Quantum Machine Learning Frameworks for Charged Particle Tracking

Laura Cappelli

Matteo Argenton, Valentina Amitrano, Concezio Bozzi,
Enrico Calore, Sebastiano Fabio Schifano

1° AI-INFN User Forum
Bologna, 11-12 Maggio 2024

INFN FERRARA

Università degli Studi di Ferrara

Finanziato dall'Unione europea NextGenerationEU

Ministero dell'Università e della Ricerca

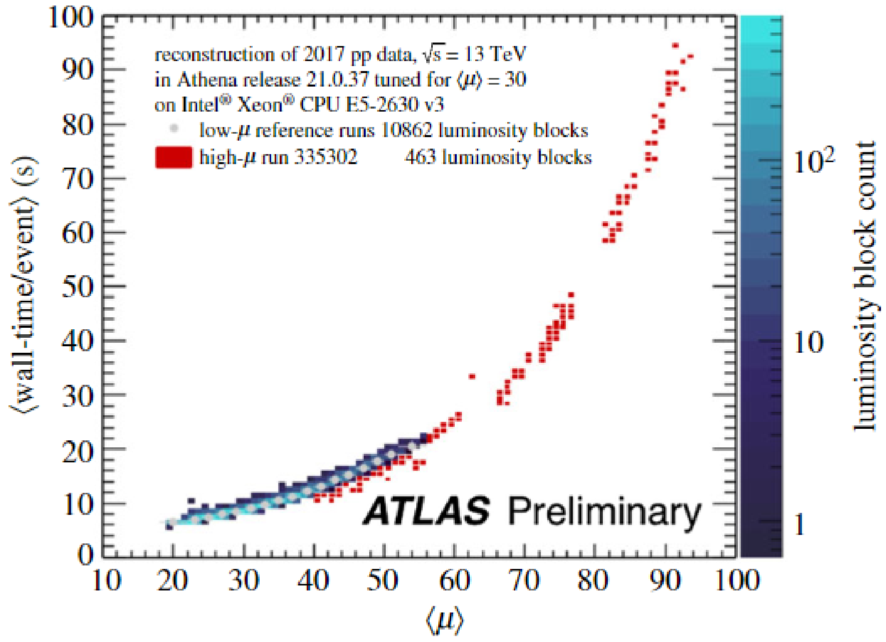
Italiadomani PIANO NAZIONALE DI RIPRESA E RESILIENZA

ICSC Centro Nazionale di Ricerca in HPC, Big Data and Quantum Computing

INTRODUCTION:
Track reconstruction problem with hybrid QGNN

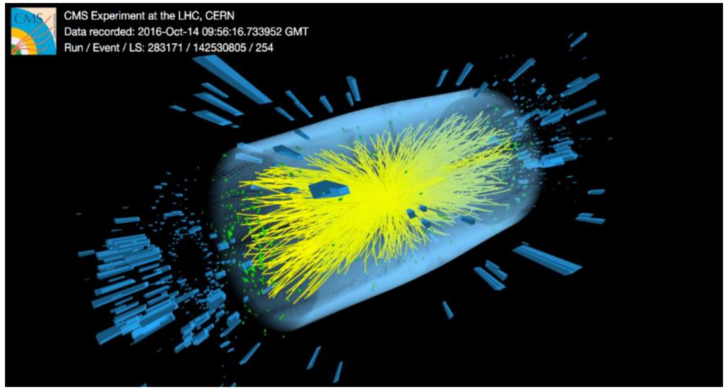
Scientific motivation

- The high-energy physics experiments at the LHC are already dealing with a huge number of particle tracks from the detectors, that need to be reconstructed
- With the High Luminosity LHC upgrade the number of proton-proton interactions per event will increase by a factor of 3-5 (140-200 collisions per beam crossing)
- A speedup in track reconstruction is mandatory

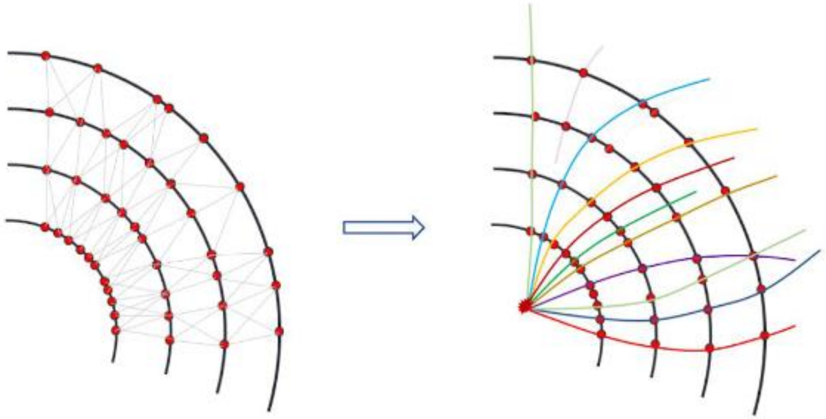


The track reconstruction problem

- An event in the LHC detectors corresponds to a particle beam crossing
  - Thousands of particles are spawned, producing hits when they interact with the detector layers
  - The average number of primary collisions per event is called pileup

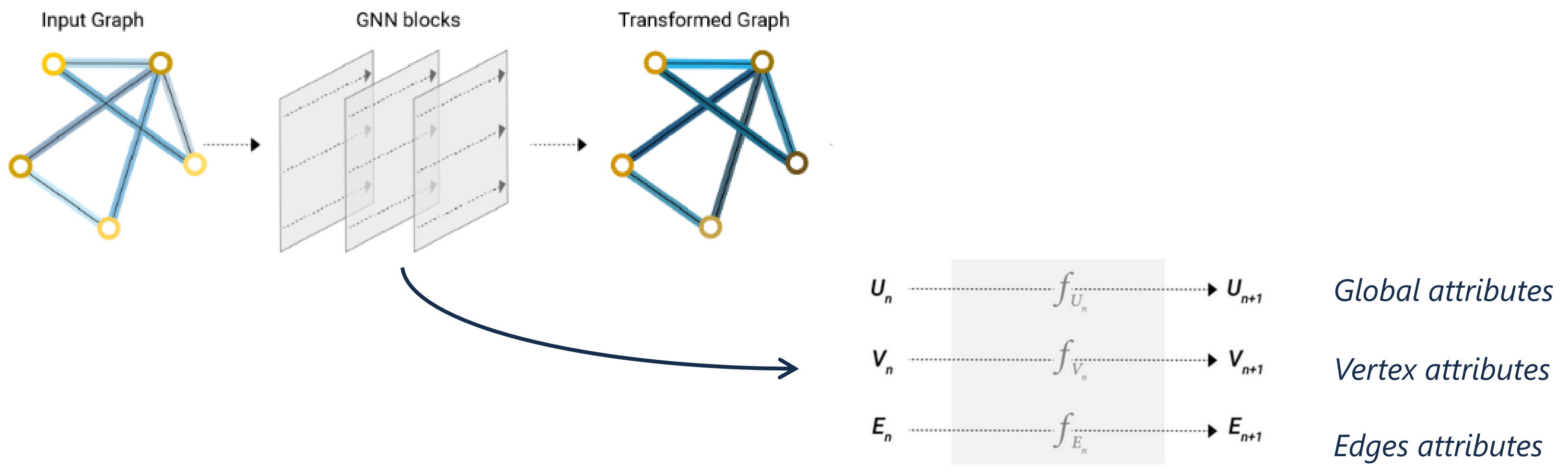


- Given a set of hits, the goal of track reconstruction is to assign labels to each of them
  - Perfect classification: all hits from a particle (and only those hits) share the same label
  - The result is a track that connects all the hits belonging to the same particle



