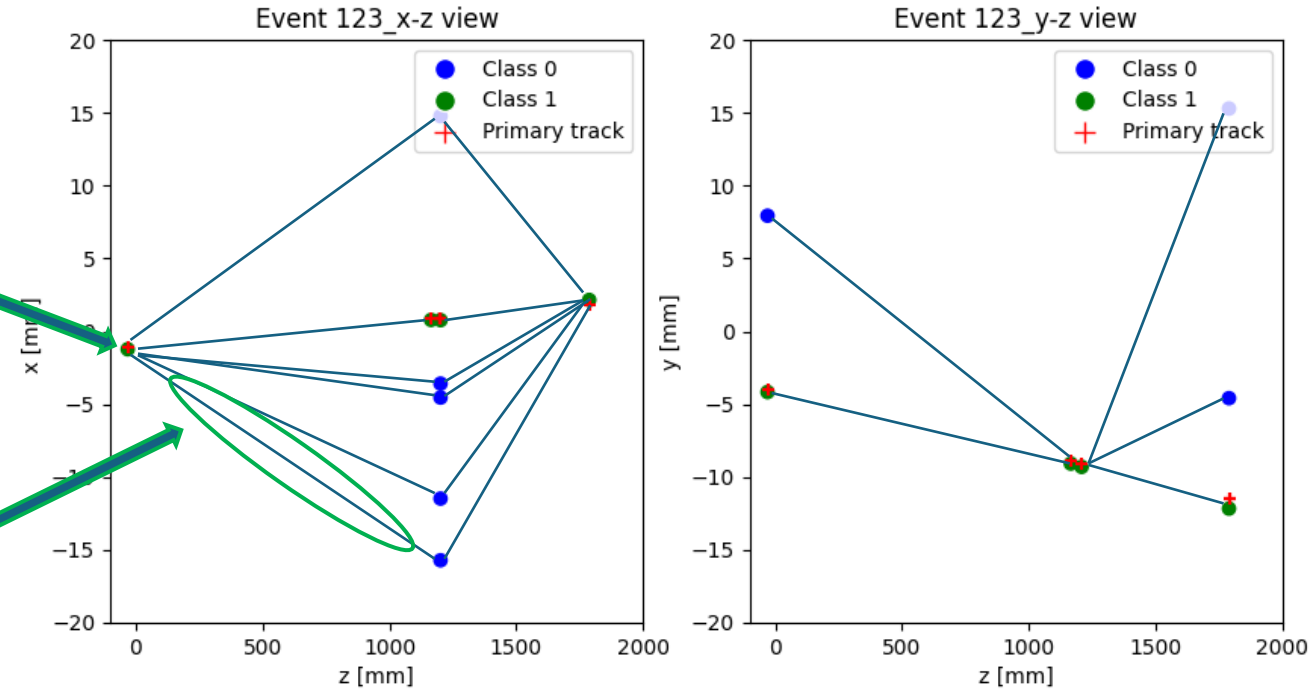
The hits inside tracking layers are clustered by applying a traditional clustering algorithm, where neighboring silicon strips with activated signals are grouped together and the barycenter of charge is calculated.

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



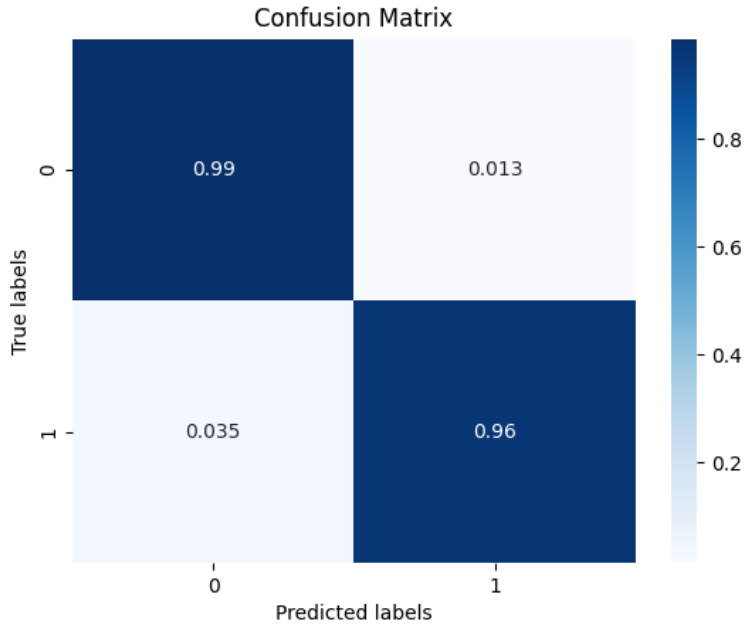
SageConv algorithm: the GNN architecture

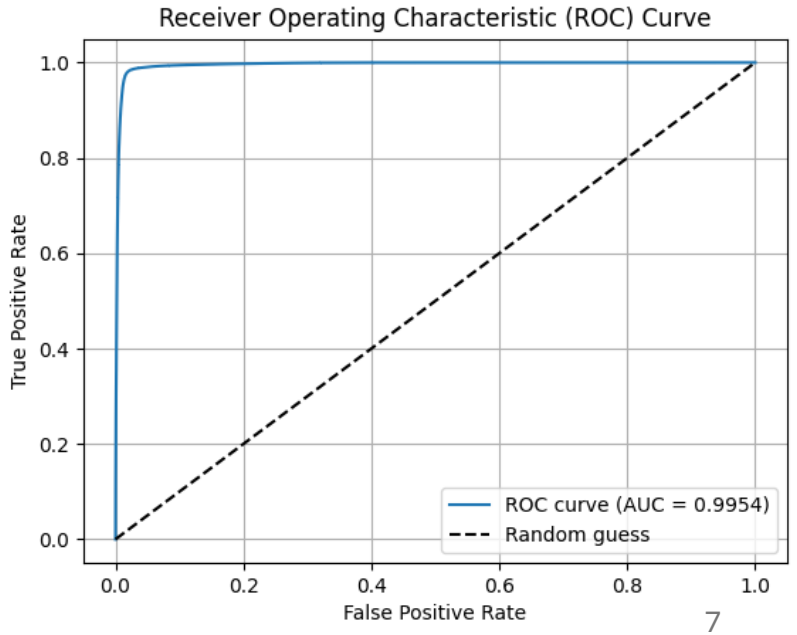
The SageConv architecture is a variant of GNN architecture.
The aggregation function takes into account the degrees of the nodes in the neighborhood.
SageConv uses the average of the representations of the neighbors, normalized by the degree of each neighbor, as the aggregate representation.
This allows 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

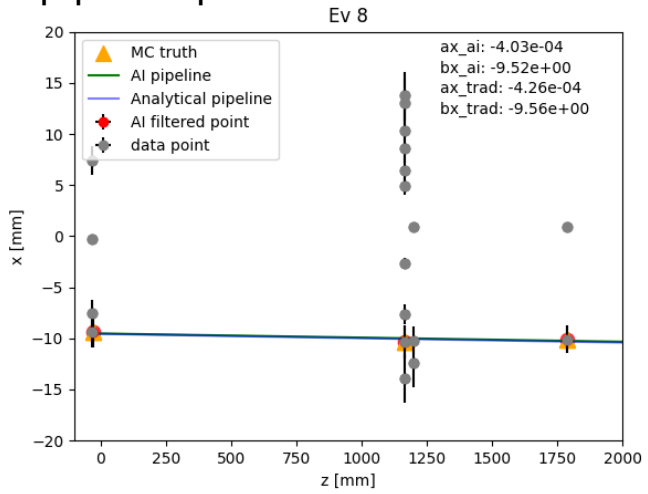


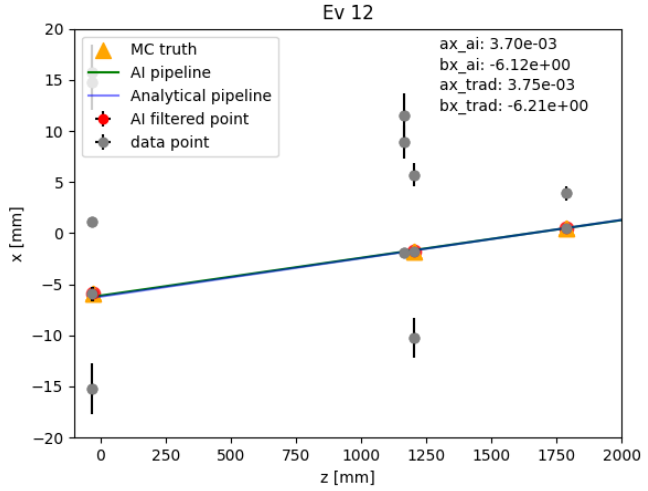


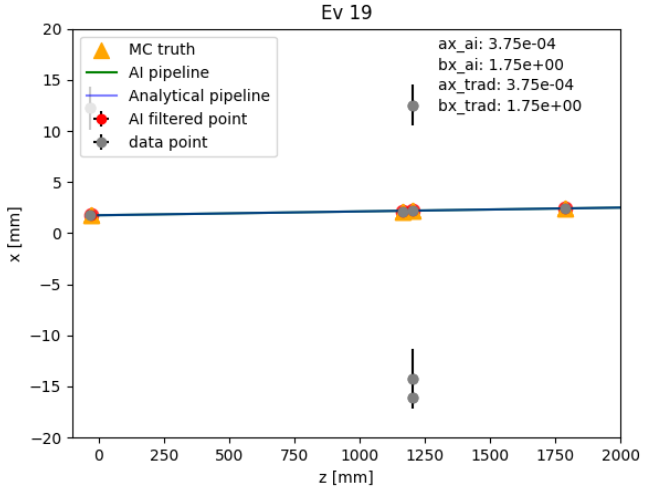
Event display: the tracks identification

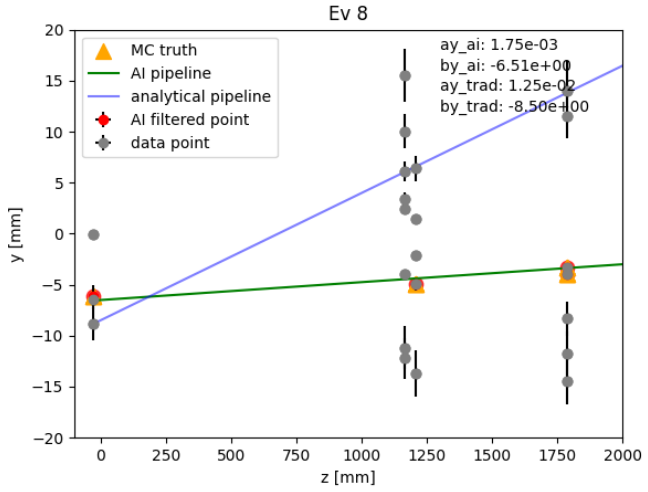
The traditional tracking pipeline minimizes the chi-square between the clusters inside each events, then fit the track with a linear function.

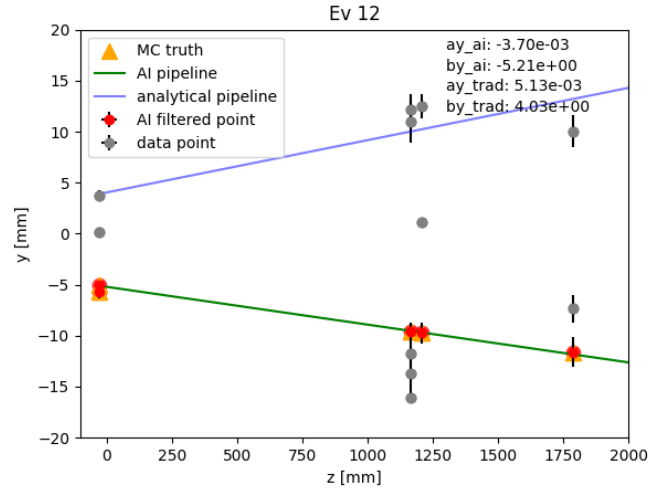
The AI pipeline performs the selection of good clusters and fit the track with the same linear function.

