

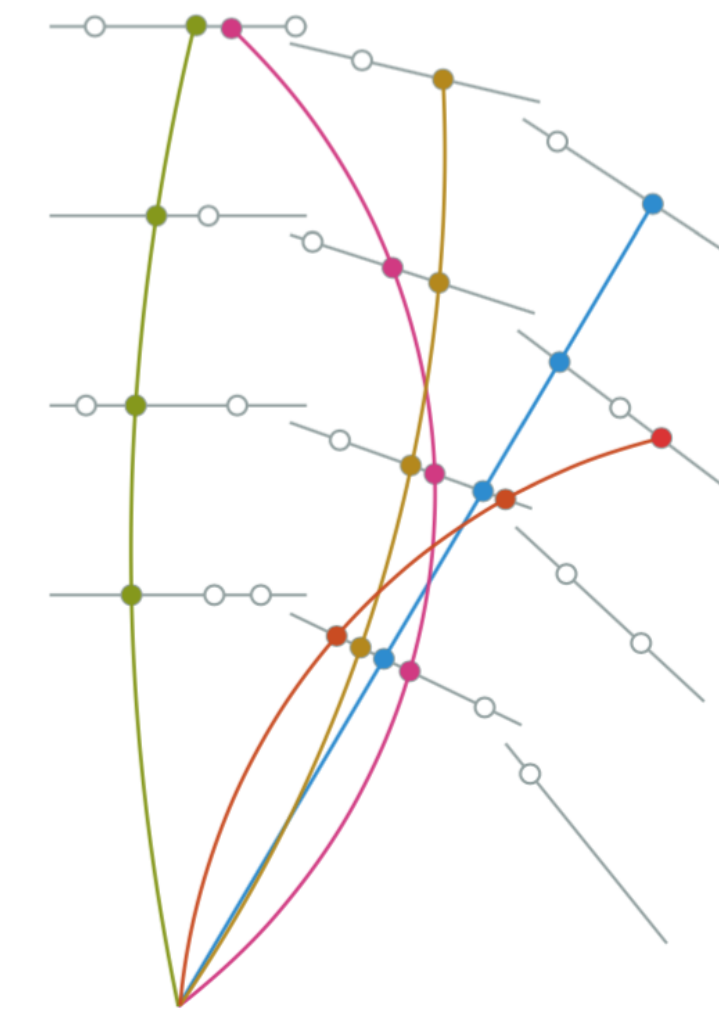
A Fast Track Reconstruction via GNN on FPGA for PANDA

by Greta Heine on Oct 24, 2022



Motivation

- accurate tracking of particle trajectories in HEP collider experiments is crucial to ensure quality of novel particle detection
- state-of-the-art track reconstruction algorithms based on Kalman filters scale computationally worse than quadratically in the number of detector hits
- Graph Neural Networks (GNNs) have already been successfully applied to this task, e.g. by DeZoort et al., 2021 and W. Esmail, 2022
- reduce computational cost by implementation on field-programmable gate arrays (FPGA)

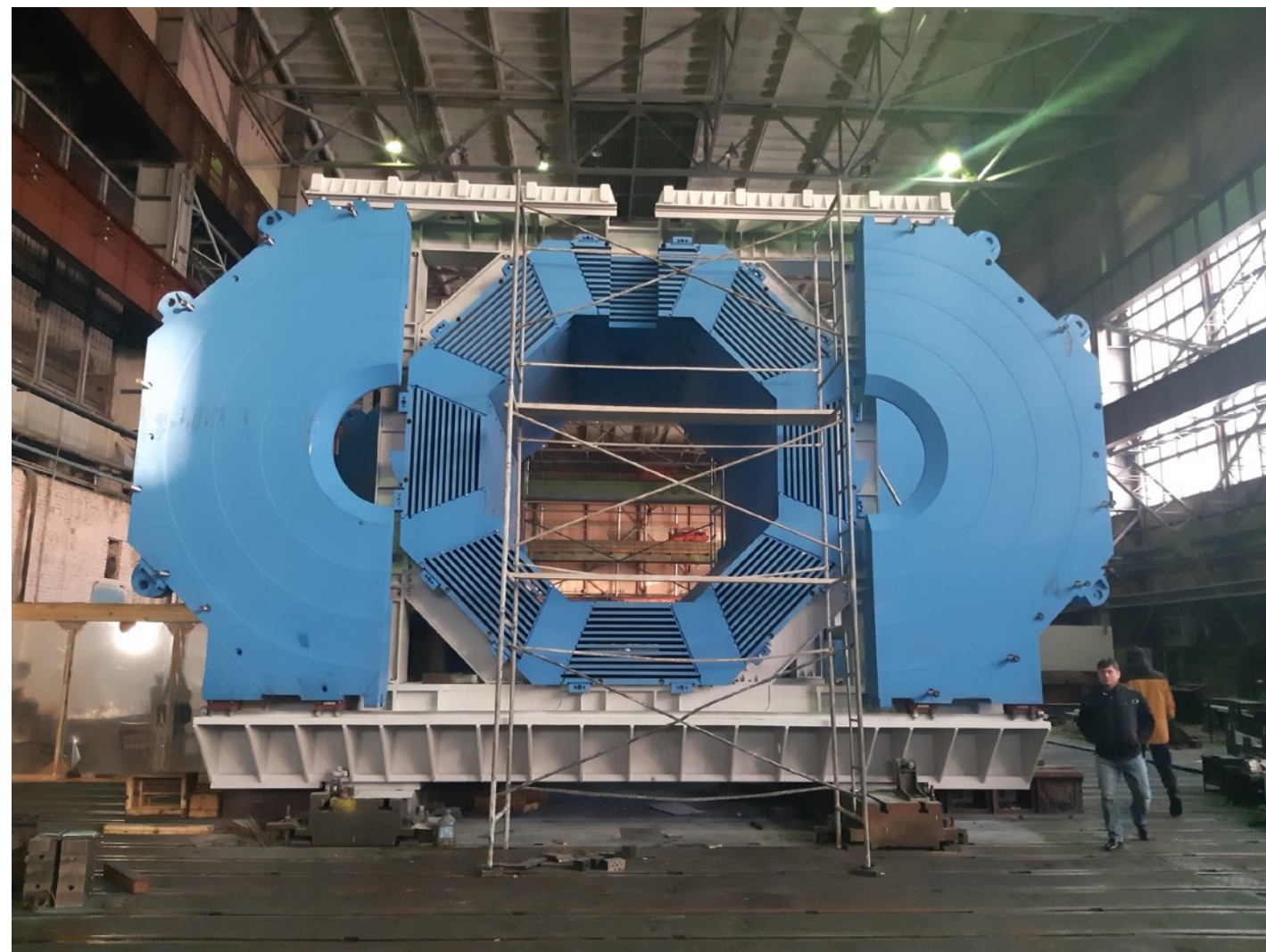


<https://arxiv.org/abs/1904.06778>



PANDA (anti-Proton ANnihilation at DArmstadt)

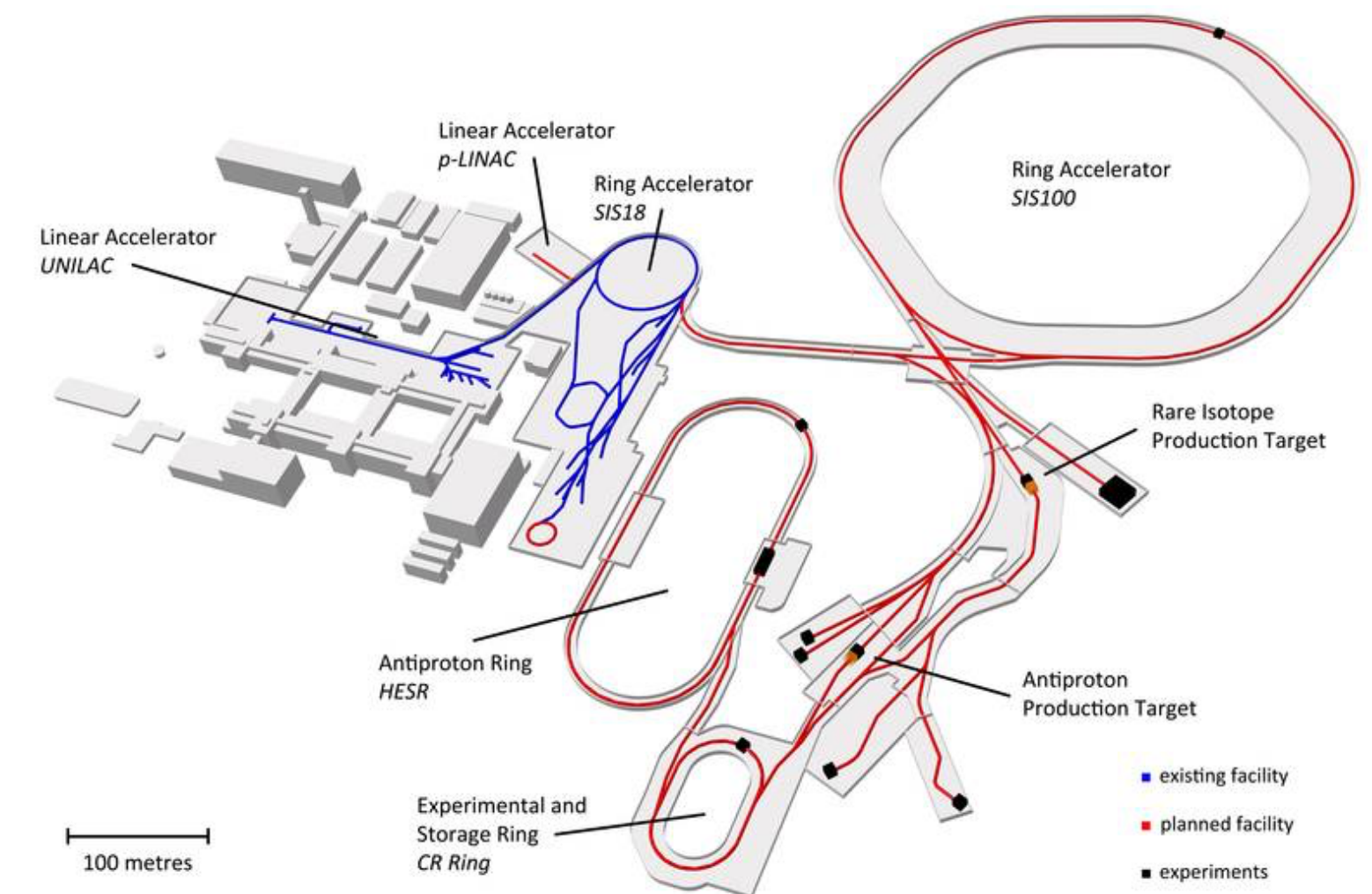
- 1 of 4 experiments at FAIR (GSI Darmstadt) which is currently under construction
- 1.5 - 15 GeV antiproton beam at fixed target
- investigation of hadron physics in the charm and multi-strange hadron sectors



<https://panda.gsi.de/article/doors-wide-open-for-panda>



<https://www.faz.net/aktuell/rhein-main/wegen-corona-jubilaeum-von-gsi-und-fair-nur-im-internet-16740977.html>



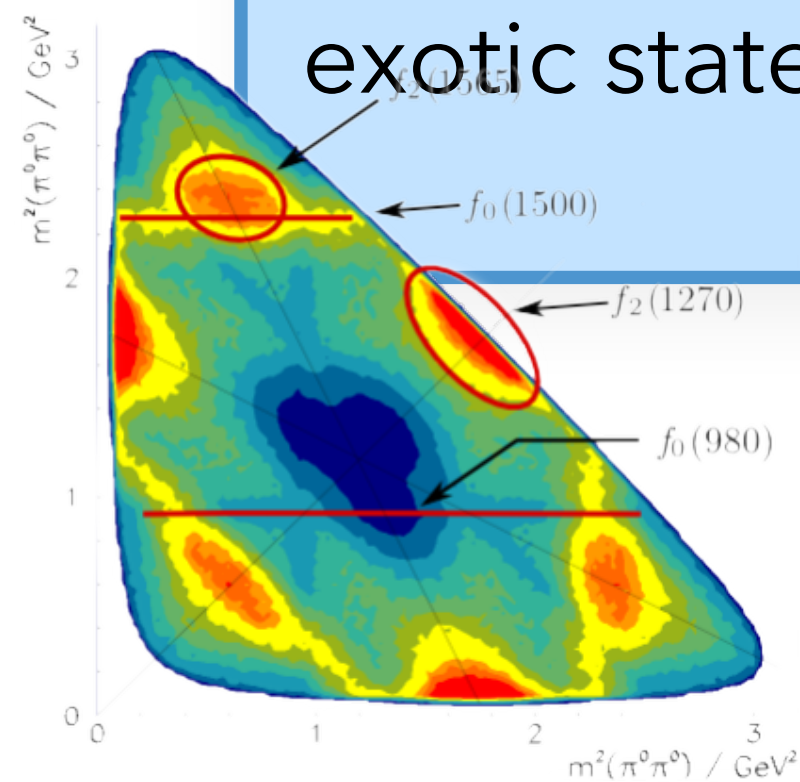
<https://www.ep1.rub.de/PandaBMBF/index.php/pandaphysik/forschungsprogramm>

PANDA Physics Program

Hadron spectroscopy

$$p\bar{p} \rightarrow$$

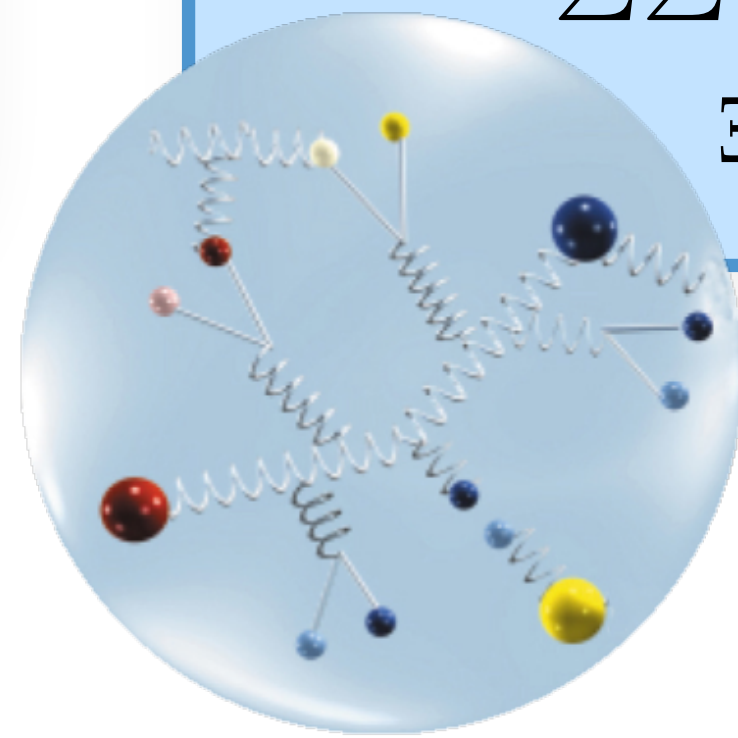
ϕ ω XYZ
 η glueballs
 D J/Ψ
 exotic states D_s



Hyperon physics

$$p\bar{p} \rightarrow$$

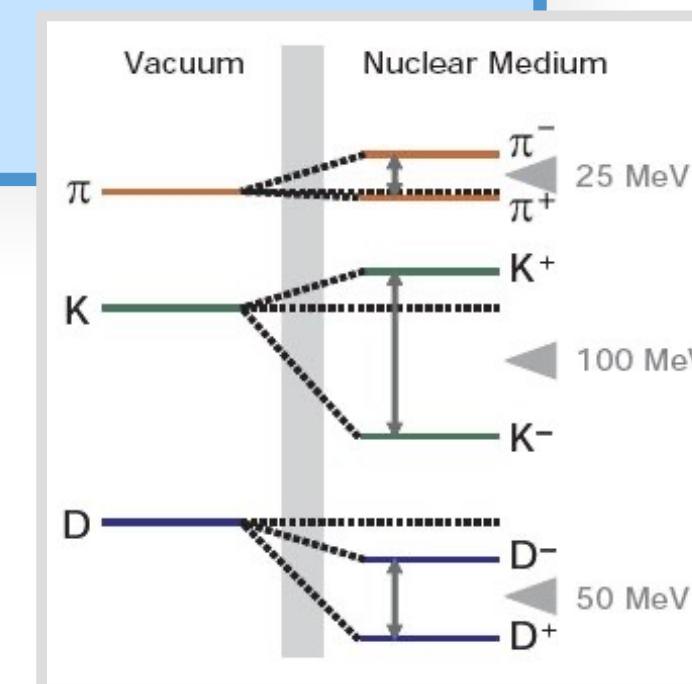
pairproduction
 $|S| = 1, 2, 3$
 $|C| = 1$
 $\Sigma \bar{\Sigma}$ $\Lambda \bar{\Lambda}$
 $\Xi \bar{\Xi}$ $\Omega \bar{\Omega}$



Proton structure

$$p\bar{p} \rightarrow$$

e^+e^-
 $\mu^+\mu^-$
 $e^+e^-\pi^0$



Hadrons in nuclei

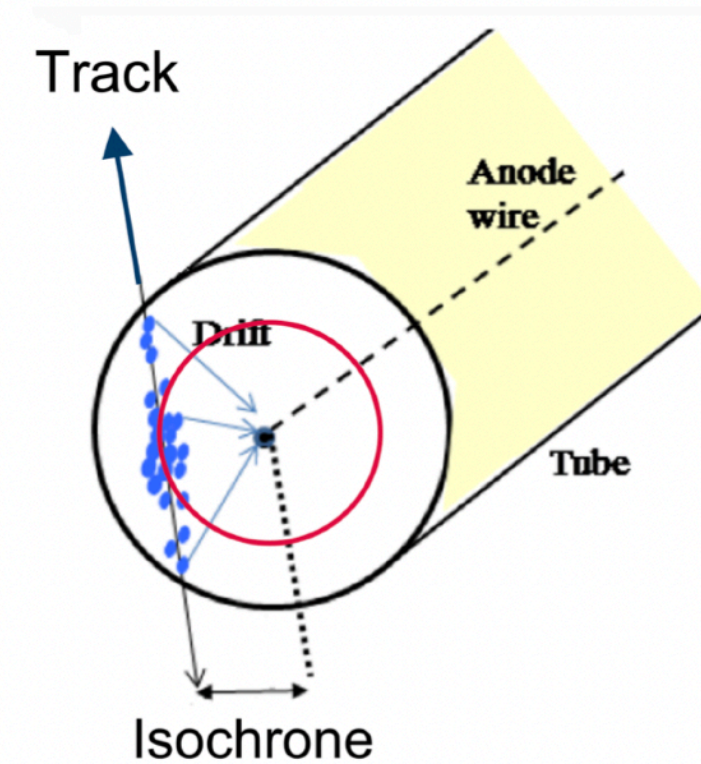
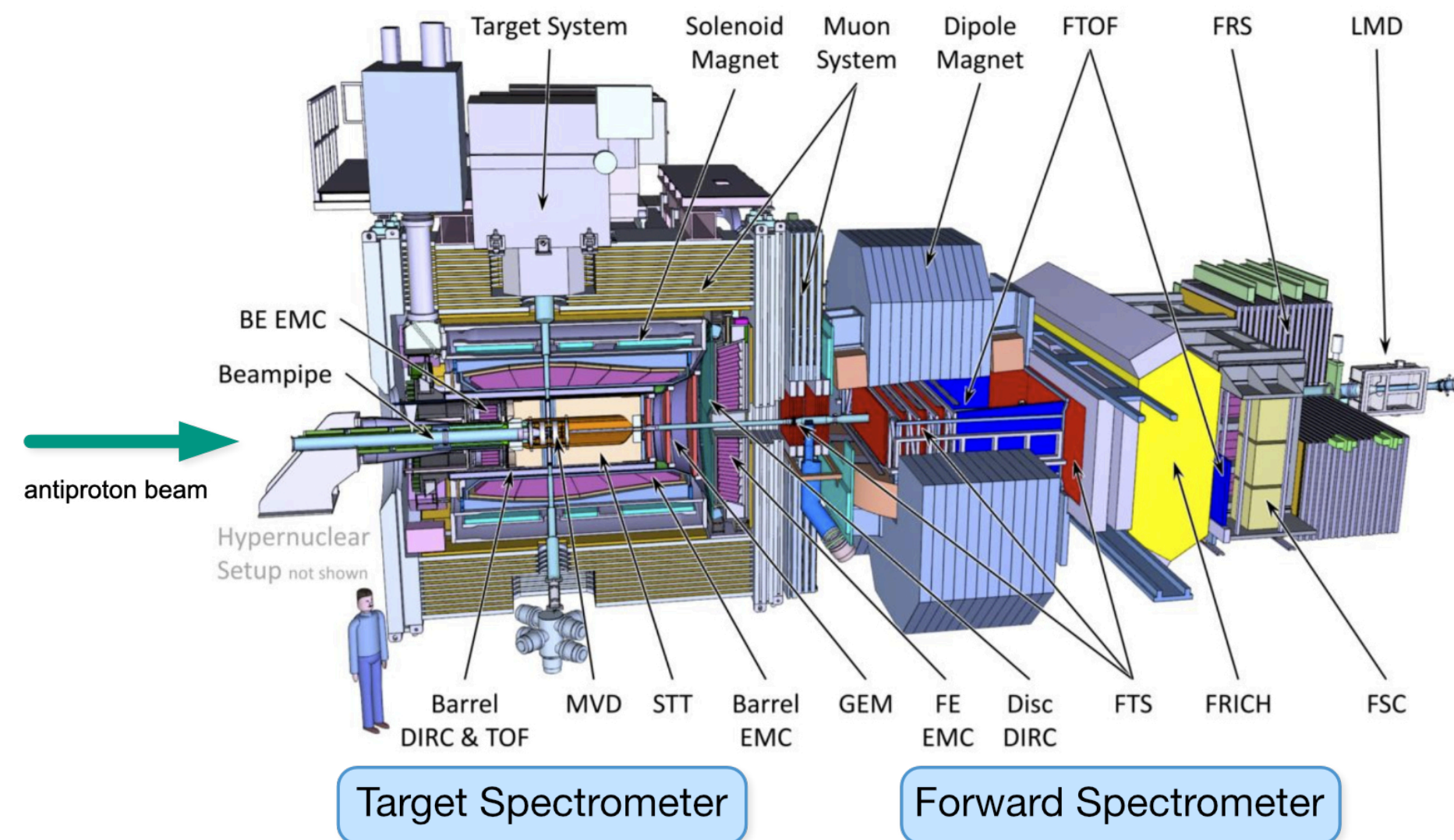
$$\bar{p}N \rightarrow$$