Graph Neural Networks

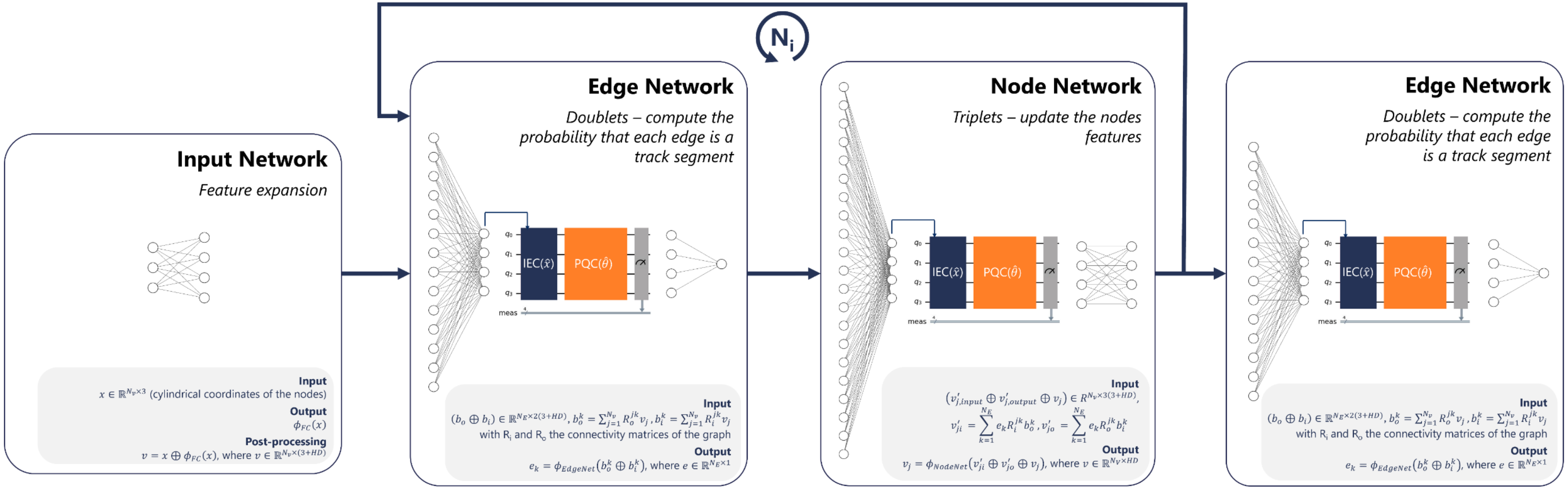
- A possible research direction is using Machine Learning
  - A GNN is an optimizable transformation on all attributes of the graph (nodes, edges, global context) that preserves graph symmetries (permutation invariances)
  - Global approach in contrast with the classical local approach



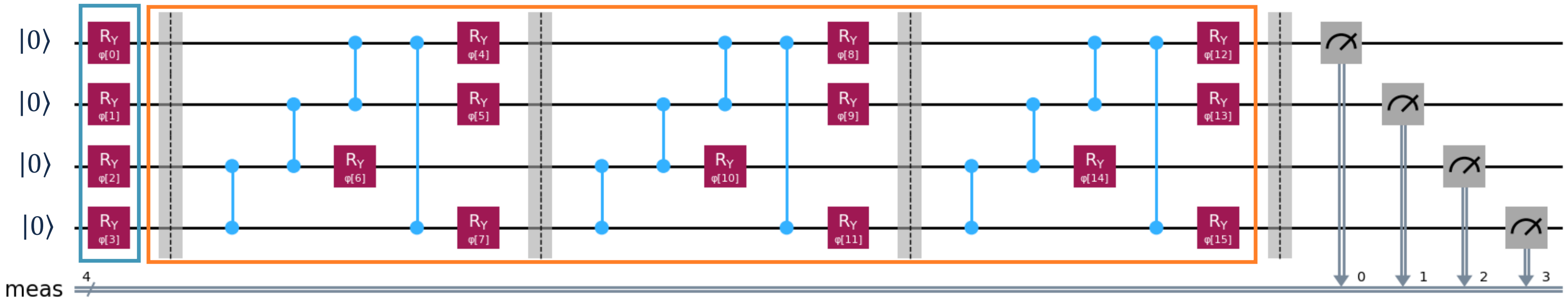
- Several groups are testing this approach with more or less promising results (e.g. EXATrkX collaboration)

Hybrid quantum GNN

- Working with the CERN Quantum Technology Initiative, we are exploring a **hybrid approach**
  - The aim is to see if there could be a quantum advantage (e.g. using parametric quantum circuits as GNN's layers)



The quantum circuit



- The quantum layer consists of:
  - An Information Encoding Circuit (IEC)
    - stores classical data into quantum states using angle encoding
  - A Parametrized Quantum Circuit (PQC)
    - rotates the input states in the Hilbert space depending on the angle parameters of the gates
    - generates entanglement between the qubits
  - Measurement of the final state
- The PQC parameters are trained to minimize the global loss function

Which technologies can we use?
Which hardware?

Quantum ML frameworks

- Most vendors are developing their own ecosystem
- Three main technologies for implementing Quantum ML Python applications:

Google AI Quantum   IBM Quantum   PENNYLANE

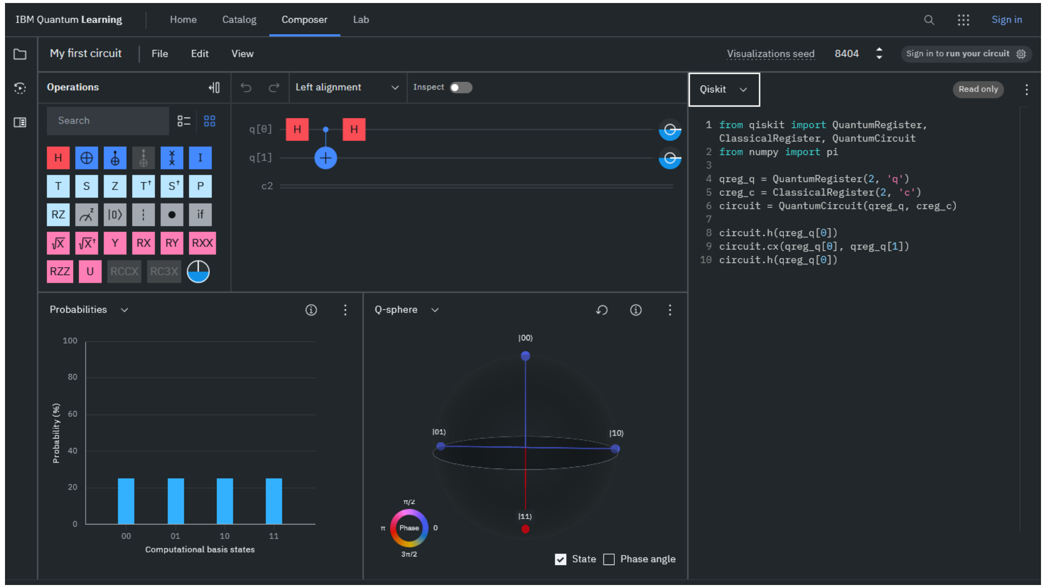
- INFN has signed an agreement with CERN to use IBM quantum hardware
  - The agreement has just expired on the 15 May 2024
- INFN is one of the main developers of QIBO
  - Open source full stack API for quantum simulation and quantum hardware control

Google AI Quantum

- Google ecosystem includes:
  - Cirq open source quantum framework for building algorithms on the NISQ era processors
  - Libraries:
    - qsim
    - TensorFlow Quantum
    - Open Fermion
  - Third Party Extensions: Pennylane, Alpine Quantum Technologies (trapped ion device), Pasqal (neutral atom)
  - Documentation with ready-to-use tutorials
  - VMs to run code on quantum simulator
- Our experience:
  - We have run tests on local simulator since the “original” code is written in Cirq + TFQ
  - We didn’t choose Google because we don’t have access to Google HW

IBM Quantum

- IBM ecosystem includes:
  - Qiskit open source SDK for working with quantum computers, both at the quantum circuits level and at higher level libraries
  - quantum hardware computing time: 10 min/month free plan vs. 600 min/month premium plan
  - Documentation and learning tools (e.g. the composer)
  - Slack channel to connect the community