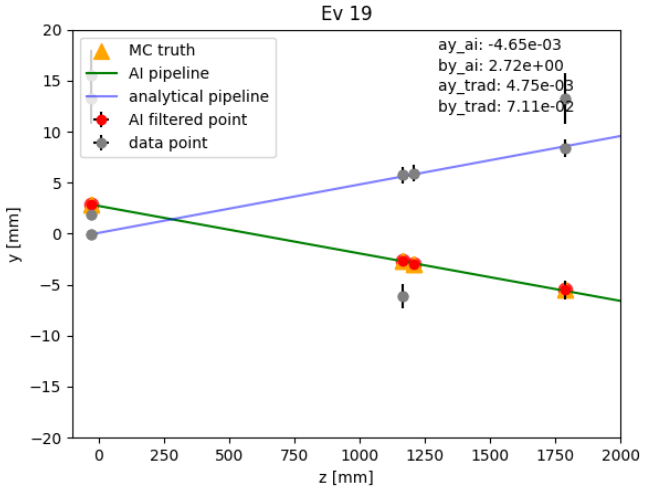




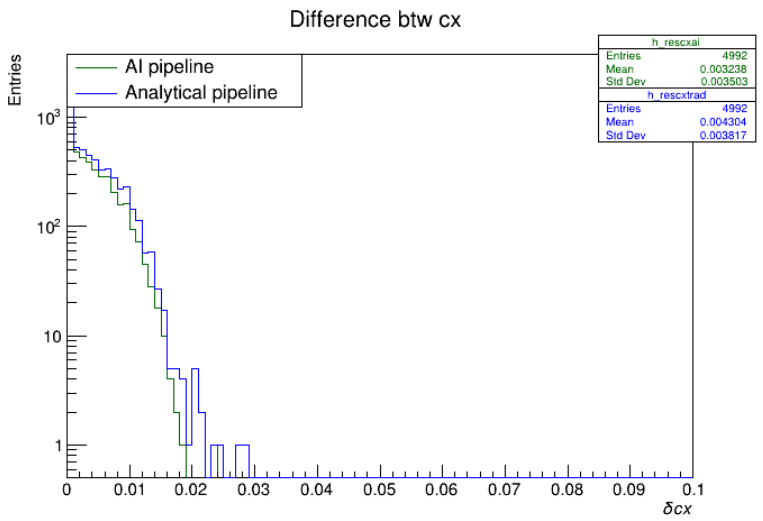


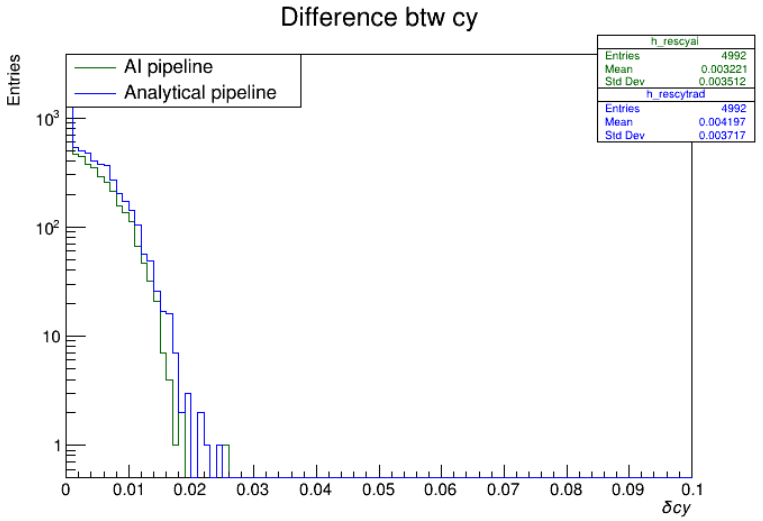


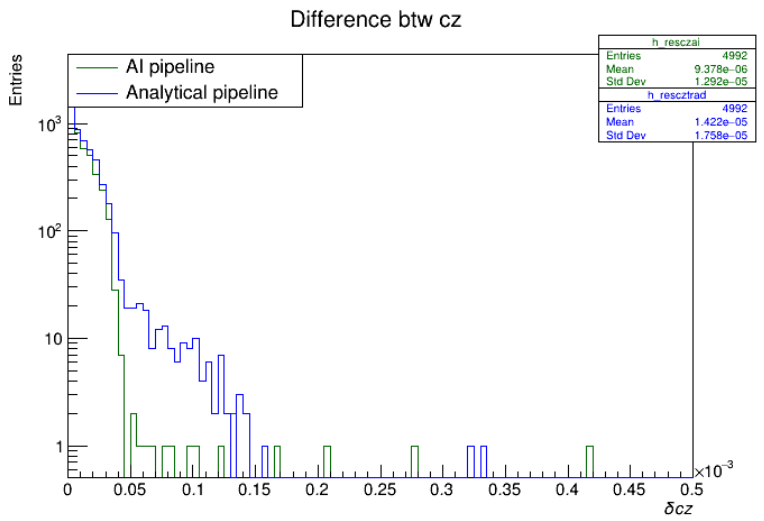




The SageConv algorithm: comparison with traditional pipeline of the director cosines







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

Spartan project: Challenges in Full GNN Implementation on FPGA

Architectural Constraints

- **Non-pipelined Design**: The structure of the SAGEConv-based GNN is not easily pipelined, making it unsuitable for direct implementation on pure FPGA without significant performance penalties.
- **Resource Usage**: The complexity of Graph Neural Networks requires a large number of logic gates and memory blocks, which scales poorly with the number of nodes and channels in the graph

Performance Bottlenecks

- **Sequential Computation**

  Unlike CNNs, GNNs require message passing and aggregation steps that are inherently sequential, limiting the effectiveness of parallel execution on FPGA.

Toolchain Limitations

- Existing tools like Vitis AI and hls4ml are not fully optimized for implementing complex GNN layers, especially when dynamic graph structures and attention mechanisms are involved

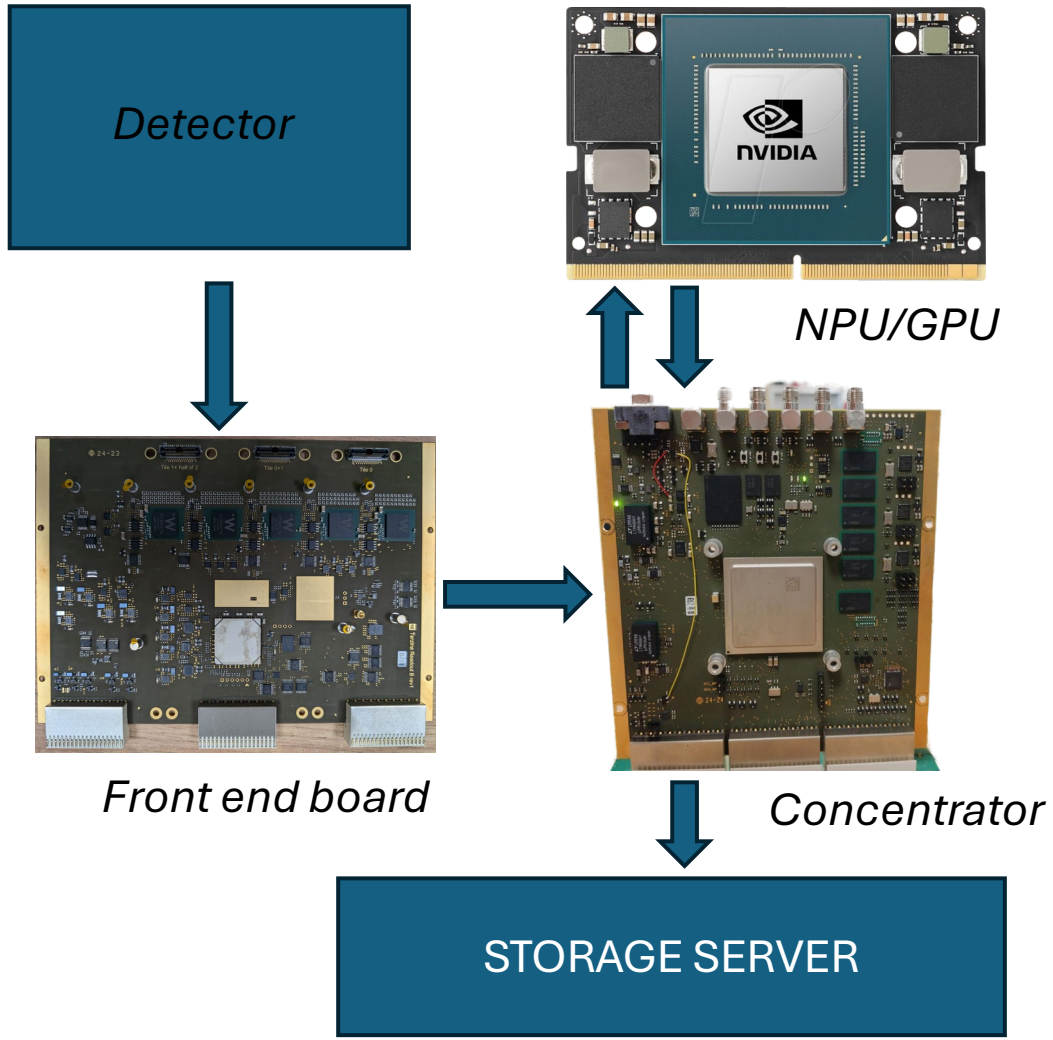
Due to the limitations above, a hybrid architecture was chosen: FPGA (Zynq Ultrascale+) for preprocessing, and NPU (Jetson Orin Nano) for inference, achieving high throughput and low power consumption.

Spartan project: Hardware Implementation Strategy

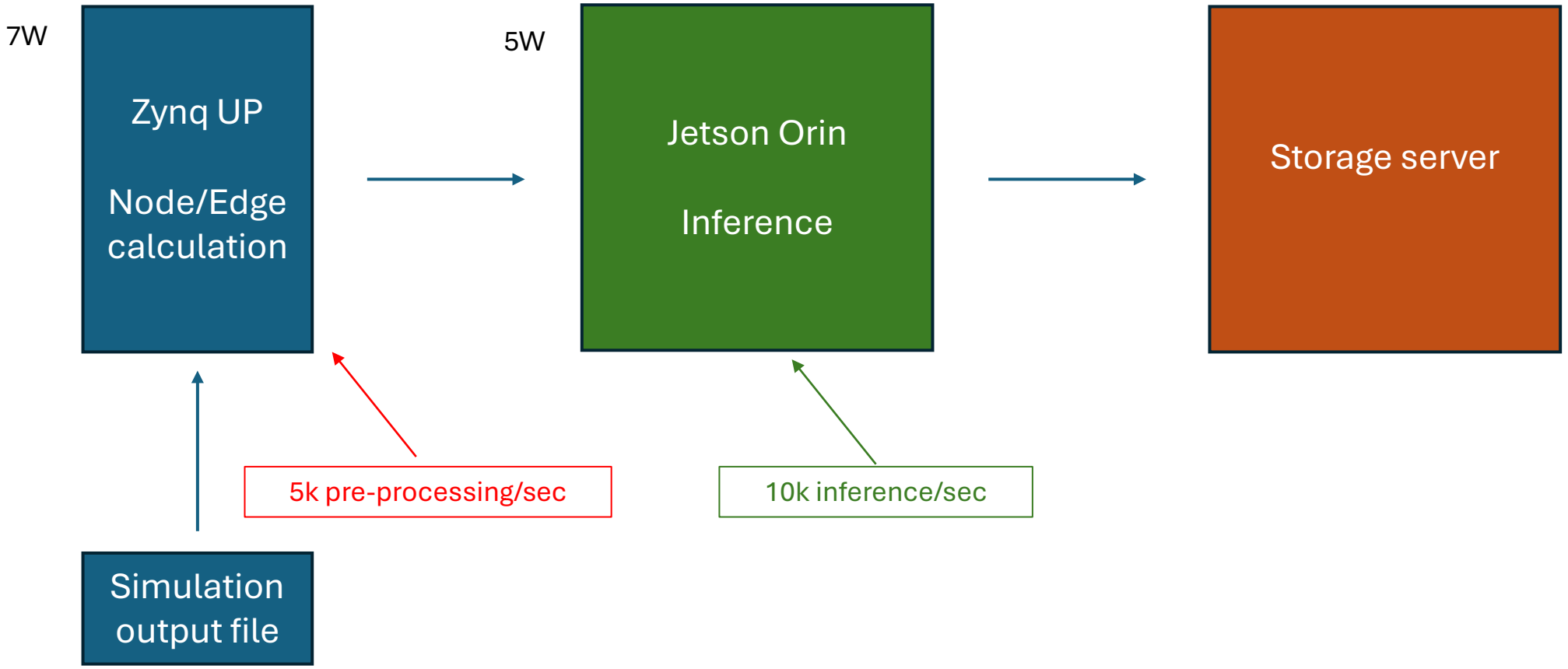
System Architecture

- **Hybrid Solution**: FPGA + NPU
  A mixed architecture combines a Xilinx Zynq Ultrascale+ FPGA with an NVIDIA Jetson Orin Nano to optimize the execution of the GNN model.