$\Xi^{-208}Pb$

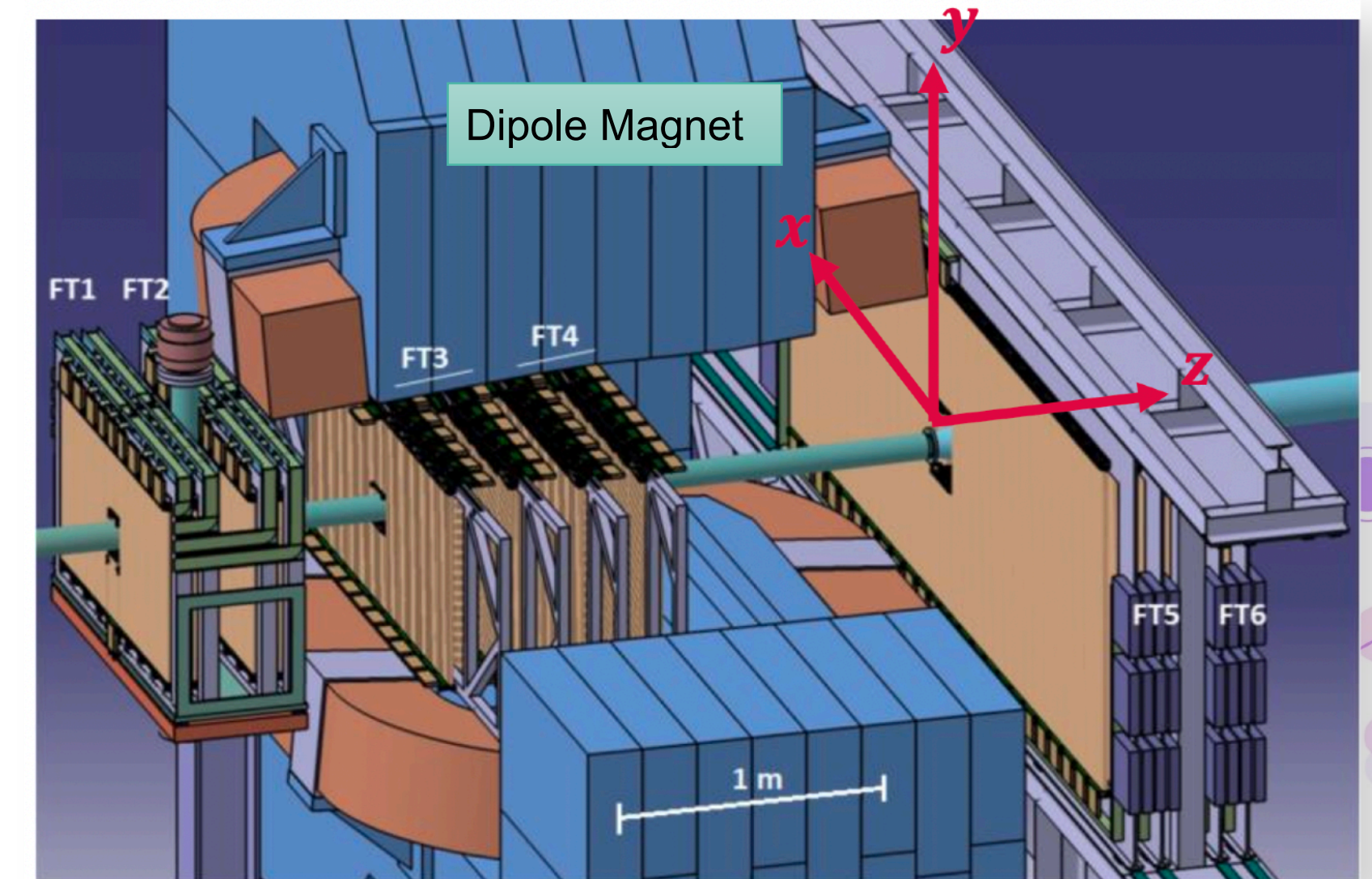


PANDA Forward Tracking System (FTS)

- three pairs of planar straw tube tracking stations FT1 to FT6
- FT1 and FT2 in front of, FT3 and FT4 inside and FT5 and FT6 behind the dipole magnet
- each straw tube with diameter of 10mm
- each tracking station consists of 4 double-layers where the two intermediate double-layers are inclined at $+5^\circ$ and -5°



Single straw tube
<https://arxiv.org/pdf/1910.07191.pdf>



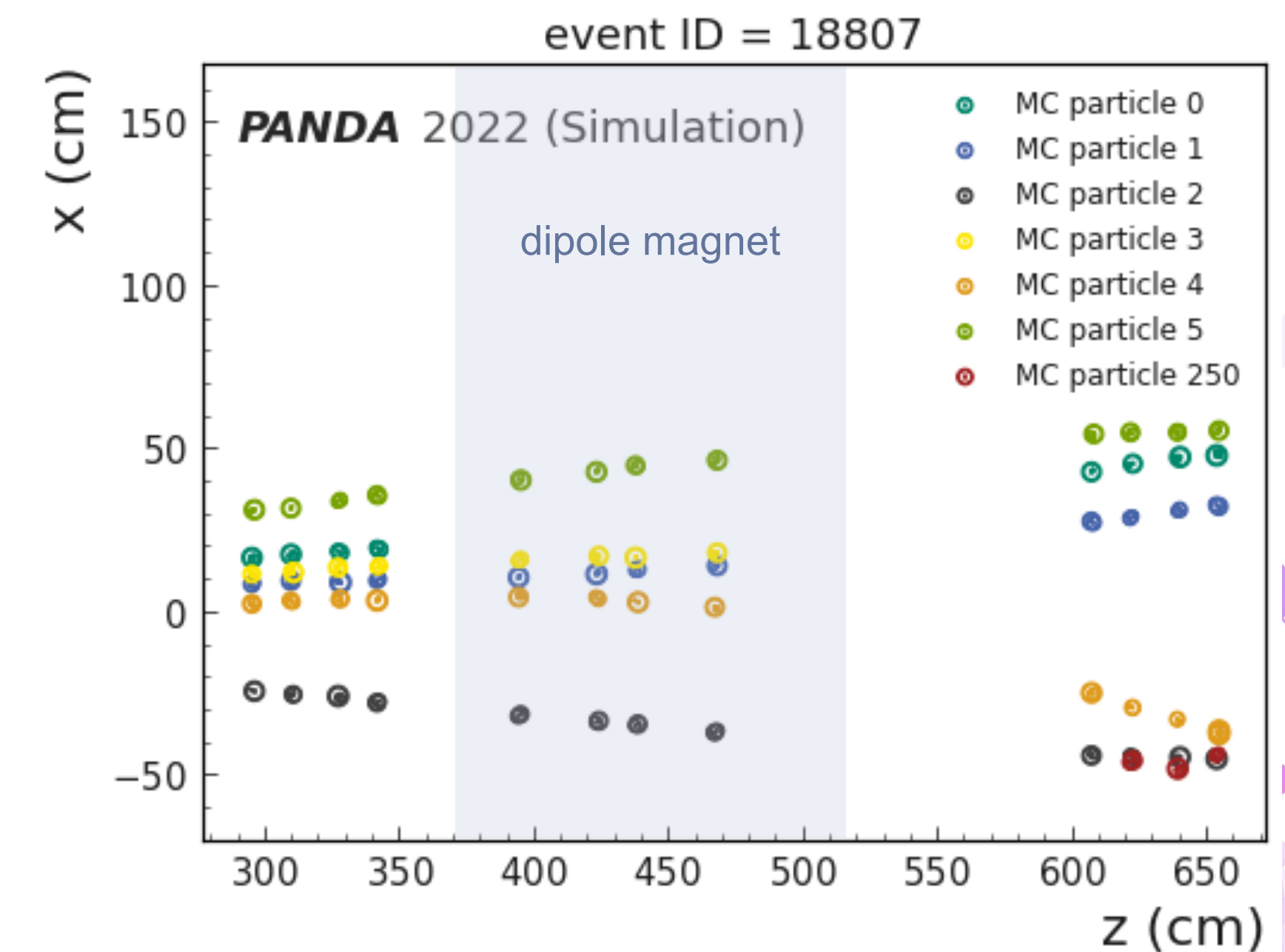
Forward Straw Tube Trackers FTS1 to FTS6
<https://arxiv.org/pdf/1910.07191.pdf>

Generation of MC training Data via PandaRoot

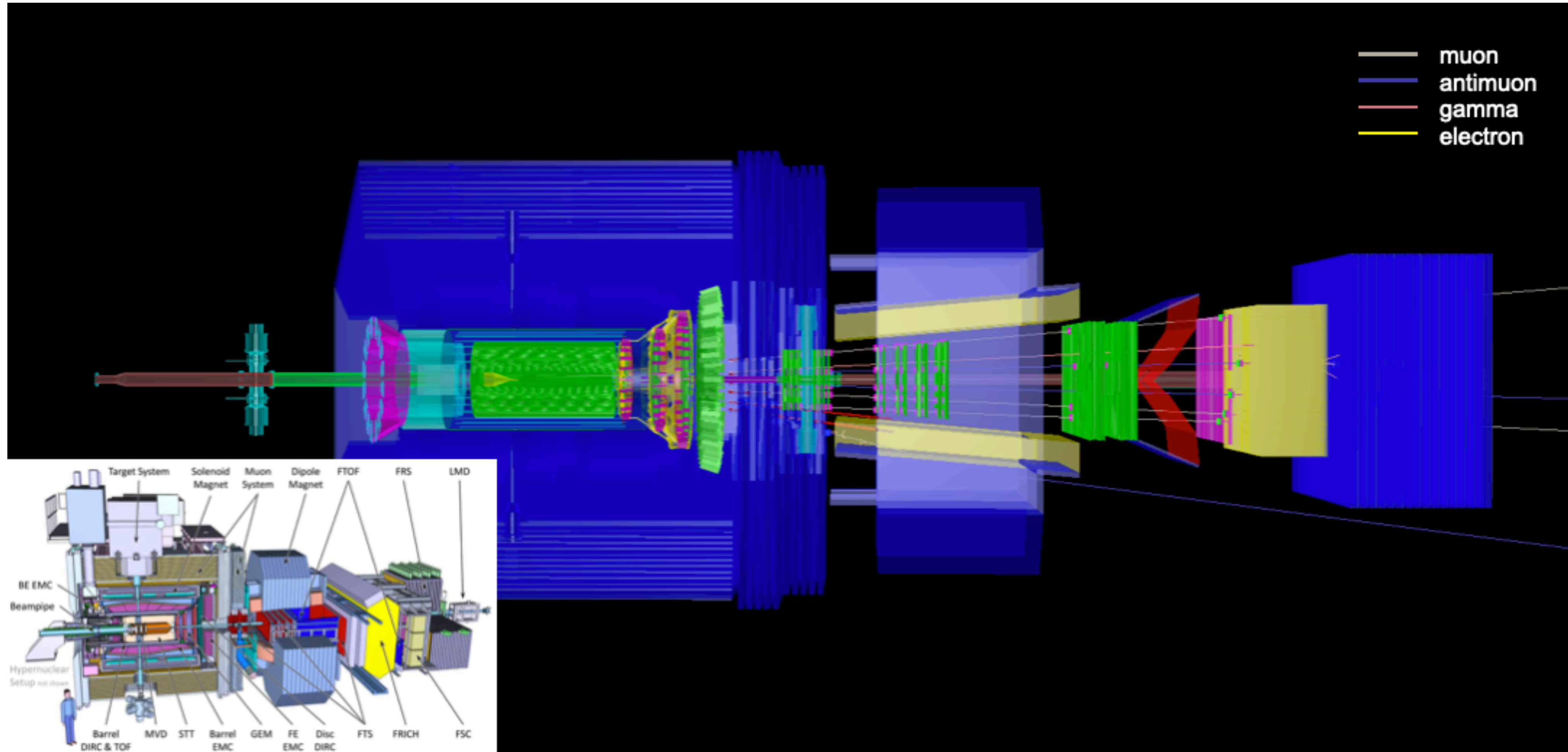


- PANDARoot: PANDA experiment's simulation and reconstruction software package
- full Panda simulation MC event generator and transport through the detector via Geant
- $n = 30.000$
- generation of 3 muons (few secondary particles) and 3 anti-muons (crossing tracks)
- particle gun box event Generator: 3 muons and 3 anti-muons in forward direction with uniform distribution of $p = [1, 10\text{GeV}]$, $\theta = [0.5^\circ, 10^\circ]$ and $\phi = [0, 2\pi]$

```
TString inputGenerator =  
„box:type(13,3):p(1,10):tht(0.5,10):phi(0,360)
```

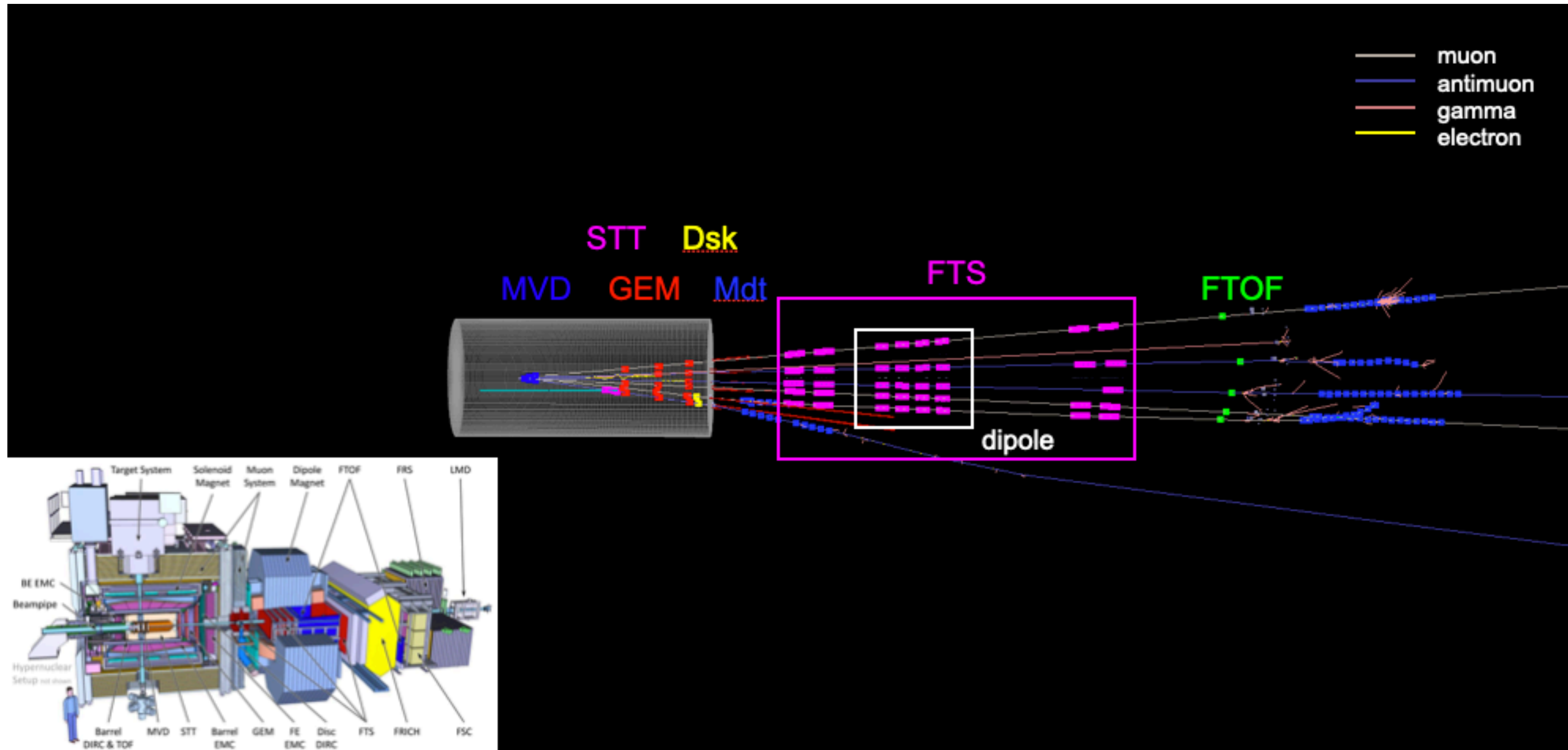


Generation of MC training Data via PANDARoot



PANDARoot simulation event display of 3 muons and 3 anti muons with detector display

Generation of MC training Data via PANDARoot

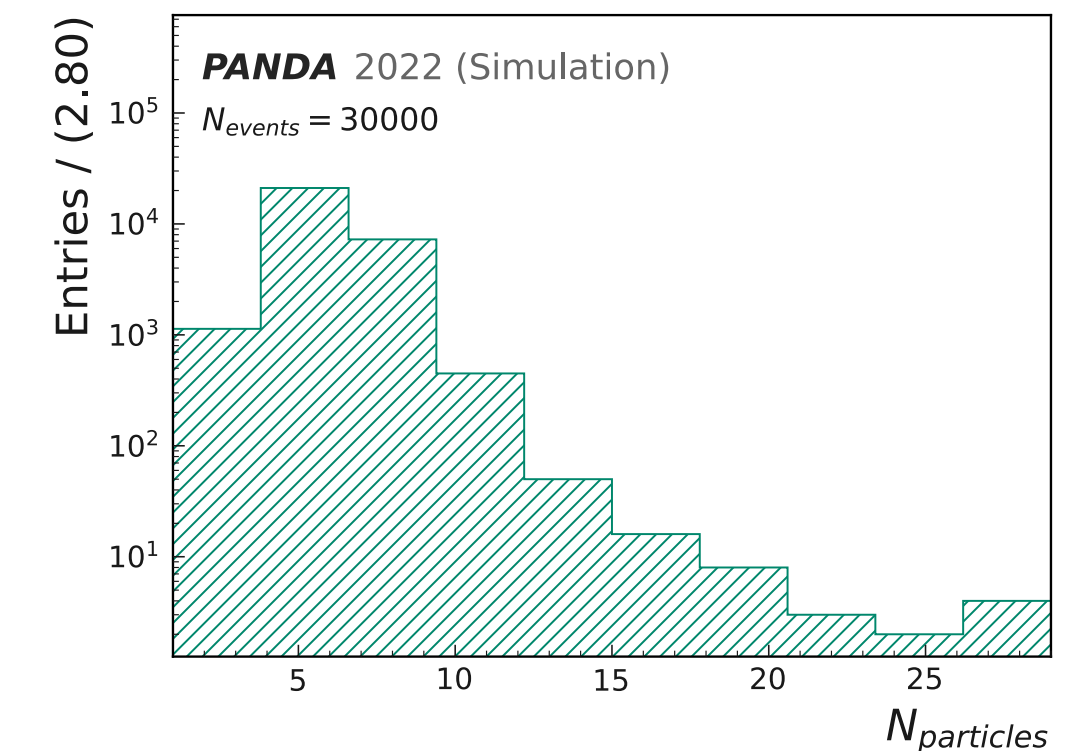
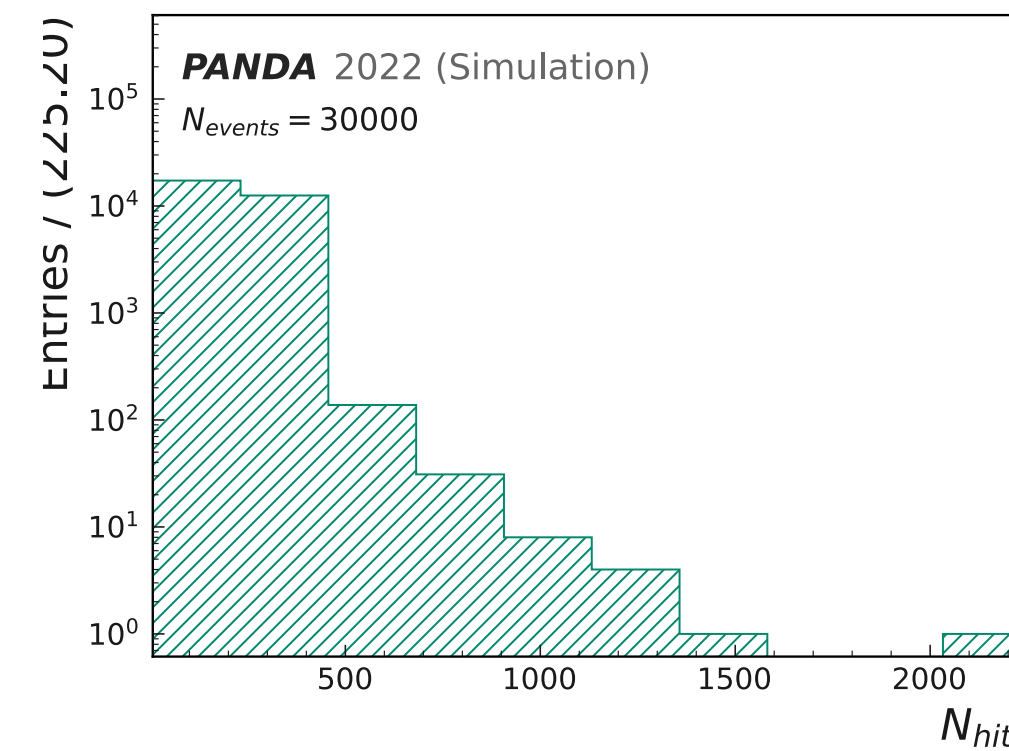
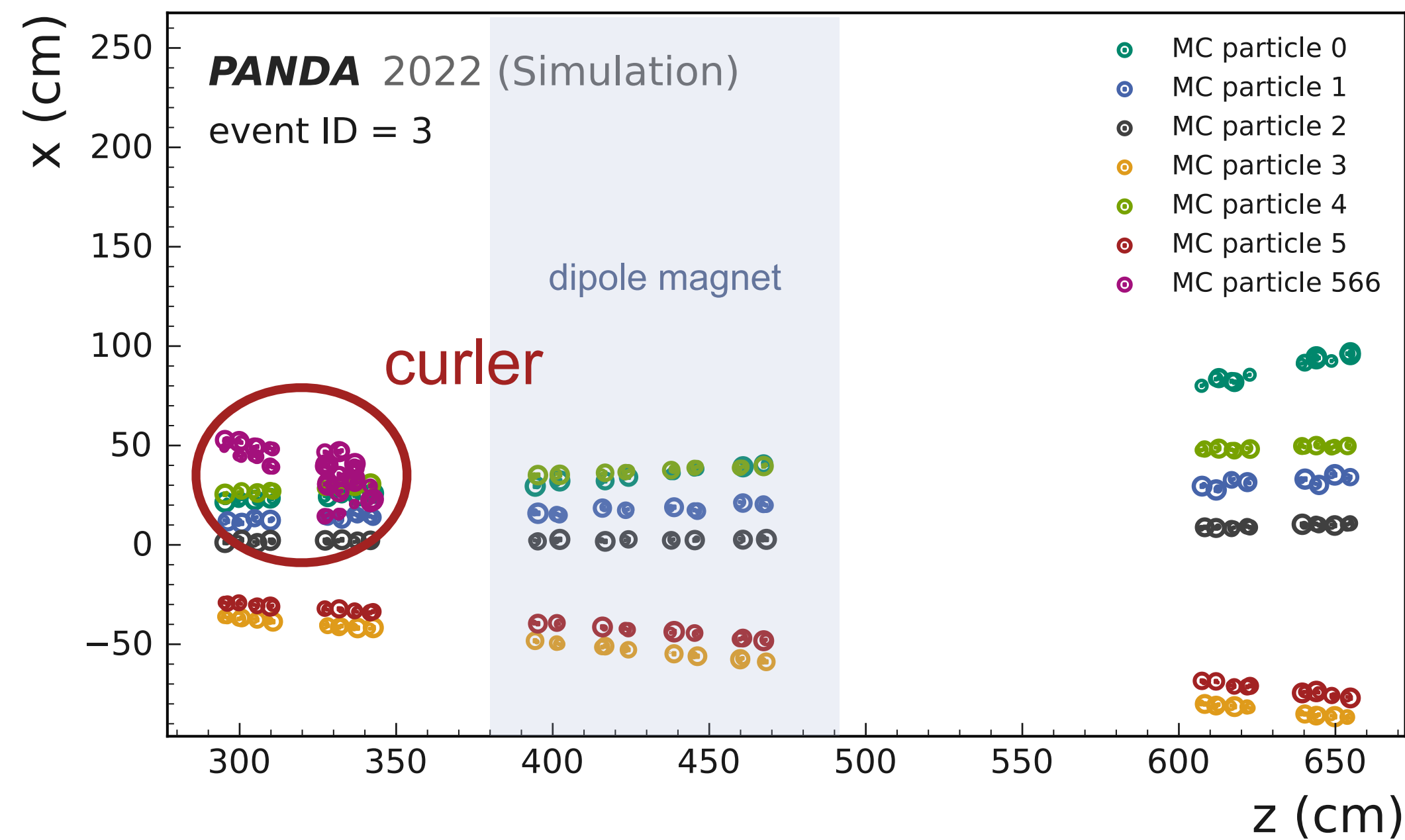


PANDARoot simulation event display of 3 muons and 3 anti muons with detector display