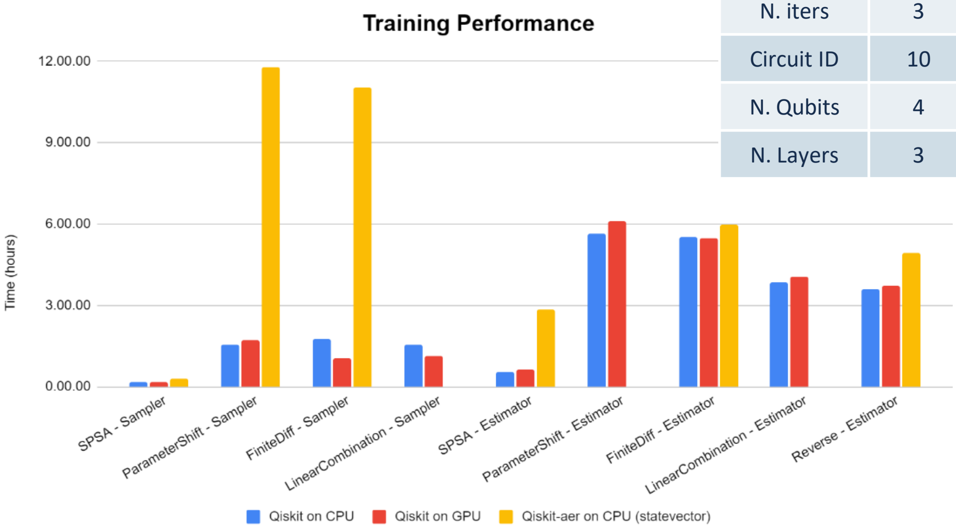


- Drawback:
  - Qiskit 1.0.0 release out in February 2024
    - Before that, a new release every month (quite unstable developing phase, even for the documentation)
  - IBM doesn’t provide functionality for ML
    - for QML it is necessary to integrate Qiskit with a third-party ML library (pyTorch is suggested)

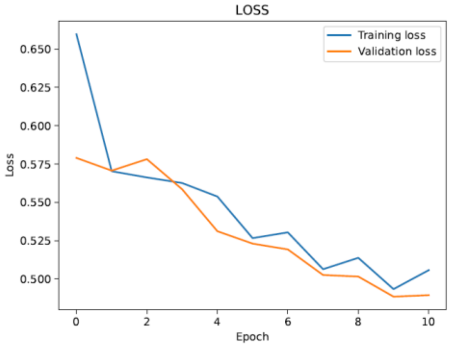
Our experience with IBMQ

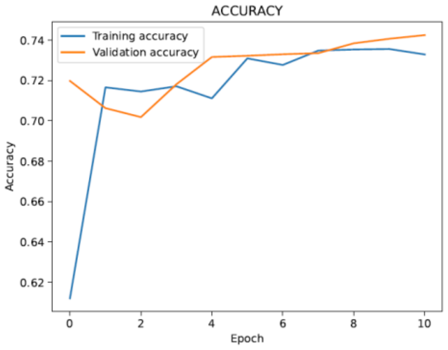
- We have implemented the hybrid NN with Qiskit + pyTorch
- Some issues we have encountered:
  - Poor support for QML
  - Very slow backpropagation with TorchConnector
    - Several tests with
      - different backpropagation algorithms
      - different simulators on both CPU and GPU (using NVIDIA cuQuantum SDK)
  - Tests on quantum hardware are slower
    - Queue time, data exchange, ...

Hyperparameters	
Hid dim	4
N. iters	3
Circuit ID	10
N. Qubits	4
N. Layers	3



Computing time of 1 epoch with 1 graph for training and 1 graph for validation (best case 10 min: ~450 sec training, ~ 150 sec validation)



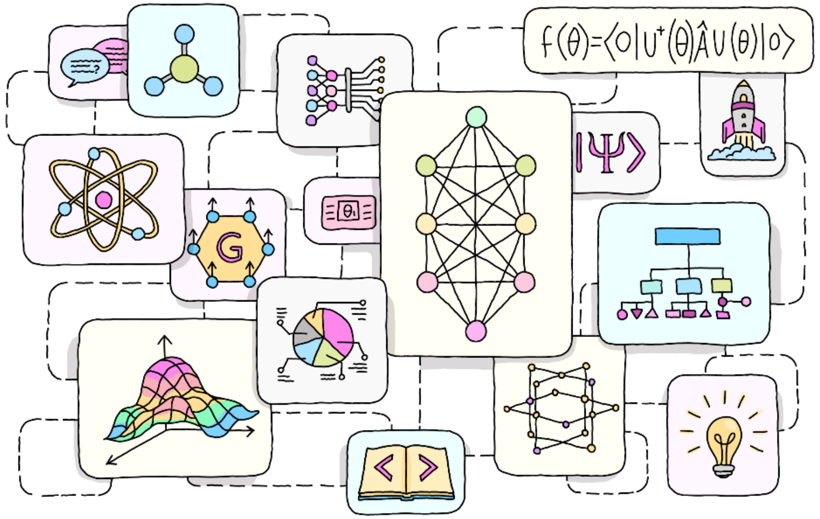


A training of the quantum network, with 50 training graphs and 50 validation graphs, for 10 epochs. The training takes about 25 hours per epoch

PENNYLANE

INFN FERRARA

- PennyLane is a cross-platform Python library by Xanadu for programming quantum computers
  - connects quantum computing to some ML frameworks, such as NumPy’s autograd, JAX, PyTorch, and TensorFlow, making them quantum-aware
  - implements the differentiable programming paradigm
    - backend-independent: circuits can be run on various kinds of simulators or hardware devices without making any changes
    - integrated with external hardware (e.g. IBM’s Qiskit, Google’s Cirq, Rigetti’s Forest, …)
    - implements a simulator that offloads quantum gate calls to the NVIDIA cuQuantum SDK
- Global community (documentation, blog, forum, support, …)

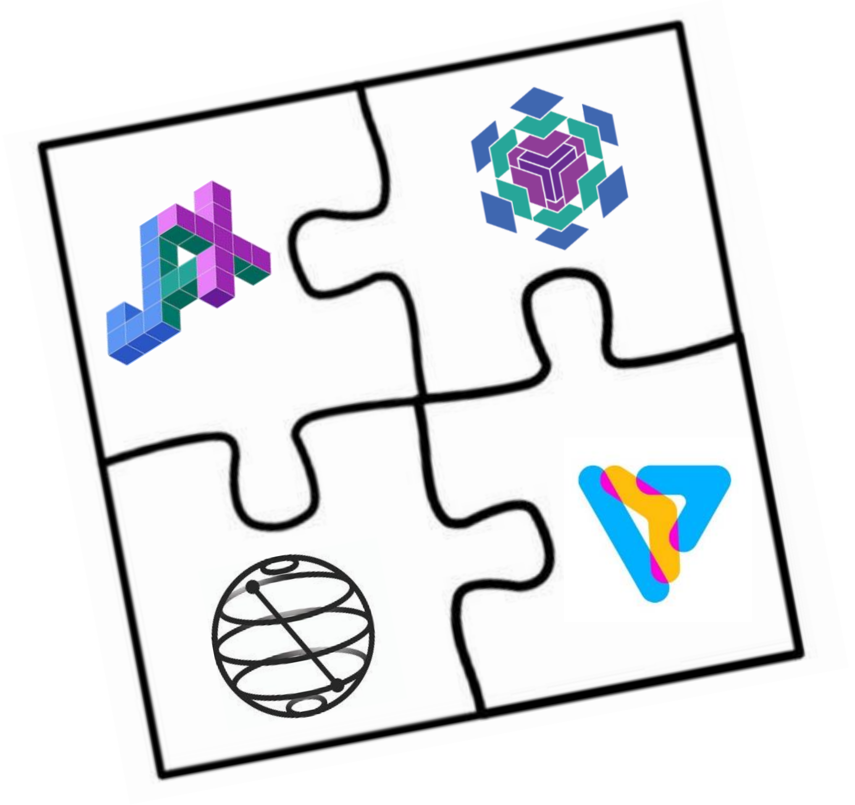


Our experience with Pennylane

- To run on IBM's backends:
  - Pennylane with pyTorch doesn't improve the training time
  - Pennylane with JAX was the game changer
- JAX is a Python library for accelerator-oriented array computation designed for high-performance numerical computing and large-scale machine learning
  - We use Flax to implement the NN
- From the 10 min Qiskit's best case, JAX and Pennylane take 30 sec for one epoch of 1 training and one validation graph on Qiskit simulator backend