- **Functional Roles**:
  - **FPGA (Zynq)** handles:
    - Front-end data acquisition
    - Preprocessing (e.g., cluster building, filtering)
    - Real-time data preparation and formatting
  - **NPU (Jetson Orin Nano)** performs:
    - GNN inference (SageConv model)
    - Track reconstruction
    - Data packaging for acquisition PC



Spartan project: Inference system: Performance



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 2027.
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.

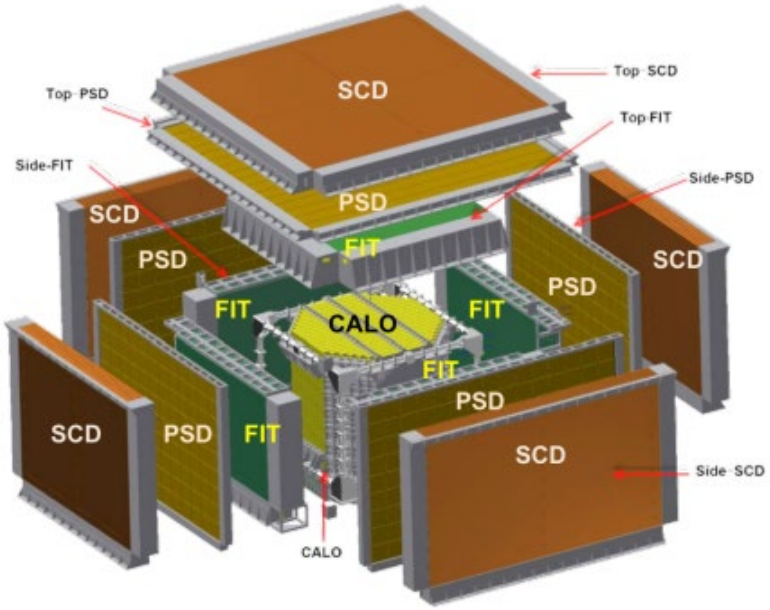
Fiber Tracker (FIT):

- 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.
It consists proton tracks simulated within a power-law energy spectrum E^{-1}, spanning an energy range from:

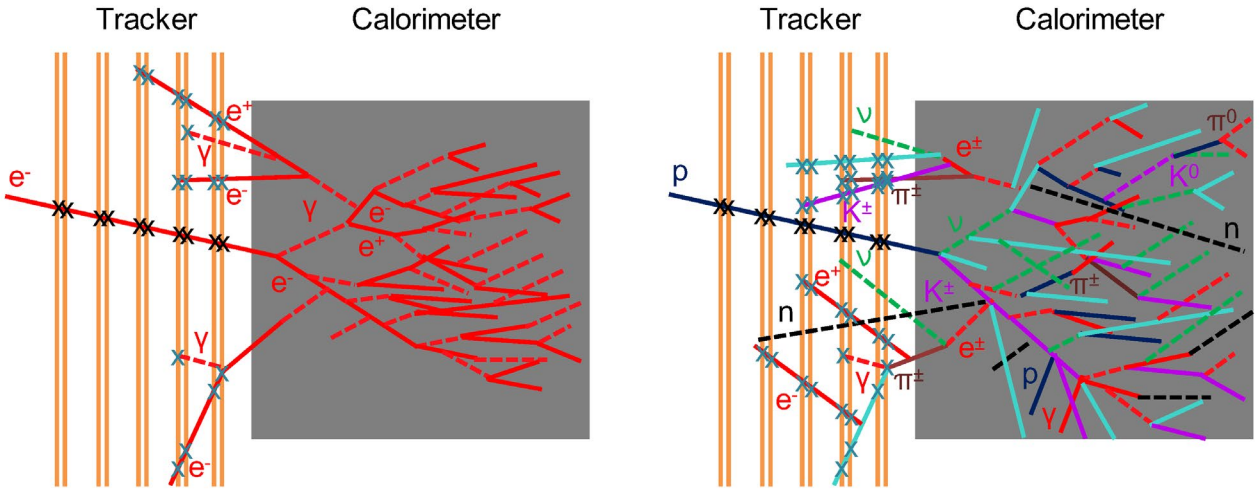
- 1 GeV - 100 GeV (66293 tracks)
- 100 GeV - 10 TeV (204229 tracks)

Tracks are distributed within a spherical region surrounding the HERD detector

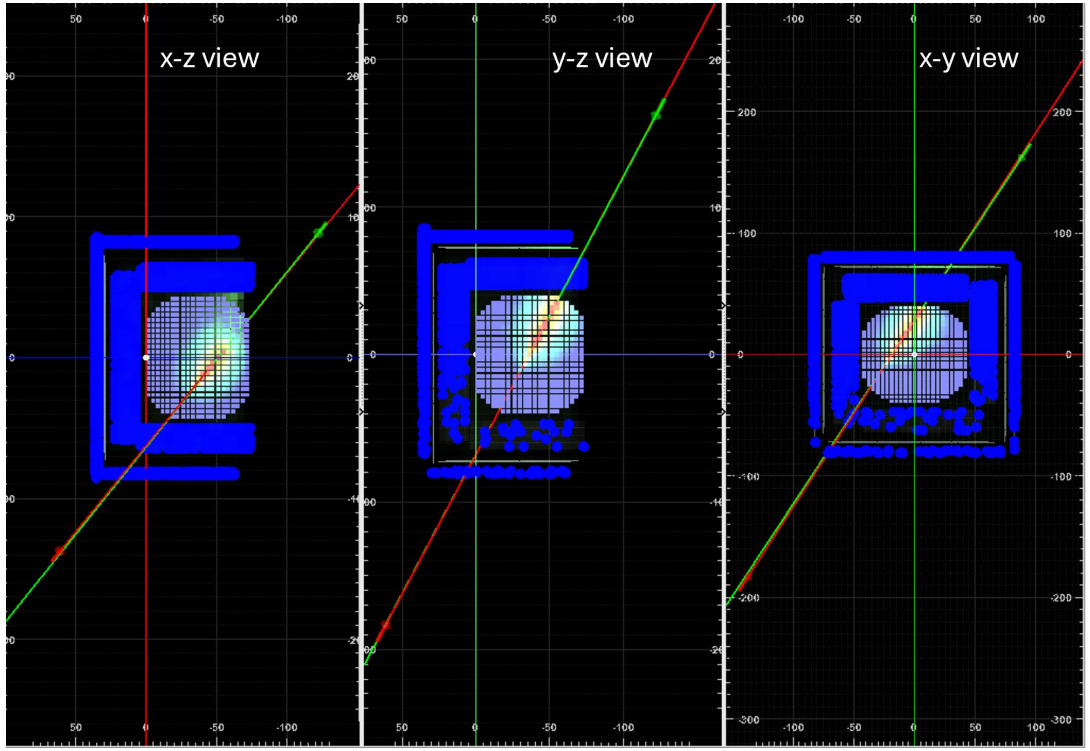


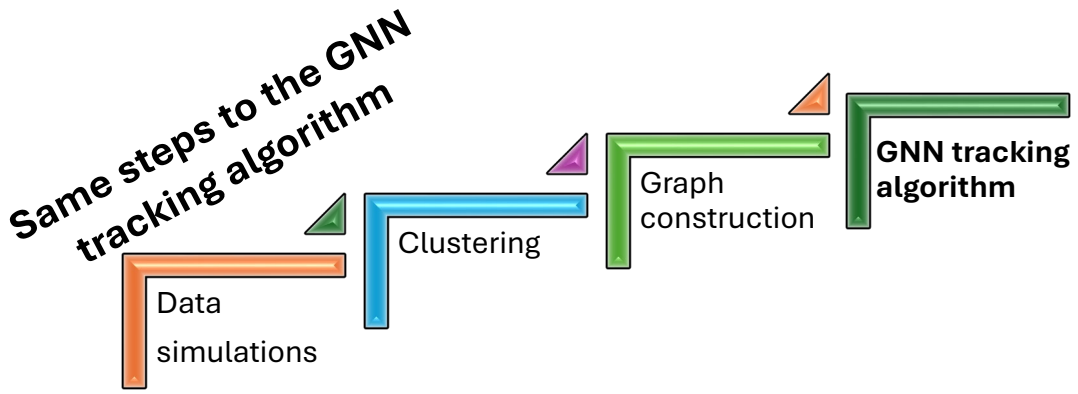
Proton tracks: backscattering and primary tracks

Protons' tracks reconstruction is a complicated challenge due to the high number of backscattering tracks!



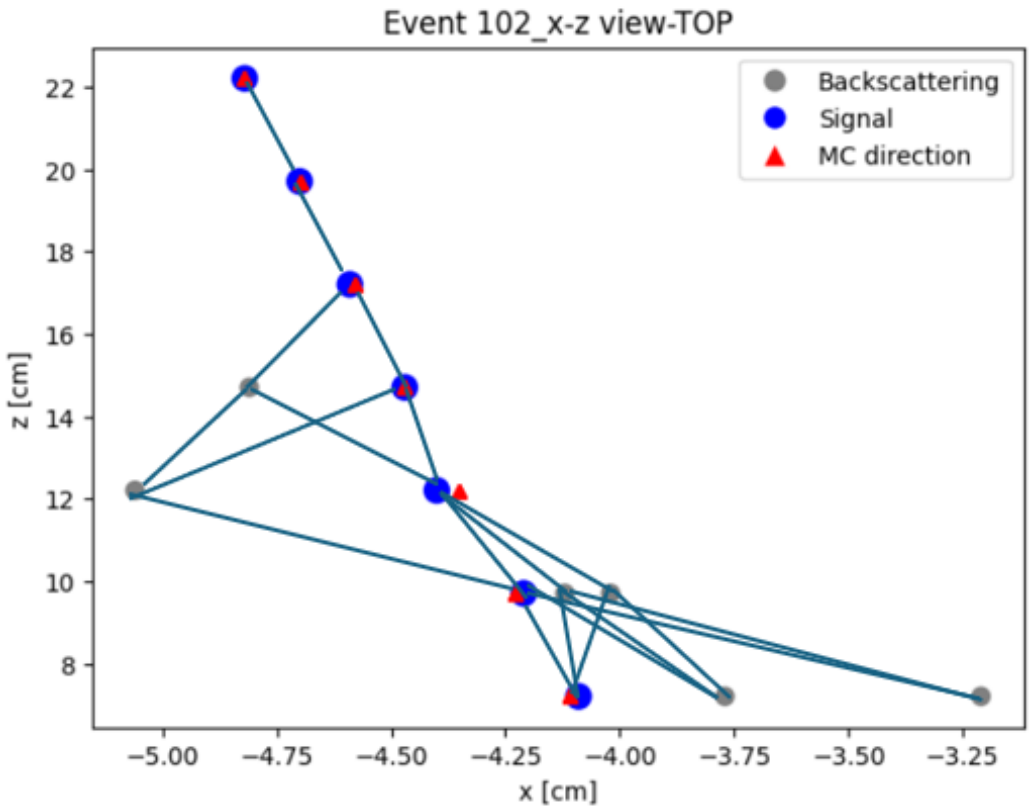
Duranti, M.; et al. Advantages and Requirements in Time Resolving Tracking for Astroparticle 337Experiments in Space. Instruments 2021, 5, 20. 338





Preprocessing phase for protons' tracks

- Tracks passing through the fiber tracker produce energy deposits identified as "hits."
- Hits are grouped using a traditional clustering algorithm that aggregates neighboring activated silicon strips.
- The charge barycenter of each cluster is then computed.
- Taking as reference the reconstructed shower calorimeter axis, clusters lying outside a cylindrical region of 3.5 cm radius are discarded.
- Data are structured in a graph format, where each node corresponds to a cluster and edges represent inter-layer connections between clusters.
- Only tracks traversing the top side of the FIT have been considered.



