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

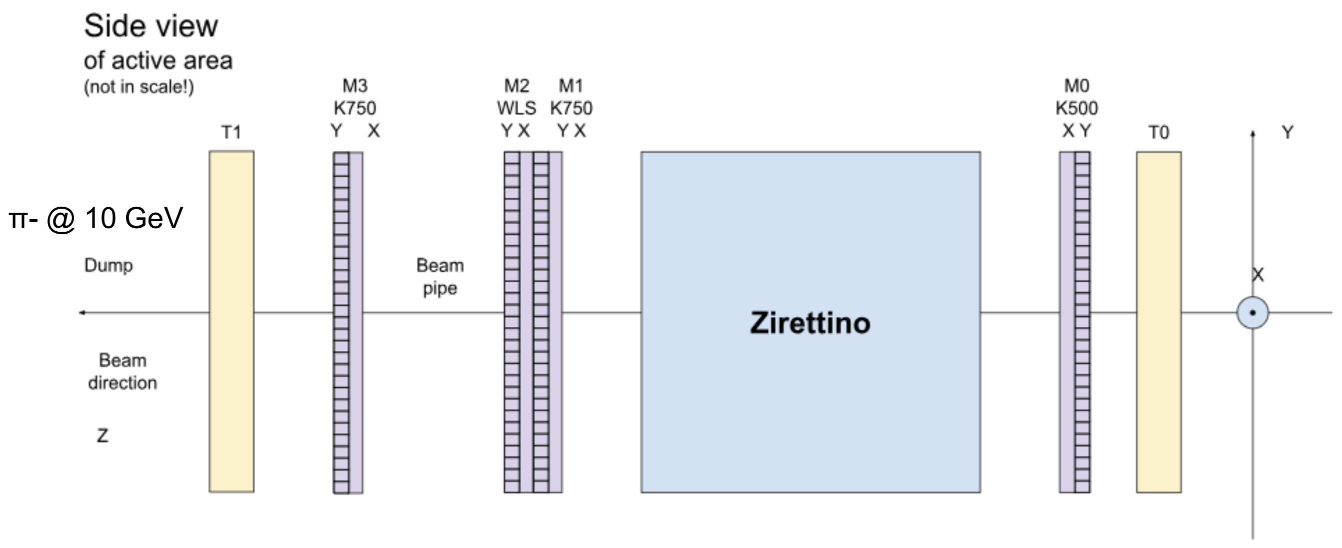
Artificial intelligence for tracking in space experiments

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

Spoke 3 General Meeting, Elba 5-9 / 05, 2024

Scientific Rationale: AI for tracking in space experiments

A range of models inspired by computer vision applications were investigated, which operated on data from tracking detectors in a format resembling images [*A deep learning method for the trajectory reconstruction of cosmic rays with the DAMPE mission, Andrii Tykhonov et al, Astroparticle Physics 146, April 2023, 102795*].
Although these approaches demonstrated potential, image-based methods encountered difficulties in adapting to the scale of realistic data, primarily due to 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. This leads to the investigation of the 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…

Technical Objectives, Methodologies and Solutions: Graph neural networks

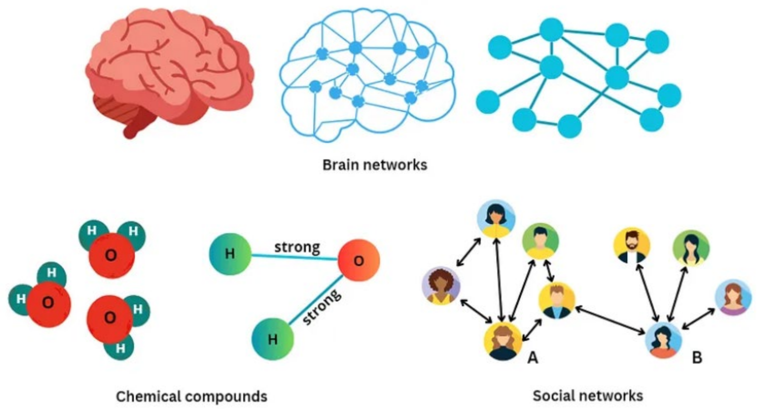
A graph represents the relations (edges or links) between a collection of entities (nodes).
Graph Neural Networks (GNNs) are a class of deep learning models that are designed to operate on graph-structured data.
They have shown remarkable success in tasks such as node classification, link prediction,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

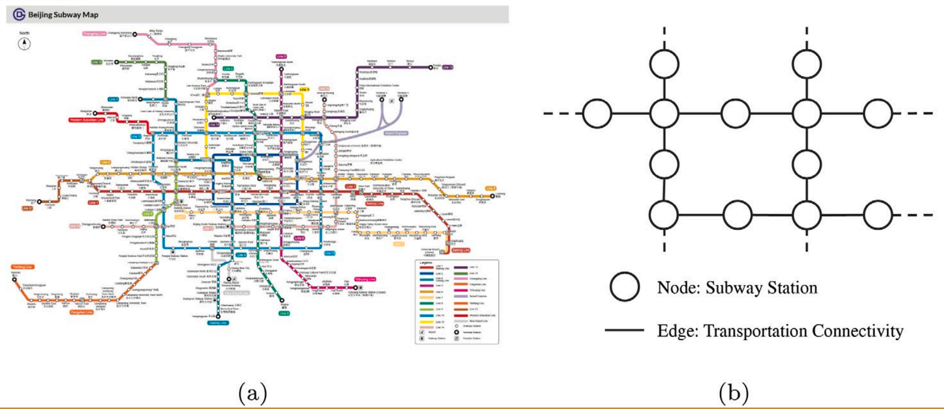
The representation of each node is typically a low-dimensional vector that encodes the node's properties and its relationships with other nodes.

The layers of a GNN are designed to capture increasingly complex features of the graph by aggregating information from the neighborhood of each node.

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.



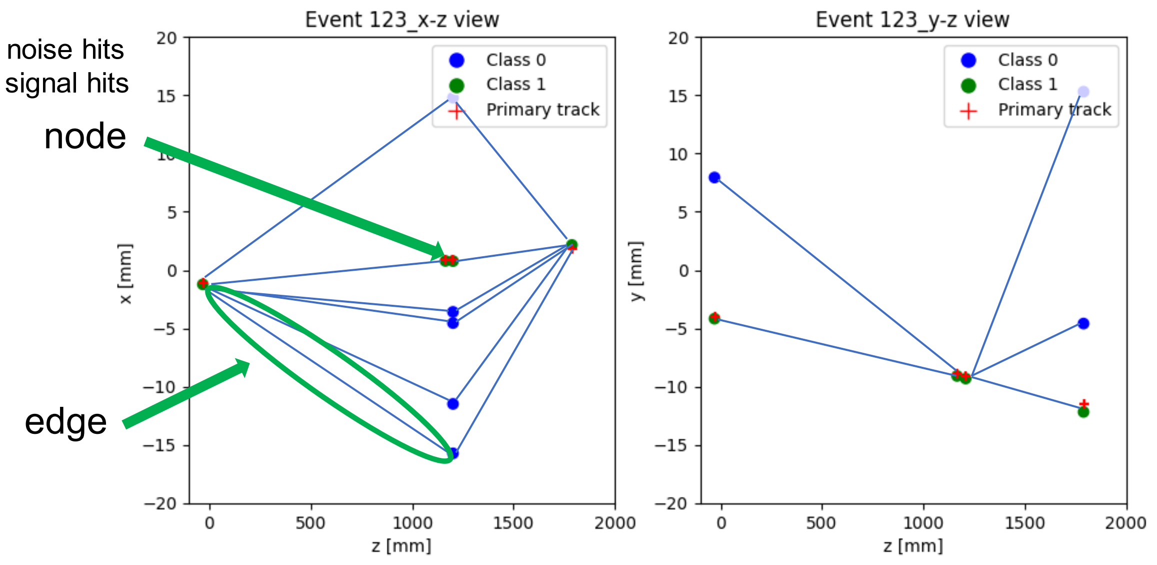
Accomplished Work, Results

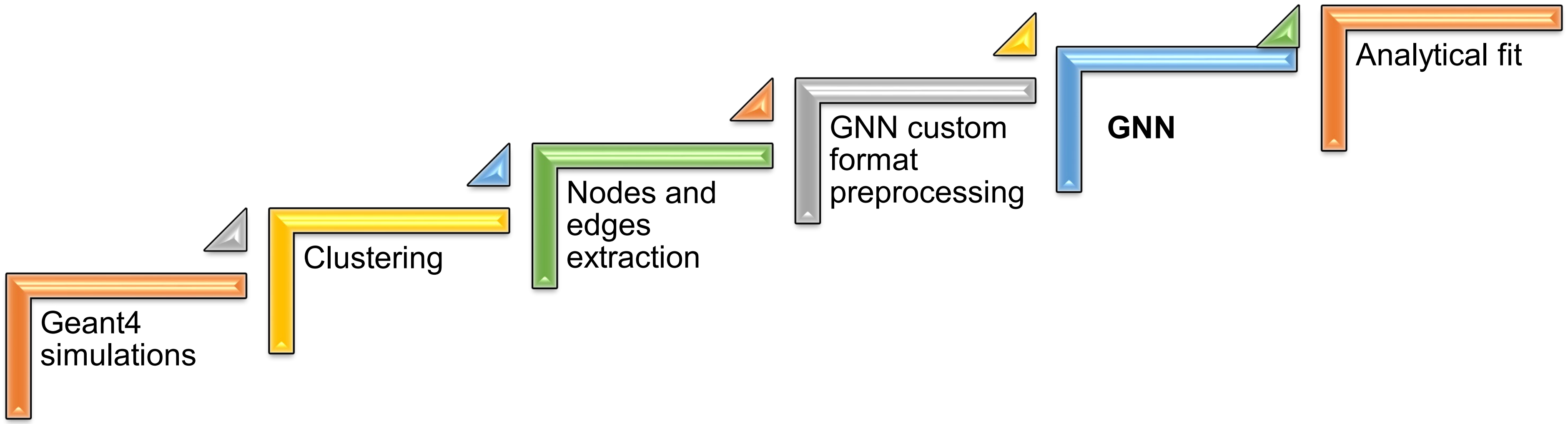
Graphs are a natural way to represent tracks!

We developed graph neural networks algorithms for node classification, by using PyTorch geometric library.

Nodes are the hits inside the tracking detector and edges are the inter-layer hit connection.