Chosen frameworks

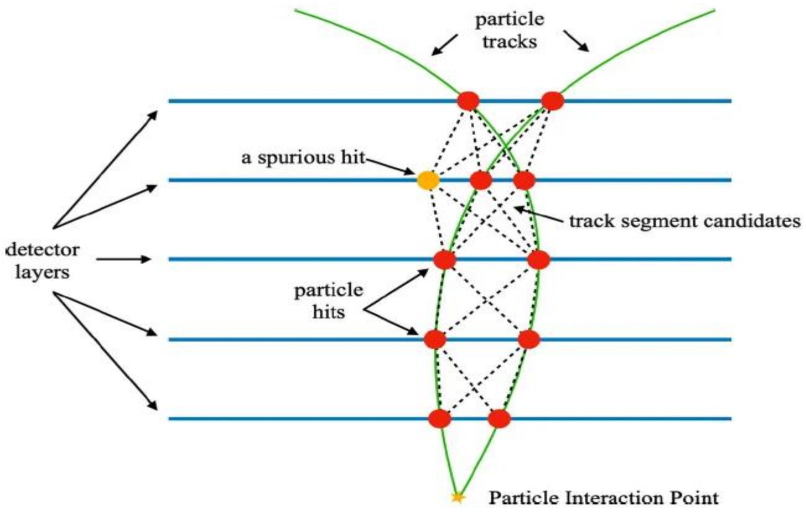
- Summary of the frameworks we have chosen:
  - Data is stored in Jax format
  - The Neural Network is defined in Flax
  - Quantum circuits are implemented in Pennylane
  - The backend for the training is the IBM Qiskit-aer simulator called by Pennylane, but the goal is to run inference on IBM quantum hardware



FIRST GOAL:
hybrid network scalability tests on (noisy) quantum hardware

Dataset and preprocessing

- Goal of preprocessing:
  - select events with pileup 10, 50, 100, 150 and 200
  - prepare the data to feed the model
- We use the [TrackML Challenge dataset]
  - Collection of thousands of simulated events with average pileup 200
  - Each event is a set of hits, so we need to build the associated graph structure

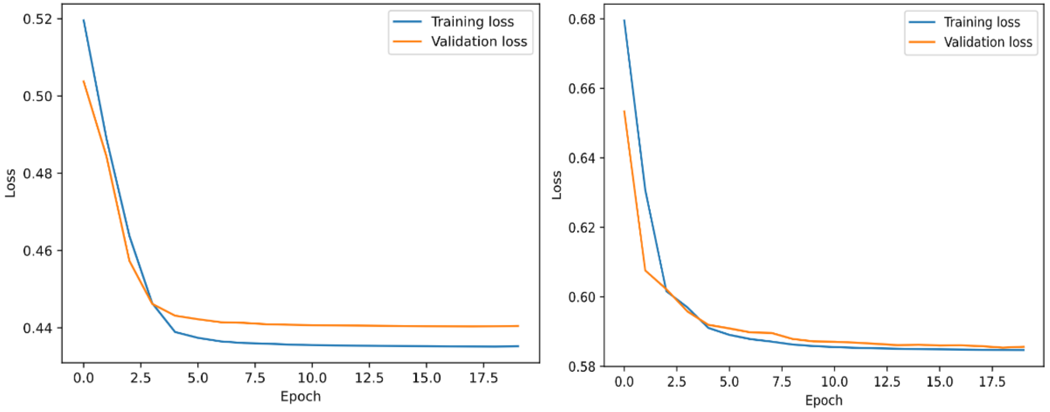


- An event is coded as a graph where:
  - Nodes are hits in a detector layer
  - Edges are track segments
  - Connections between hits in adjacent layers can be seen as candidate edges
- The network should learn to recognize true and fake edges

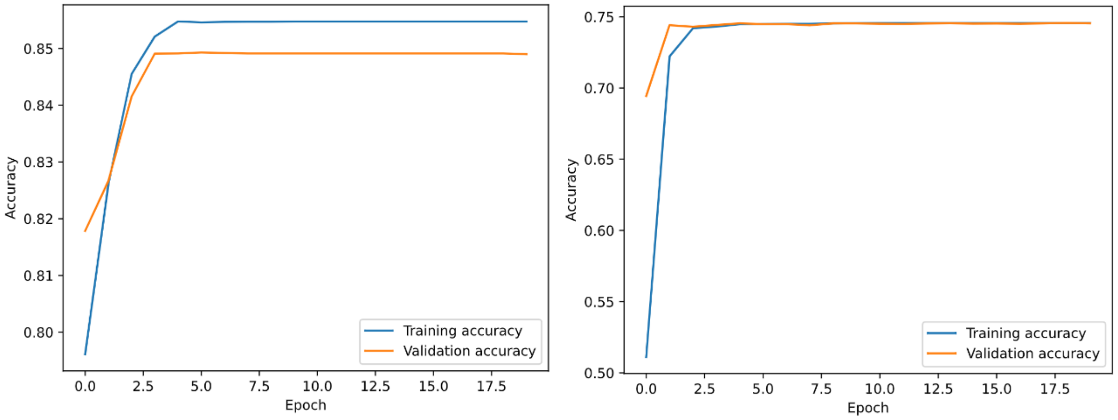
Training

- We have trained the network to perform scalability tests
  - graphs of pileup 10, 50, 100, 150 and 200
  - 35 training graphs and 10 validation graphs
  - Noiseless local simulator: jax-pennylane default backend

Loss on pileup 50 and 200



Accuracy on pileup 50 and 200

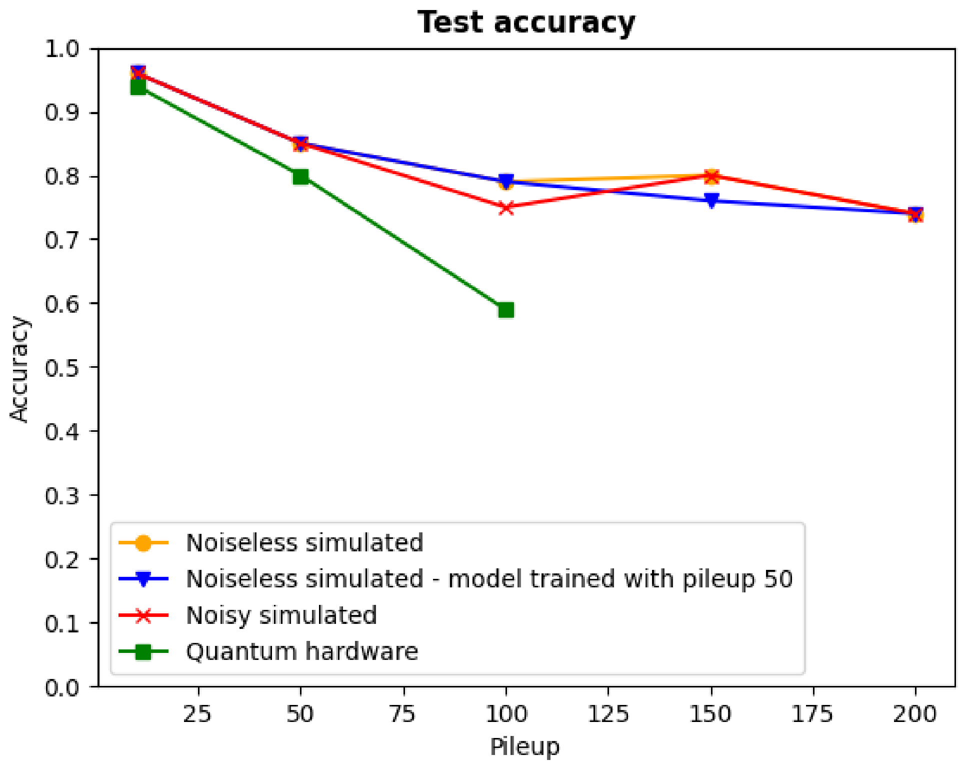


Inference

- We have run the tests on 10 graphs with different backends:
  - Noiseless Qiskit and Pennylane simulators
  - Noisy Qiskit and Pennylane simulators
  - Noiseless Pennylane simulator fixing a model of pileup 50
  - IBM's quantum hardware (IBM_Osaka)

pileup	Accuracy on noisless simulator (training model to match pileup)	Accuracy on noisless simulator (training model on pileup 50)	Accuracy on noisy simulator	Accuracy on quantum hardware* (IBM Osaka)
10	0.96	0.96	0.96	0.94
50	0.85	0.85	0.85	0.80
100	0.79	0.79	0.75	0.59
150	0.80	0.76	0.80	-
200	0.74	0.74	0.74	-