The Interaction Network: a variant of Graph Neural Network

The Interaction network architecture developed is inspired by the one developed by the HEP.TrkX project (https://heptrkx.github.io/).
It consists of:

- **Feature encoder stage** to initializes the graph representation
- two core modules: an **edge network** and a **node network**.

The **edge network** computes edge weights using the features of the start and end nodes;
The **node network** updates each node's features based on the aggregated edge-weighted features of neighboring nodes.
Both the EdgeNetwork and NodeNetwork are implemented as Multi-Layer Perceptrons.

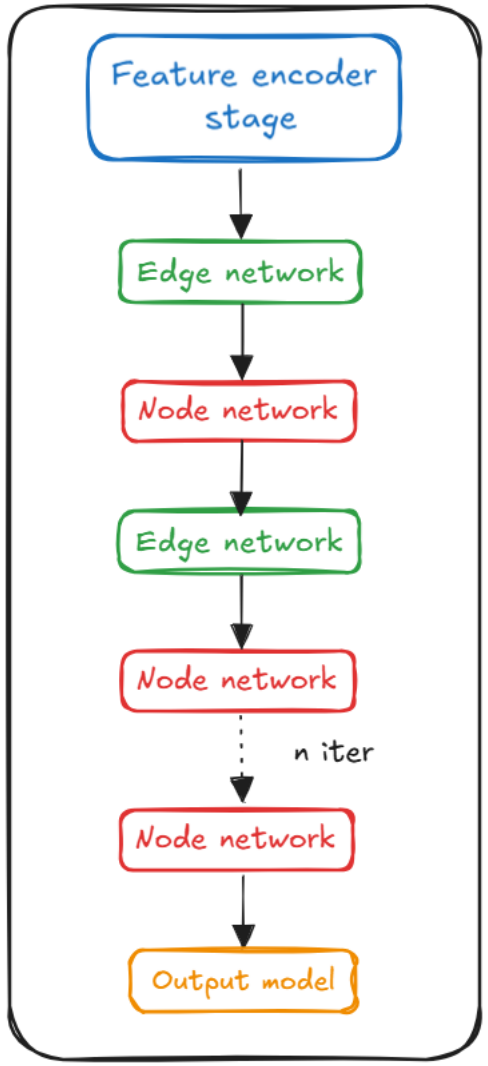
Two version of the **feature encoder stage** have been tested.

- the first version applies two independent layer fully connected networks to transform the raw input features of the nodes and edges,
- the second version uses a Multi-Head Feature Attention mechanism. Instead of directly projecting input features through fully connected layers, this encoder employs a multi-head attention module to learn more expressive representations of the input node features.

The node and edge network are applied recursively for a fixed number of iterations N.

At each iteration, edge weights are recomputed by the edge network, and node features are updated by the node network using the aggregated information from neighboring nodes.

After a fixed number of iterations, the final node representations are passed to the **output module**, which consists of a **linear classifier with dropout regularization.**



The loss function

Strong class imbalance: backscattering clusters dominate over the signal ones in high energy range

Solution: Focal Loss

$$L = -\alpha_t (1 - p_t)^{\gamma} \log(p_t)$$

p_t is the model's predicted probability for the correct class:

p_t low → the model is uncertain, the scaling factor $(1 - p_t)^{\gamma}$ will be large, making the loss higher for this sample.

γ is the focusing parameter

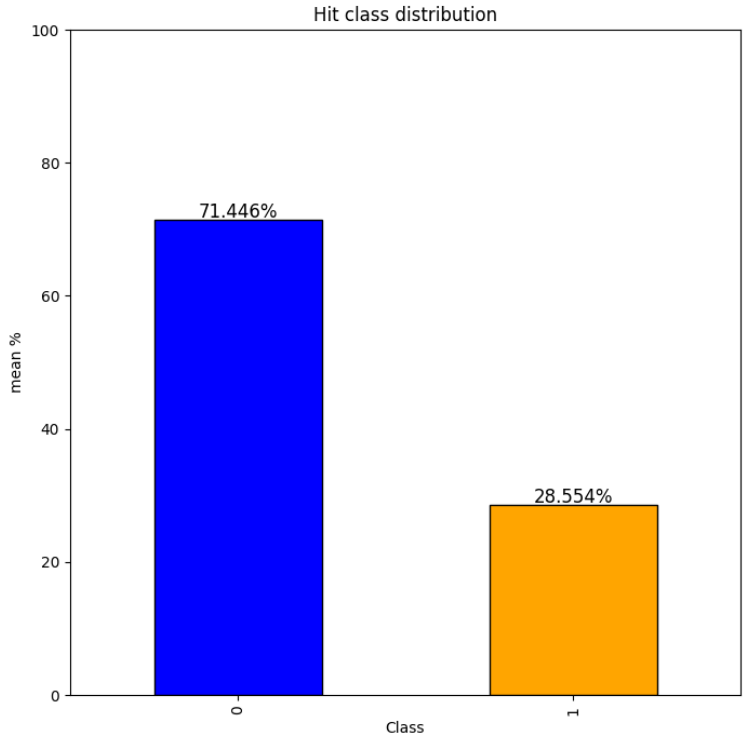
- $\gamma = 0$ → standard cross-entropy
- Larger γ → the contribution of well-classified examples, with high predicted probability is progressively down-weighted, while misclassified or hard examples receive greater emphasis

α_t class-balancing factor

- Higher α_t for minority class
- Lower α_t for majority class

In this work, α_t computed dynamically per batch to:

- Better handling of imbalance
- Stronger focus on difficult samples
- Improved learning efficiency & robustness



The training procedure

Ray Tune framework for hyperparameters search!

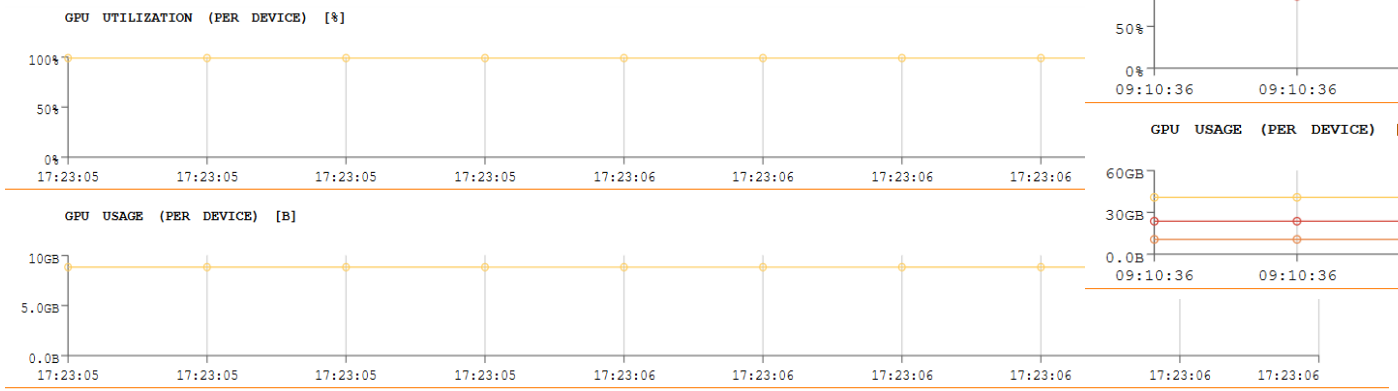
The hyperparameter which have been tuned are:

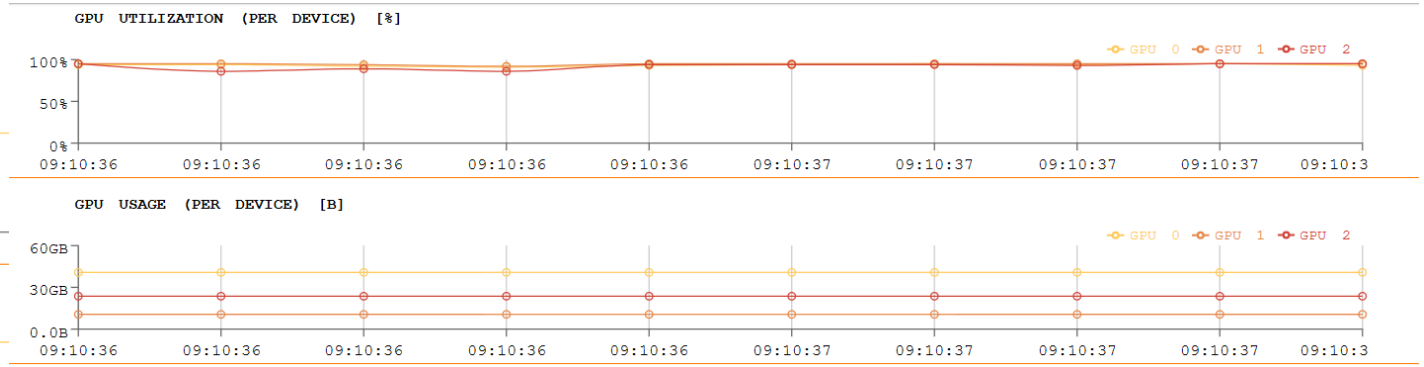
- the number of iteration of edge and node network
- the activation function
- the gamma parameter for the focal loss
- the number of layer inside node and edge network
- the hidden dimension of each layer
- the number of heads for the Multi-Head Feature Attention approach,
- the learning rate
- the batch size

The training process integrates a learning rate scheduler: it reduces the learning rate by a factor of 0.5 whenever the loss does not improve over a 5 of epochs.
The best model has been chosen as the one that maximize the f1 score among 50 trials.

Two distinct hardware setups were employed to test the model's scalability and efficiency.

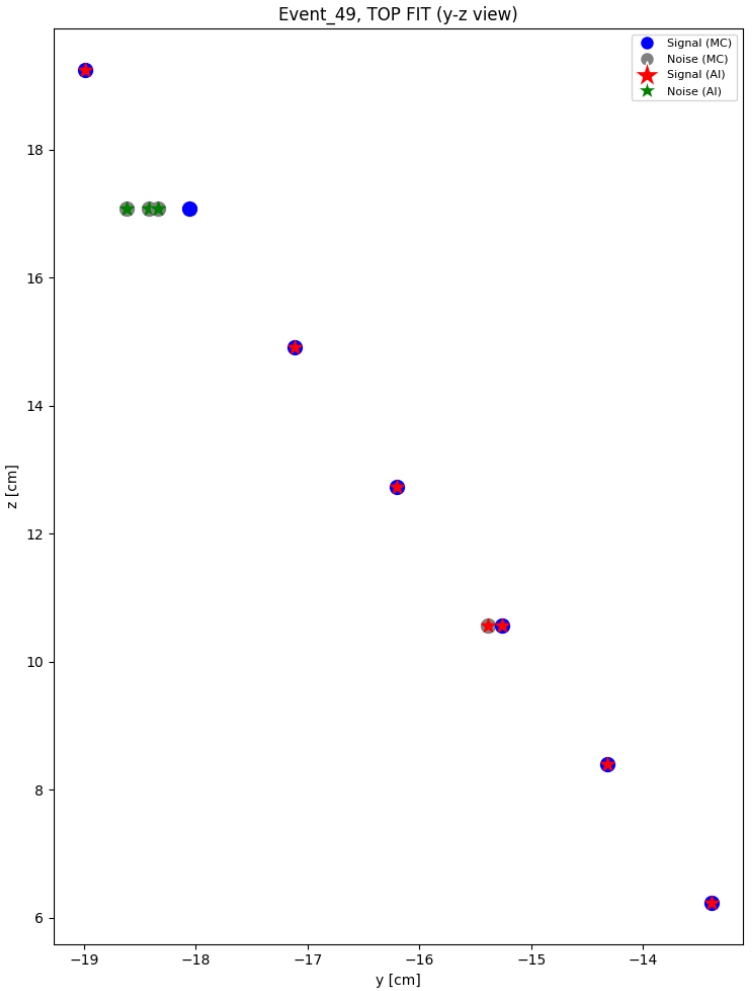
- distributed training across three NVIDIA A100 GPUs, each with 40 GB of memory
- a single NVIDIA H100 GPU with 94GB of memory