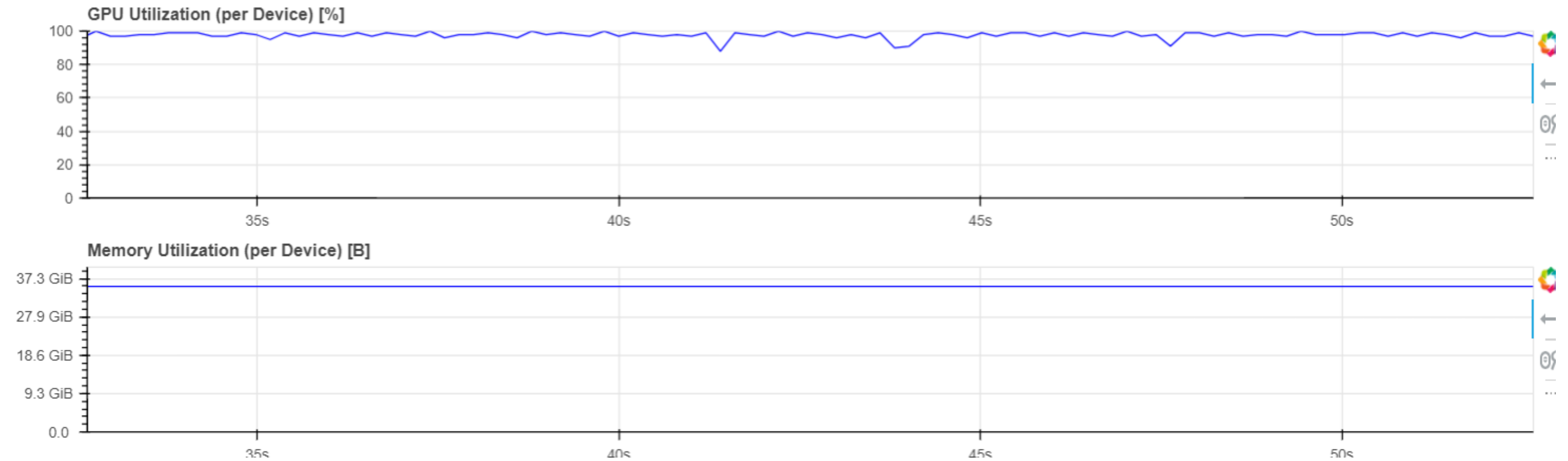
Class 0: noise hits
Class 1: signal hits

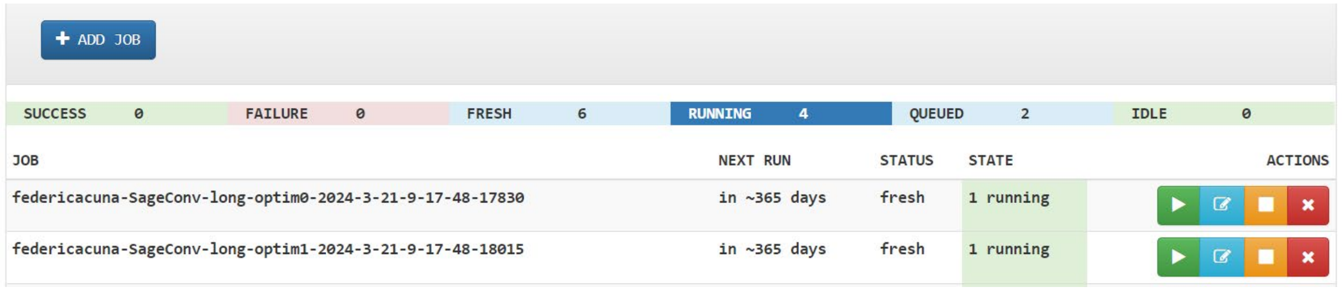




Software development tools

- The software development has been accomplished by using an in-local JupyterHub with GPU. The GPU for the JupyterHub instance is a partitioned GPU created from a 40GB Nvidia A100. In details, from a single A100, 7 GPU were generated, each with 5 GB of dedicated memory. The computing power of each partitioned GPU is also 1/7 of an A100.
- Tests on the GPU utilization have been accomplished by using a in-local JupyterLab instance with 40GB Nvidia A100-GPU.
- Training of the GNNs has been accomplished by using a in-local GPUs cluster. The available GPUs are NVIDIA A100 40GB and NVIDIA V100 32GB.





SUCCESS	FAILURE	FRESH	RUNNING	QUEUED	IDLE
0	0	6	4	2	0

JOB	NEXT RUN	STATUS	STATE	ACTIONS
federicacuna-SageConv-long-optim0-2024-3-21-9-17-48-17830	in ~365 days	fresh	1 running	
federicacuna-SageConv-long-optim1-2024-3-21-9-17-48-18015	in ~365 days	fresh	1 running	

Thanks to dr G.Donvito and dr G.Vino from Recas-Bari.

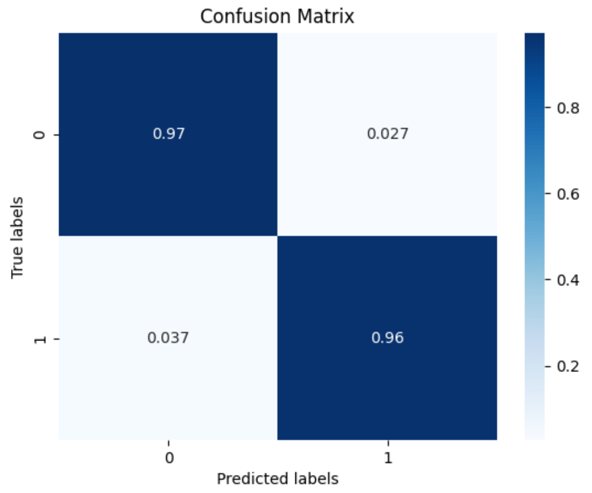
Finanziato dall'Unione europea NextGenerationEU
Ministero dell'Università e della Ricerca
Italiadomani PIANO NAZIONALE DI RIPRESA E RESILIENZA
ICSC Centro Nazionale di Ricerca in HPC, Big Data and Quantum Computing

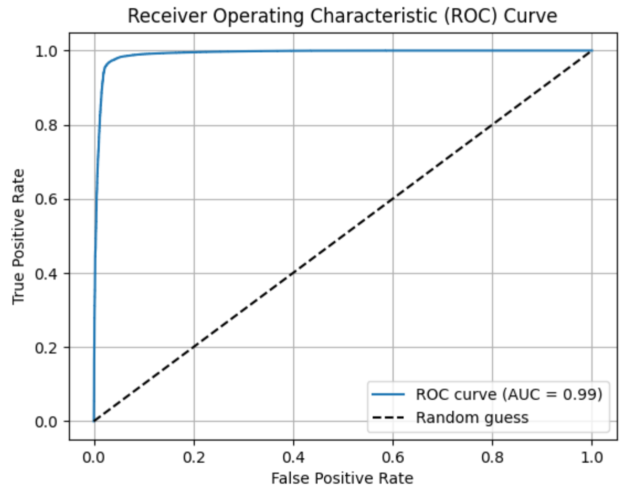
The SageConv algorithm

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. It also uses skip connections to facilitate gradient flow during training. Specifically, the output of each layer is combined with the input representation of the node. The concatenated vector is then passed through a fully connected layer to produce the final output of the layer.

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

- 2,6 million event
- 500 epochs,
- lr =0.0001,
- aggregation function=mean

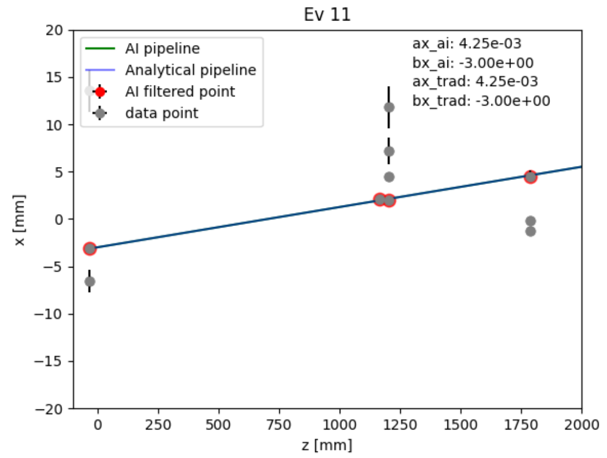


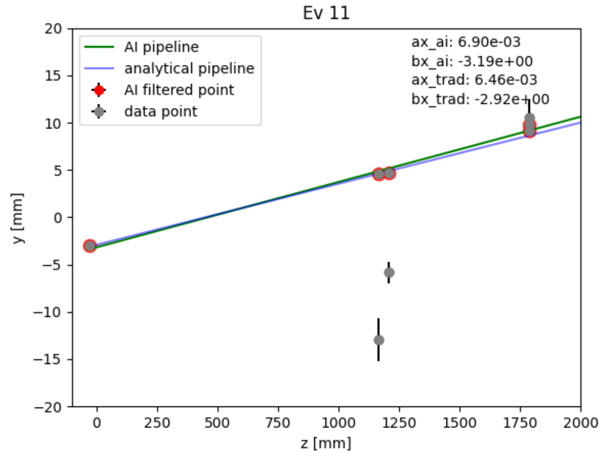


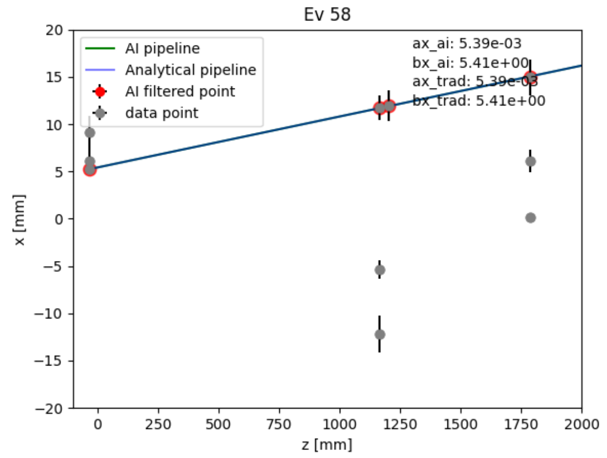
Performances on test data
accuracy 0.9665
recall 0.9631
precision 0.9873

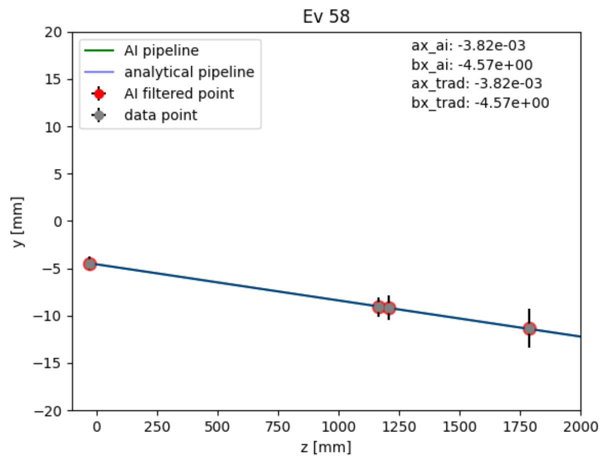
The SageConv algorithm: comparison with traditional pipeline

The traditional tracking pipeline minimizes the chi-square between the hits 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.