**Test set reduced due to issues in QPU time and resources availability*



Summary

Summary

- Finding the optimal combination of tools for QML projects is strictly related to the available hardware
  - QC frameworks are still in a development phase, as the quantum hardware
- On quantum hardware:
  - the inferred accuracies of the hybrid QGNN show a decrease compared to those obtained on noisy simulators
  - the execution time is still too long to allow training
- The inferred accuracies show that we could train model on small pileup and run tests using bigger pileup
- Further developments of our work could include the exploration of different encoding schemes, quantum circuits based on expressivity, and GNN architectures…

… Checking how QC frameworks will evolve!

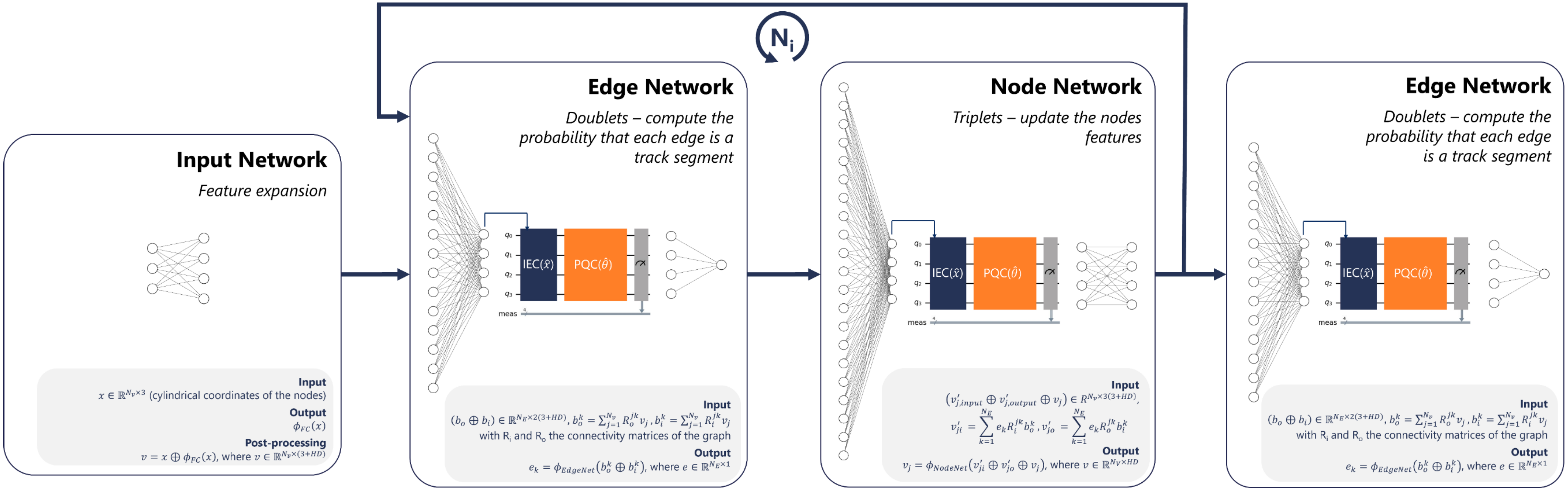
References

- *Graph Neural Network, https://distill.pub/2021/gnn-intro/*
- *EXATrkX, https://exatrkx.github.io/*
- *CERN QTI, https://quantum.cern/*
- *Google AI Quantum, https://quantumai.google*
- *IBM Quantum, https://www.ibm.com/quantum*
- *cuQuantum, https://developer.nvidia.com/cuquantum-sdk*
- *Pennylane, https://pennylane.ai/*
- *Jax, https://jax.readthedocs.io/en/latest/*
- *Flax, https://flax.readthedocs.io/en/latest/*
- *Qibo, https://qibo.science/*
- *TrackML, https://www.kaggle.com/competitions/trackml-particle-identification*
- *Tüysüz C. et al. Hybrid quantum classical graph neural networks for particle track reconstruction. Quantum Mach. Intell. 3, 29 (2021). https://doi.org/10.1007/s42484-021-00055-9*
- *Sim S et al. (2019) Expressibility and Entangling Capability of Parameterized Quantum Circuits for Hybrid Quantum-Classical Algorithms. Advanced Quantum Technologies 2(12):1900070, DOI:10.1002/qute.201900070*