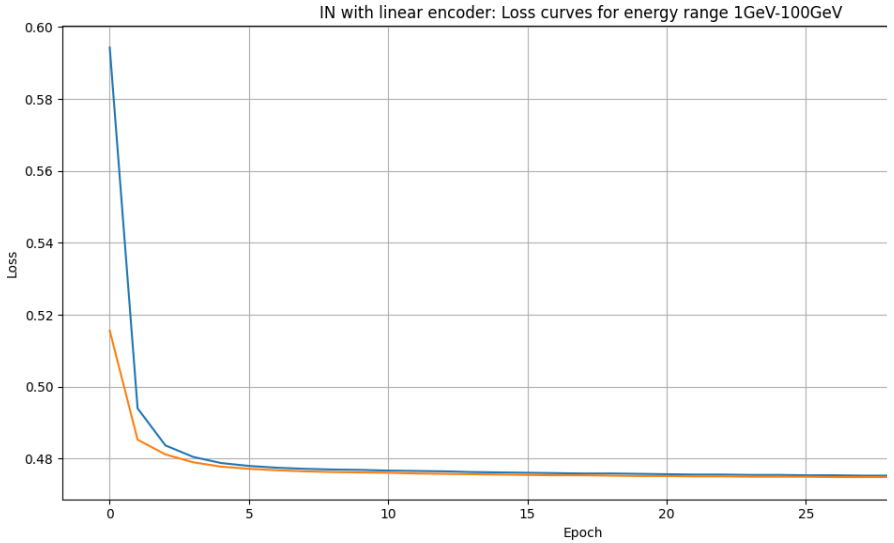


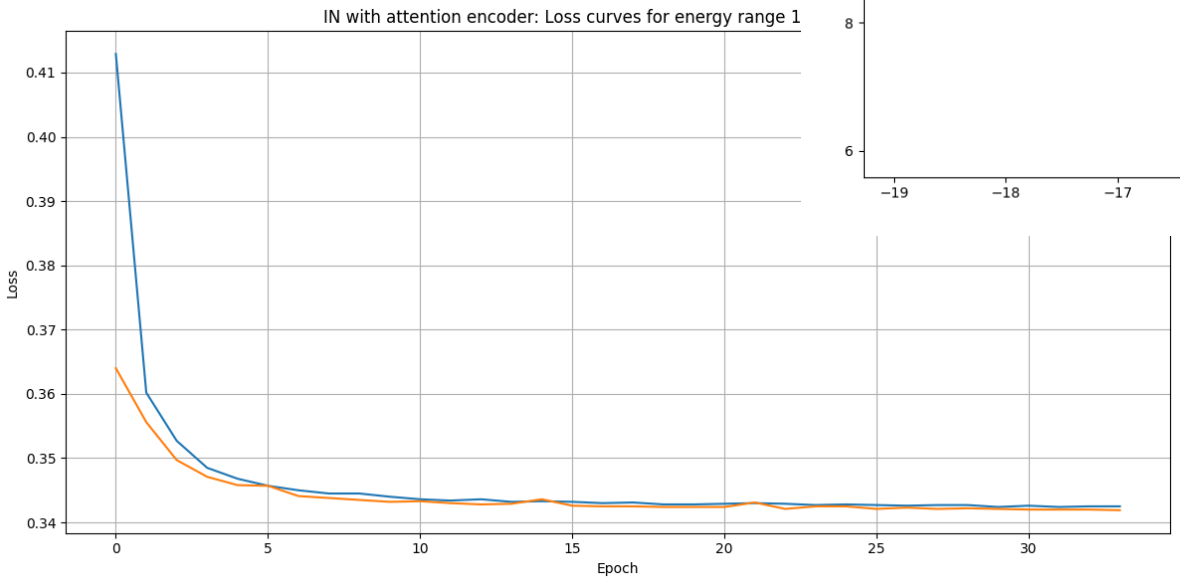


PRELIMINARY RESULTS

Evaluation metrics for the 1 GeV – 100 GeV energy range		
Metrics	**IN with linear**	**IN with attention encoder**
Accuracy	80.14	80.11
Precision	80.20	81.15
Recall	86.43	86.67
F1-score	80.19	80.18

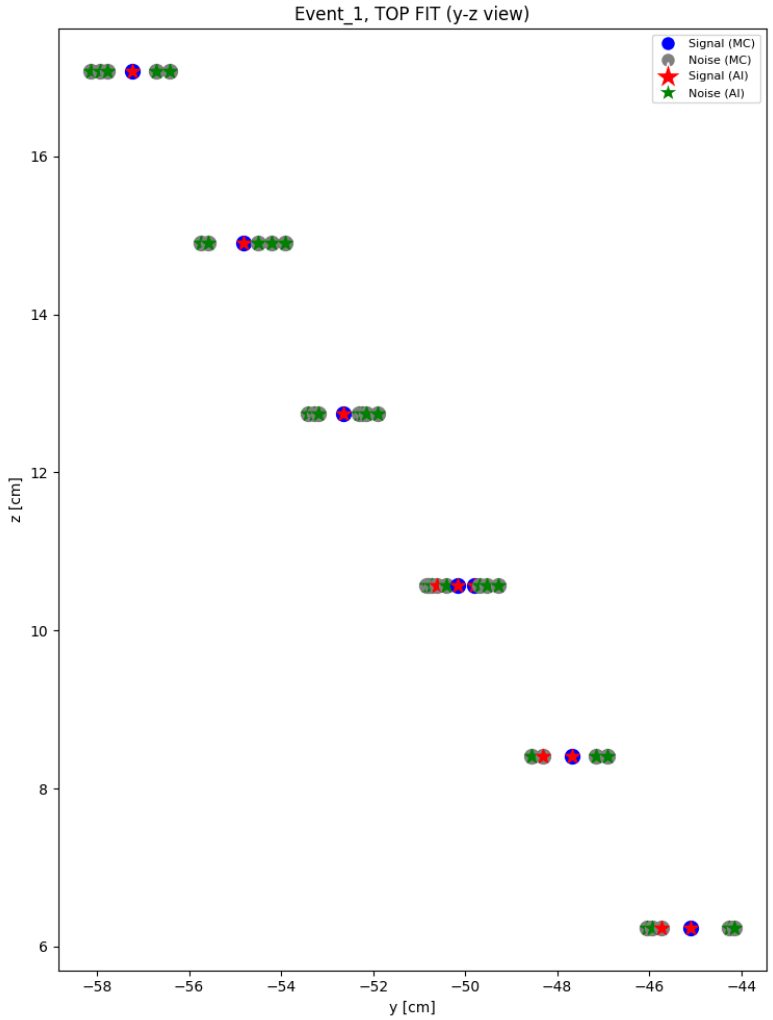


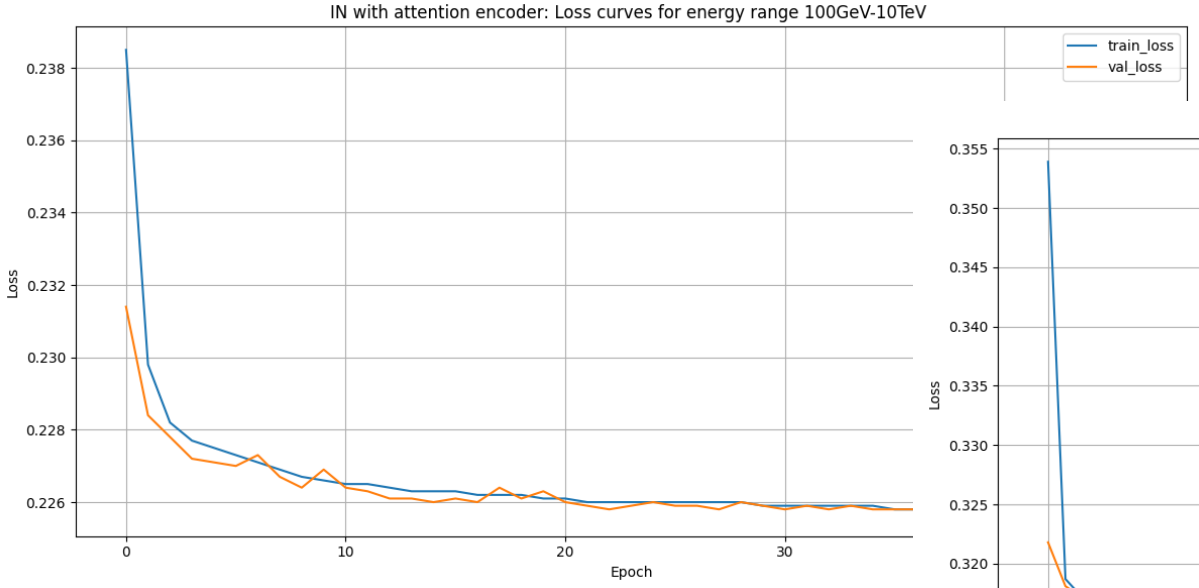


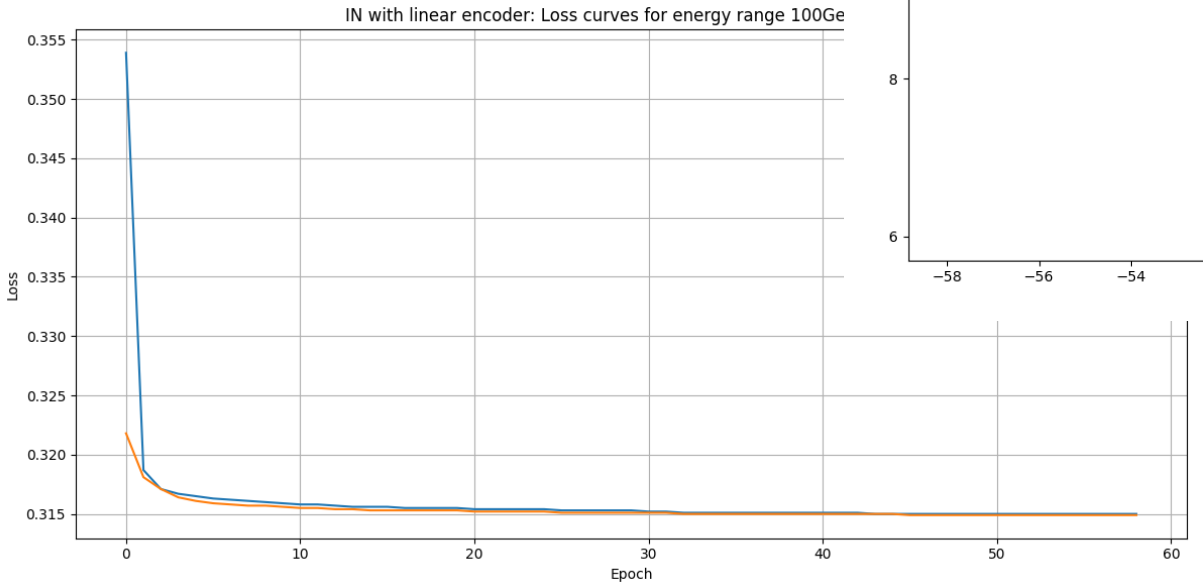


PRELIMINARY RESULTS

Evaluation metrics for the 100 GeV – 10 TeV energy range		
Metrics	**IN with linear encoder**	**IN with attention encoder**
Accuracy	81.85	82.11
Precision	77.68	79.65
Recall	71.31	71.29
F1-score	74.11	73.97







Conclusions

At low energies (1–100 GeV), both models achieve competitive and balanced performance. The attention encoder shows the highest recall (86.67 %), demonstrating a stronger ability to correctly identify signal hits and higher precision (81.15%), reflecting a better suppression of false positives.