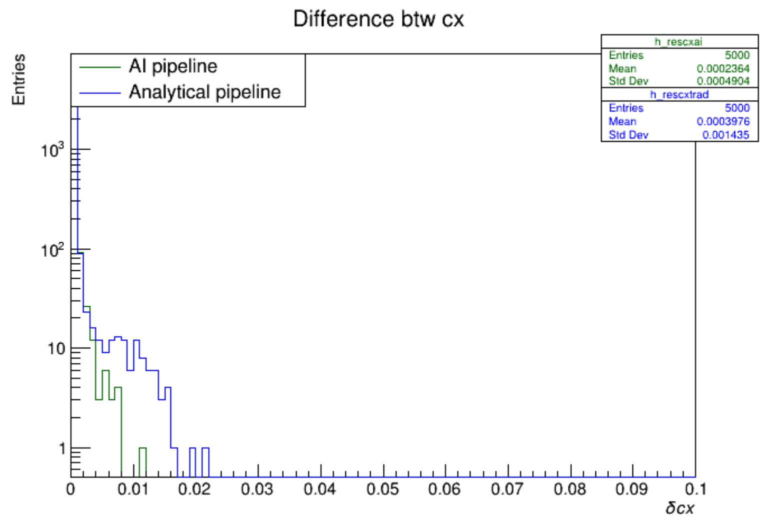


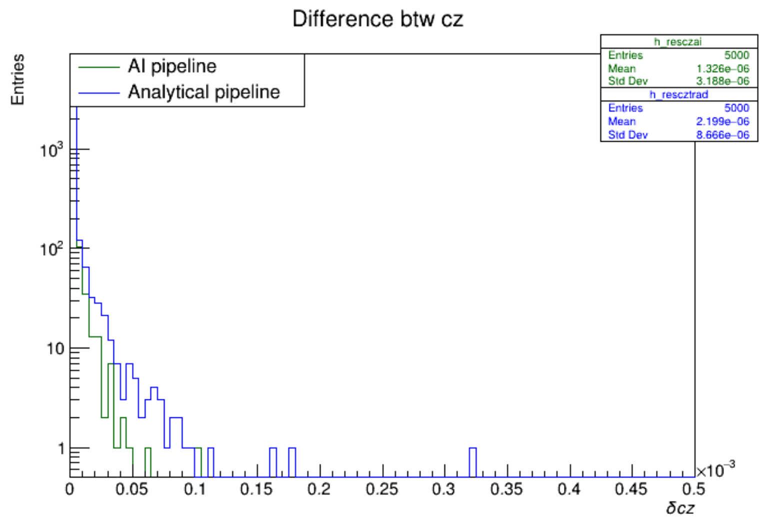


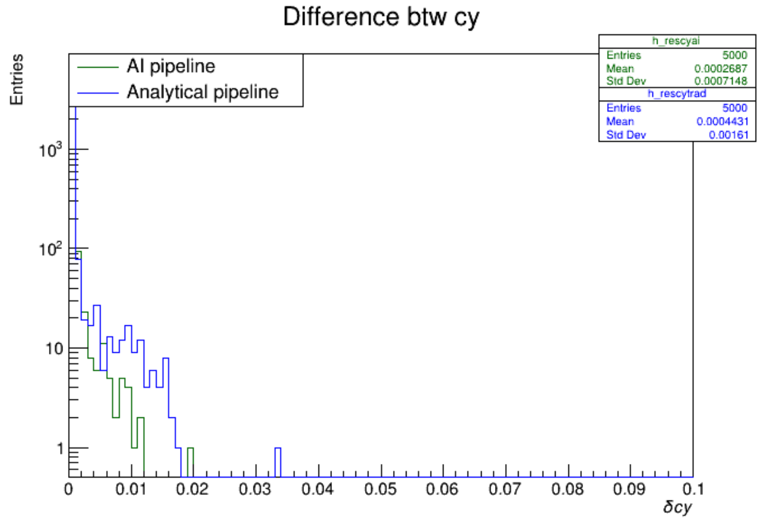


The SageConv algorithm: comparison with traditional pipeline

To evaluate the AI pipeline, we compare the differences between the monte carlo director cosines and the ones evaluated by applying the AI and traditional pipeline







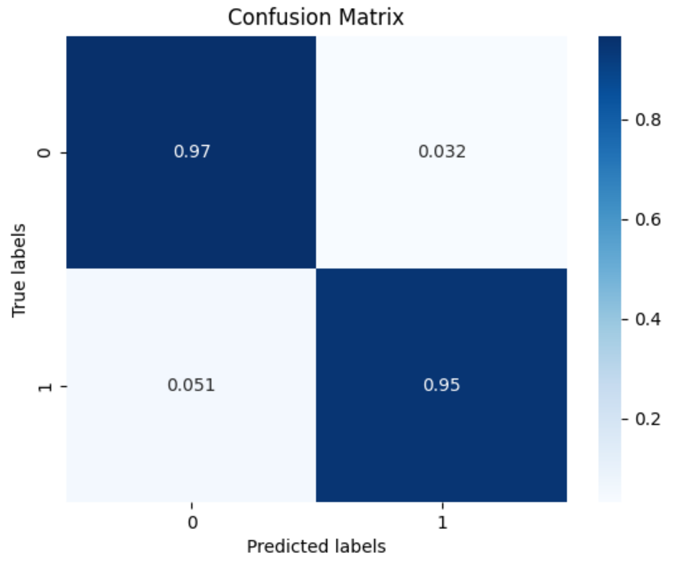
To process 5000 tracks the analytical pipeline takes 9 min and 15 s, the AI pipeline takes 114 ms!

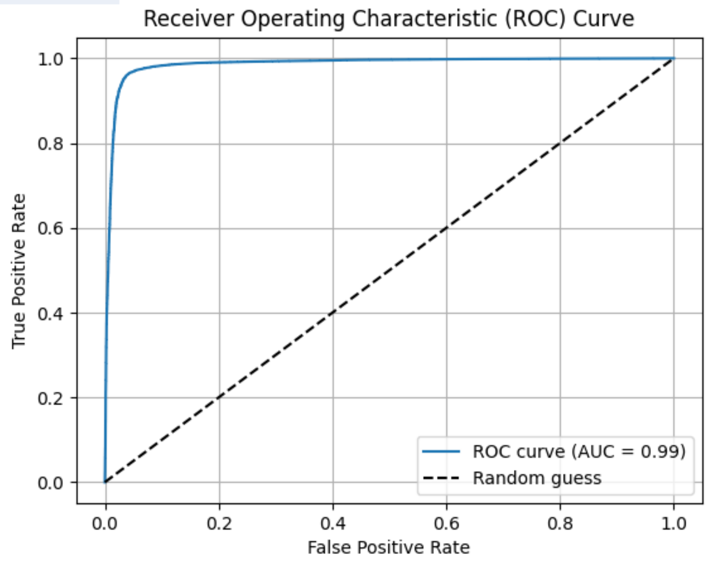
GAT algorithm

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

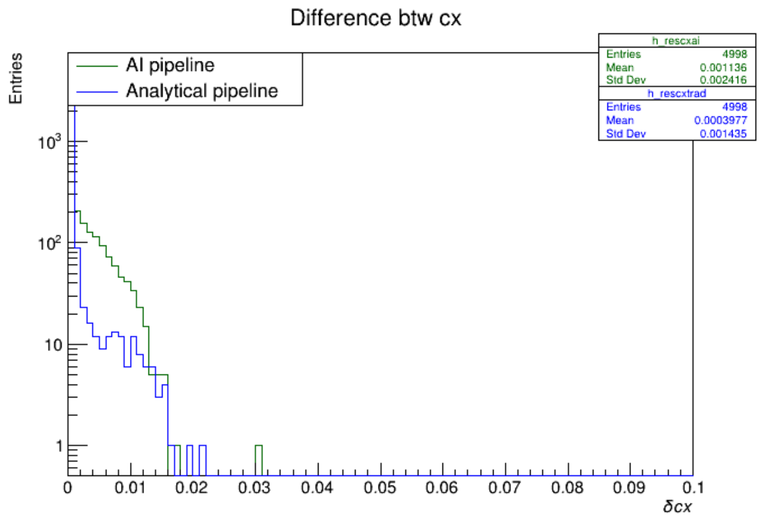
600 epochs
2,6 million events
learning rate: 1e-4