Thank you for your attention

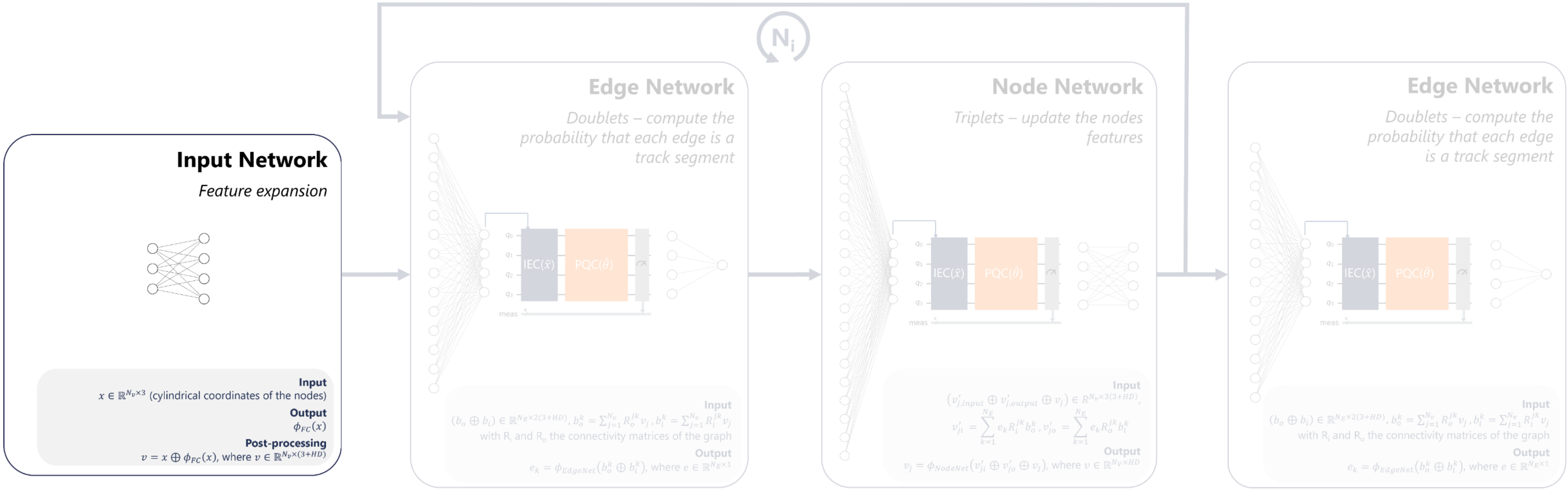
BACKUP

The network



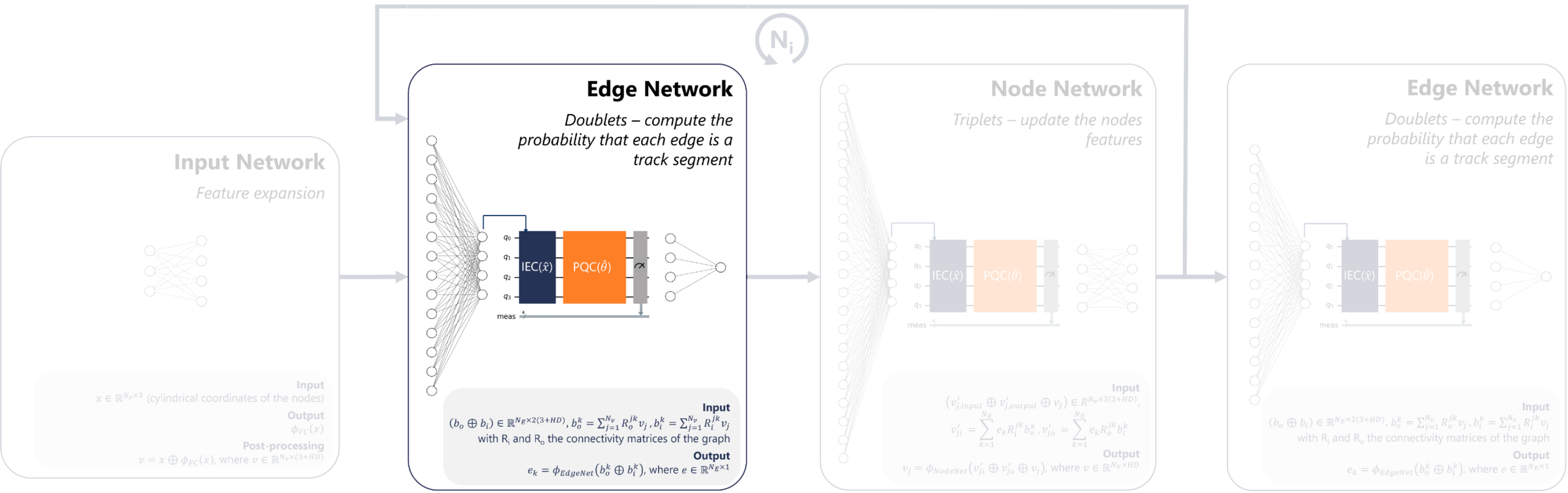
N_i
Input Network
Feature expansion
Input
$x \in \mathbb{R}^{N_v \times 3}$ (cylindrical coordinates of the nodes)
Output
$\phi_{FC}(x)$
Post-processing
$v = x \oplus \phi_{FC}(x)$, where $v \in \mathbb{R}^{N_v \times (3+HD)}$
Edge Network
Doublets – compute the probability that each edge is a track segment
q_0 q_1 q_2 q_3 meas
IEC($\hat{x}$)
PQC($\hat{\theta}$)
Input
$(b_o \oplus b_i) \in \mathbb{R}^{N_E \times 2(3+HD)}$, $b_o^k = \sum_{j=1}^{N_v} R_o^{jk} v_j$, $b_i^k = \sum_{j=1}^{N_v} R_i^{jk} v_j$
with R_i and R_o the connectivity matrices of the graph
Output
$e_k = \phi_{EdgeNet}(b_o^k \oplus b_i^k)$, where $e \in \mathbb{R}^{N_E \times 1}$
Node Network
Triplets – update the nodes features
q_0 q_1 q_2 q_3 meas
IEC($\hat{x}$)
PQC($\hat{\theta}$)
Input
$(v'_{j,input} \oplus v'_{j,output} \oplus v_j) \in R^{N_v \times 3(3+HD)}$,
$v'_{ji} = \sum_{k=1}^{N_E} e_k R_i^{jk} b_o^k$, $v'_{jo} = \sum_{k=1}^{N_E} e_k R_o^{jk} b_i^k$
Output
$v_j = \phi_{NodeNet}(v'_{ji} \oplus v'_{jo} \oplus v_j)$, where $v \in \mathbb{R}^{N_V \times HD}$
Edge Network
Doublets – compute the probability that each edge is a track segment
q_0 q_1 q_2 q_3 meas
IEC($\hat{x}$)
PQC($\hat{\theta}$)
Input
$(b_o \oplus b_i) \in \mathbb{R}^{N_E \times 2(3+HD)}$, $b_o^k = \sum_{j=1}^{N_v} R_o^{jk} v_j$, $b_i^k = \sum_{j=1}^{N_v} R_i^{jk} v_j$
with R_i and R_o the connectivity matrices of the graph
Output
$e_k = \phi_{EdgeNet}(b_o^k \oplus b_i^k)$, where $e \in \mathbb{R}^{N_E \times 1}$

The network

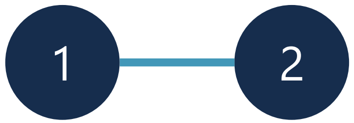


N_i
Input Network
Feature expansion
Input
$x \in \mathbb{R}^{N_v \times 3}$ (cylindrical coordinates of the nodes)
Output
$\phi_{FC}(x)$
Post-processing
$v = x \oplus \phi_{FC}(x)$, where $v \in \mathbb{R}^{N_v \times (3+HD)}$
Edge Network
Doublets – compute the probability that each edge is a track segment
IEC($\vec{x}$)
PQC($\vec{\theta}$)
meas
Input
with R_i and R_o the connectivity matrices of the graph
Output
Node Network
Triplets – update the nodes features
IEC($\vec{x}$)
PQC($\vec{\theta}$)
meas
Input
Output
Edge Network
Doublets – compute the probability that each edge is a track segment
IEC($\vec{x}$)
PQC($\vec{\theta}$)
meas
Input
with R_i and R_o the connectivity matrices of the graph
Output