In the higher energy range (100 GeV-10 TeV), both models maintain stable performance, with the attention encoder achieving slightly higher accuracy (82.11% vs. 81.85%) and precision (79.65% vs. 77.68%).
Recall values remain nearly identical (around 71%), indicating that both approaches struggle to recover all true signal hits in the presence of the intense backscattering environment.

Good!
…. but we can do better…
…but …

- The simulation framework used to generate the dataset (HERDSoftware) is **currently no longer maintained**:
  - the clustering procedure is not optimized, and the labeling of signal and backscattering clusters lacks precision.
  - there is no any kind of traditional tracking algorithm for comparison.

- The current approach also stops at node-level classification, i.e. the pattern recognition stage, without yet including the final track fitting step (e.g., linear regression for trajectory extraction).

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

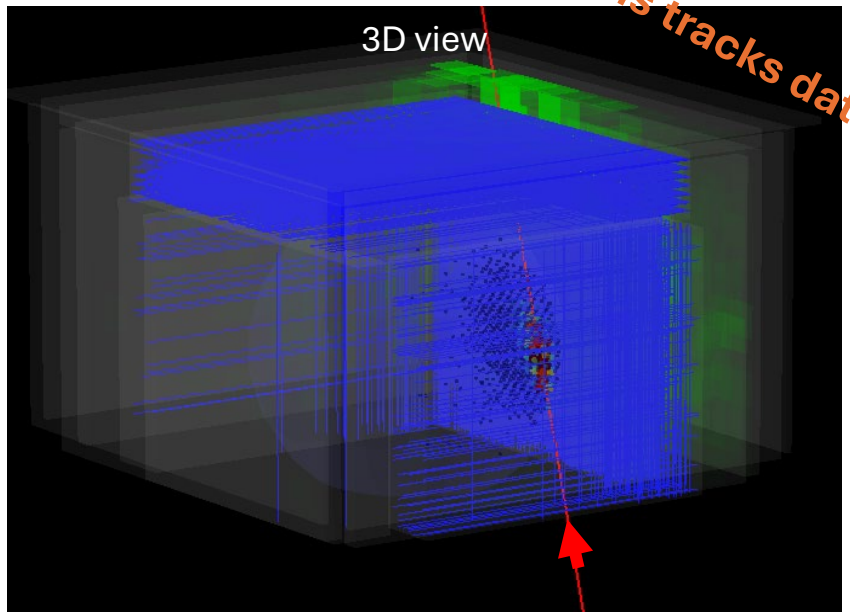
Preprocessing phase: prepare data for the GNN—electron tracks

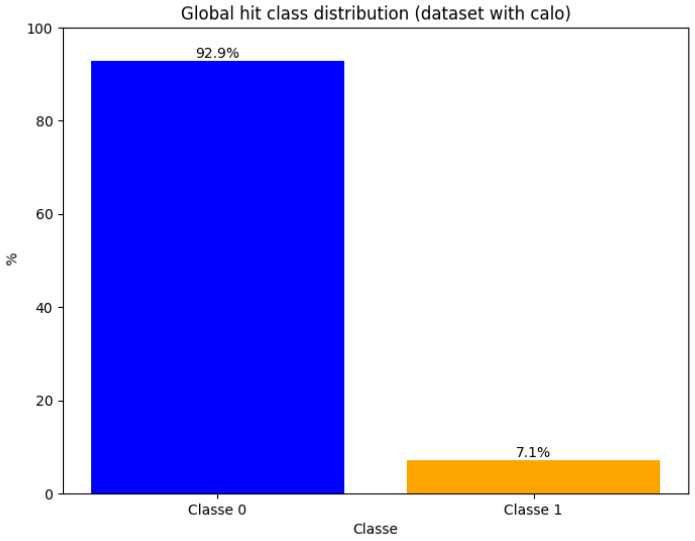
Data are highly imbalanced, since there are many bacskattering tracks originating from the calorimeter, which interfere with the correct identification of the primary particle's trajectory .

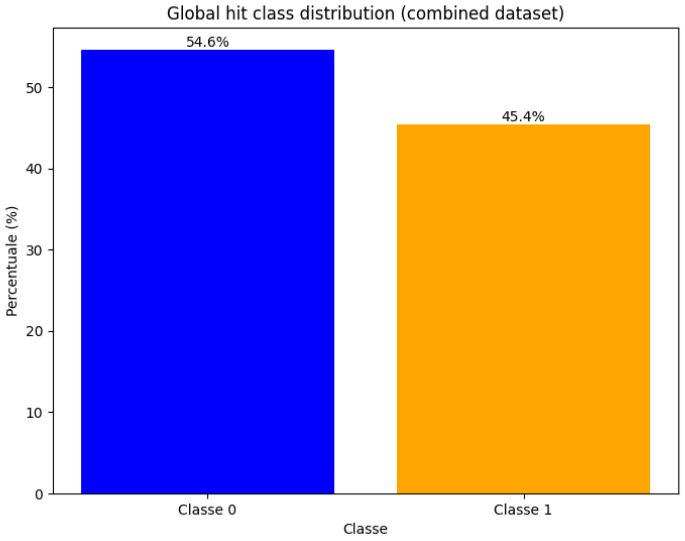
To mitigate the imbalance:

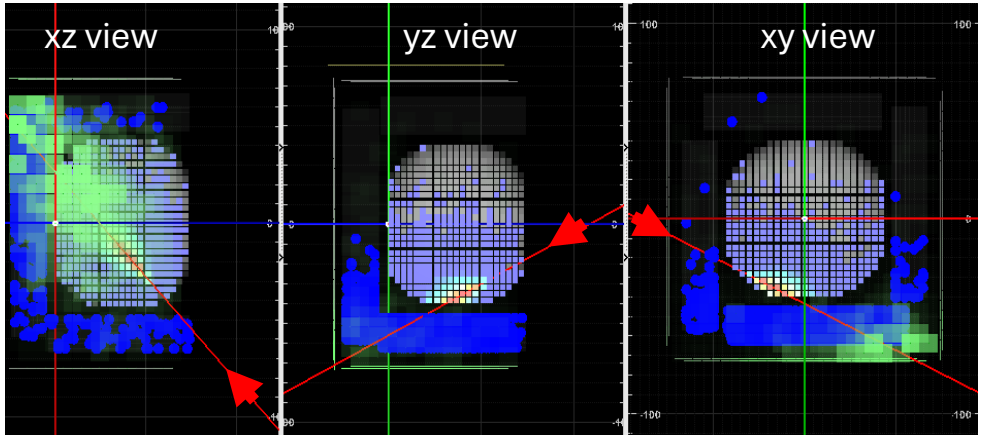
1. Clustering algorithm
2. Cut of noise clusters “far away” from the primary track
3. Adding simulated events without calorimeter

Electrons tracks dataset









The GNN algorithm: AI hardware
Tools for GNN training algorithm

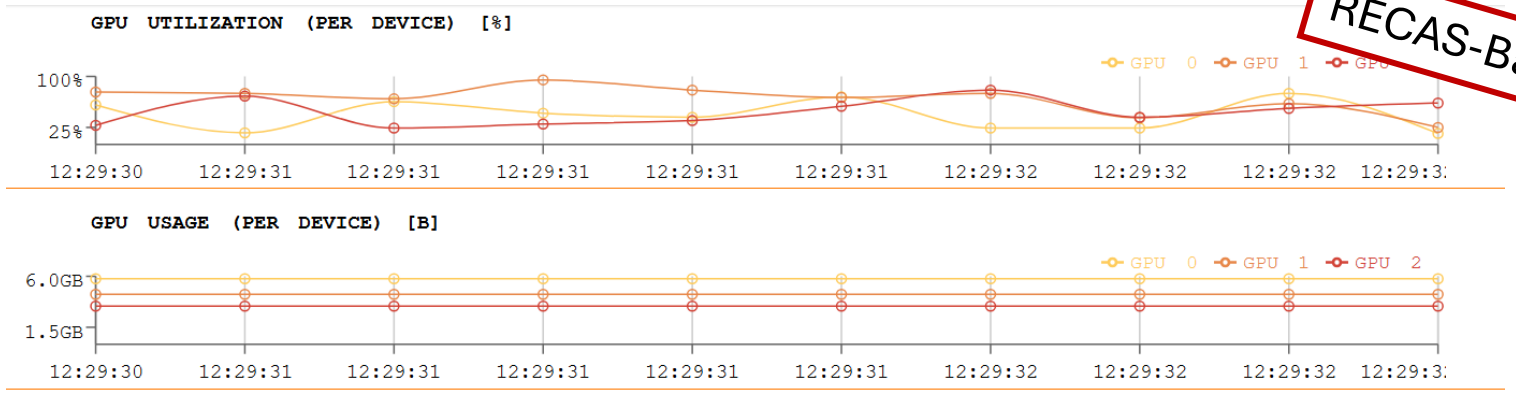
- SageConv architecture
- 18 layers
- Mean aggregation function
- 128 hidden size
- Adam optimizer
- Binary cross entropy loss function