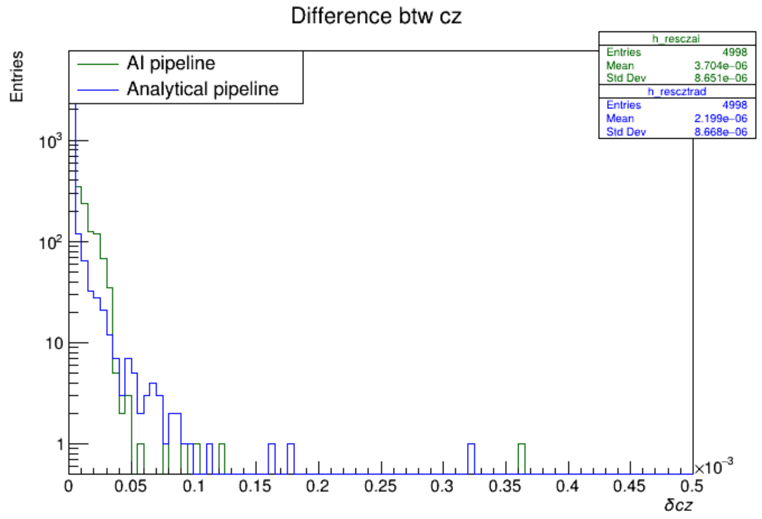


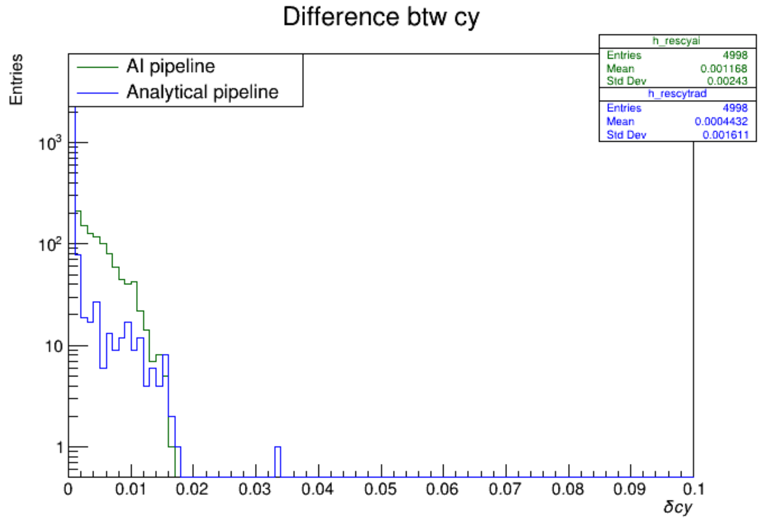


Performances on test data
Accuracy: 0.9557
Recall: 0.9493
Precision: 0.9849

The GAT algorithm: comparison with traditional pipeline

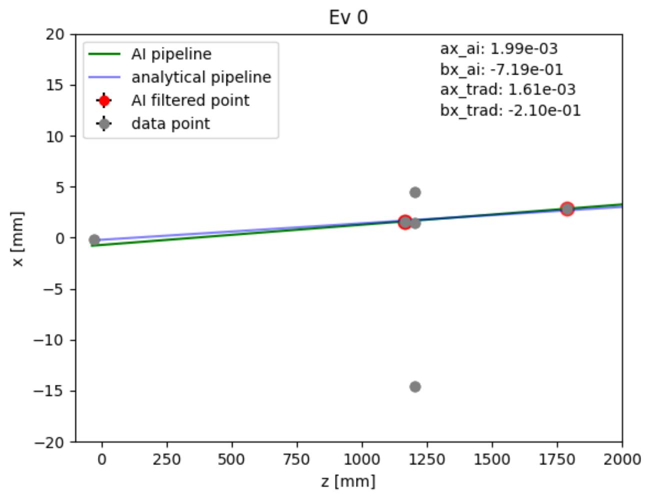


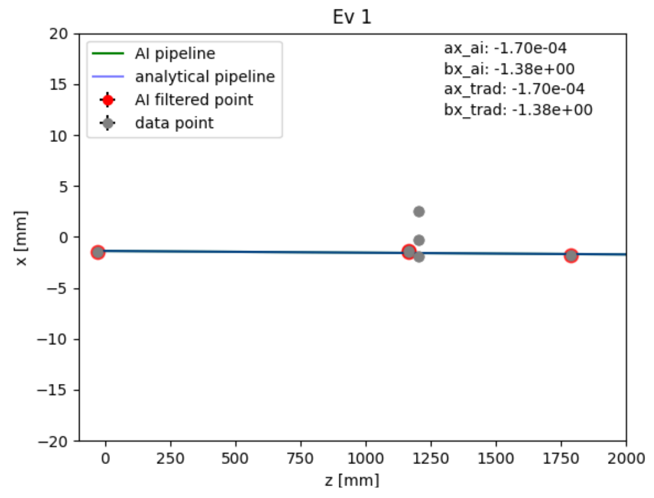


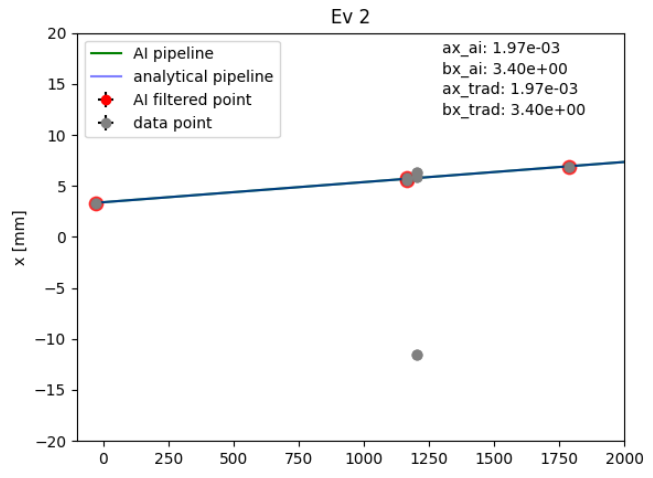


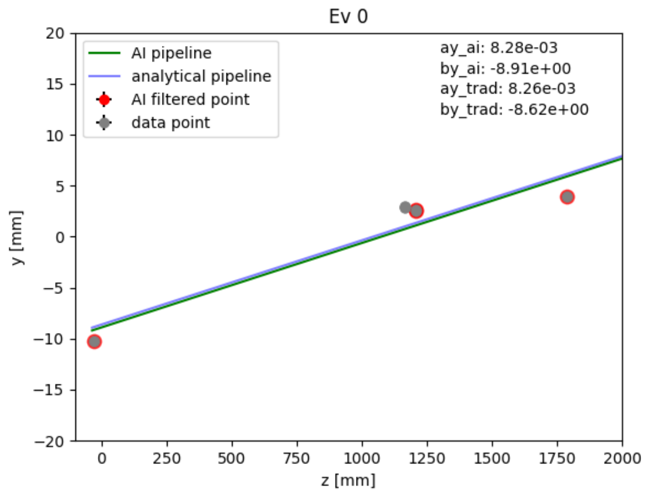
The Gat performances are similar in terms of time consumption to the SageConv ones.
Anyway, for two events out of 5000, it does not recognize good hits (false negatives).