Raw ROOT Data Input

■ Conversion from Root to pandas dataframe using uproot

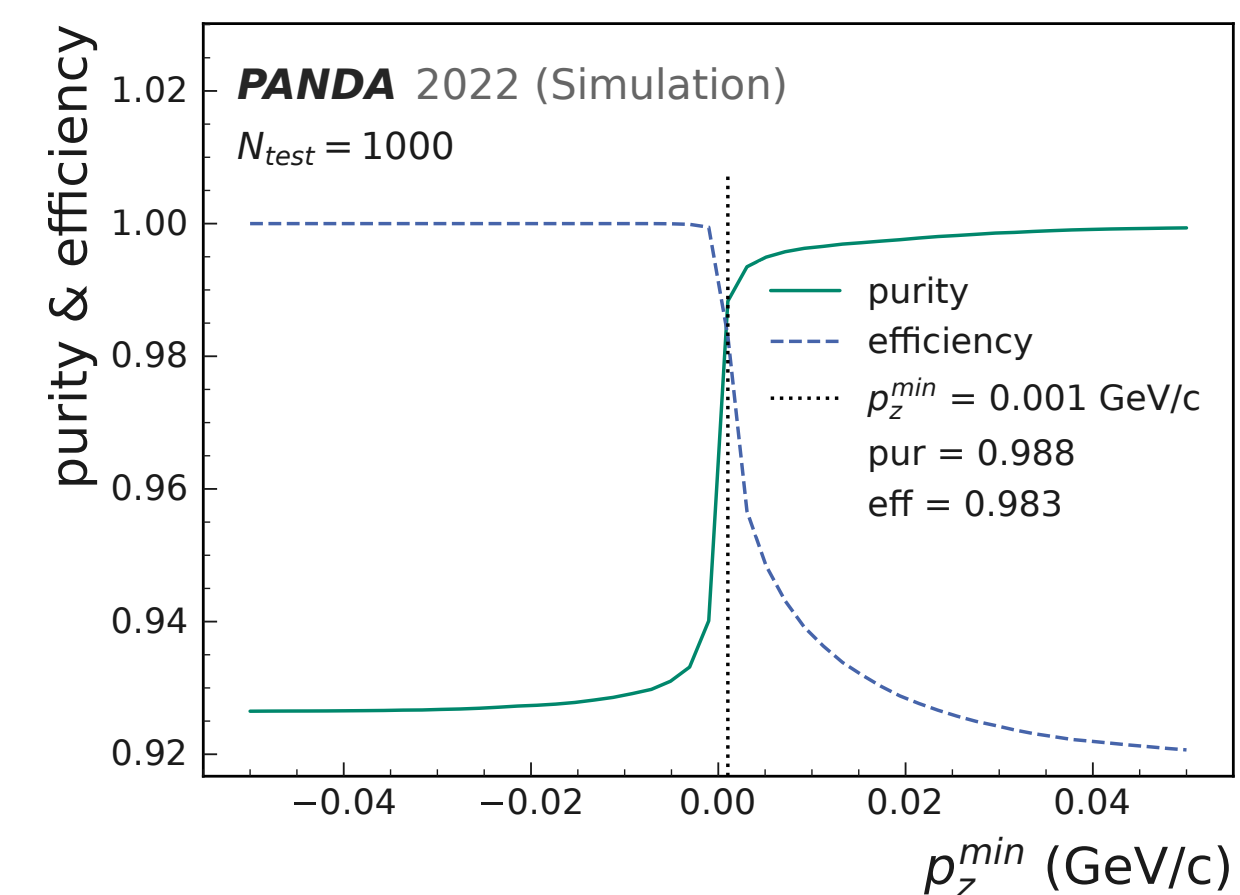
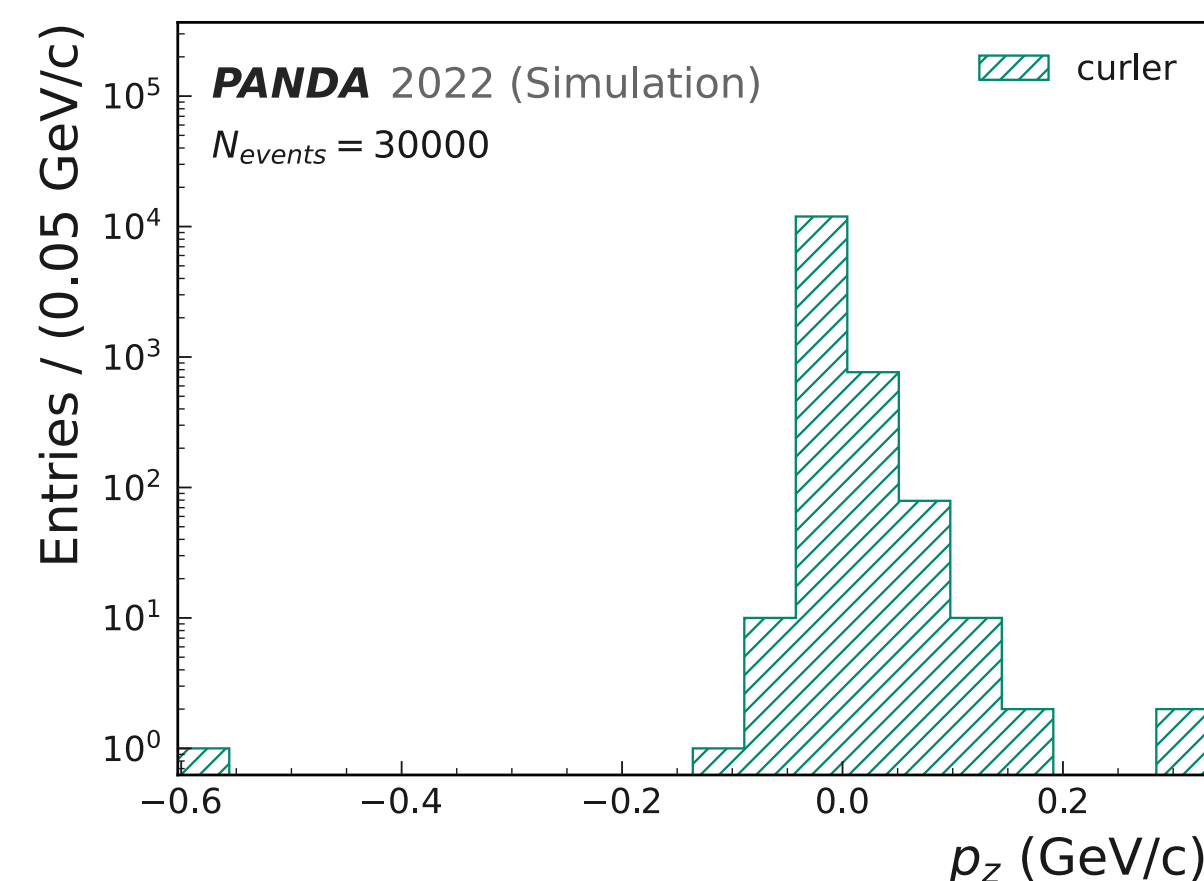
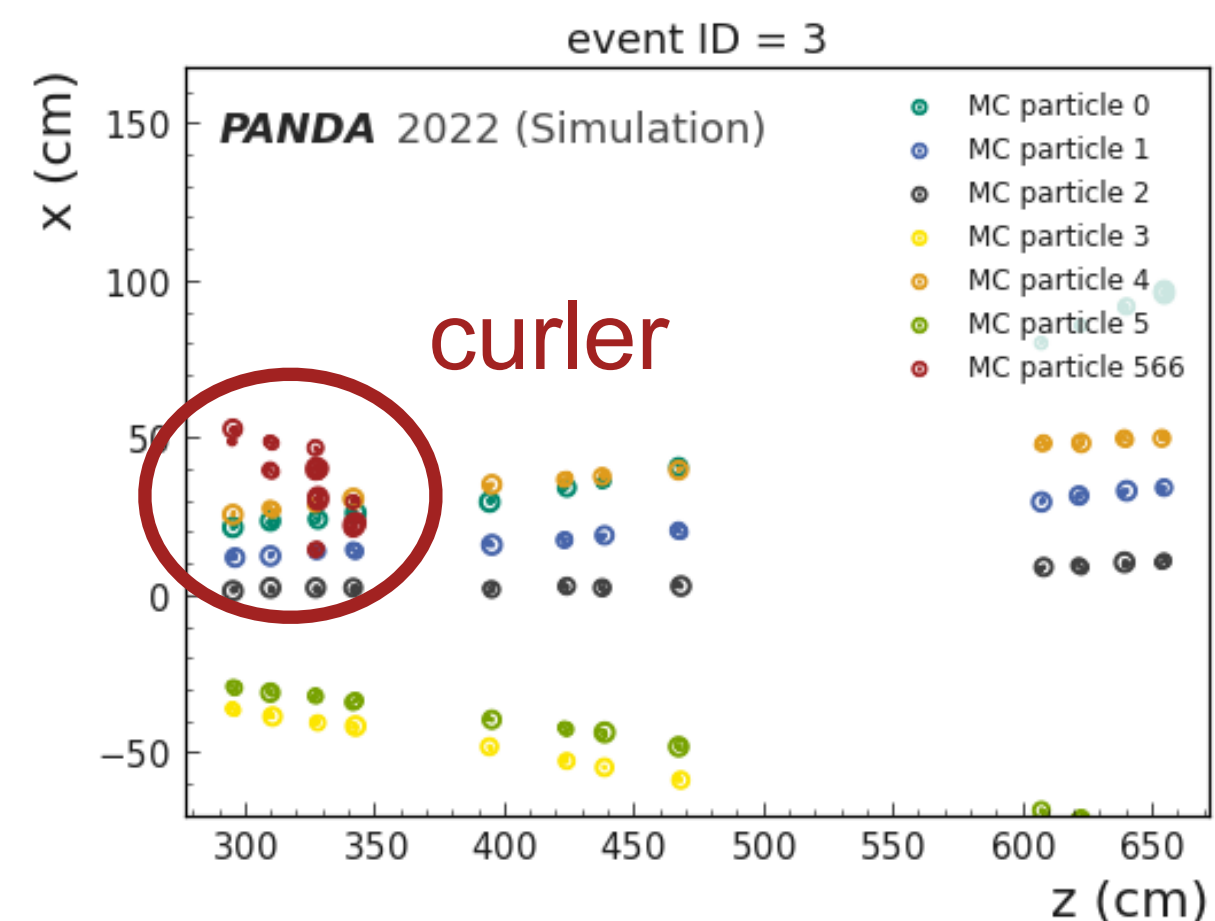
■ features: event ID, hit ID, x, z, isochrone radius, momentum pz, MC particle ID, layer ID



event_id	hit_id	x	z	iso	pz	particle_id	layer_id	
0	0	-15.907500	606.994995	0.453287	2.604892	5	33	
1	0	1	-15.402500	607.869690	0.168081	2.604809	5	34
2	0	6	-16.917500	621.489990	0.377410	2.604239	5	39
3	0	7	-17.422501	622.364685	0.016627	2.604213	5	40
4	0	8	-19.947500	638.994995	0.403070	2.603821	5	41
...
3312107	29999	232	3.787500	622.364685	0.127759	7.105216	0	40
3312108	29999	233	4.292500	638.994995	0.418415	7.105163	0	41
3312109	29999	234	4.797500	639.869690	0.053052	7.105146	0	42
3312110	29999	239	5.302500	653.489990	0.086463	7.105002	0	47
3312111	29999	240	5.807500	654.364685	0.382681	7.104968	0	48

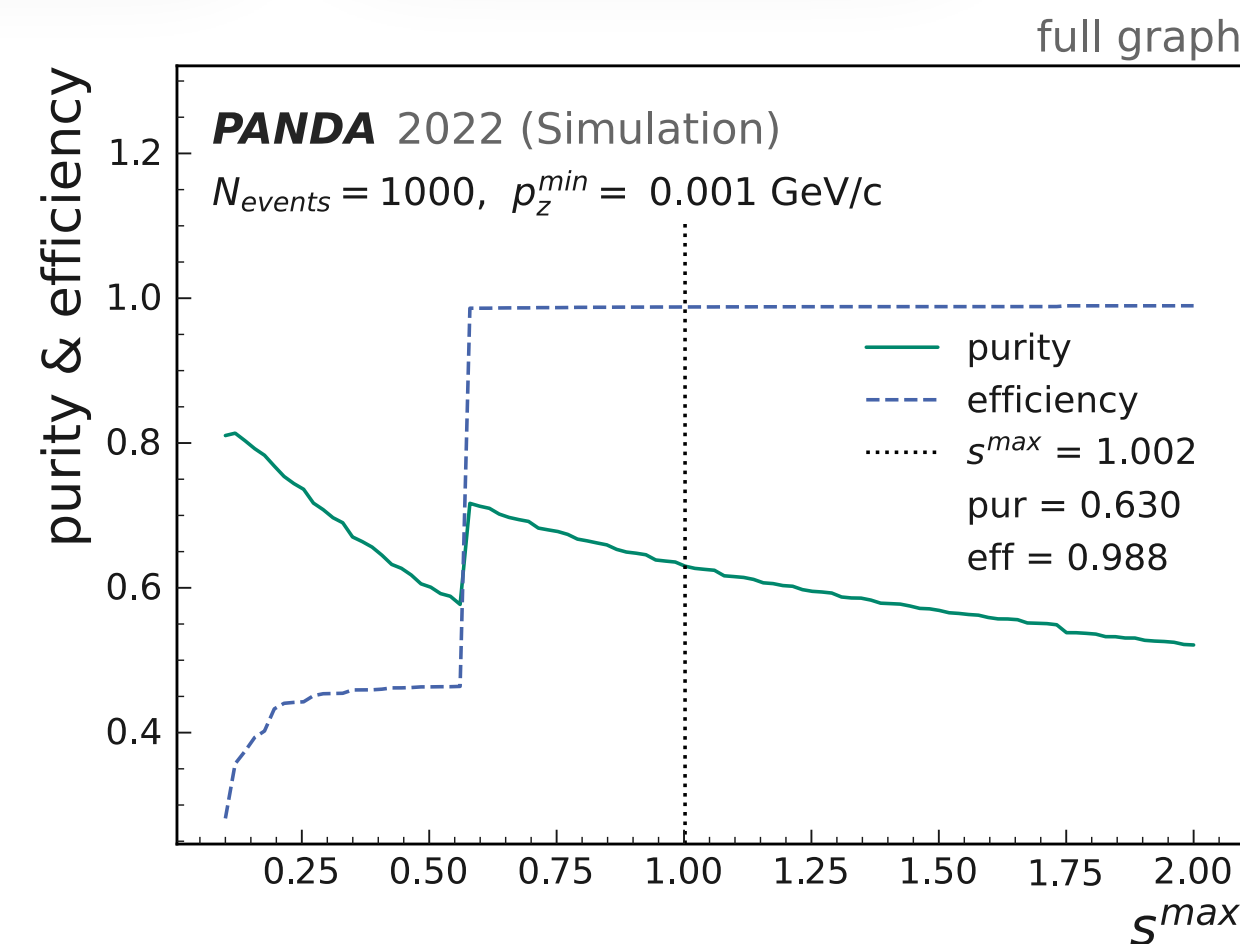
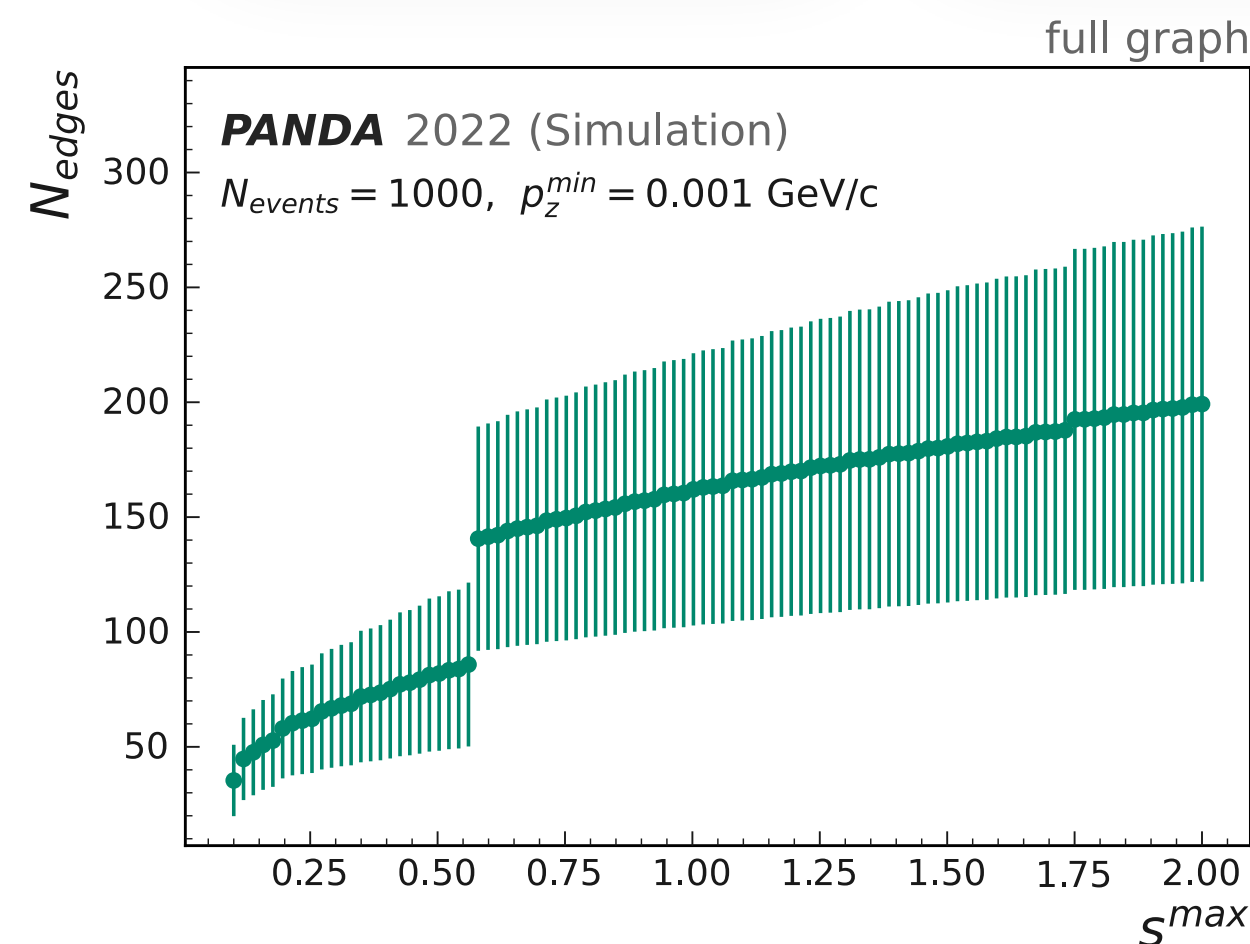
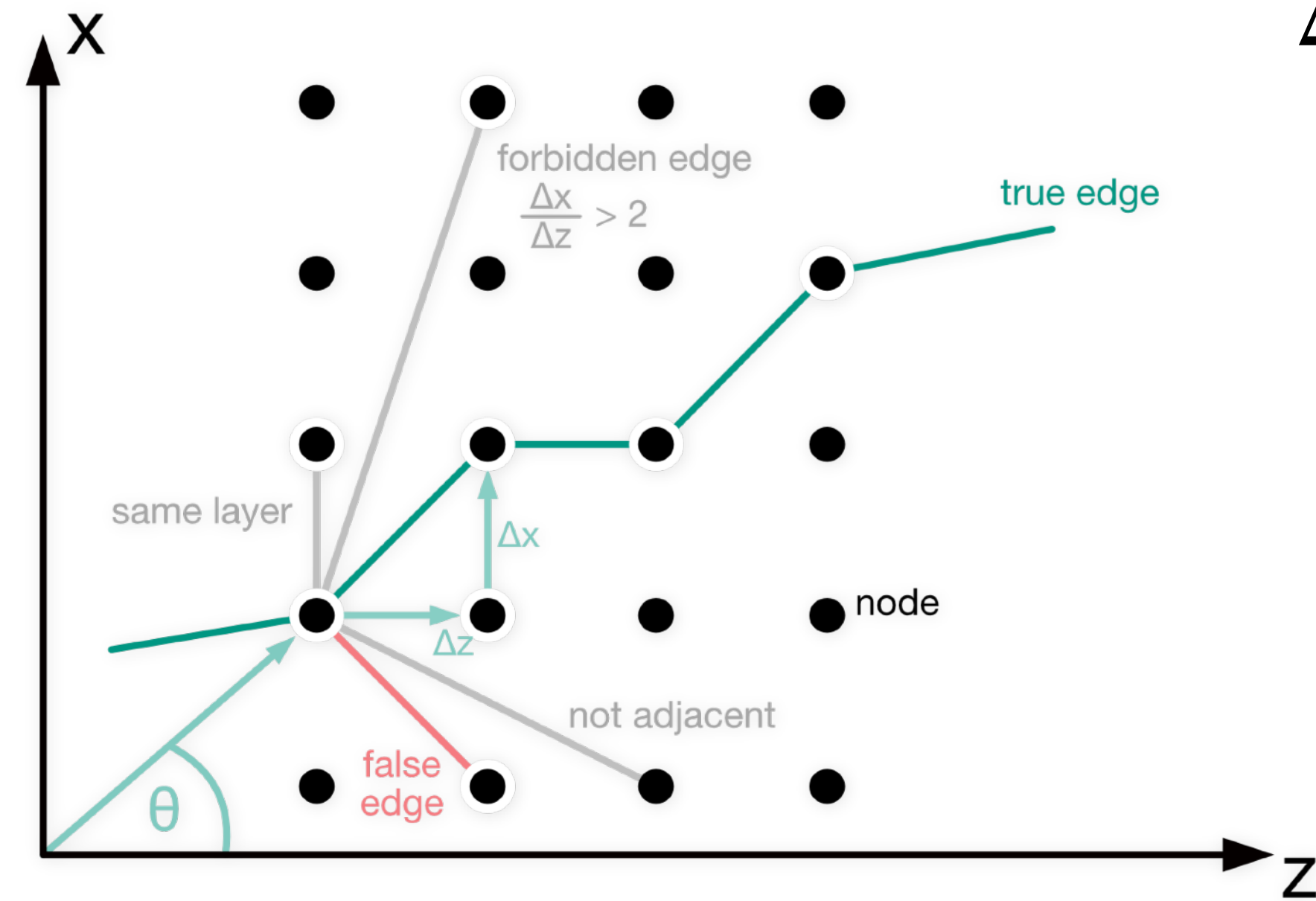
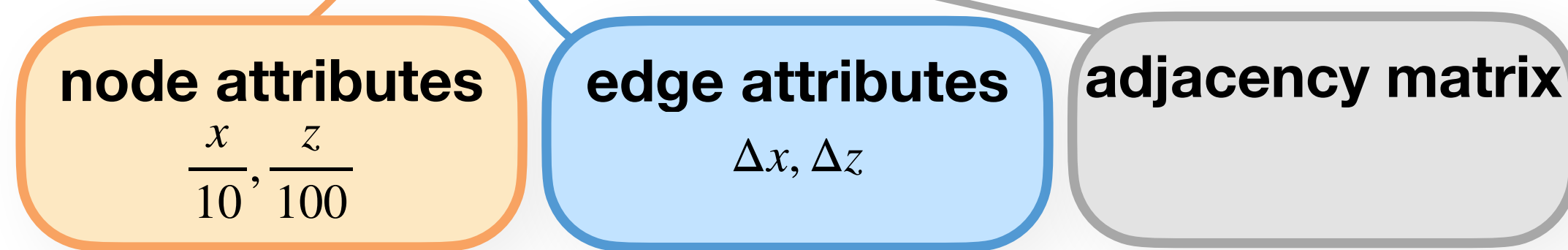
Preprocessing of Training Data