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

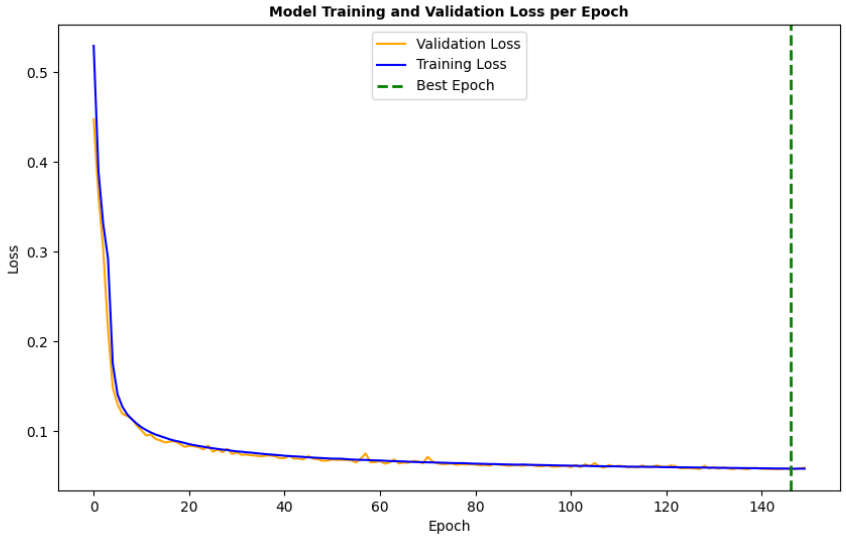
RECAS-Bari

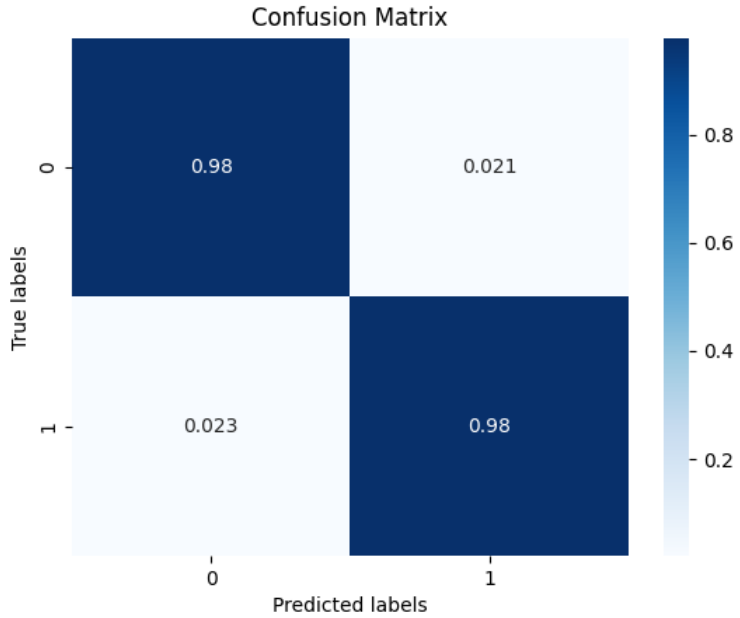


Leonardo-HUB

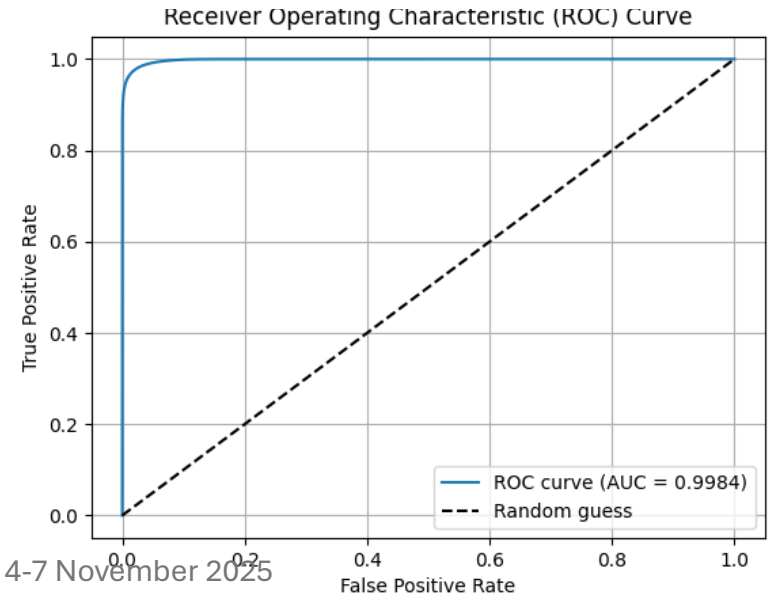
```
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 530.30.02              Driver Version: 530.30.02    CUDA Version: 12.
|-------------------------------+----------------------+----------------------+
| GPU  Name                 Persistence-M| Bus-Id        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      |
|=============================================================================|
+-----------------------------------------------------------------------------+
```

Main Results: GNN algorithm evaluation

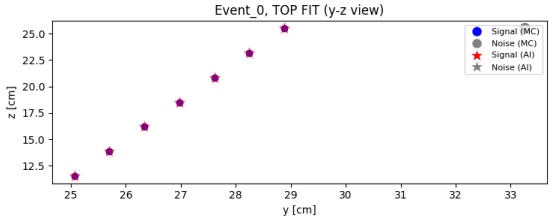


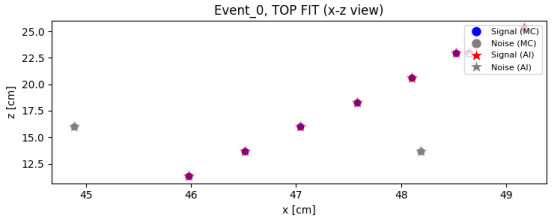


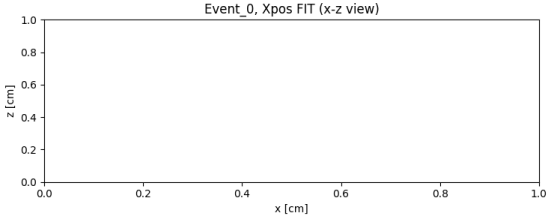
Metrics	Values
Accuracy	97.80%
Recall	97.65%
Precision	97.81%
F1-score	97.73%
ROC AUC	99.84%

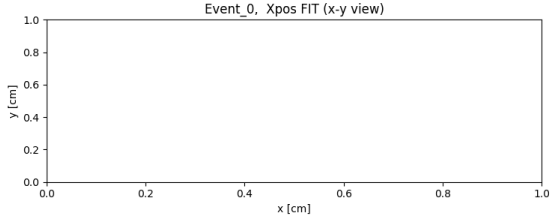


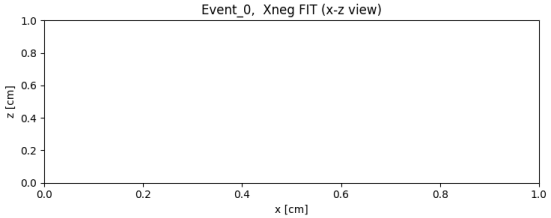
Main Results: events display

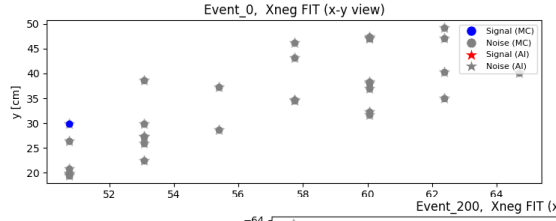


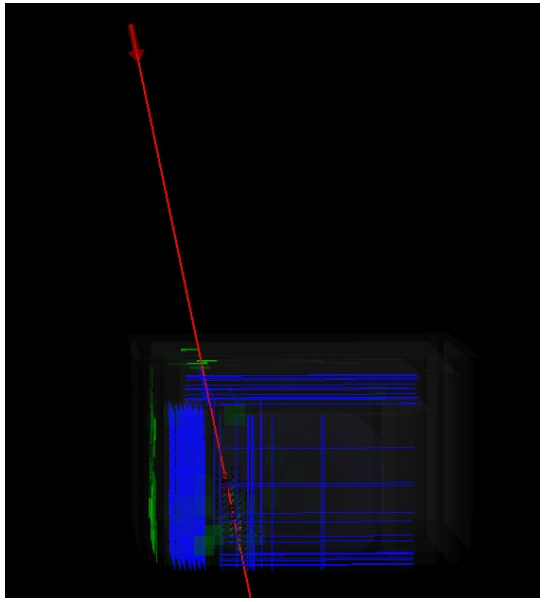




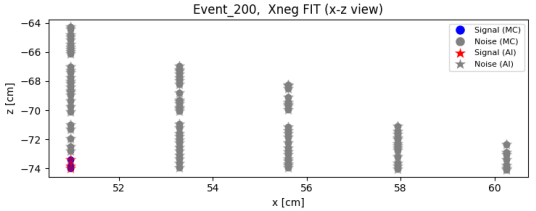


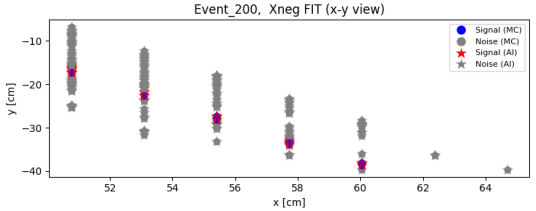


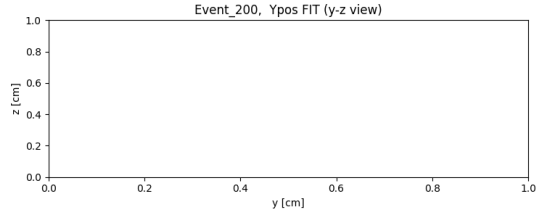


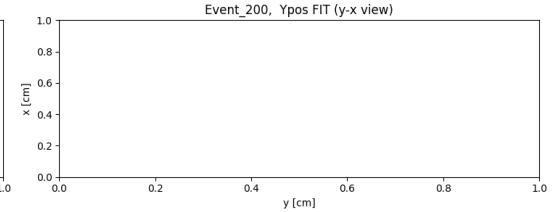


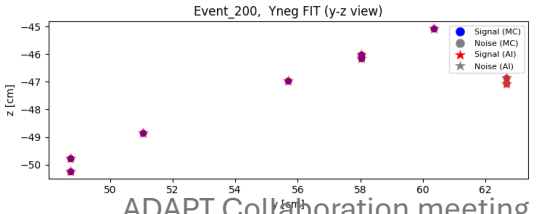
Paper submitted at PDP Conference!!!
https://ieeexplore.ieee.org/xp
l/conhome/10974743/proceeding