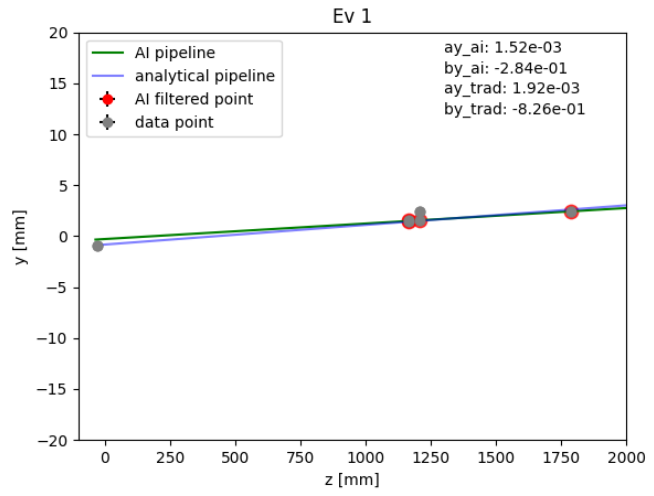
Beam test data
SageConv algorithm

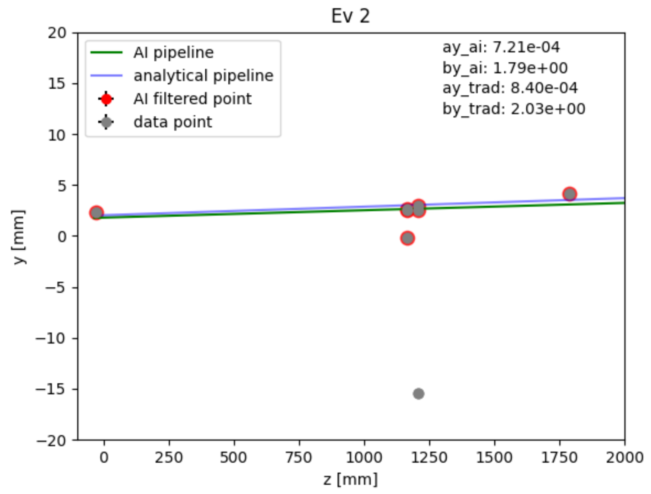




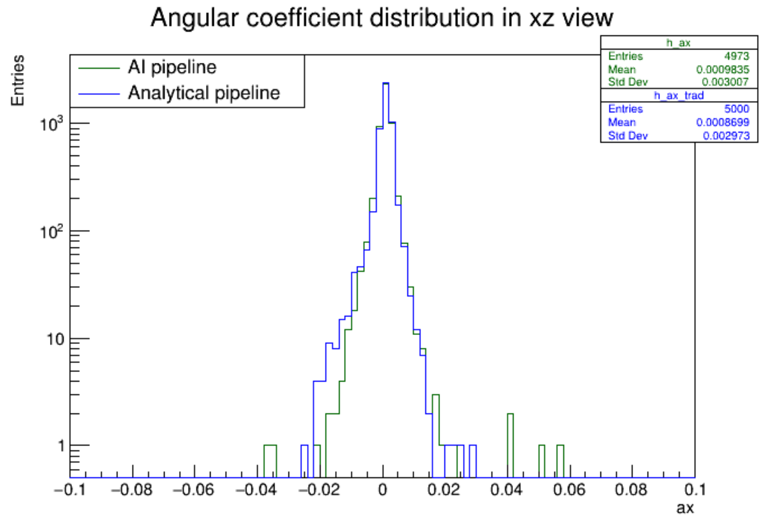


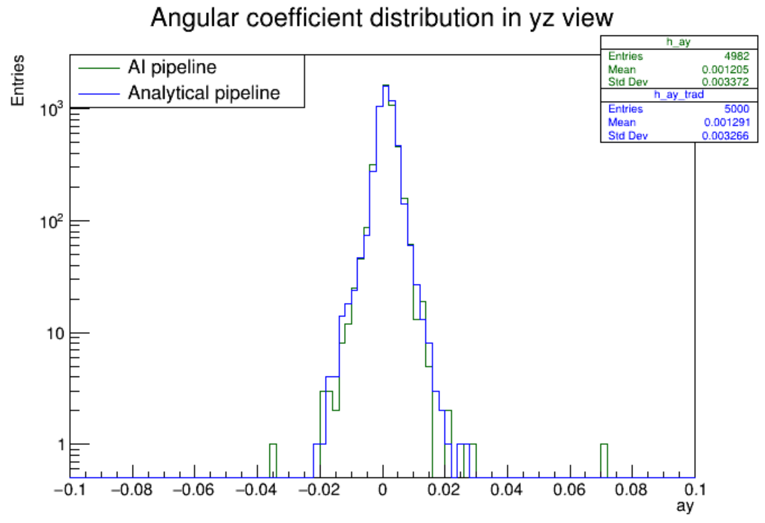


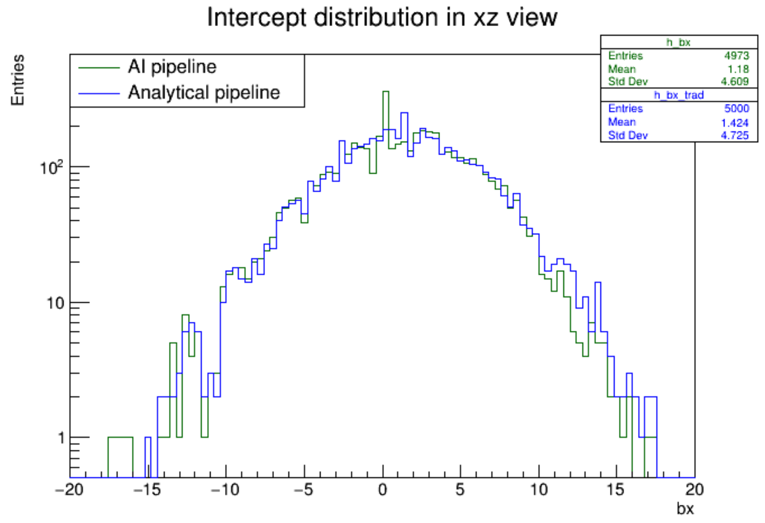


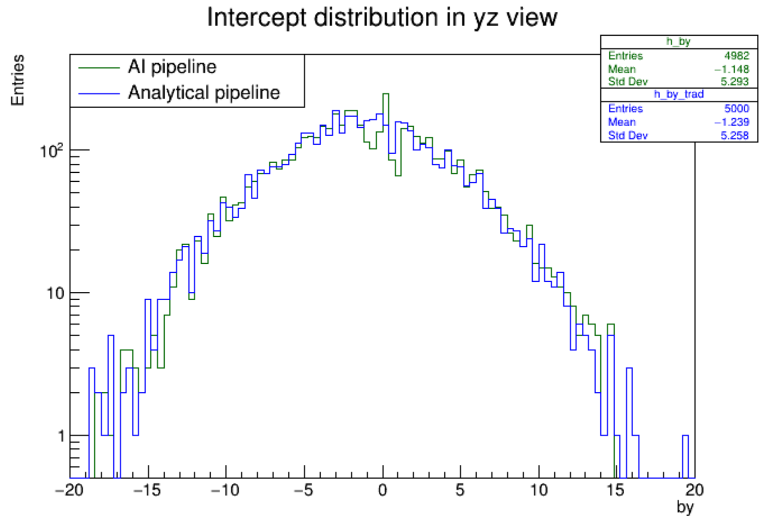


Track fit parameters distributions

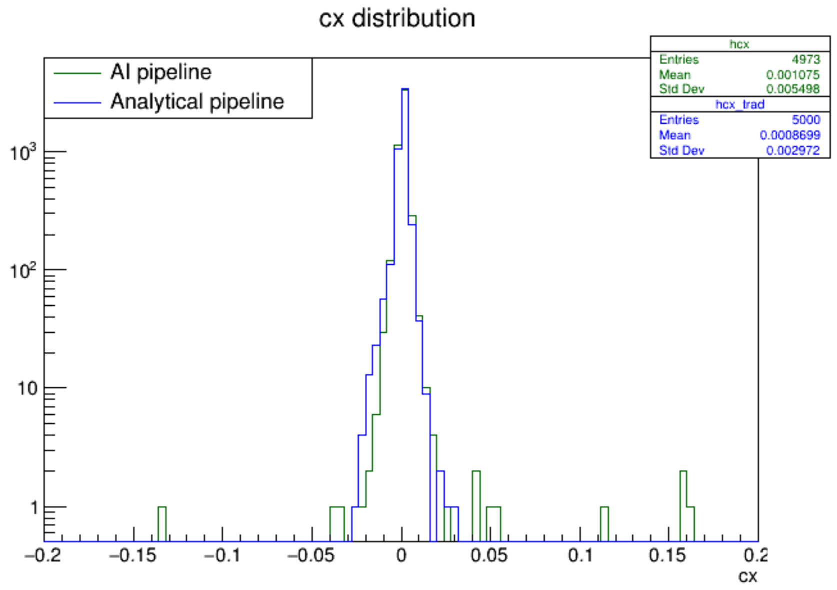


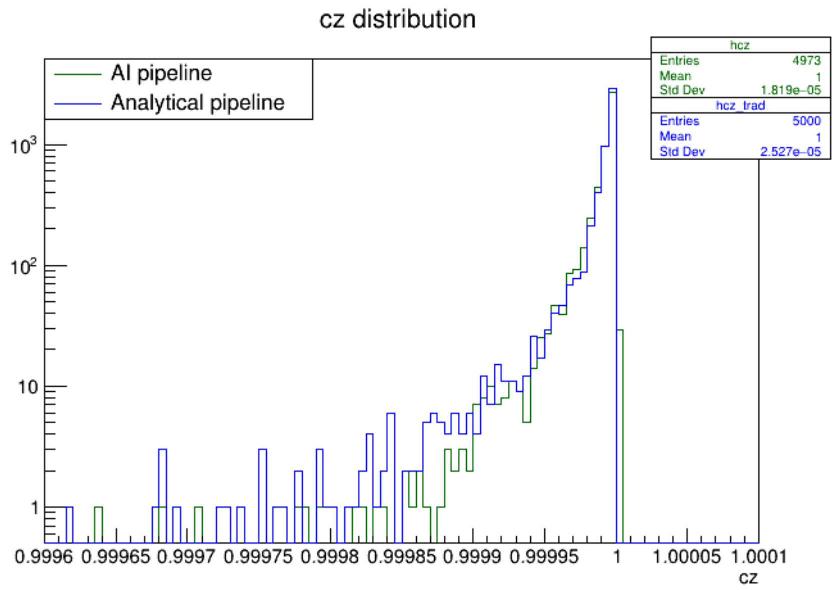


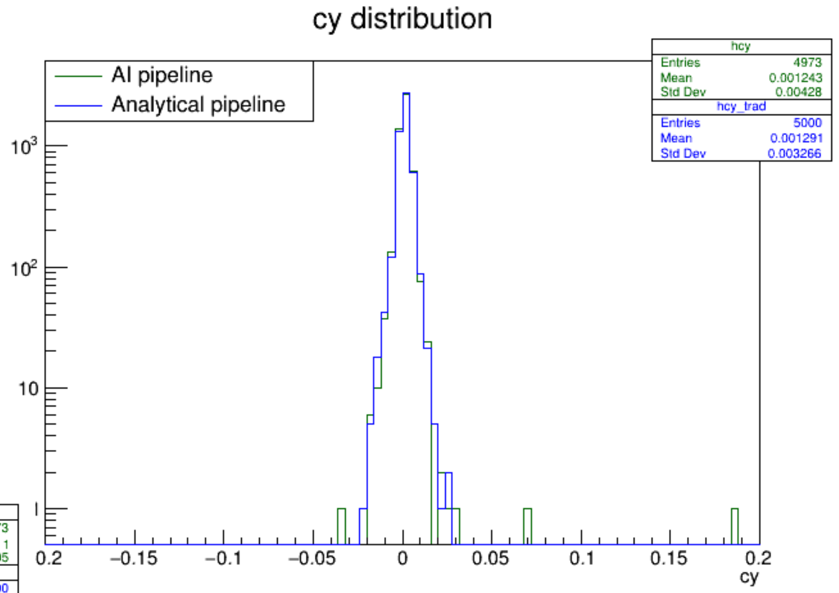




Cosine directors distributions







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

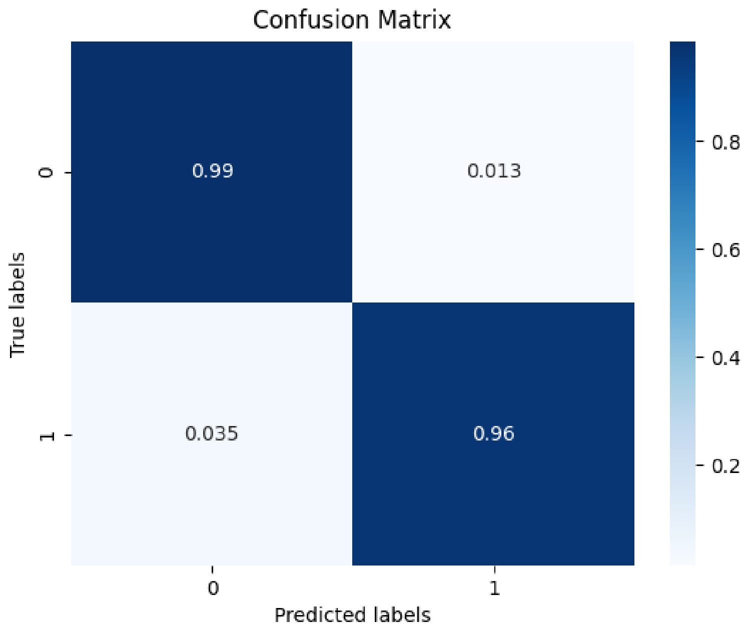
What happens in a high noise environment?

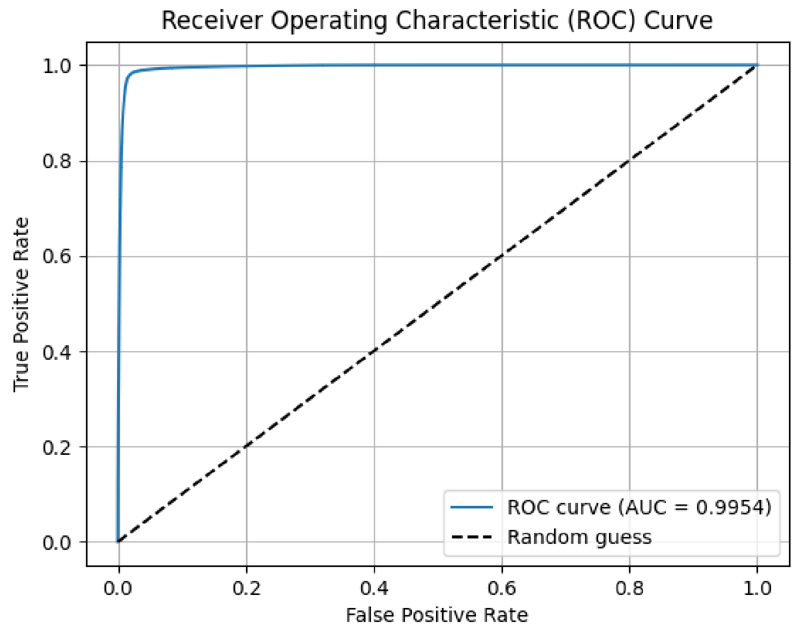
Evaluating a high-noise environment can be beneficial because it often occurs that real-world data is subject to detectors aging, environment with high levels of charged particles, backscattering tracks…

SageConv algorithm:
- 4 million event,
- 550 epochs,
- lr =0.0001,
- 7 GNN layer
- aggregation function=mean