- omit skewed layers and apply layer mapping for new adjacent layer labels
- in order to avoid looping tracks (curlers) and tracks leaving the detector early, only consider particles with
 - $p_z > 0.001$ GeV/c (85.4% of curlers removed and only 1.7% of non-curlers lost)
- additional same-layer filter and hit index reordering



Graph Building

- GNNs are a central method of geometric deep learning based on a graph representation of data
- build all possible edges (track segments) between hits of adjacent layers with a slope $s = \frac{\Delta x}{\Delta z} < 1$
- graph $G (X, E, I)$

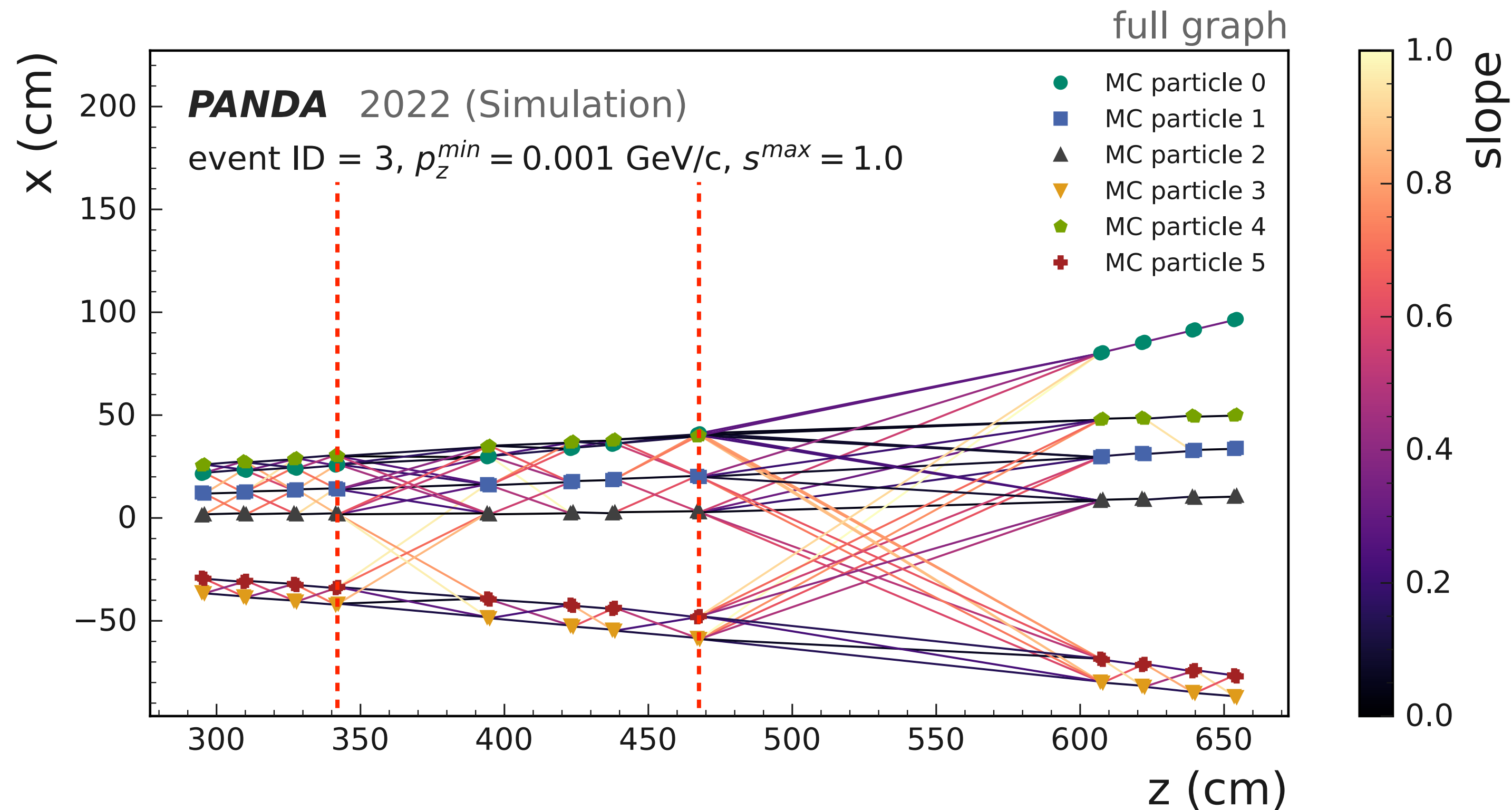


$$\text{purity} = \frac{N_{\text{true edges} < \text{threshold}}}{N_{\text{edges} < \text{threshold}}}$$

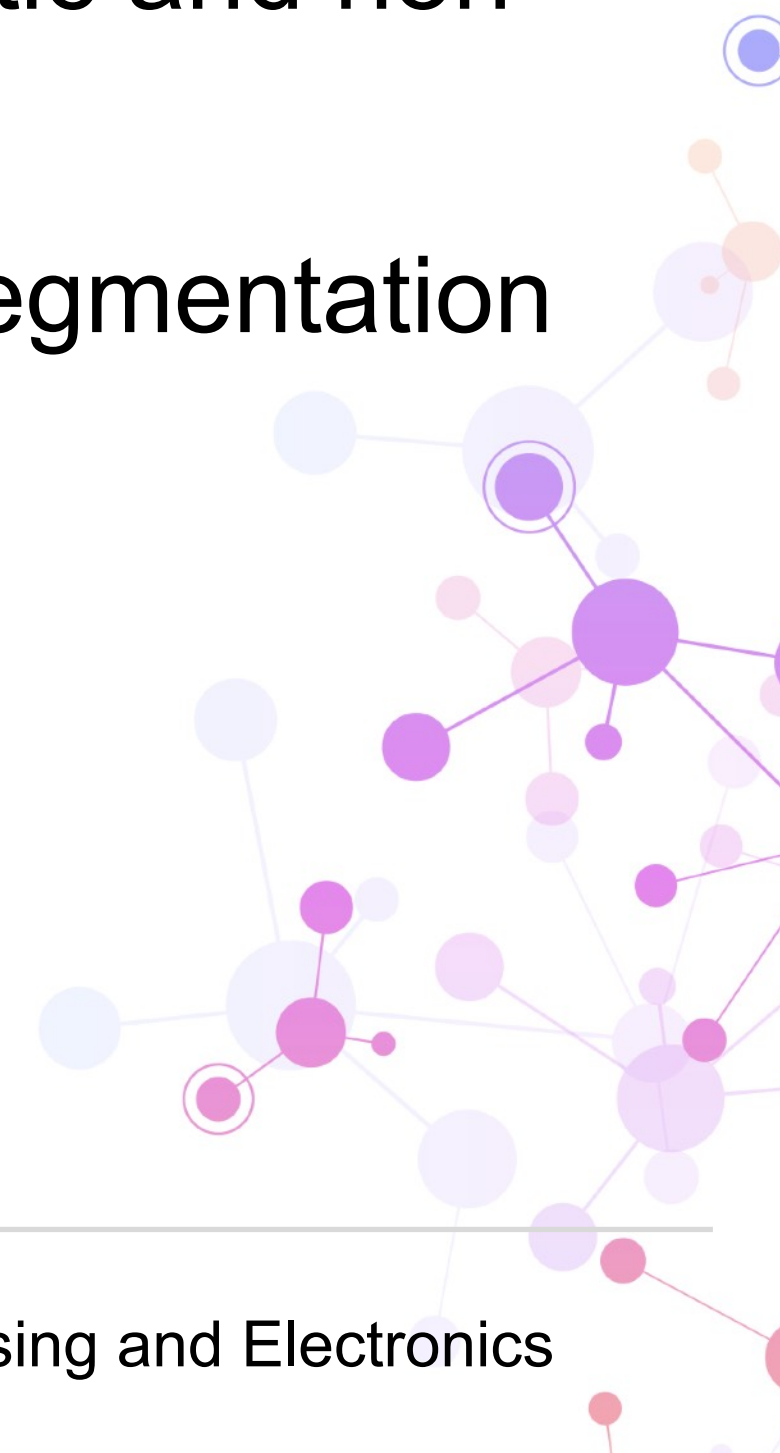
$$\text{efficiency} = \frac{N_{\text{true edges} < \text{threshold}}}{N_{\text{true edges}}}$$

Graph Encoding	Tensors
Node Attributes	$X = (x_i, z_i) \in \mathbb{R}^{N_{\text{nodes}} \times 2}$
Edge Attributes	$E = (\Delta x_{k,ij}, \Delta z_{k,ij}) \in \mathbb{R}^{N_{\text{edges}} \times 2}$
Adjacency Matrix	$I = [I_{i,k}, I_{j,k}] \in \mathbb{N}^{2 \times N_{\text{edges}}}$
Target Vector	$y \in \{0, 1\}^{N_{\text{edges}}}$

Graph Building

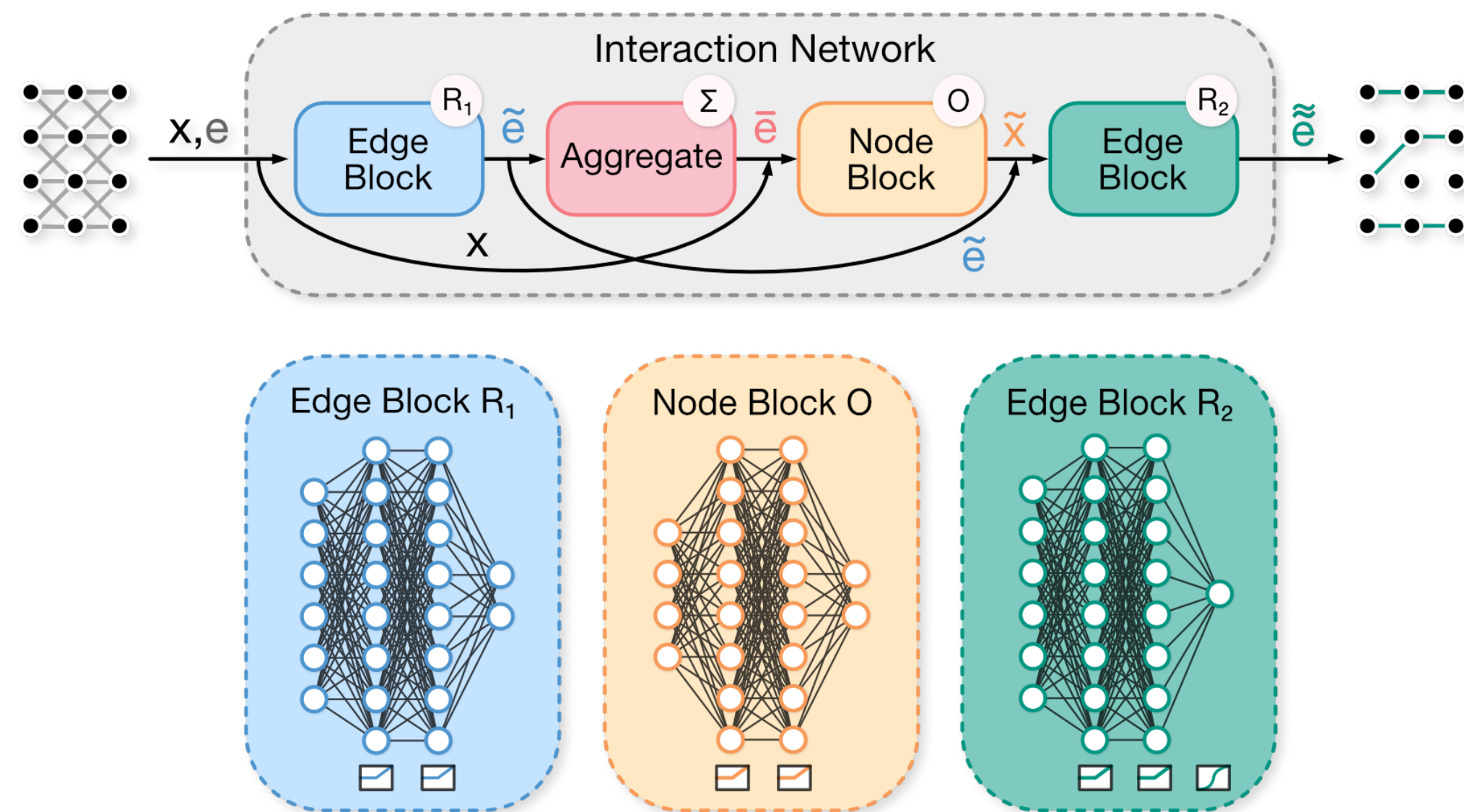


- graph segmentation in 3 parts
- ▶ graph size reduction (faster hls4ml compilation, reduced FPGA occupancy)
- ▶ separation of magnetic and non-magnetic regions
- ▶ equivalent to DAQ segmentation



GNN Architecture

- interaction network model by Battaglia et al. with PyTorch Geometric
- **Edge block R1** (linear feed forward network with ReLu activation) for edge features
- **Node block O** (linear feed forward network with ReLu activation) for hit features
- **final Edge block R2** with sigmoid activation, 1-dim output



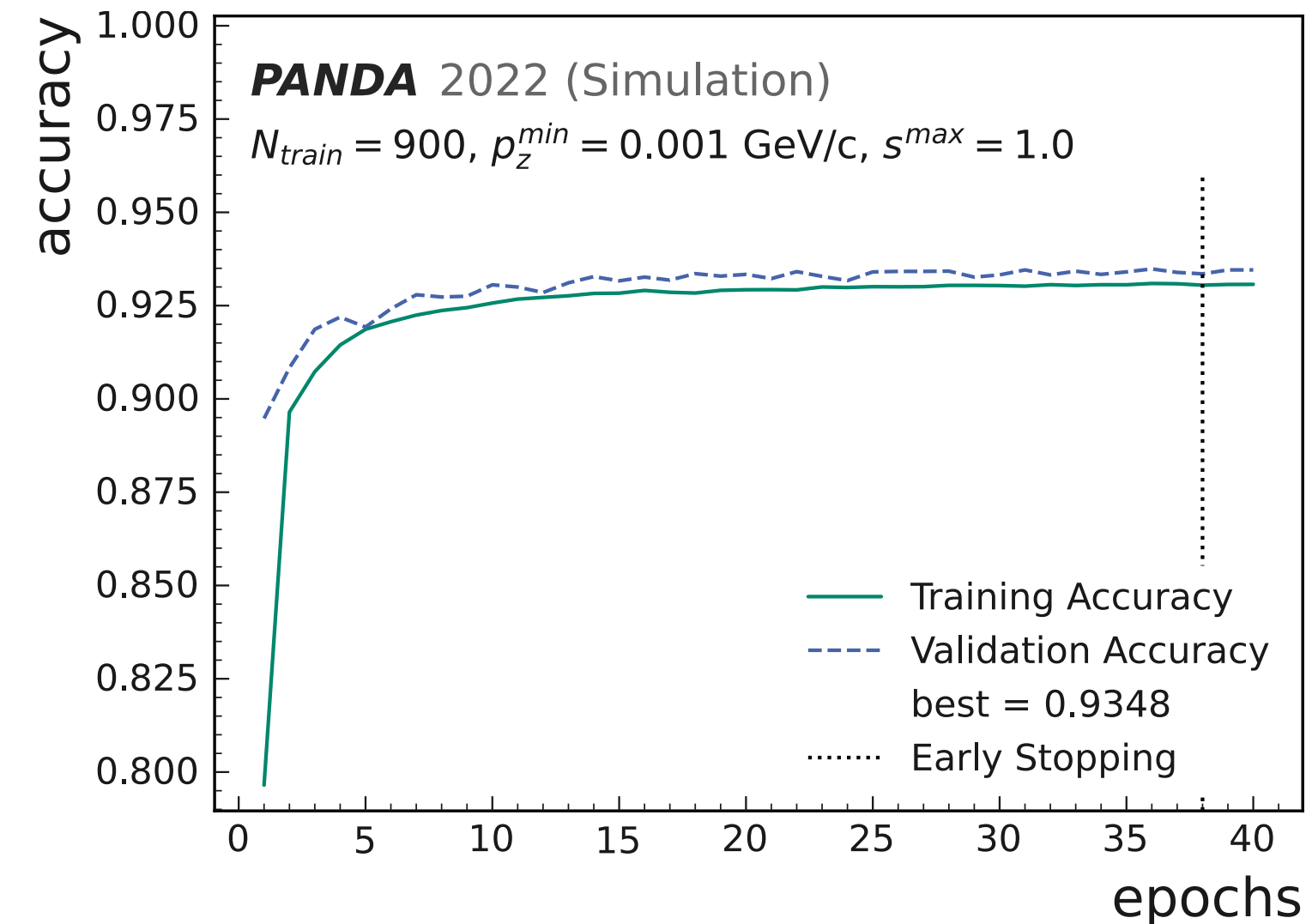
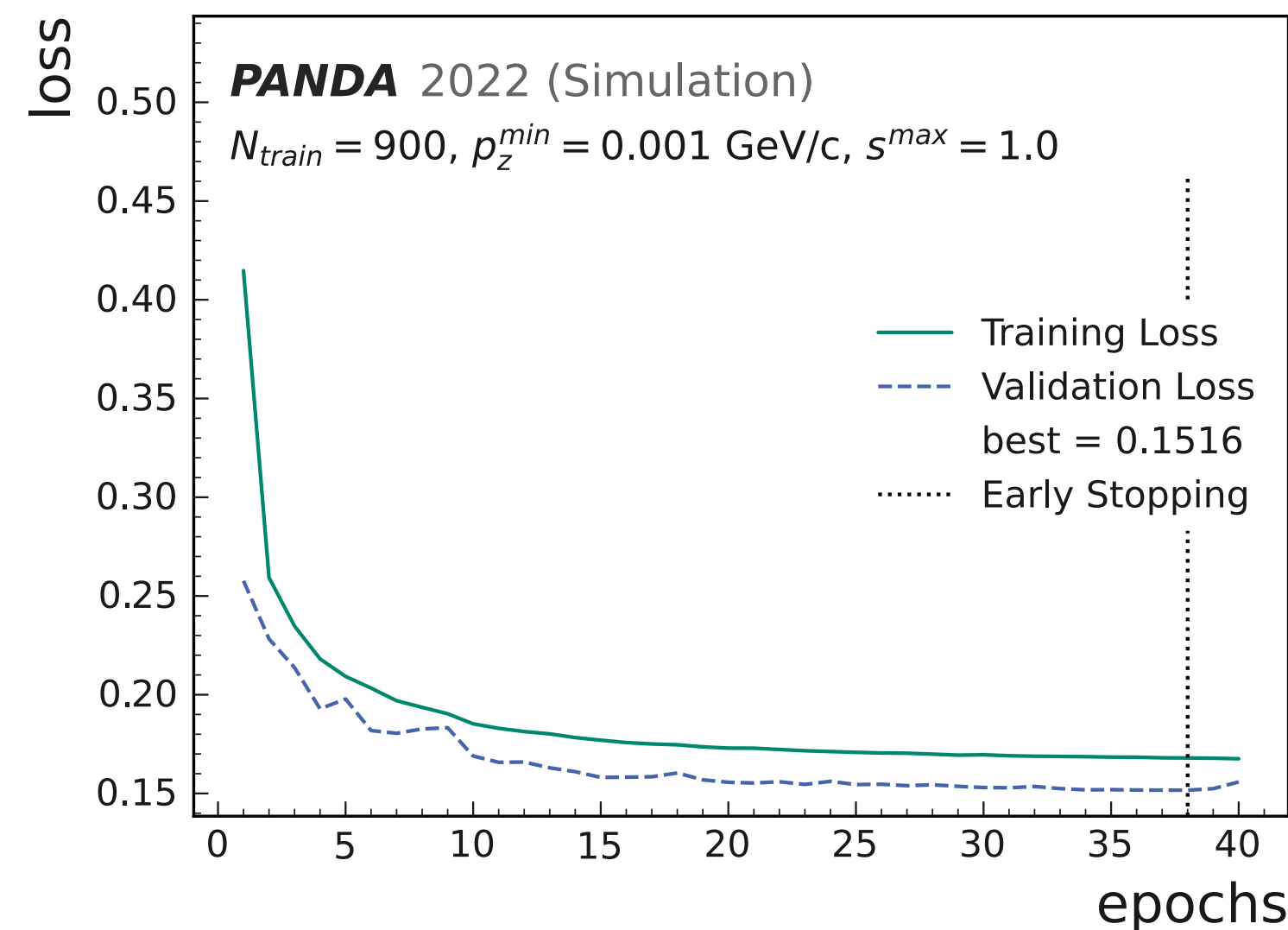
```
InteractionNetwork(node_dim: 2, edge_dim: 2, hidden_size: 8)
```

Modules	Parameters
R1.layers.0.weight	48
R1.layers.0.bias	8
R1.layers.2.weight	64
R1.layers.2.bias	8
R1.layers.4.weight	16
R1.layers.4.bias	2
O.layers.0.weight	32
O.layers.0.bias	8
O.layers.2.weight	64
O.layers.2.bias	8
O.layers.4.weight	16
O.layers.4.bias	2
R2.layers.0.weight	48
R2.layers.0.bias	8
R2.layers.2.weight	64
R2.layers.2.bias	8
R2.layers.4.weight	8
R2.layers.4.bias	1