The network

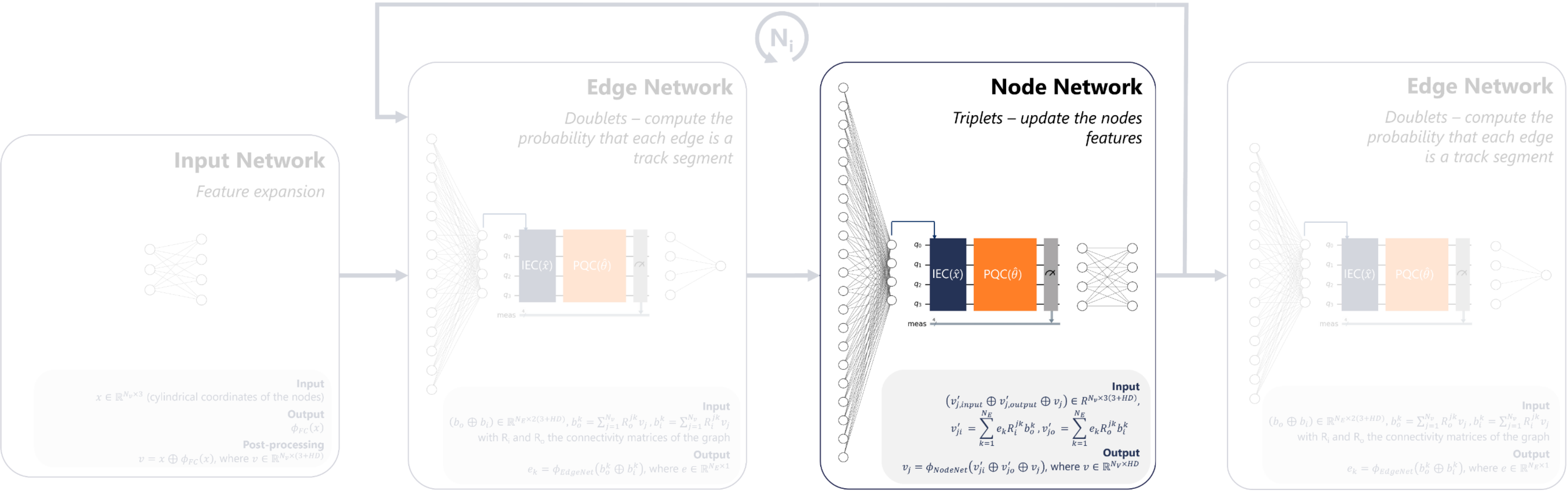


Input Network
Feature expansion
Input
$x \in \mathbb{R}^{N_V \times 3}$ (cylindrical coordinates of the nodes)
Output
$\phi_{FC}(x)$
Post-processing
$v = x \oplus \phi_{FC}(x)$, where $v \in \mathbb{R}^{N_V \times (3+HD)}$
Edge Network
Doublets – compute the probability that each edge is a track segment
q_0
q_1
q_2
q_3
IEC($\hat{x}$)
PQC($\hat{\theta}$)
meas
Input
$(b_o \oplus b_i) \in \mathbb{R}^{N_E \times 2(3+HD)}$, $b_o^k = \sum_{j=1}^{N_v} R_o^{jk} v_j$, $b_i^k = \sum_{j=1}^{N_v} R_i^{jk} v_j$
with R_i and R_o the connectivity matrices of the graph
Output
$e_k = \phi_{EdgeNet}(b_o^k \oplus b_i^k)$, where $e \in \mathbb{R}^{N_E \times 1}$
N_i
Node Network
Triplets – update the nodes features
Input
Output
Edge Network
Doublets – compute the probability that each edge is a track segment
Input
Output

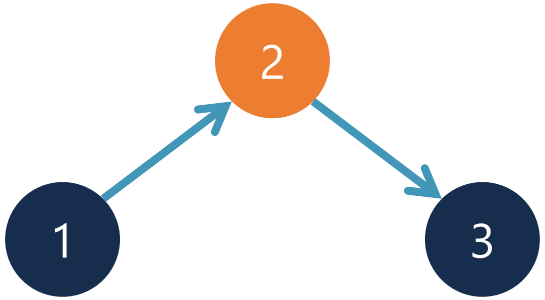


1
2

The network

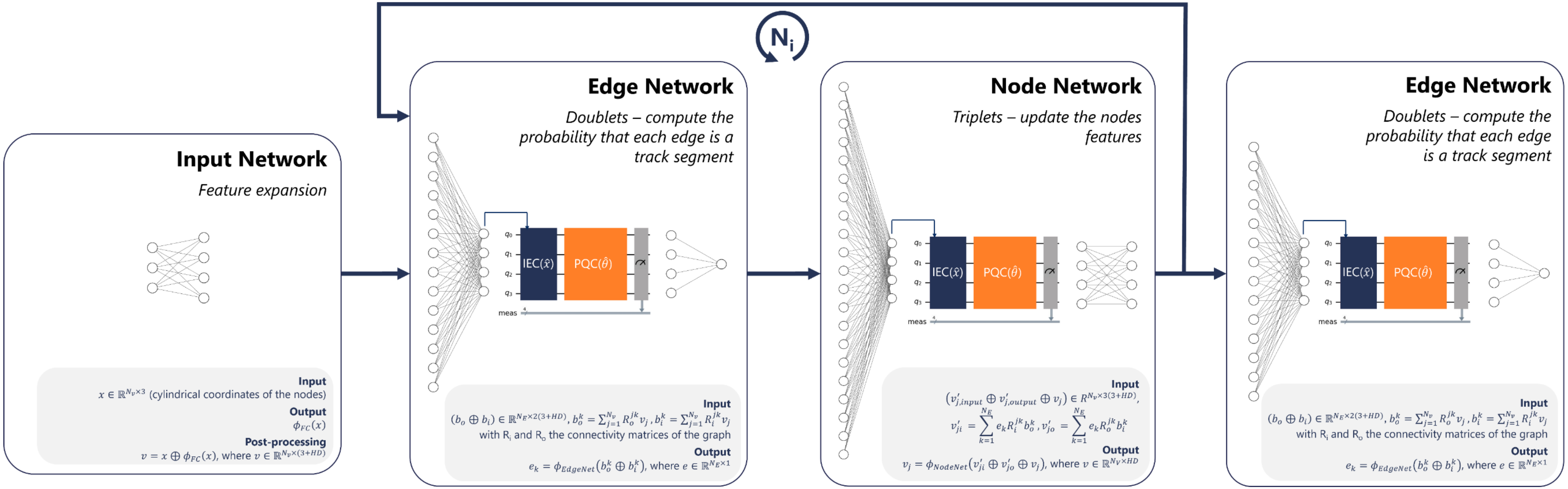


N_i
Input Network
Feature expansion
Input
$x \in \mathbb{R}^{N_V \times 3}$ (cylindrical coordinates of the nodes)
Output
$\phi_{FC}(x)$
Post-processing
$v = x \oplus \phi_{FC}(x)$, where $v \in \mathbb{R}^{N_V \times (3+HD)}$
Edge Network
Doublets – compute the probability that each edge is a track segment
q_0 q_1 q_2 q_3
IEC($\hat{x}$)
PQC($\hat{\theta}$)
meas
Input
$(b_o \oplus b_i) \in \mathbb{R}^{N_E \times 2(3+HD)}$, $b_o^k = \sum_{j=1}^{N_V} R_o^{jk} v_j$, $b_i^k = \sum_{j=1}^{N_V} R_i^{jk} v_j$
with R_i and R_o the connectivity matrices of the graph
Output
$e_k = \phi_{EdgeNet}(b_o^k \oplus b_i^k)$, where $e \in \mathbb{R}^{N_E \times 1}$
Node Network
Triplets – update the nodes features
q_0 q_1 q_2 q_3
IEC($\hat{x}$)
PQC($\hat{\theta}$)
meas
Input
$(v'_{j,input} \oplus v'_{j,output} \oplus v_j) \in R^{N_v \times 3(3+HD)}$,
$v'_{ji} = \sum_{k=1}^{N_E} e_k R_i^{jk} b_o^k$, $v'_{jo} = \sum_{k=1}^{N_E} e_k R_o^{jk} b_i^k$
Output
$v_j = \phi_{NodeNet}(v'_{ji} \oplus v'_{jo} \oplus v_j)$, where $v \in \mathbb{R}^{N_V \times HD}$
Edge Network
Doublets – compute the probability that each edge is a track segment
q_0 q_1 q_2 q_3
IEC($\hat{x}$)
PQC($\hat{\theta}$)
meas
Input
$(b_o \oplus b_i) \in \mathbb{R}^{N_E \times 2(3+HD)}$, $b_o^k = \sum_{j=1}^{N_V} R_o^{jk} v_j$, $b_i^k = \sum_{j=1}^{N_V} R_i^{jk} v_j$
with R_i and R_o the connectivity matrices of the graph
Output
$e_k = \phi_{EdgeNet}(b_o^k \oplus b_i^k)$, where $e \in \mathbb{R}^{N_E \times 1}$