Balanced and standardized dataset

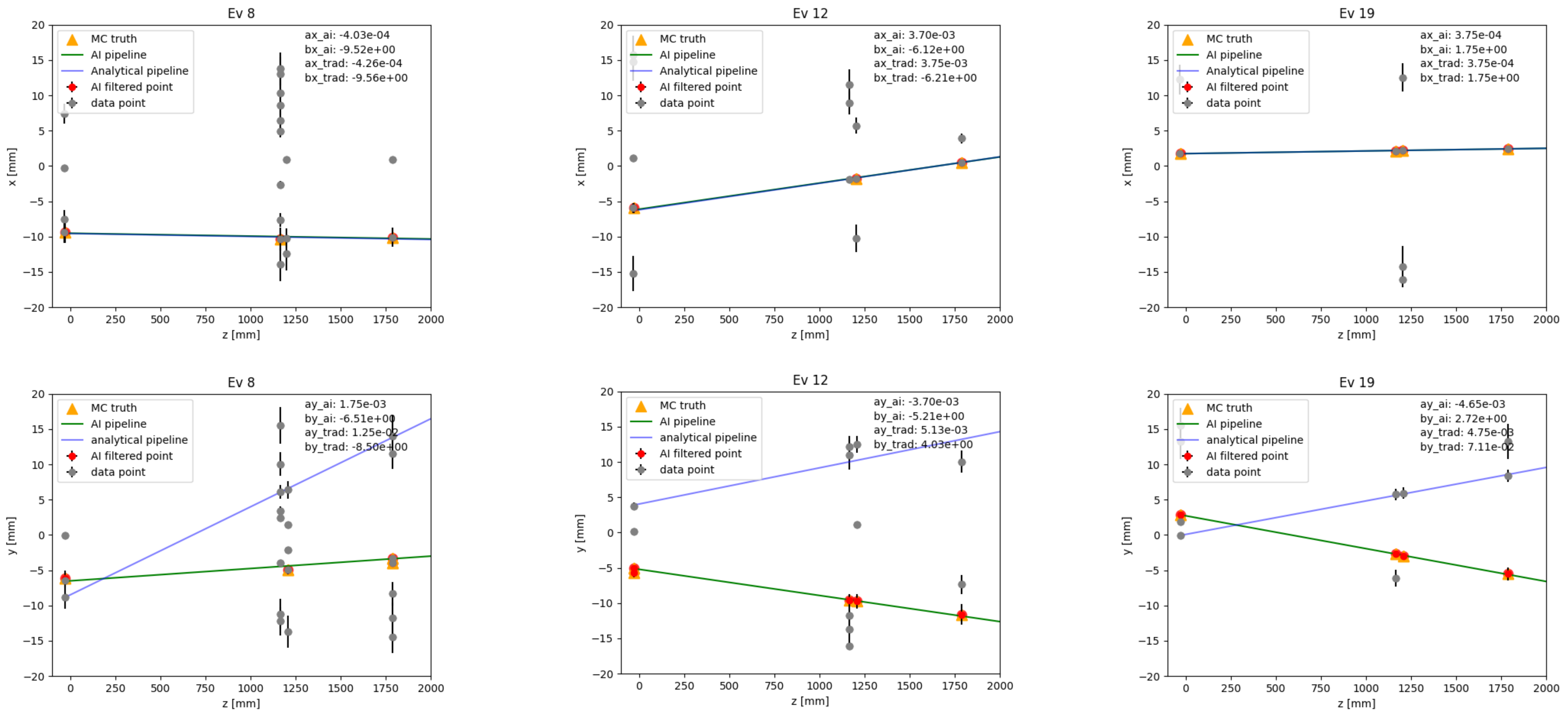
Preliminary





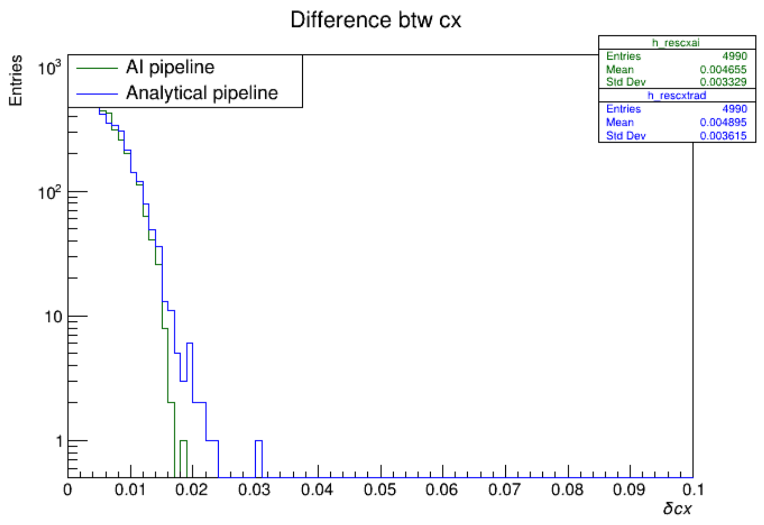
Accuracy:	0,9761
Recall:	0,9649
Precision:	0,9851

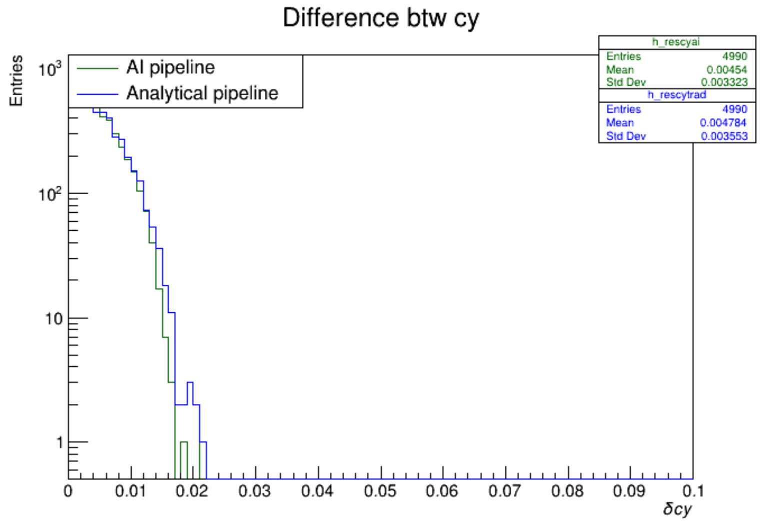
Event display

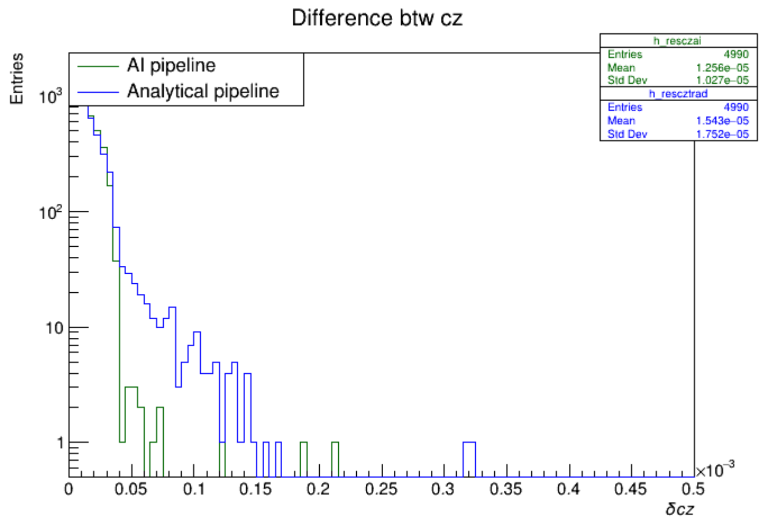


Finanziato dall'Unione europea NextGenerationEU
Ministero dell'Università e della Ricerca
Italiadomani PIANO NAZIONALE DI RIPRESA E RESILIENZA
ICSC Centro Nazionale di Ricerca in HPC, Big Data and Quantum Computing

The SageConv algorithm: comparison with traditional pipeline







To process 5000 tracks the analytical pipeline takes 102 min, the AI pipeline takes 140 ms!

Timescale, Milestones and KPIs

Timescale	Milestones	KPI
May-September	Optimization of the GNN algorithms	Link github
September –December	Preliminary analysis and gnn development on more complex tracking data	Link github
January-?	Development of a preliminary unified AI architecture for tracker and calorimeter (see M.Bossa talk) in space experiment	Link github

Spoke 3 Technical Workshop, Trieste October 9 / 11, 2023

- Improving the GNN tracking algorithm (*by the end of the year*): adding link prediction, using PyTorch libraries… ✓
- Testing the algorithm on big datasets, by using docker container implementation (*by the end of the year/first months of 2024* - **April 2024 checkpoint**) ✓
- Adapt the algorithm to in orbit or future space experiments, like DAMPE and HERD (*2024/2025*) **…ongoing…**

Next Steps and Expected Results

The GNN shows promising performances on tracking purposes over the analytical/traditional approach.

All the software development has been saved on https://github.com/federicacuna/TB_Sept_2023_ml, https://github.com/federicacuna/TestBeam_T10_2023 , https://github.com/federicacuna/nuses_Ml (access on request).

There is room for improvement, by looking on more complex problem with more sophisticated tracker, where we can investigate more elaborated GNN architecture like the interaction networks, which can reconstruct multiple tracks inside the tracker.

Next step:

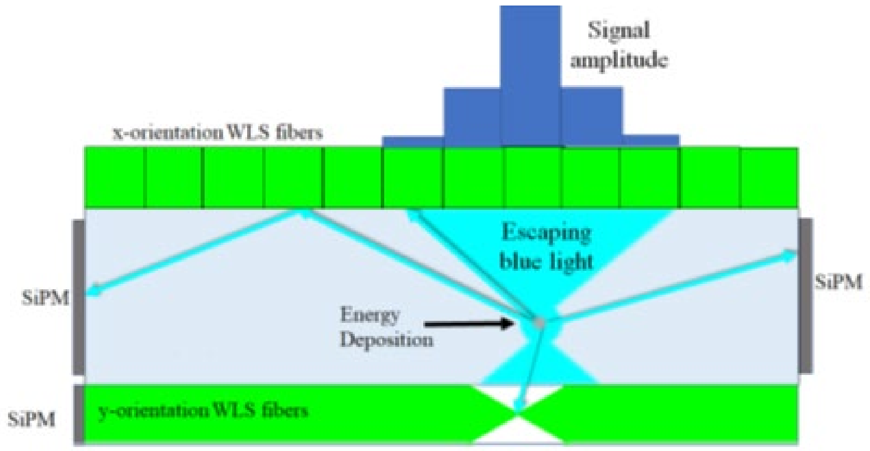
- Optimization of the GNN algorithm
- Preliminary analysis and GNN development on more complex tracking data
- Development of a preliminary unified AI architecture for tracker and calorimeter (see M.Bossa talk) in space experiment

Thank you!