Total Trainable Params: 413

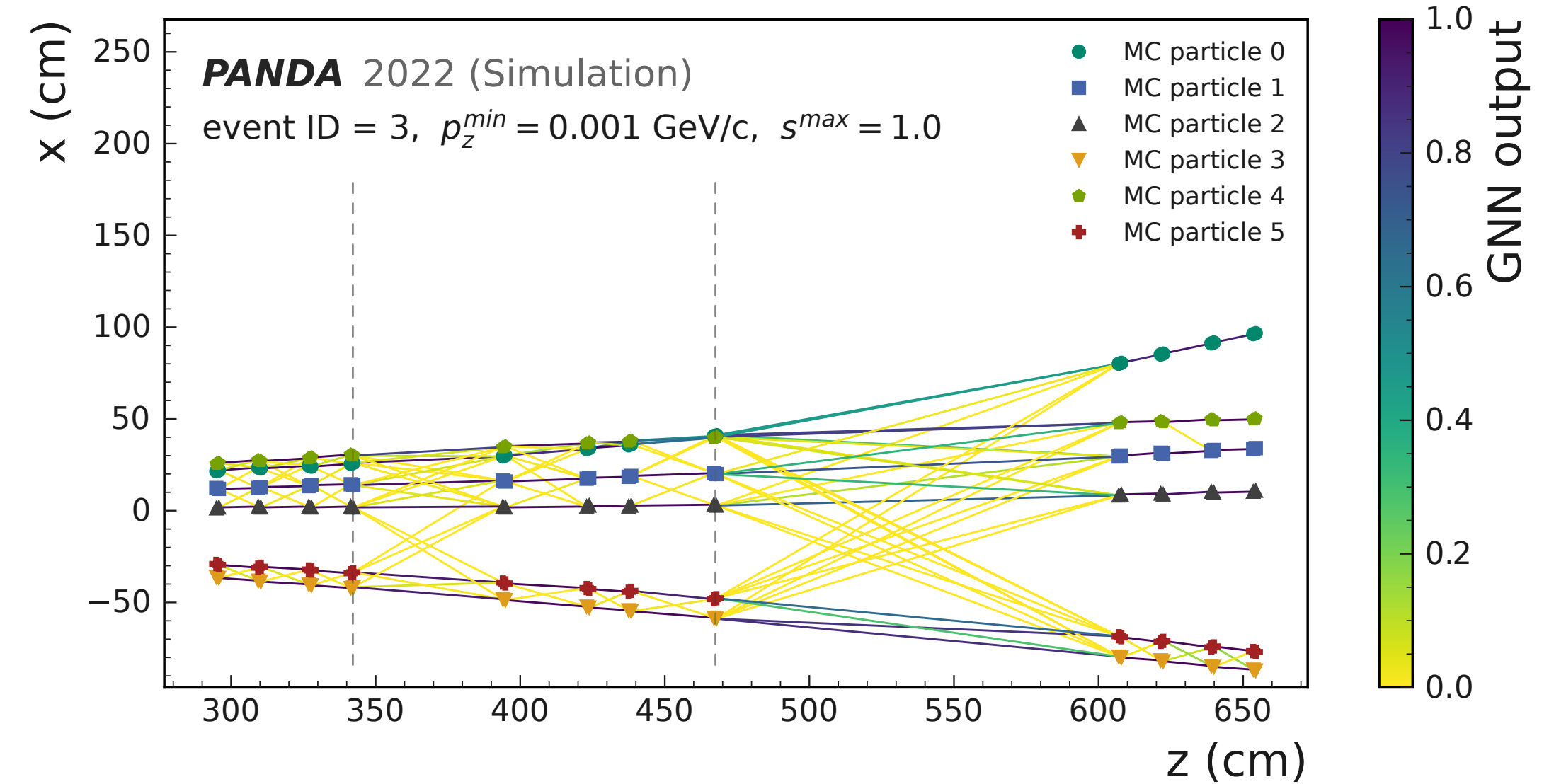
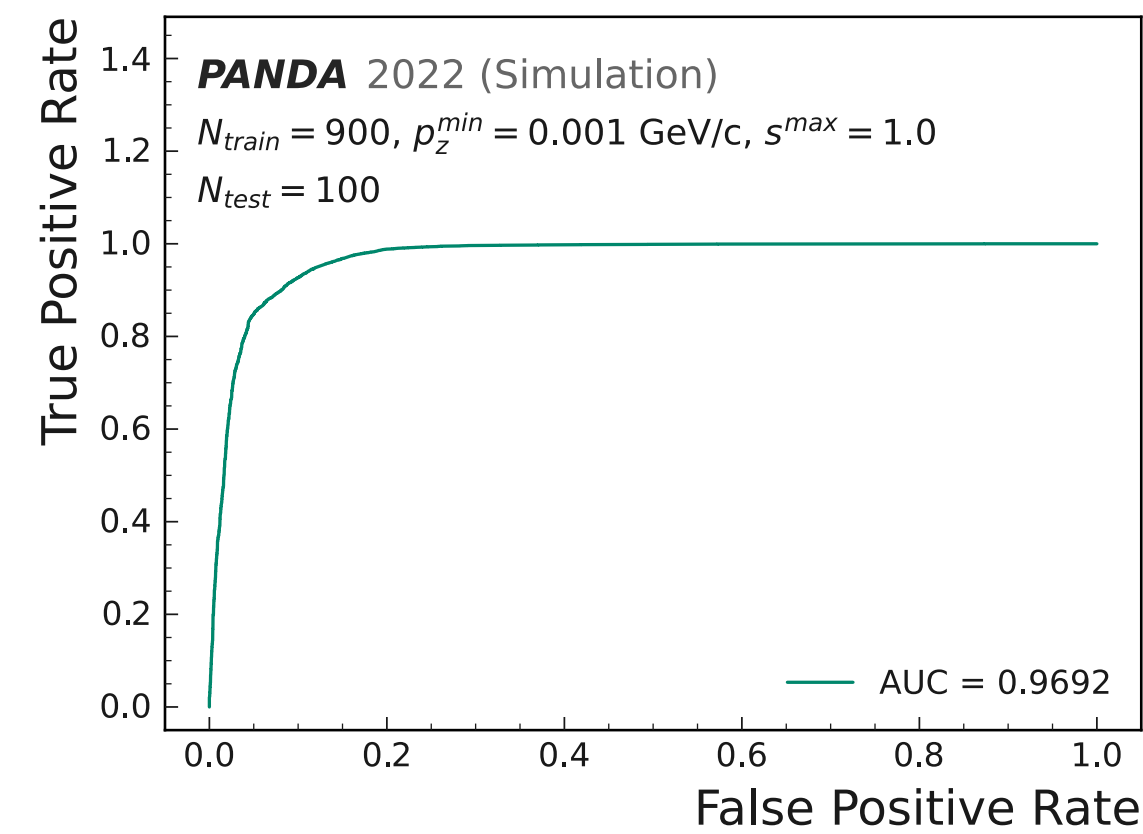
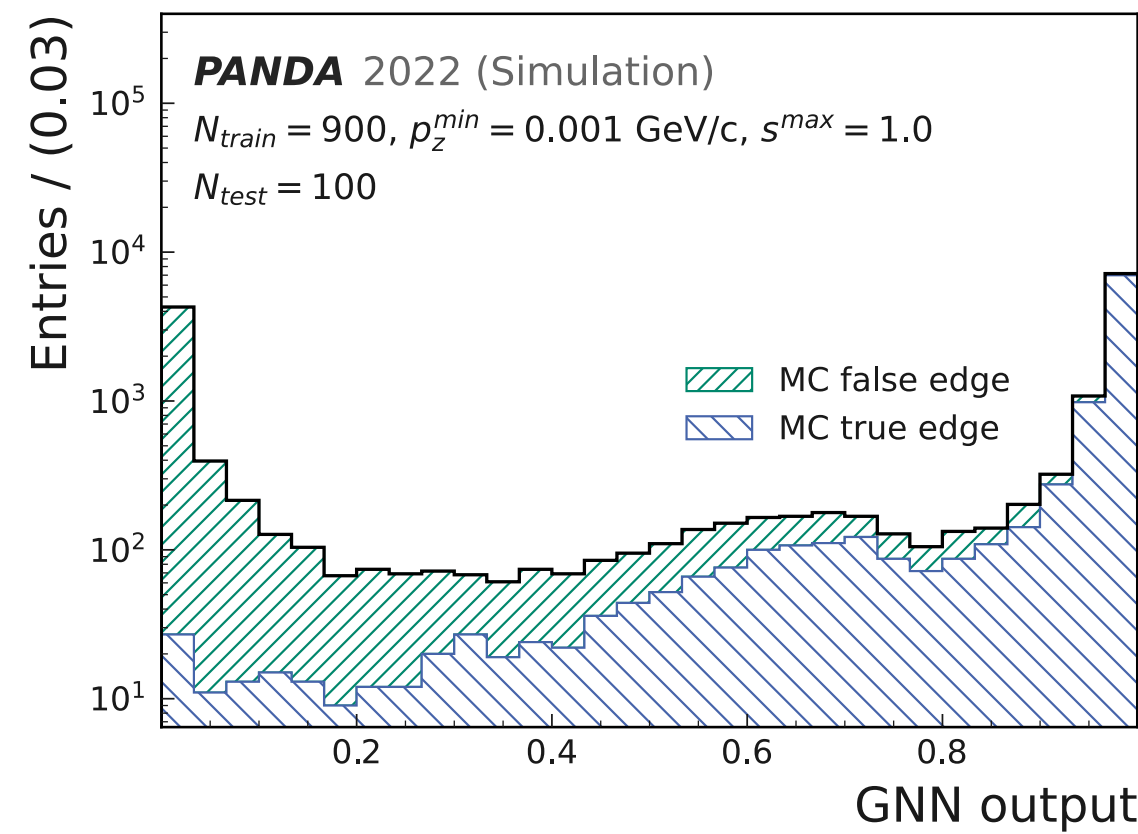
Training and Validation

- divide data into 80% training, 10% validation, 10% test data
- **Optimizer:** Adam optimizer, **loss:** Binary Cross Entropy
- **Epochs:** max. 40 (early stopping after 10 epochs without improvement)
- **learning rate** $lr = 6.5 \cdot 10^{-4}$, **learning rate decay** $\gamma = 0.86$, **step size** = 3 (via optuna hyper parameter search)

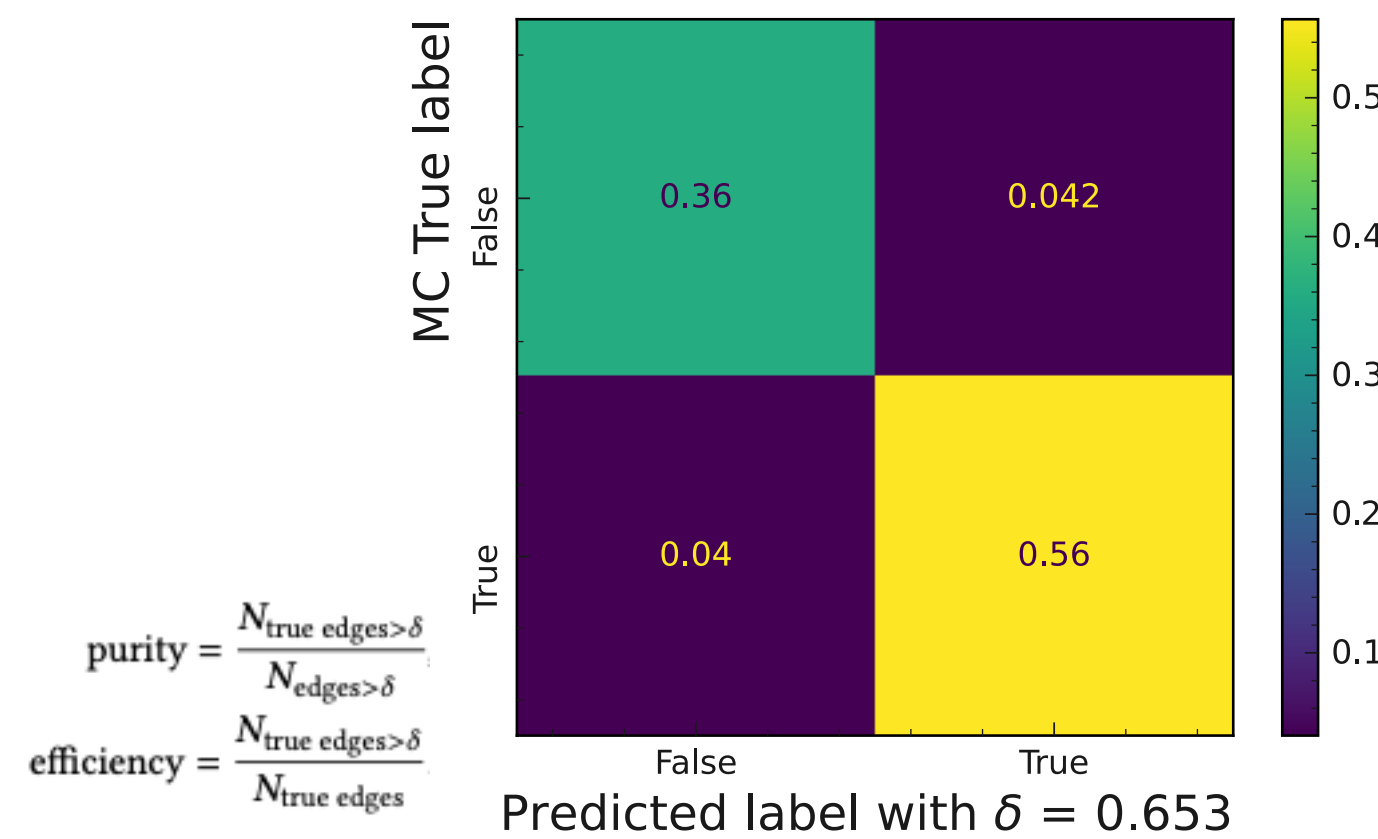
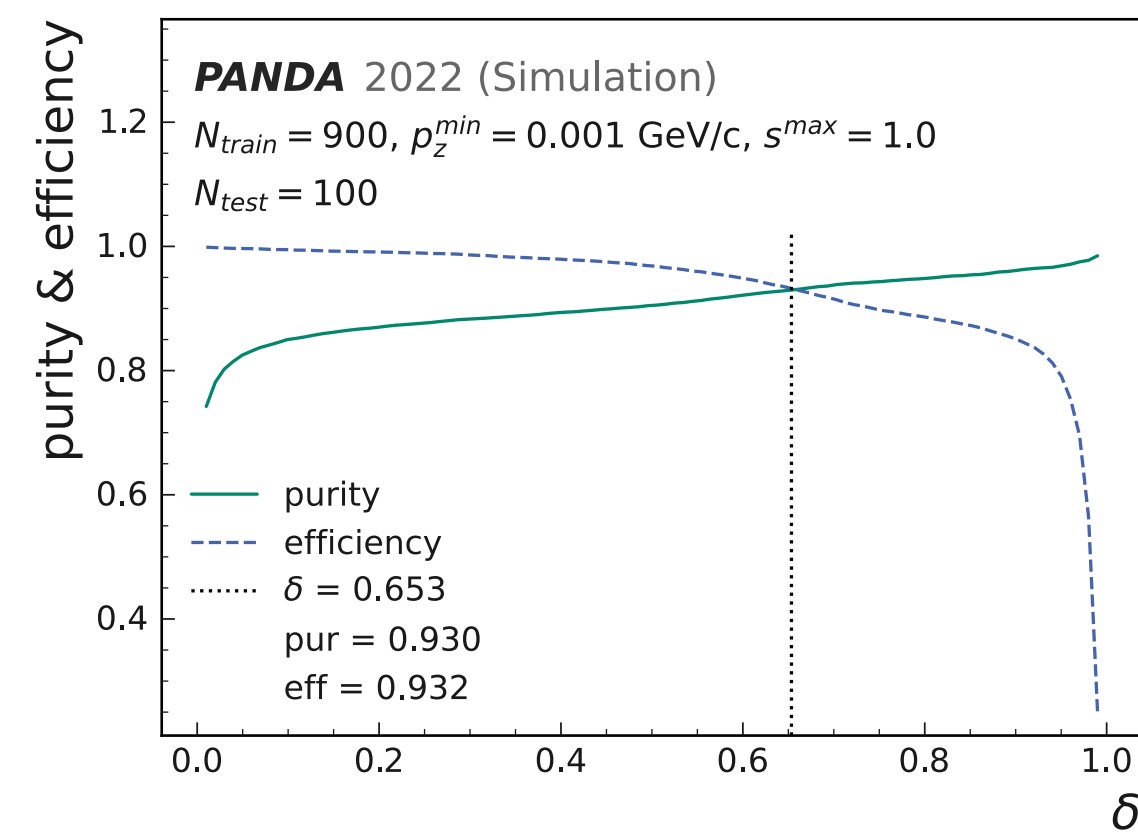


GNN Edge Classification

classification results

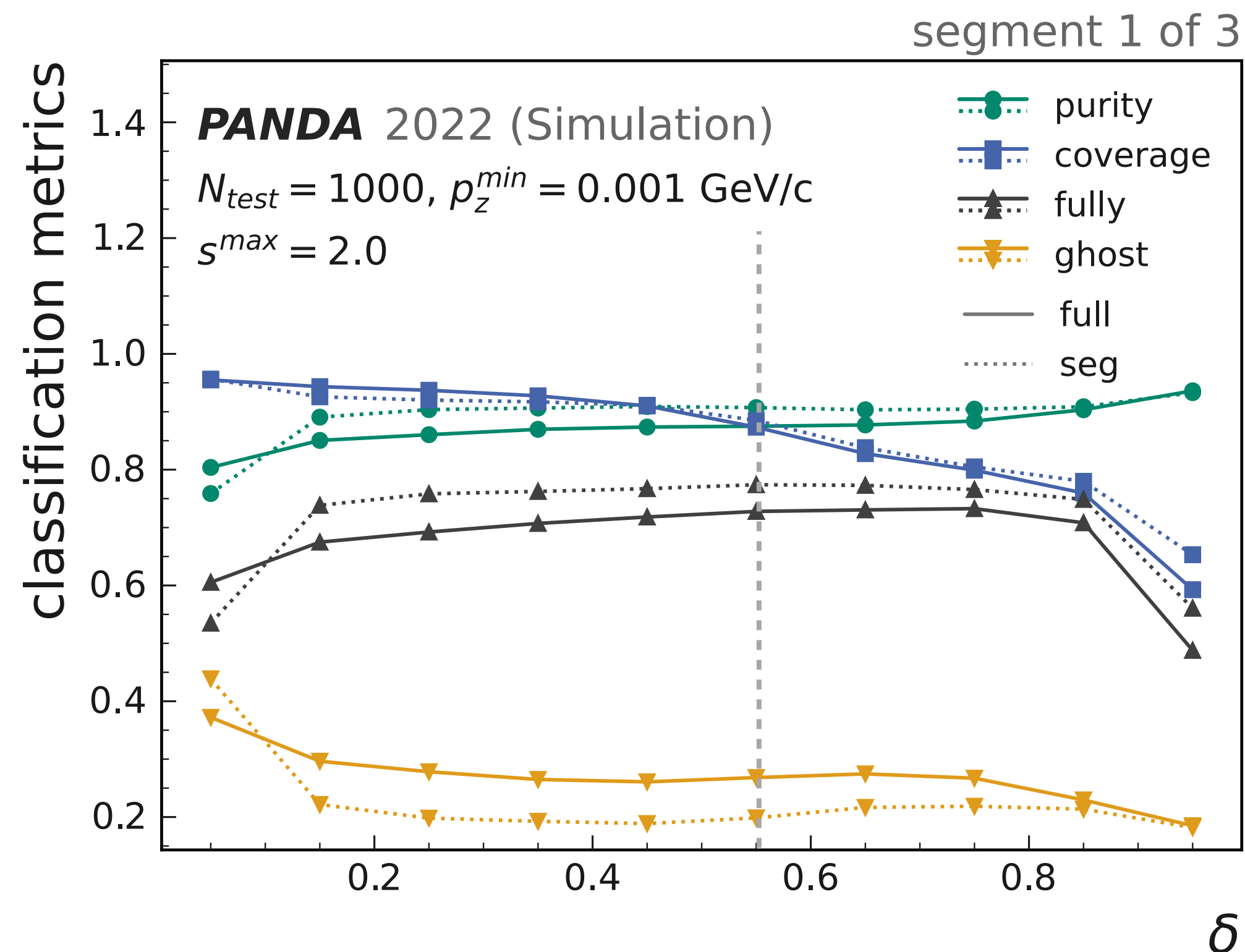


application of classification threshold δ



Tracklet Finding

- **tracklets**: track candidates for each of the 3 segments (max. length = 8 hits)
- tracklets computed via Python package NetworkX



$$\text{tracklet purity} = \frac{1}{N_{\text{tracklets}}} \sum_{N_{\text{tracklets}}} \frac{N_{\text{majority particle}}}{N_{\text{tracklet hits}}}$$

$$\text{MC coverage} = \frac{1}{N_{\text{tracklets}}} \sum_{N_{\text{tracklets}}} \frac{N_{\text{majority particle}}}{N_{\text{MC}}}$$

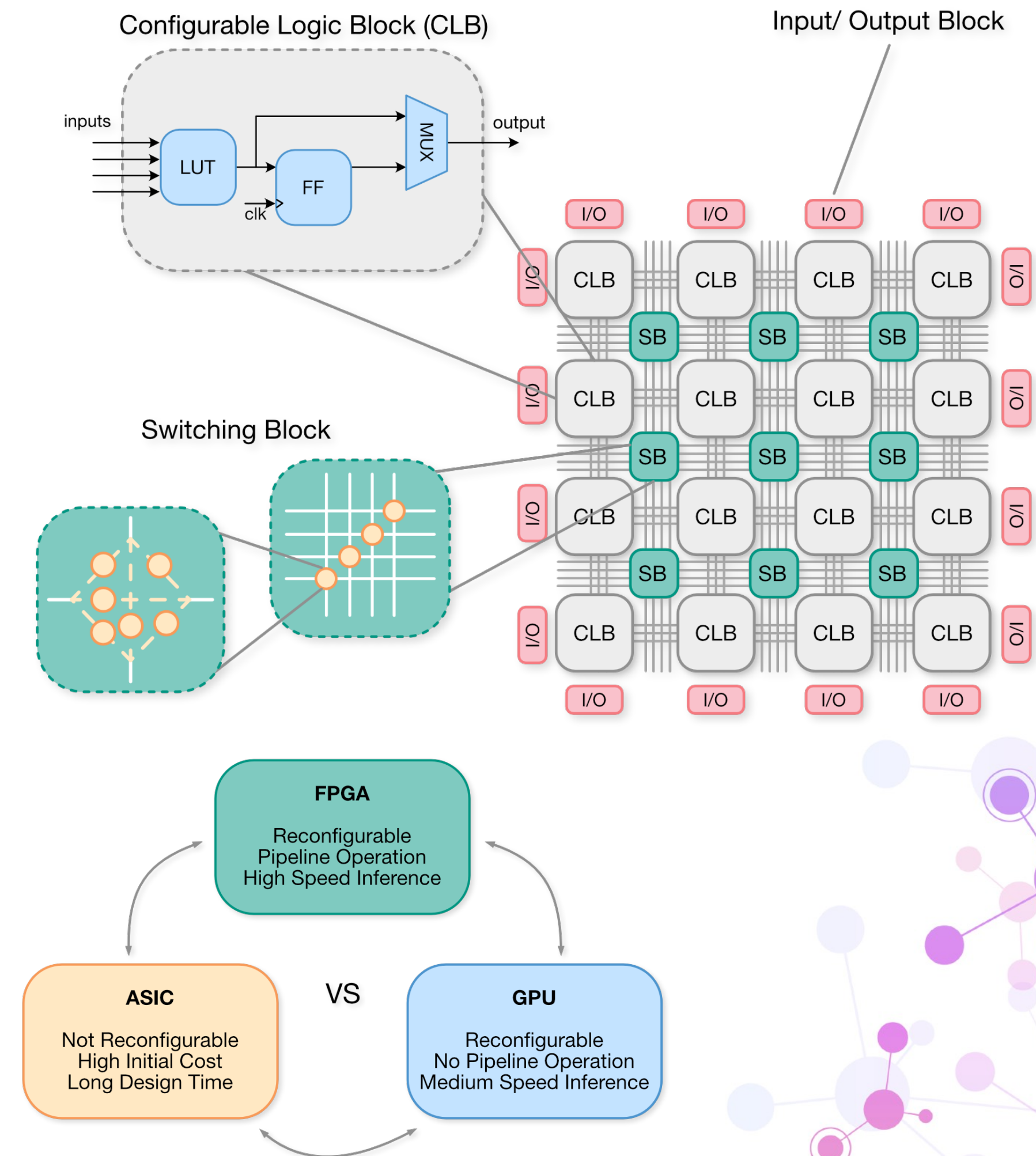
$$\text{fully found rate} = \frac{N_{\text{full}}}{N_{\text{tracklets}}}$$

$$\text{ghost rate} = \frac{N_{\text{ghost}}}{N_{\text{tracklets}}}$$

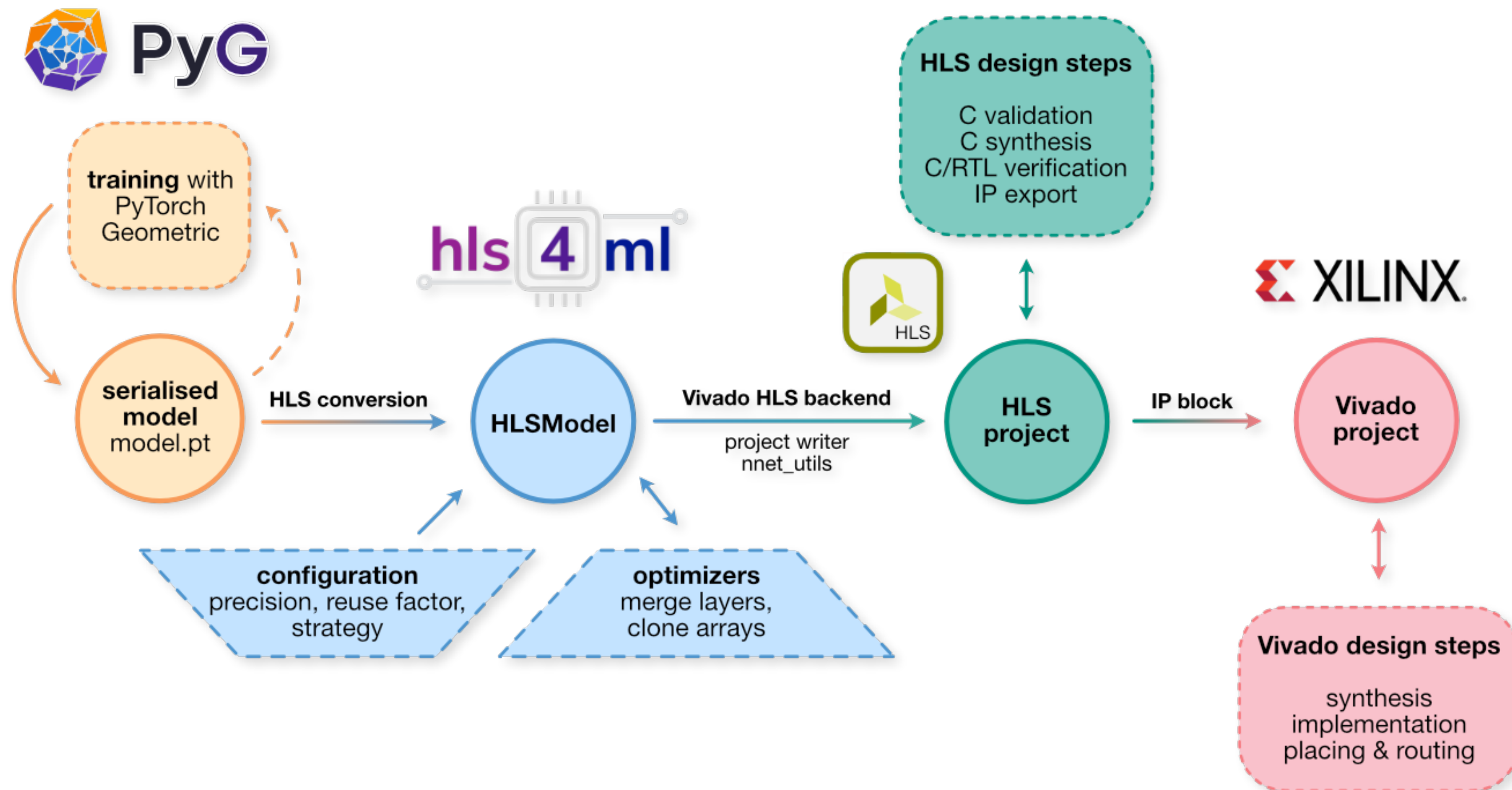


FPGA Technology

- programmable digital integrated circuits
- FPGAs support high data throughput and low latency data processing at high flexibility
- common in HEP DAQ systems
- general tradeoff between resource usage and latency/throughput requirements
- design optimisations needed to meet performance requirements
- building blocks: lookup tables (LUTs), flip-flops (FFs), block random access memories (BRAMs) and digital signal processors (DSPs)