2
1
3

The network



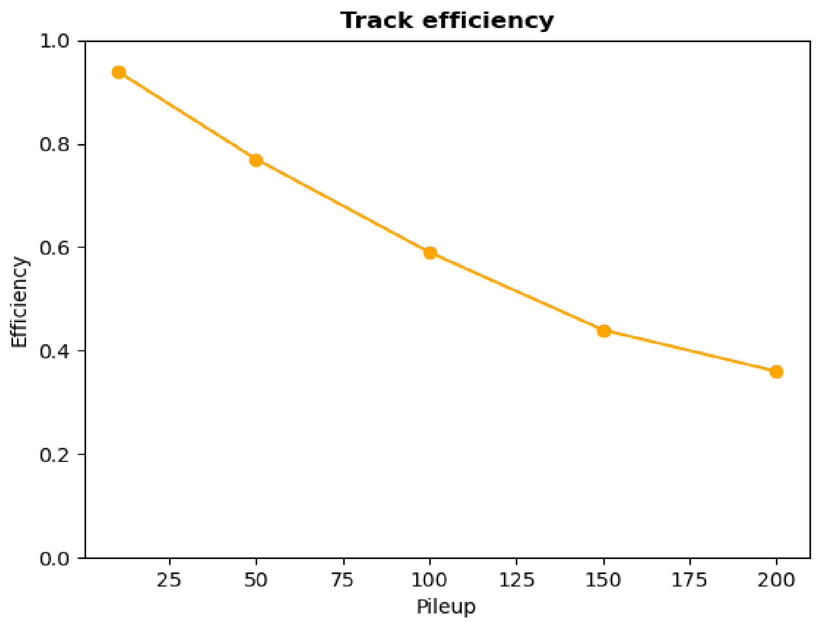
N_i
Input Network
Feature expansion
Input
$x \in \mathbb{R}^{N_v \times 3}$ (cylindrical coordinates of the nodes)
Output
$\phi_{FC}(x)$
Post-processing
$v = x \oplus \phi_{FC}(x)$, where $v \in \mathbb{R}^{N_v \times (3+HD)}$
Edge Network
Doublets – compute the probability that each edge is a track segment
q_0
q_1
q_2
q_3
meas
IEC($\hat{x}$)
PQC($\hat{\theta}$)
Input
$(b_o \oplus b_i) \in \mathbb{R}^{N_E \times 2(3+HD)}$, $b_o^k = \sum_{j=1}^{N_v} R_o^{jk} v_j$, $b_i^k = \sum_{j=1}^{N_v} R_i^{jk} v_j$
with R_i and R_o the connectivity matrices of the graph
Output
$e_k = \phi_{EdgeNet}(b_o^k \oplus b_i^k)$, where $e \in \mathbb{R}^{N_E \times 1}$
Node Network
Triplets – update the nodes features
q_0
q_1
q_2
q_3
meas
IEC($\hat{x}$)
PQC($\hat{\theta}$)
Input
$(v'_{j,input} \oplus v'_{j,output} \oplus v_j) \in R^{N_v \times 3(3+HD)}$,
$v'_{ji} = \sum_{k=1}^{N_E} e_k R_i^{jk} b_o^k$, $v'_{jo} = \sum_{k=1}^{N_E} e_k R_o^{jk} b_i^k$
Output
$v_j = \phi_{NodeNet}(v'_{ji} \oplus v'_{jo} \oplus v_j)$, where $v \in \mathbb{R}^{N_V \times HD}$
Edge Network
Doublets – compute the probability that each edge is a track segment
q_0
q_1
q_2
q_3
meas
IEC($\hat{x}$)
PQC($\hat{\theta}$)
Input
$(b_o \oplus b_i) \in \mathbb{R}^{N_E \times 2(3+HD)}$, $b_o^k = \sum_{j=1}^{N_v} R_o^{jk} v_j$, $b_i^k = \sum_{j=1}^{N_v} R_i^{jk} v_j$
with R_i and R_o the connectivity matrices of the graph
Output
$e_k = \phi_{EdgeNet}(b_o^k \oplus b_i^k)$, where $e \in \mathbb{R}^{N_E \times 1}$

Track efficiency

- Track efficiency is computed as:

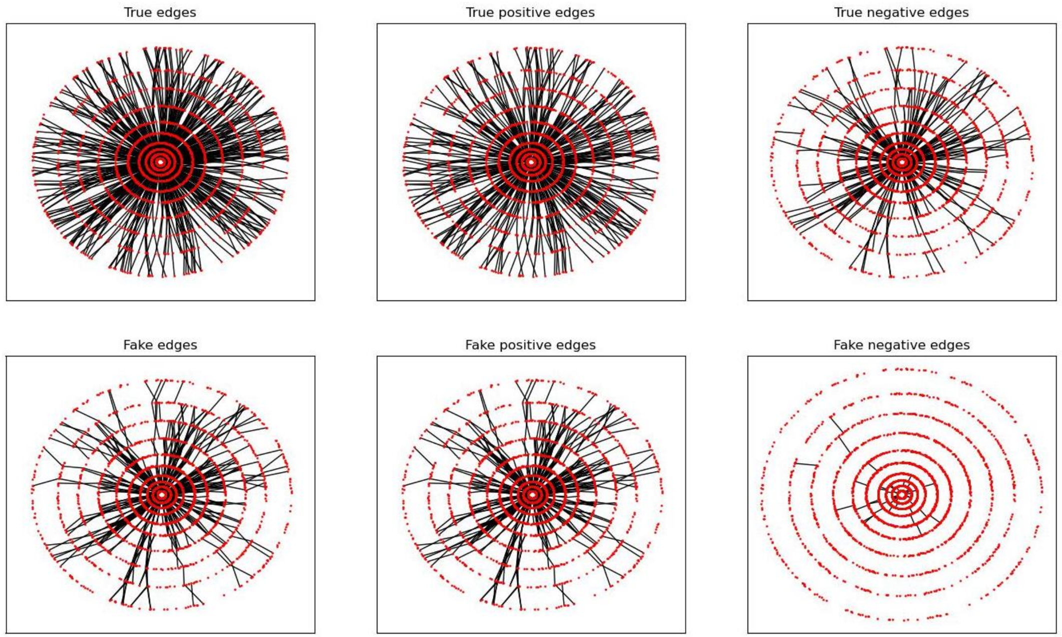
$$\frac{\#\ correctly\ reconstructed\ tracks}{\#\ tracks}$$

Correctly = 70% of correctly identified edges



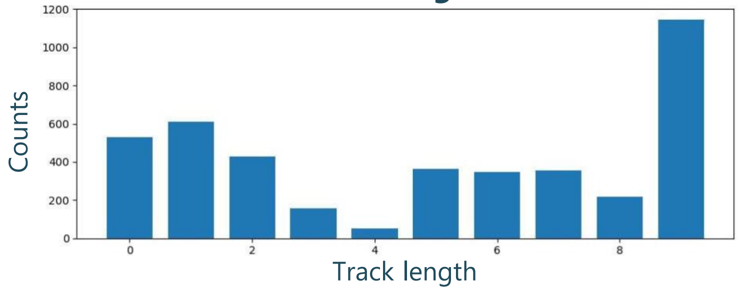
pileup	Particles detected	Track efficiency*
10	46	0.94
50	206	0.77
100	420	0.59
150	668	0.44
200	804	0.36

Network's errors

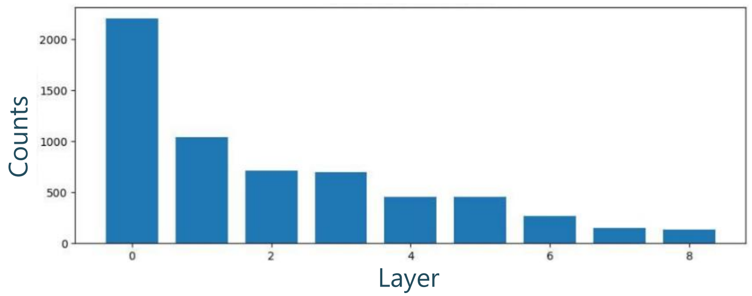


Graph with pileup 200

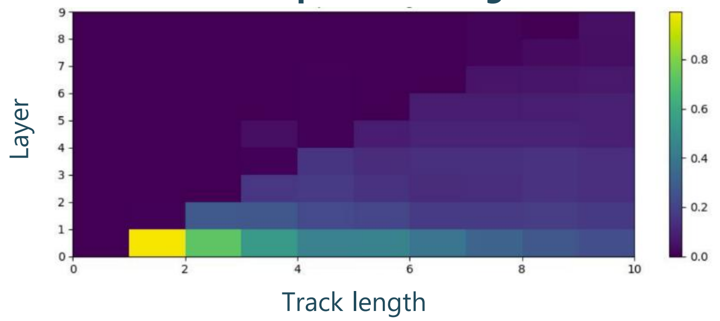
Tracks length



Error per layer



Error per track length



Analysis on 10 graphs with pileup 150