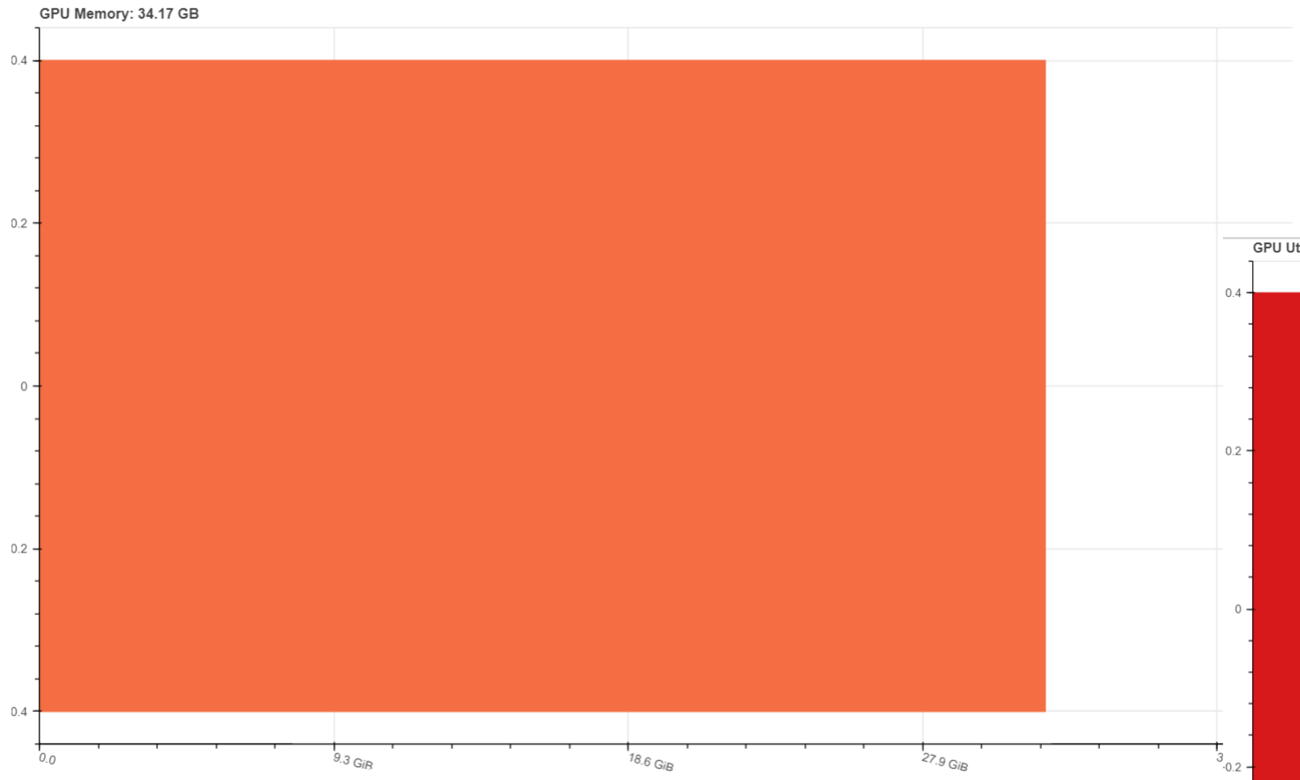
Zirettino

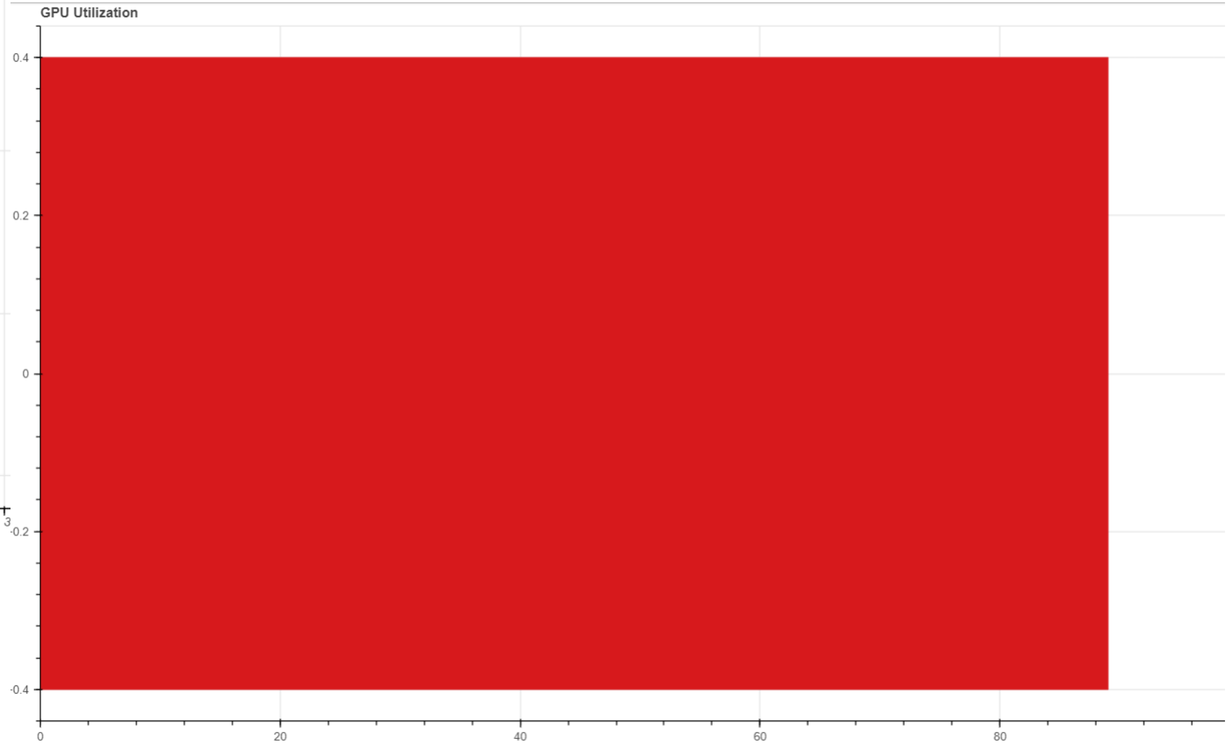
The Zirè experiment is part of the NUSES (NeUtrino and Seismic Electromagnetic Signals) space mission, proposed by the Gran Sasso Science Institute (GSSI) in collaboration with the Istituto Nazionale di Fisica Nucleare (INFN) and by Thales Alenia Space Italy (TAS-I), involving many Institutes and Universities from Europe and abroad. Zirè will perform measurements of electrons, protons and light nuclei from few up to hundreds of MeVs, for the study of low energy CRs, space weather phenomena and possible Magnetosphere-Litosphere-Ionosphere Coupling (MILC) signals. A further goal of the experiment is to test new tools for the detection of photons in the energy range 0.1 MeV - 50 MeV, allowing the investigation of transient phenomena and steady gamma sources.
[The Zirè experiment on board the NUSES space mission, 38th International Cosmic Ray Conference, *R. Aloisio, et al.,* https://doi.org/10.22323/1.444.0139].
Zirettino is the prototype of Zirè with 1 X-Y modules (FTK), a PST which consists of scintillating bars and a pixelated calorimeter.
Each module of the FTK is equipped with two planes (views) of fibers oriented along two orthogonal directions (X-view and Y-view).
Each view consists of two staggered layers (ribbons) of round scintillating fibersThe width of each layer is about 3.25 cm, to match the size of the SiPM array. Fibers of different diameters, i.e. 500 μm and 750 μm, have been used configurations with different read-out pitches for the SiPM strips.
[The light tracker based on scintillating fibers with SiPM readout of the Zire instrument on board the NUSES space mission, Mazziotta, Mario Nicola and Pillera, Roberta, https://pos.sissa.it/444/083].

WLS+LYSO CRYSTAL: Imaging calorimeter



- Scintillation light isotopically produced in the scintillator crystal
- Fibers subtended by the acceptance cone corresponding to the total internal reflection angle collect the light to photodetectors
- Crossed fiber planes allow to centroid the interaction point in the scintillator crystal


GPU Memory: 34.17 GB
0.4
0.2
0
0.2
0.4
0.0
9.3 GiB
18.6 GiB
27.9 GiB
3


GPU Utilization
0.4
0.2
0
-0.2
-0.4
0
20
40
60
80

The architecture of graph neural networks

Each node in the graph has a feature vector representing its attributes (categorical or continuous).

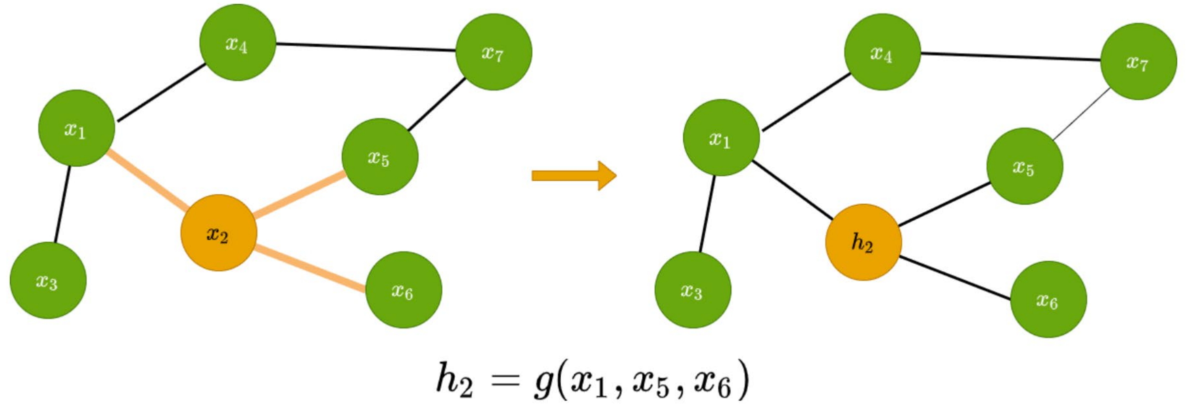
Edges between nodes can also have associated features, capturing more nuanced information about the relationships between nodes.

The core idea of GNNs is to iteratively update the node representations by **passing and aggregating messages from neighboring nodes**.

This process, called **message passing**, involves computing a message vector for each neighboring node using a message function.

The message function takes the features of the sender node, receiver node and edge (if present) as input and produces a message vector.