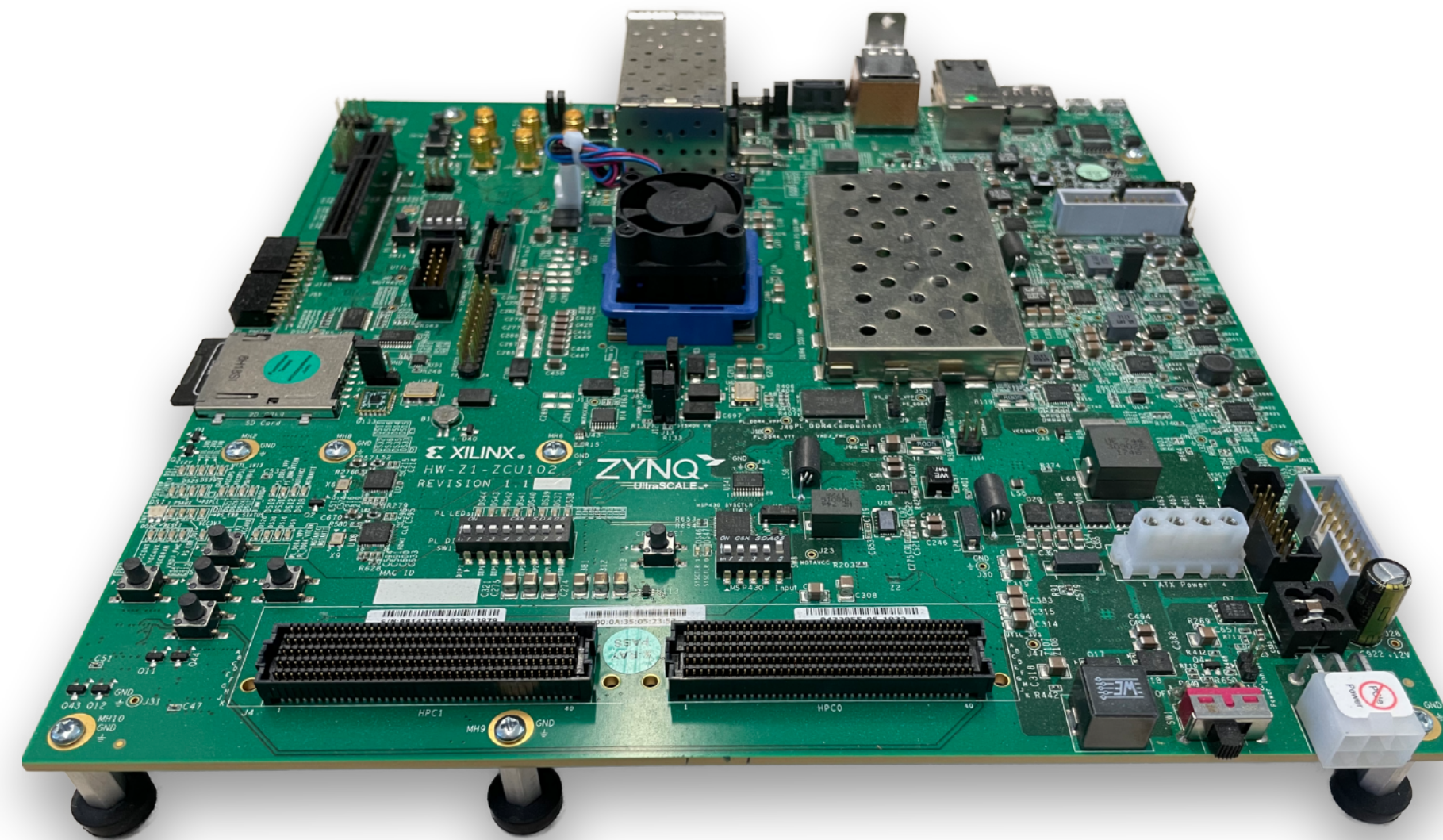
Highlevel Synthesis for Machine Learning (hls4ml)



- automatic translation of ML models to hardware level
- based on Vivado HLS
- fast prototyping by automated workflow
- several configuration parameters

Working Environment and Benchmark Model

- Xilinx Zynq® UltraScale+ MPSoC FPGA (part number xczu11eg-ffvc1760-2-e)
- no variable-size input data possible → truncation and zero-padding using 95% rule (corresponds to graphs with 49 nodes and 98 edges)
- benchmark model: node and edge dimension $D_{\text{node}} = D_{\text{edge}} = 2$, number of nodes $N_{\text{nodes}} = 28$, $N_{\text{edges}} = 56$, reuse factor $RF = 8$



Design Optimization: Quantization

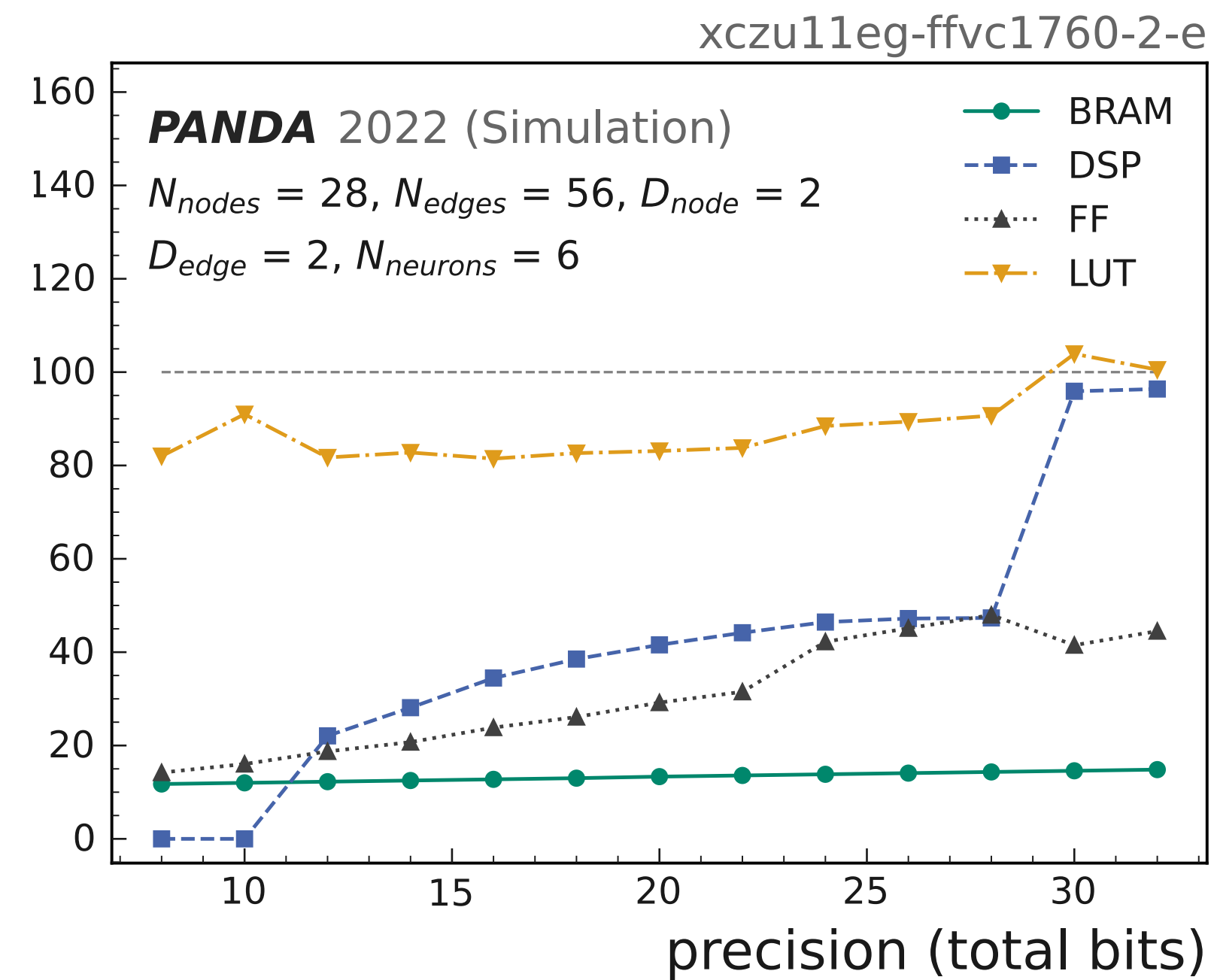
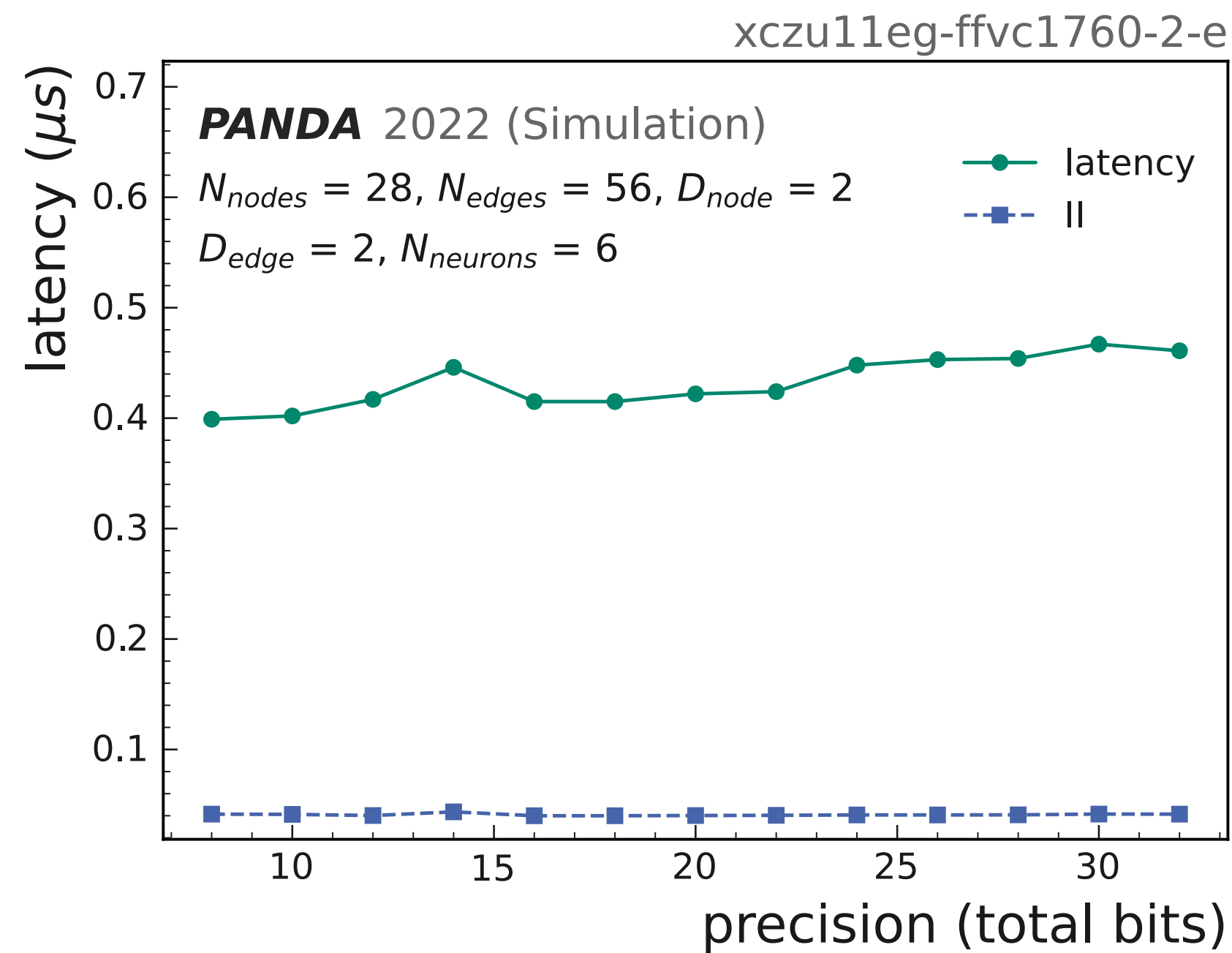
- reduction of number of bits used representing the NN model (precision)

- data representation by arbitrary precision fixed-point data format $ap_fixed\langle W, I \rangle$

- bit widths directly affect resource usage

integer length
signed integer bits above
binary point

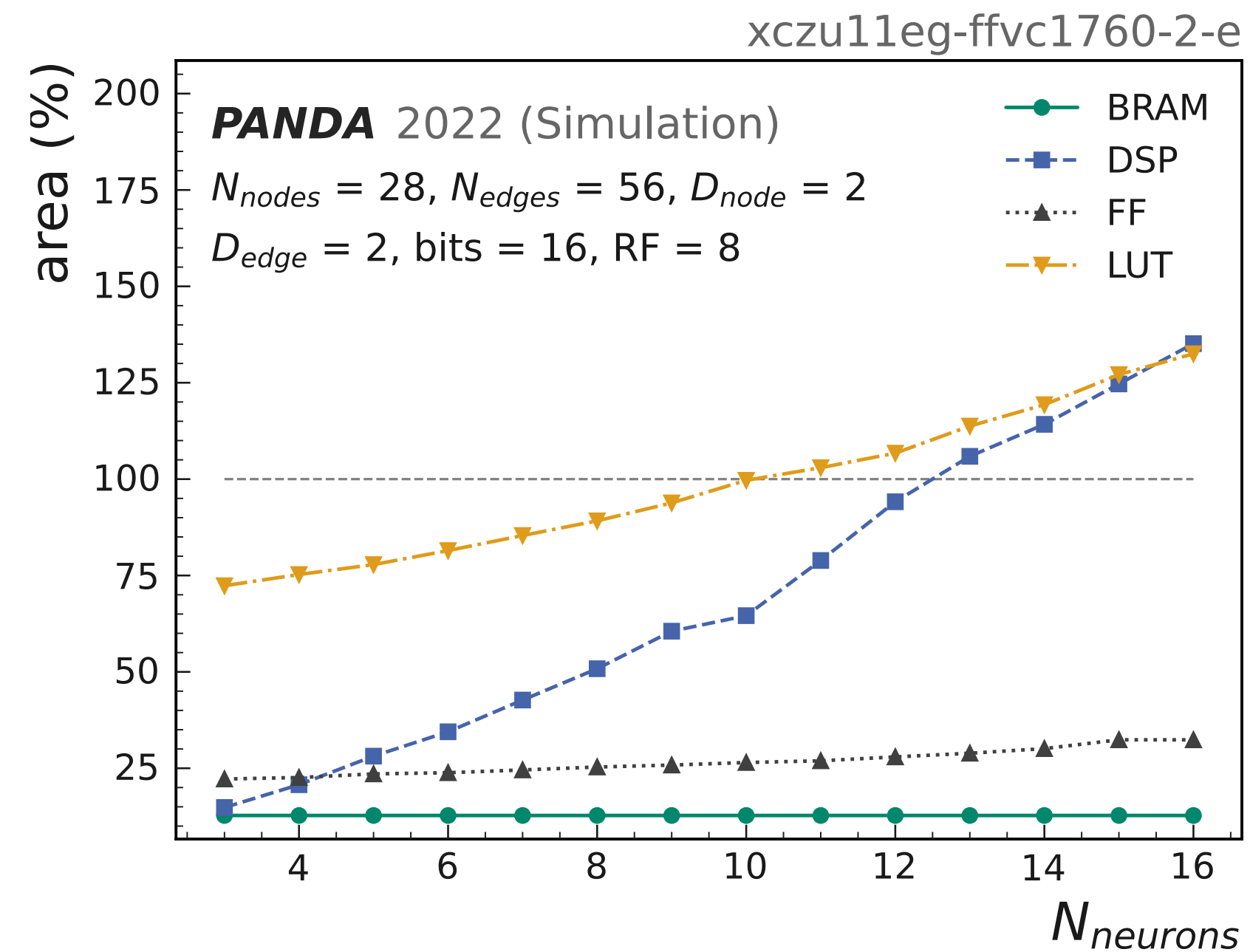
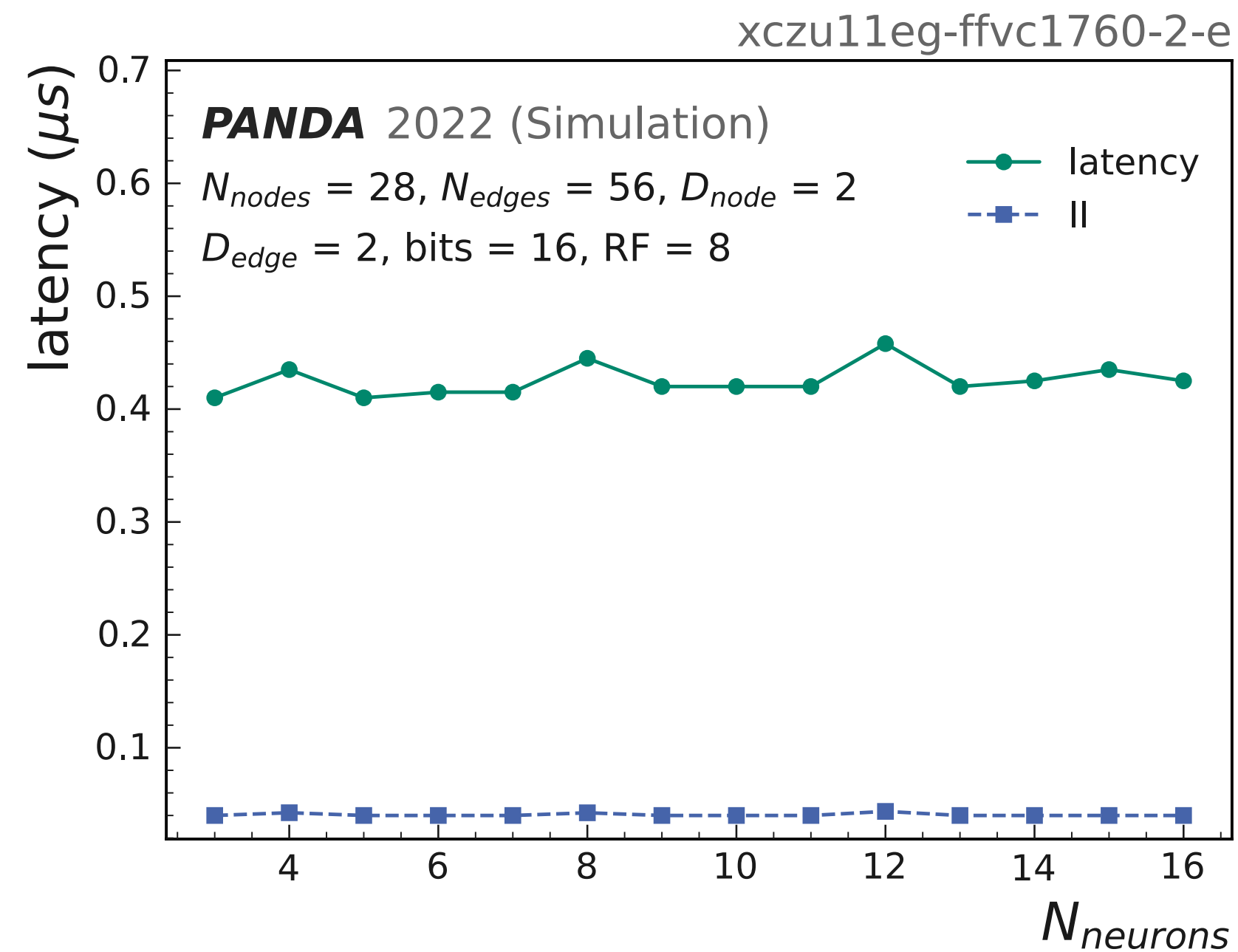
word length
total number of bits