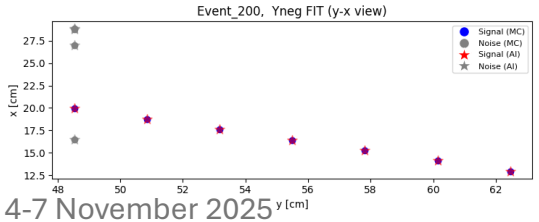


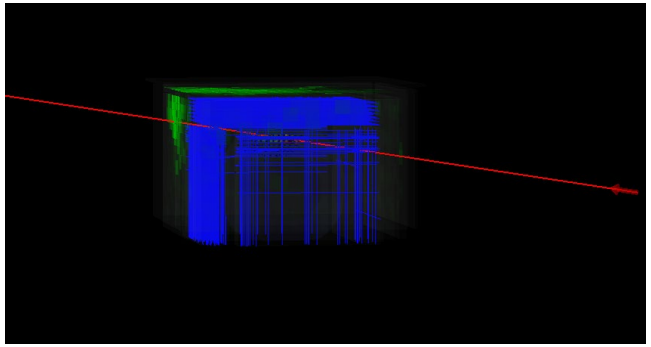




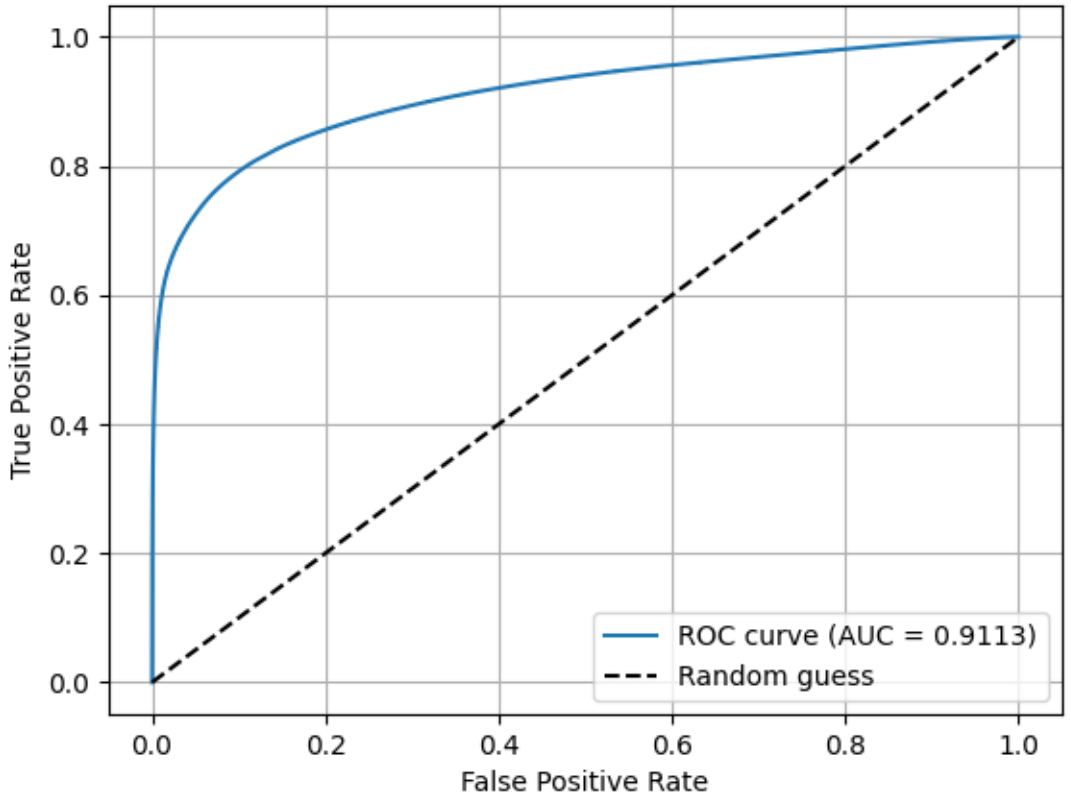


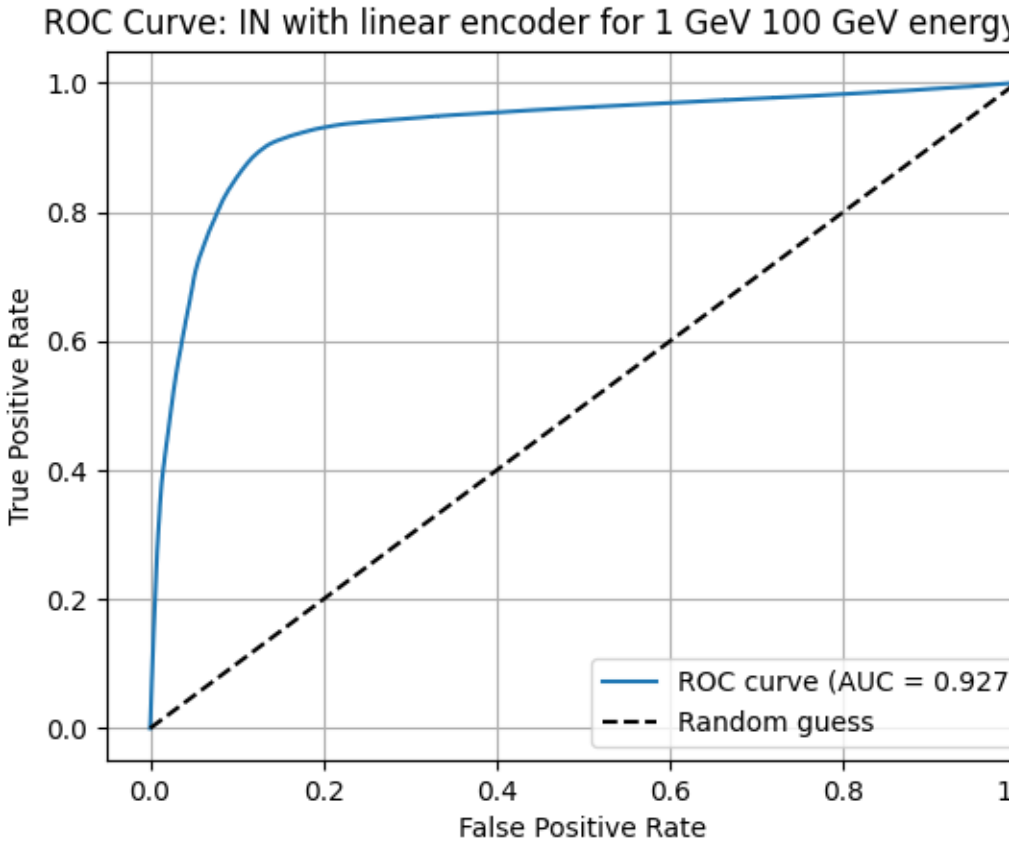






ROC Curve: IN with linear encoder for 100 GeV 10 TeV energy range





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 .

Cross-Entropy vs. Focal Loss

Standard Cross-Entropy

•Loss: $L = -\sum y \, log(p)$

•All samples contribute equally

•Biased toward the majority class in imbalanced datasets

•Easy examples keep dominating the loss

Focal Loss

•Loss: $L = -\alpha_t (1 - p_t)^{\gamma} \log(p_t)$

•Down-weights easy, well-classified samples

•Focuses on hard and misclassified examples

•Class-balancing factor α_t gives more weight to minority class

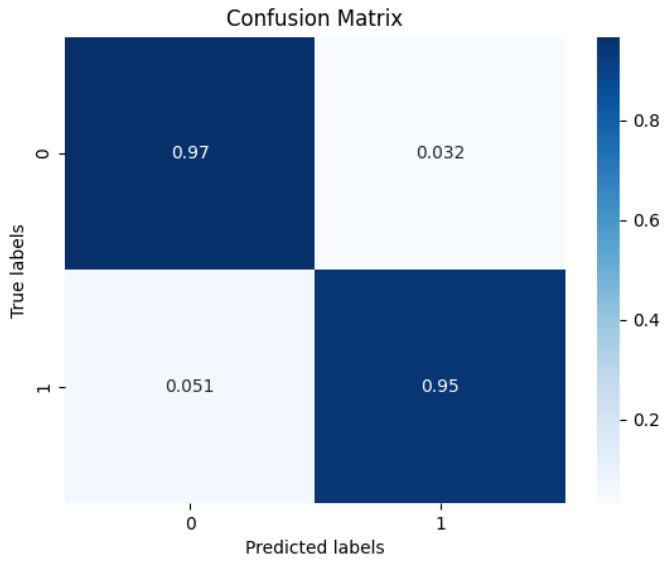
•Designed for **imbalanced and noisy data**

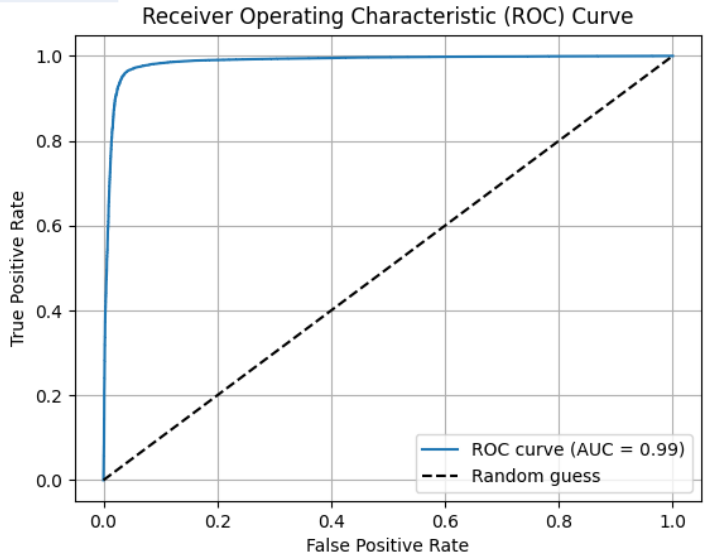
GAT algorithm

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

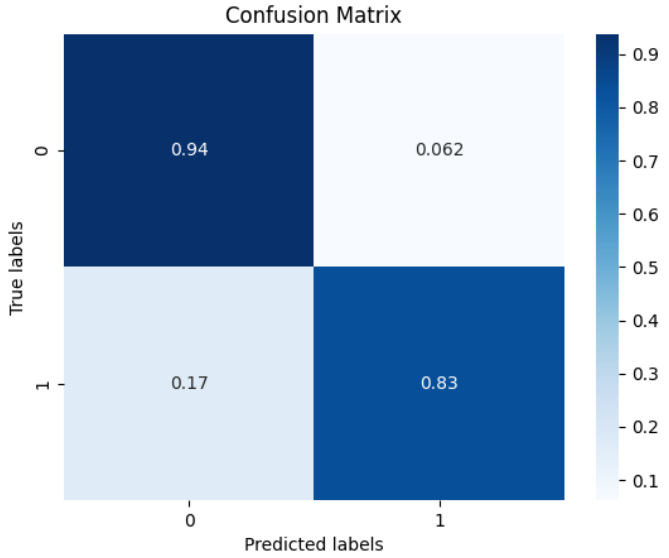


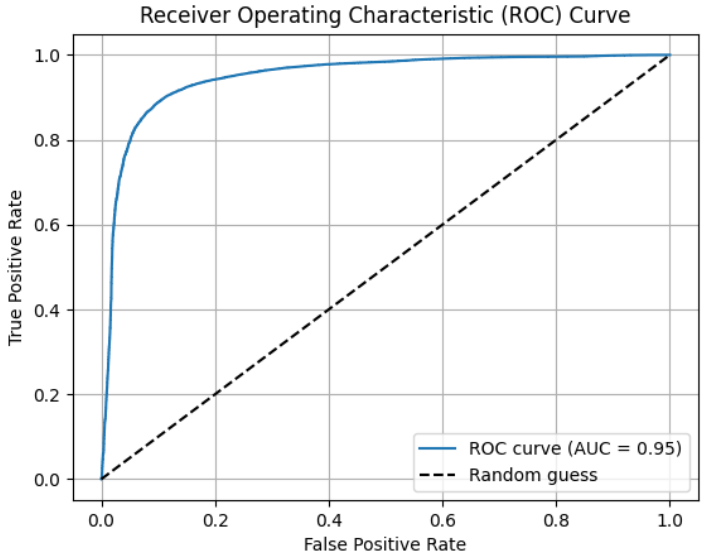


Performances on test data
Accuracy: 0.9557
Recall: 0.9493
Precision: 0.9849

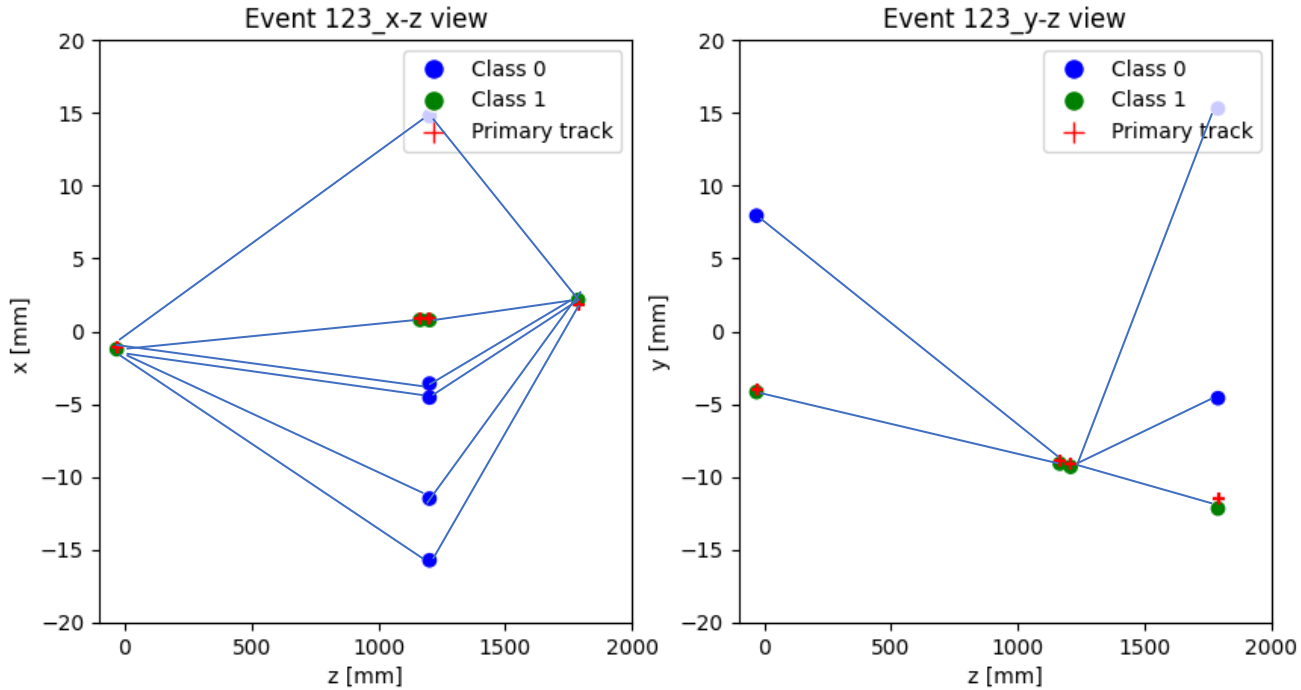
GCN algorithm

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


Event 123_x-z view
Class 0
Class 1
Primary track
x [mm]
z [mm]
Event 123_y-z view
Class 0
Class 1
Primary track
y [mm]
z [mm]

Aspect	Convolutional Neural Networks (CNNs)	Graph Neural Networks (GNNs)
Data Structure	Operate on regular, grid-like data structures (e.g., images), where spatial relationships are fixed.	Operate on graph-structured data, accommodating arbitrary sizes and complex topologies, ideal for irregular detector geometries.
Application in Particle Tracking	Transform detector hits into 2D or 3D images; apply convolutional filters to detect patterns corresponding to particle trajectories.	Model detector hits as nodes in a graph; use message passing to aggregate information from neighboring nodes, capturing complex spatial relationships.
Advantages	- Well-established architectures with extensive tooling and support. - Efficient for data with regular spatial structures.	- Naturally handle irregular and sparse data. - Adaptable to complex detector geometries. - Capture intricate relationships between hits.
Limitations	- May struggle with sparse data and complex topologies.	- Potentially higher computational complexity. - May require more sophisticated training and tuning.