This message-passing process is performed through multiple layers or "hops" to capture higher-order relationships within the graph.



$$h_2 = g(x_1, x_5, x_6)$$

GNN applications:

- ❖ Link prediction
- ❖ Node classification
- ❖ Graph classification

- *A Gentle Introduction to Graph Neural Networks, https://distill.pub/2021/gnn-intro/*
- *Introducing TensorFlow Graph Neural Networkshttps://blog.tensorflow.org/2021/11/introducing-tensorflow-gnn.html*
- *TensorFlow-GNN: An End-To-End Guide For Graph Neural Networks, https://towardsdatascience.com/tensorflow-gnn-an-end-to-end-guide-for-graph-neural-networks-a66bfd237c8c*

Graph convolutional neural network

In traditional neural networks, linear layers apply a **linear transformation** to the incoming data. T
his transformation converts input features *x* into hidden vectors *h* through the use of a weight matrix $\mathbf{W}$.
Ignoring biases for the time being, this can be expressed as:

$$h = \mathbf{W}x$$

With graph data, an additional layer of complexity is added through the **connections between nodes**.
These connections matter because, typically, in networks, it's assumed that similar nodes are more likely to be linked to each other than dissimilar ones, a phenomenon known as network homophily.
We can enrich our **node representation** by merging its features with those of its neighbors.
This operation is called convolution, or neighborhood aggregation. Let's represent the neighborhood of node *i* including itself as *Ñ*.

$$h_i = \sum_{j \in \tilde{\mathcal{N}}_i} \mathbf{W}x_j$$

Unlike filters in Convolutional Neural Networks (CNNs), our weight matrix $\mathbf{W}$ is unique and shared among every node. But there is another issue: nodes do not have a **fixed number of neighbors** like pixels do.
How do we address cases where one node has only one neighbor, and another has 500? If we simply sum the feature vectors, the resulting embedding *h* would be much larger for the node with 500 neighbors. To ensure a **similar range** of values for all nodes and comparability between them, we can normalize the result based on the **degree** of nodes, where degree refers to the number of connections a node has.

$$h_i = \frac{1}{\deg(i)} \sum_{j \in \tilde{\mathcal{N}}_i} \mathbf{W}x_j$$

Introduced by Kipf et al. (2016), the graph convolutional layer has one final improvement.
The authors observed that features from nodes with numerous neighbors propagate much more easily than those from more isolated nodes.
To offset this effect, they suggested assigning **bigger weights** to features from nodes with fewer neighbors, thus balancing the influence across all nodes.
This operation is written as:

$$h_i = \sum_{j \in \tilde{N}_i} \frac{1}{\sqrt{\deg(i)}\sqrt{\deg(j)}} \mathbf{W} x_j$$

GraphSAGE

Hamilton et al. introduced GraphSAGE in 2017 (see item [1] of the *Further reading* section) as a framework for inductive representation learning on large graphs (with over 100,000 nodes). Its goal is to generate node embeddings for downstream tasks, such as node classification.

Two main components of GraphSAGE:

- Neighbor sampling
- Aggregation

Neighbor sampling
Instead of adding every neighbor in the computation
graph, we sample a predefined number of them. For instance, we choose only to keep (at most) three
neighbors during the first hop and five neighbors during the second hop.
Aggregation
Now that we've seen how to select the neighboring nodes, we still need to compute embeddings. This is performed by the aggregation operator (or aggregator). In GraphSAGE, the authors have proposed
three solutions:

- mean aggregator
- long short-term memory (LSTM) aggregator
- pooling aggregator

SageConv

SageConv is an improvement over GraphSAGE in that it uses a more expressive convolutional operator, which allows it to capture more complex features. The key difference between SageConv and GraphSAGE is in the aggregation function used. The SageConv aggregation function takes into account the degrees of the nodes in the neighborhood. SageConv, unlike GraphSAGE, 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.
It also uses skip connections to facilitate gradient flow during training. Specifically, the output of each layer is combined with the input representation of the node. The concatenated vector is then passed through a fully connected layer to produce the final output of the layer.

Advantages of SageConv
SageConv has several advantages over other GNN architectures. Firstly, the use of the degree-normalized aggregation function allows SageConv to capture more fine-grained information about the local neighborhood of each node, which can be particularly useful in tasks such as link prediction and graph classification.

Secondly, the skip connections used in SageConv help to alleviate the problem of vanishing gradients that can occur in deep neural networks. By preserving the information from previous layers, skip connections facilitate gradient flow and allow the model to learn more effectively.

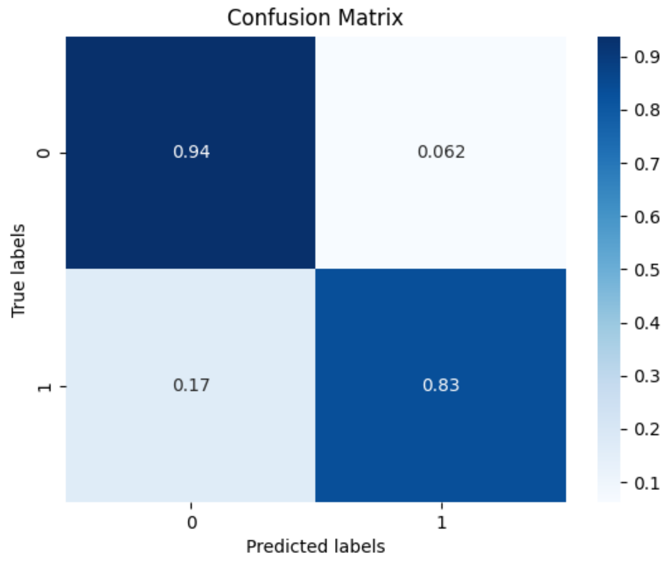
Finally, SageConv is computationally efficient, as it does not require any expensive matrix inversions or eigendecompositions that are typically required in other GNN architectures.

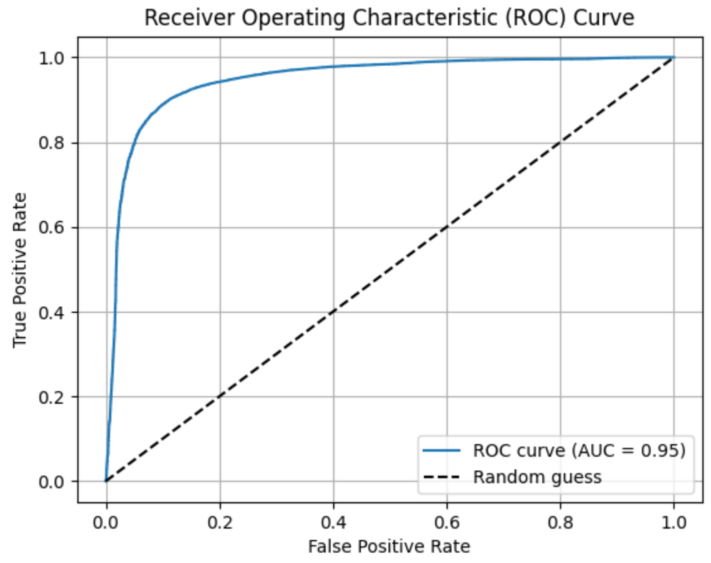
GAT

Graph Attention Networks (GATs) are a variant of Graph Neural Networks (GNNs) that leverage attention mechanisms for feature learning on graphs. Introduced by Veličković et al. in 2018, GATs offer a more nuanced approach to aggregating neighborhood information compared to traditional GNNs.
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.
The attention-based approach allows GATs to assign different weights to different neighbors, providing a more flexible and potentially more expressive model. It also offers a level of interpretability, as the attention coefficients can be seen as indicating the importance of each neighbor.

GCN algorithm

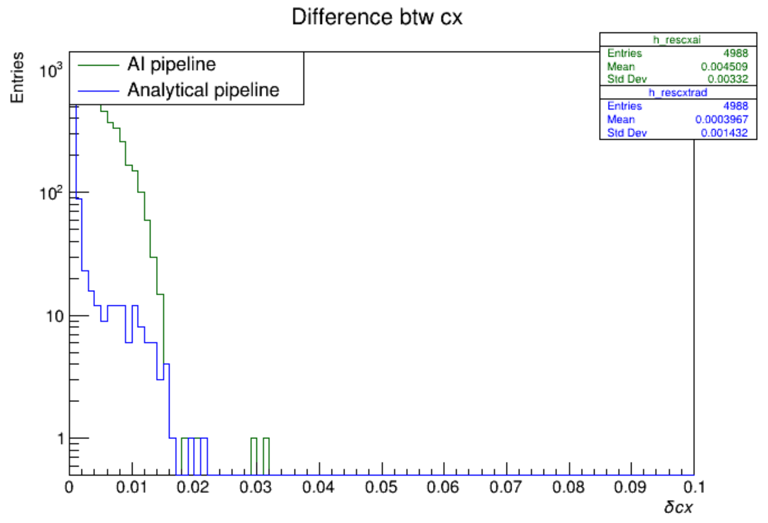
The general idea of GCN is to apply convolution over a graph. Instead of having a 2-D array as input as in the classical CNN algorithm, GCN takes a graph as an input

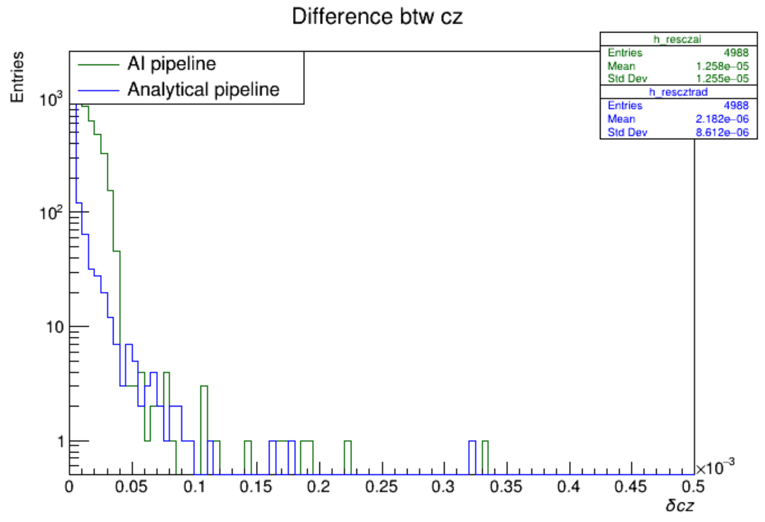


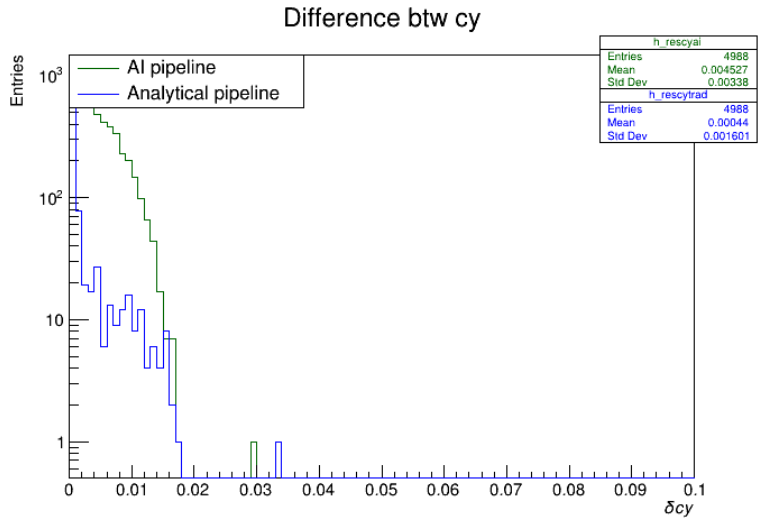


Algorithm performances:
1500 epochs
2 million events
lr 5e-4
Accuracy: 0,8662
Recall: 0,8326
Precision: 0,9663

The GCN algorithm: comparison with traditional pipeline







The GCN performances are similar in terms of time consumption to the SageConv ones.
Anyway, for 11 events out of 5000, it does not recognize good hits (false negatives).