here $I = W/2$

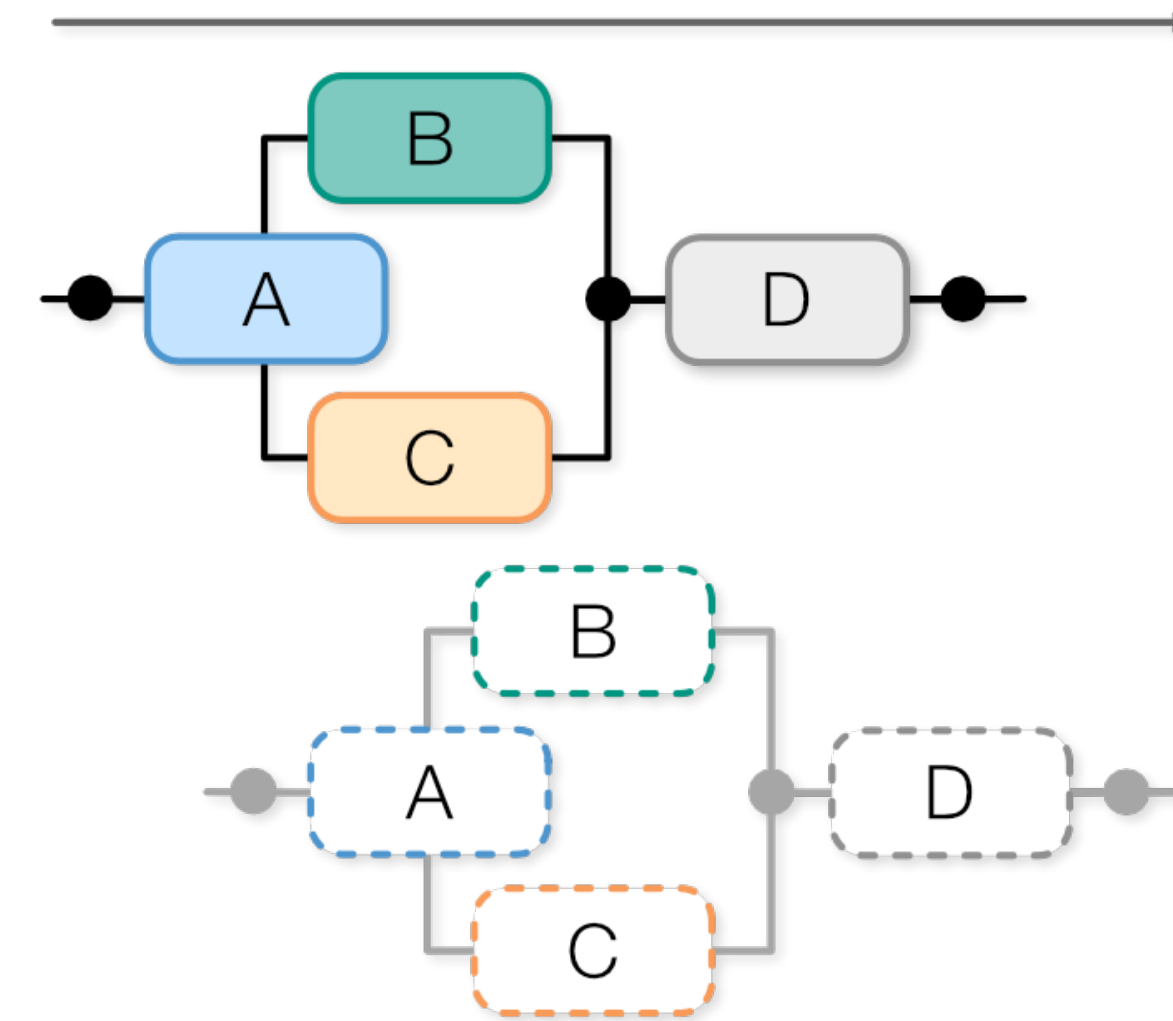
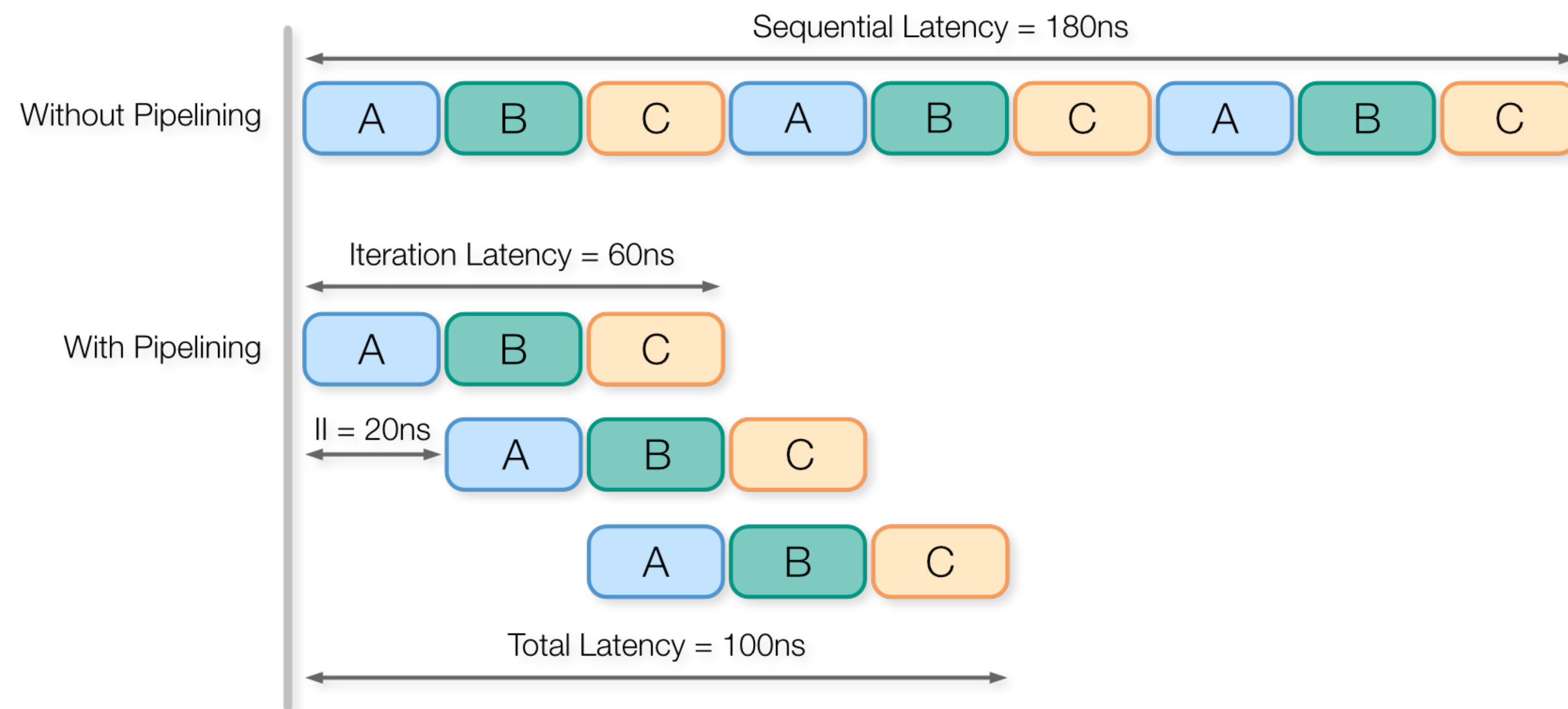
Design Optimization: Compression

- reduction of model parameter number
- number of model parameters strongly depends on the number of hidden nodes/ neurons



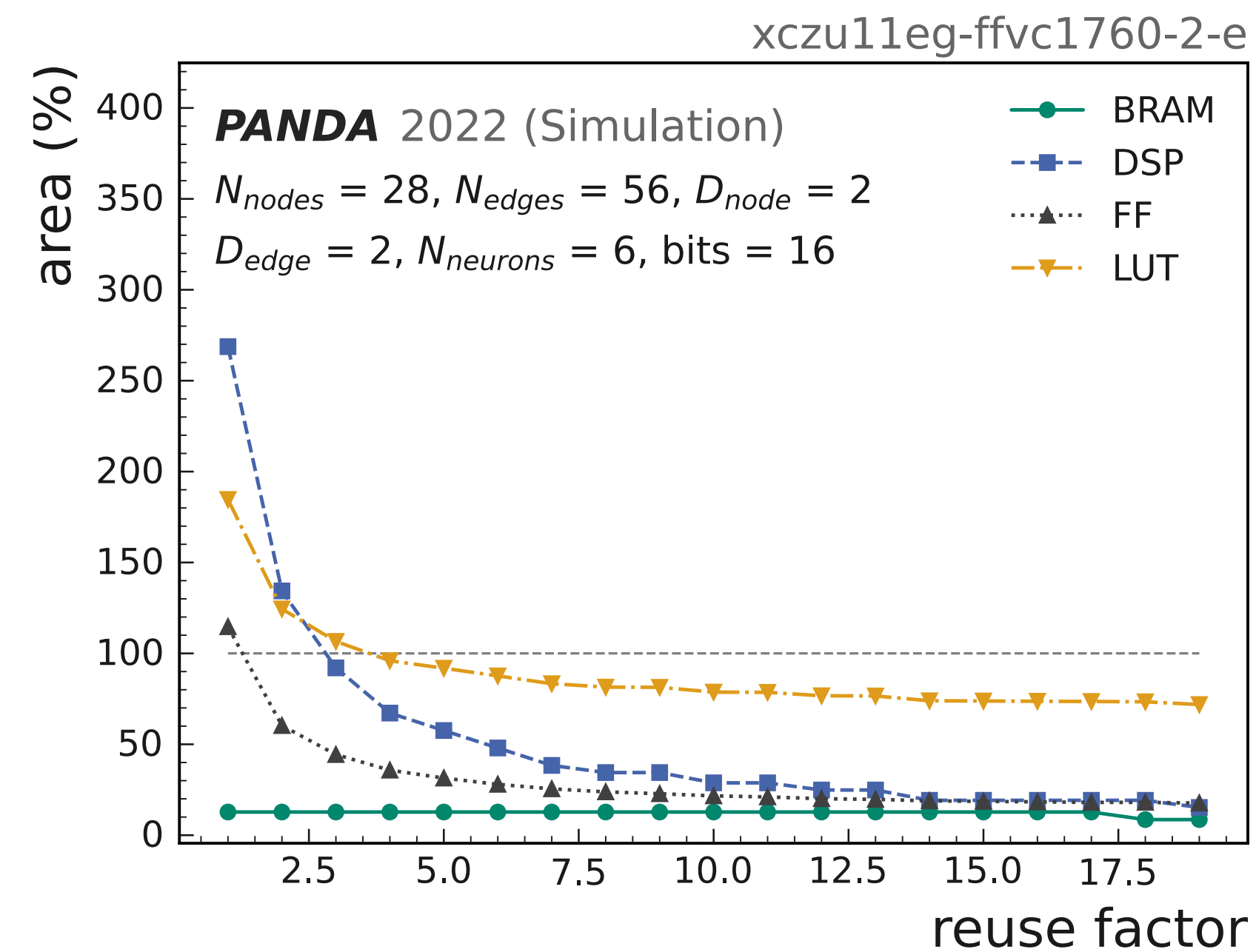
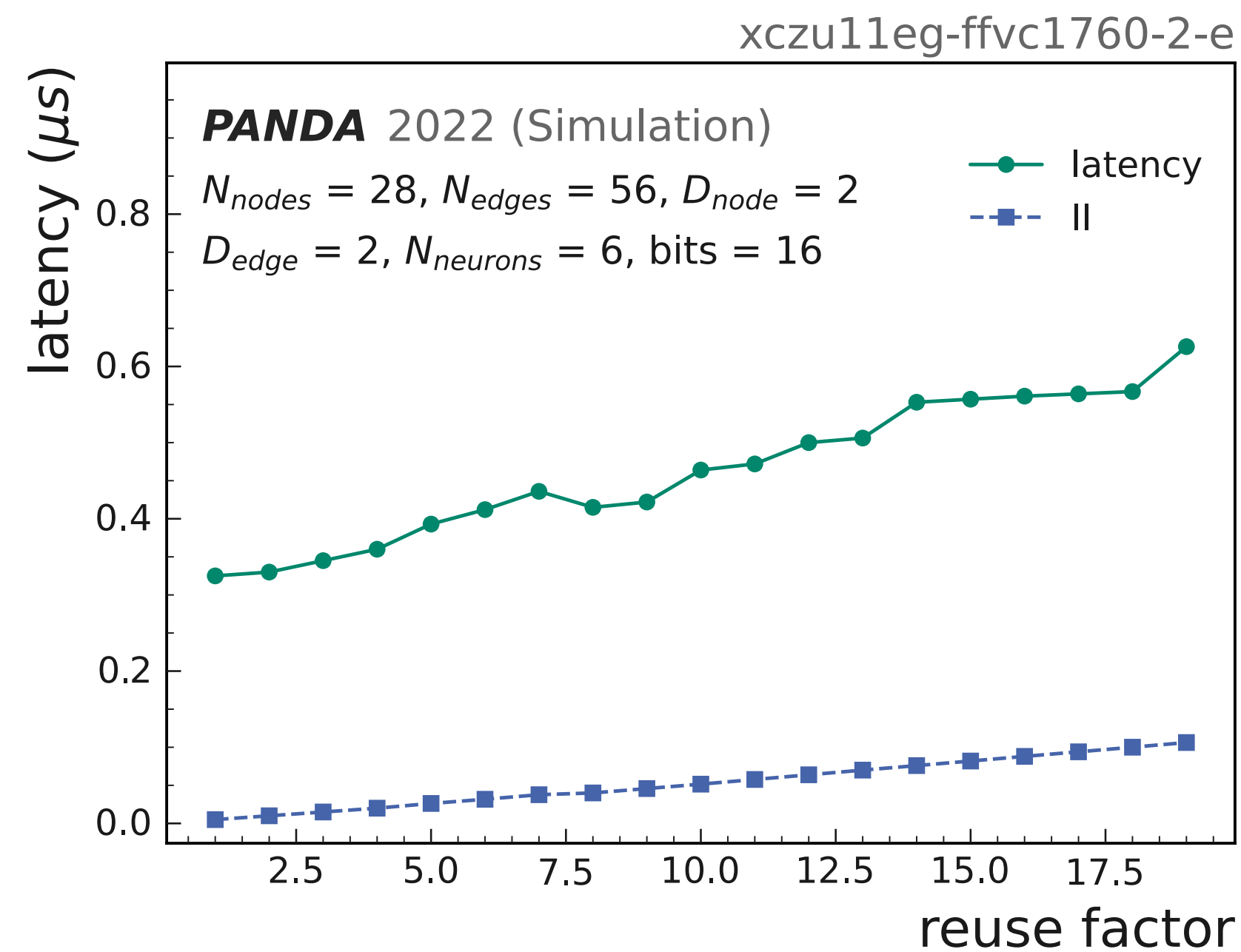
Design Optimization: Pipelining

- key advantage of FPGAs: throughput acceleration by parallelisation and pipelining
- total latency = iteration latency + $II \cdot (\text{number of functions} - 1)$
- pipelining includes task parallelism, pipelining within a run, of runs or within a task



Design Optimization: Pipelining

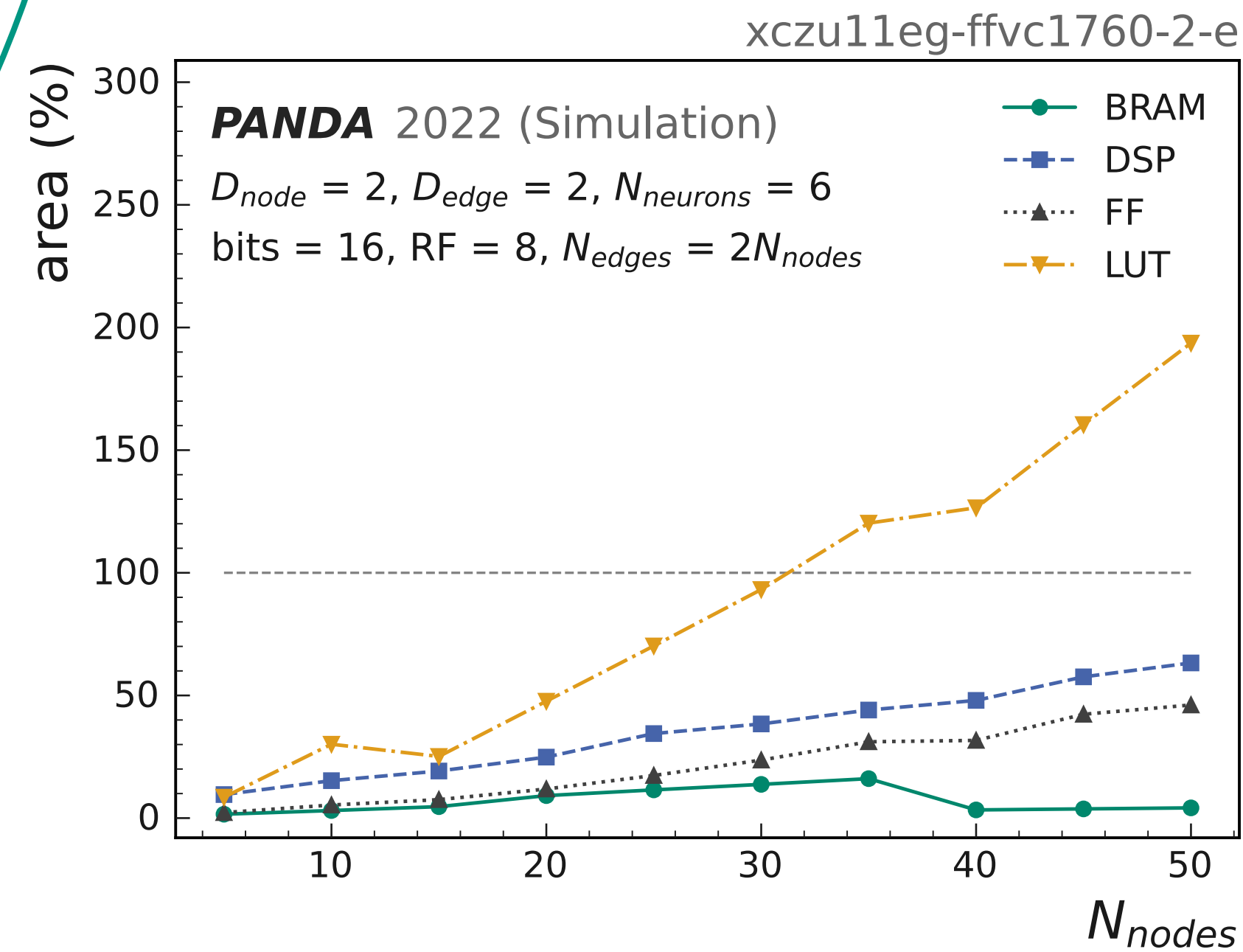
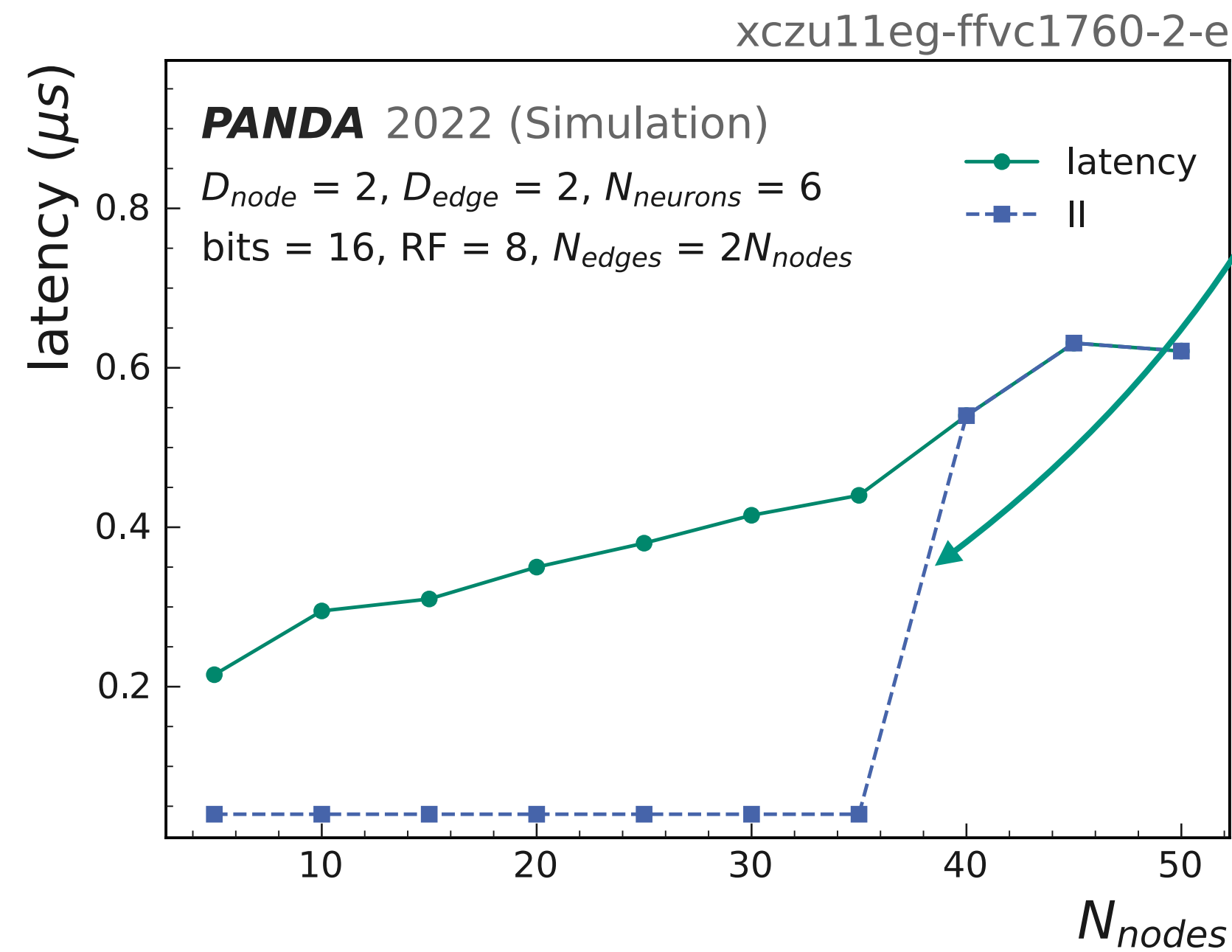
- key advantage of FPGAs: throughput acceleration by parallelisation
- initiation interval II equals the hls4ml internal reuse factor RF



Design Optimization: Graph Dimensions

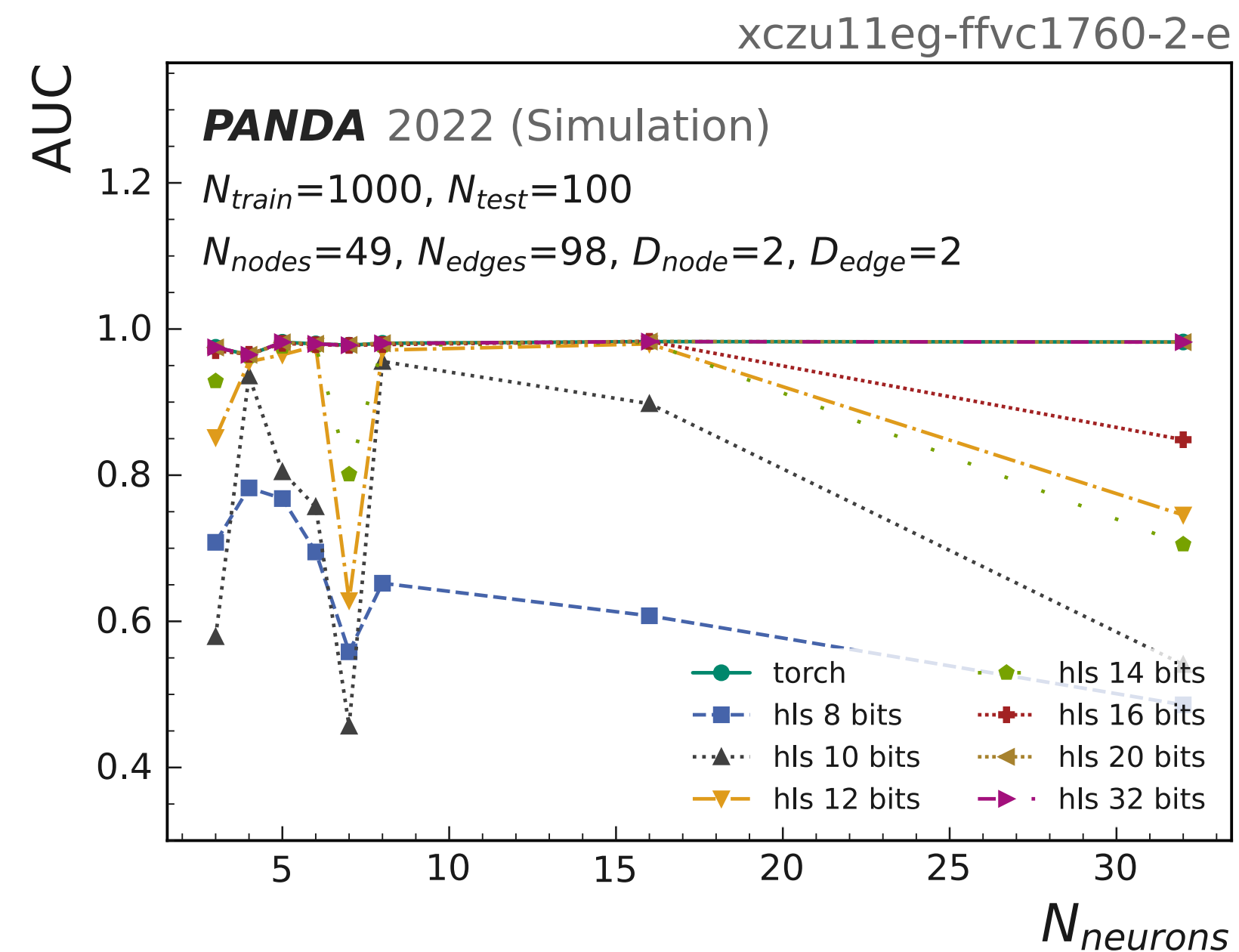
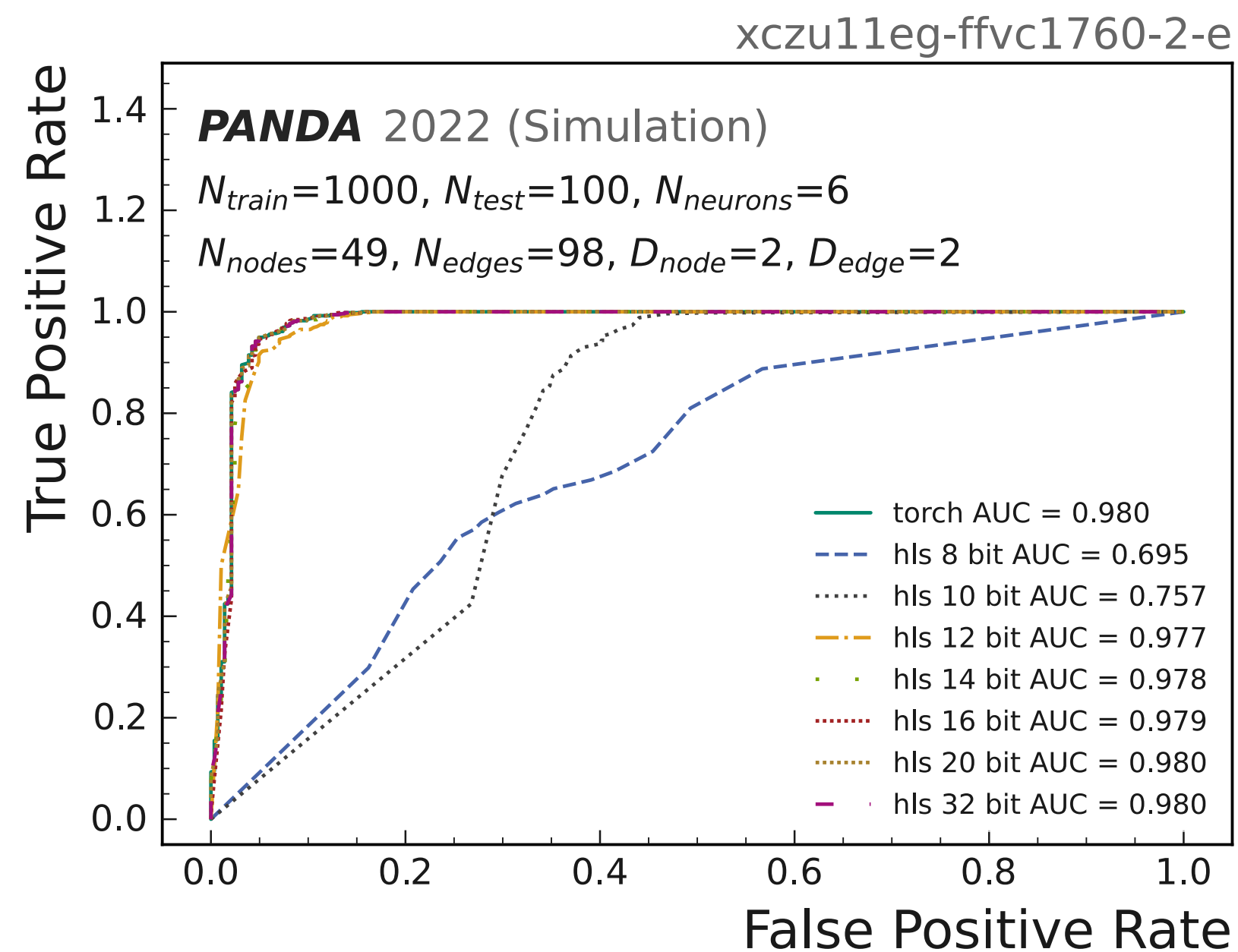
■ ratio of $N_{edges} = N_{nodes}$

functions executed sequentially (too complicated control-flow)



Classification Performance

- graph size $N_{\text{nodes}} = 49$, $N_{\text{edges}} = 98$, number of neurons $N_{\text{neurons}} = 6$
- precision = $ap_{\text{fixed}}\langle 16, 8 \rangle$
- **problem:** this configuration exceeds resources



Design Optimization

- goal: find a design that satisfies latency, resource and classification requirements
- hands-on adjustment using Vivado HLS design directives (pragmas) including ARRAY_PARTITION, UNROLL, PIPELINE, DATAFLOW

#	RF	Description	Latency [cycles]	II [cycles]	DSP [%]	LUT [%]	FF [%]	BRAM [%]
1	1	throughput-optimized design	103	103	546	386	88	2
2	8	throughput-optimized design	124	124	63	193	46	4
3	16	throughput-optimized design	139	139	34	175	42	4
4	32	throughput-optimized design	173	173	19	169	40	4
5	16	edge-aggregation without array partitioning	384	247	34	85	17	4
6	16	edge-aggregation with array partitioning factor = 2	311	247	34	85	17	4
7	16	# 5 with dataflow design	290	247	34	104	18	113
8	16	loop unrolling with factor parallelization factor = 16	85	85	34	169	20	4
9	16	#6 with 2ns clock period	345	247	34	87	23	4

Conclusion

- successful application of a GNN approach on charged particle tracking
- edge classification with true edge efficiency of 98.9% at true edge purity of 63.0%
- Interaction network architecture with only 275 parameters used
- tracklet finding with up to 77% of full found tracklets
- application on FPGA using 34% of DSPs and 85% of LUTs at a total latency of $0.99\mu\text{s}$ enabling real-time analysis



Outlook

- include skewed detector layer hits for 3D information
- include next-to-next layer and same-layer edges
- extend to full track finding and reconstruction
- investigate more sophisticated FPGA design optimization methods
- implement on Versal board with much more resources using Vitis AI
- integration in PandaRoot
- application on real FPGA



THE END



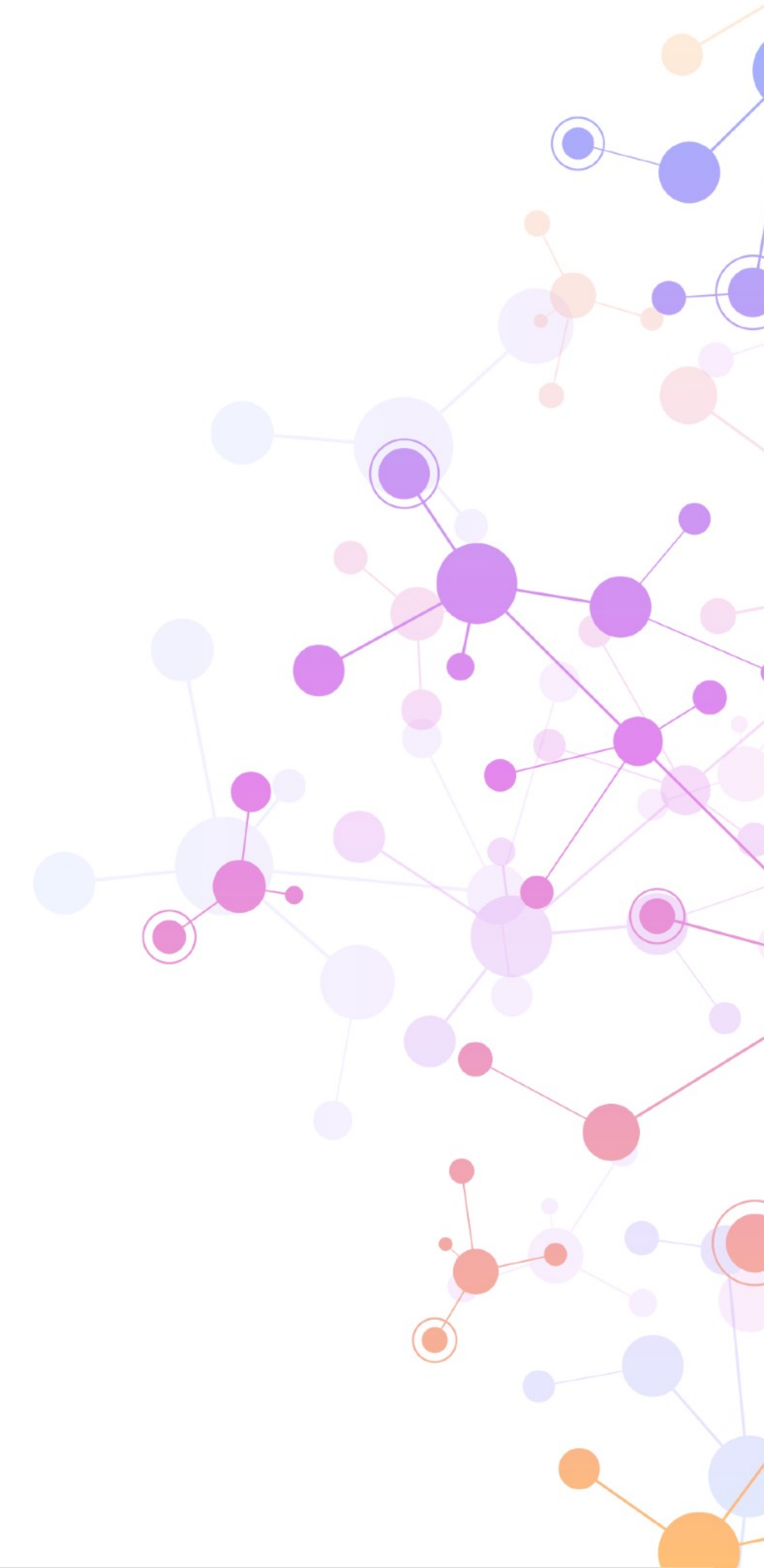
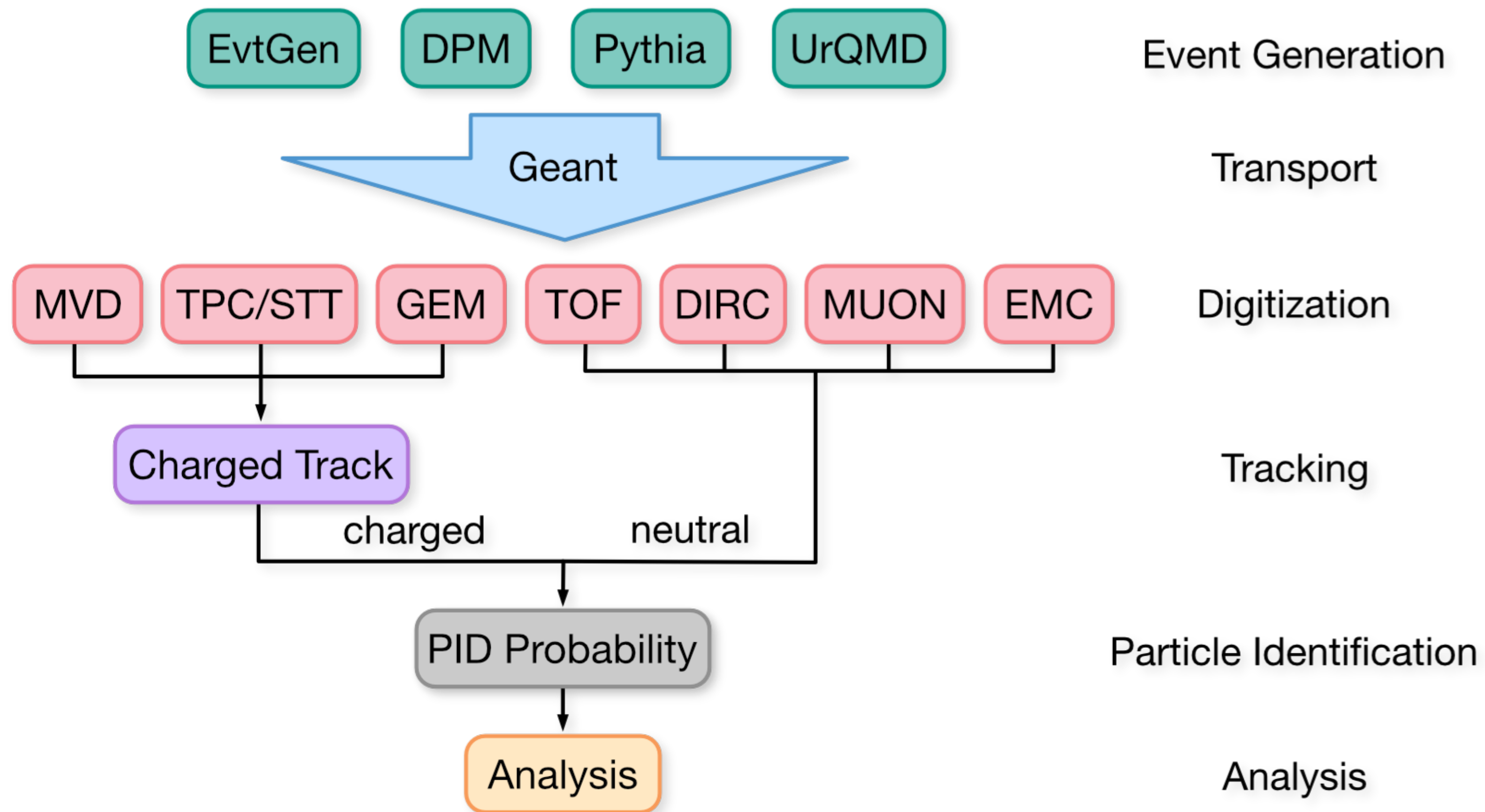
BACKUP



Definition of metrics

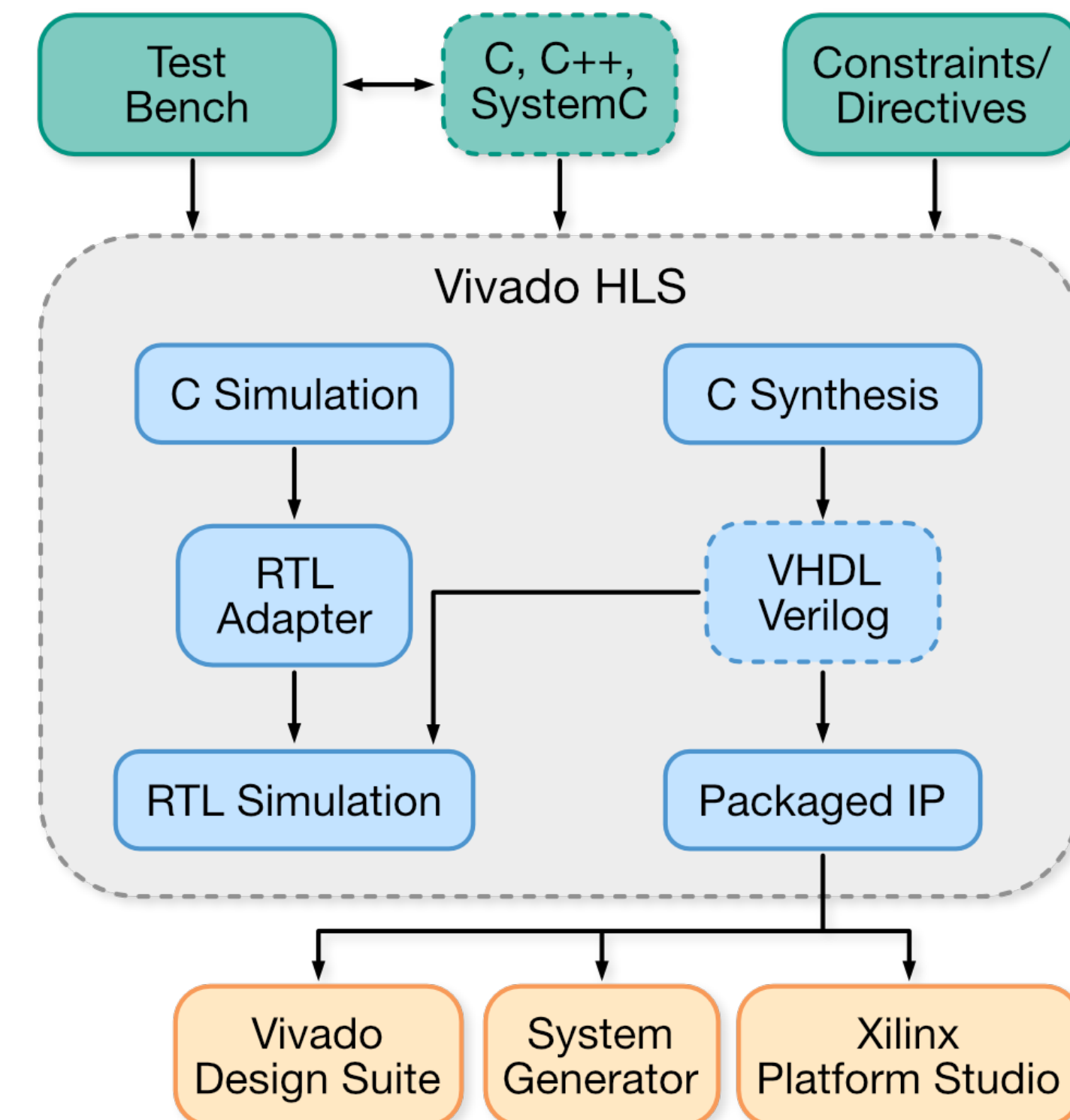
- **Purity** (precision, positive predictive value): $PPV = \frac{TP}{TP + FP}$
- **Efficiency** (sensitivity, recall, hit rate, true positive rate): $TPR = \frac{TP}{TP + FN}$
- **Specificity** (selectivity, true negative rate): $TNR = \frac{TN}{N} = \frac{TN}{TN + FP} = 1 - FPR$
- **Miss rate** (false negative rate): $FNR = \frac{FN}{P} = \frac{FN}{FN + TP} = 1 - TPR$
- **Accuracy** (ACC): $ACC = \frac{TP + TN}{P + N} = \frac{TP + TN}{TP + TN + FP + FN}$

		Predicted		
		False	True	
Actual	False 60	True Negative (TN) 50	False Positive (FP) 10	Specificity $\frac{TN}{TN+FP}$
	True 105	False Negative (FN) 5	True Positive (TP) 100	Efficiency $\frac{TP}{TP+FN}$
		Negative Predictive Value	Purity $\frac{TP}{TP+FP}$	Accuracy $\frac{TP}{TP+TN+FP+FN}$



Vivado/ Vivado HLS

- Vivado is an integrated design environment of Xilinx device systems
- Vivado HLS is a compiler enabling conversion of high-level languages such as C, C++ and System C to HDLs
- Vivado HLS design workflow consists of C synthesis, C simulation, C/RTL Co-Simulation, package and export RTL design as Vivado IP block to Vivado



Number of GNN Parameters

- number of model parameters for each block

$$N_B = N_w + N_b = (D_{in} + D_{hidden} + D_{out}) \cdot D_{hidden} + (2D_{hidden} + D_{out})$$

- relational model R

$$R_1(2 \cdot D_{node} + D_{edge}, D_{edge}, N_{hidden})$$

- object model O

$$O(D_{node} + D_{edge}, D_{node}, N_{hidden})$$

- relational model output block

$$R_2(2 \cdot D_{node} + D_{edge}, 1, N_{hidden})$$

- total number of parameters

$$N_{IN} = N_w + N_b$$

with

$$N_w = D_{hidden} \cdot (6D_{node} + 4D_{edge} + 3D_{hidden} + 1)$$

$$N_b = 6D_{hidden} + D_{edge} + D_{node} + 1$$

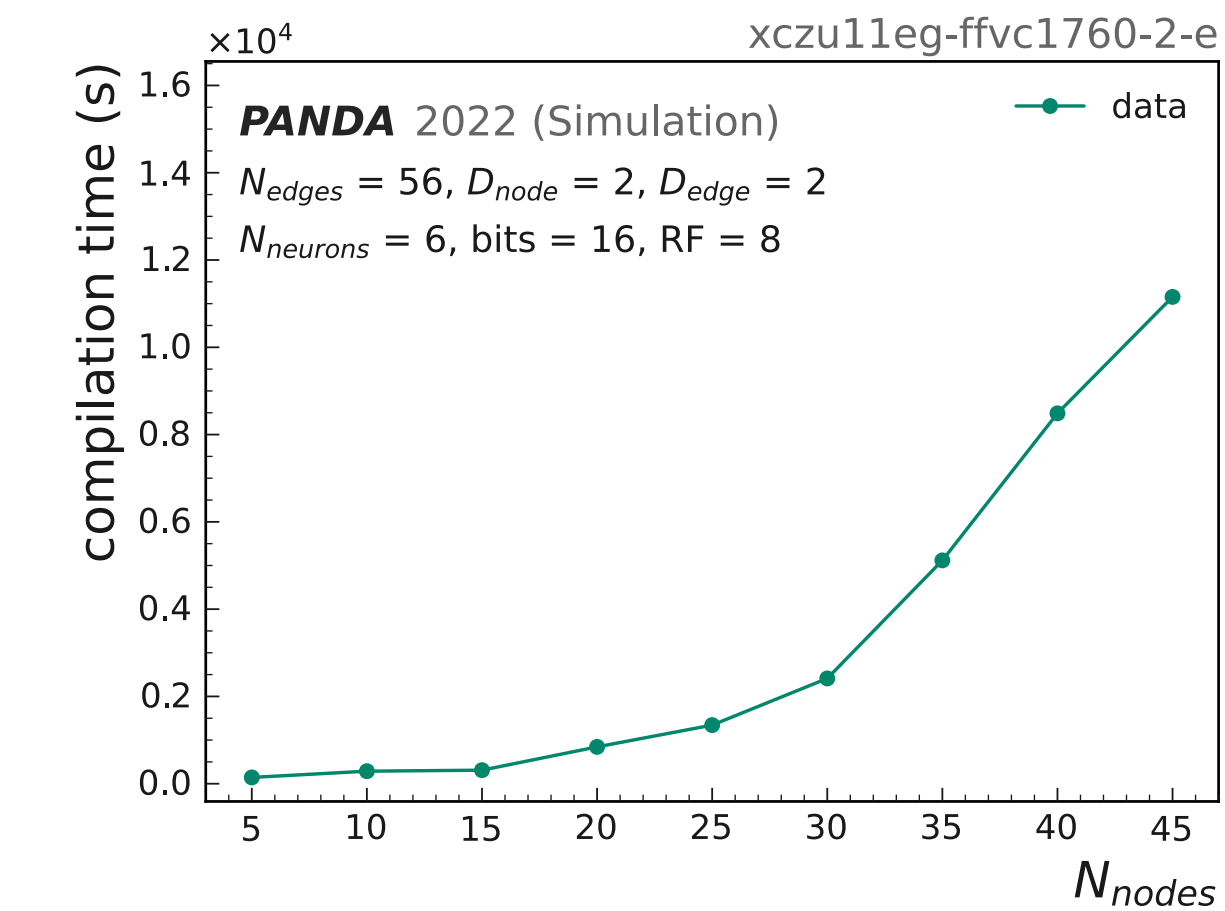
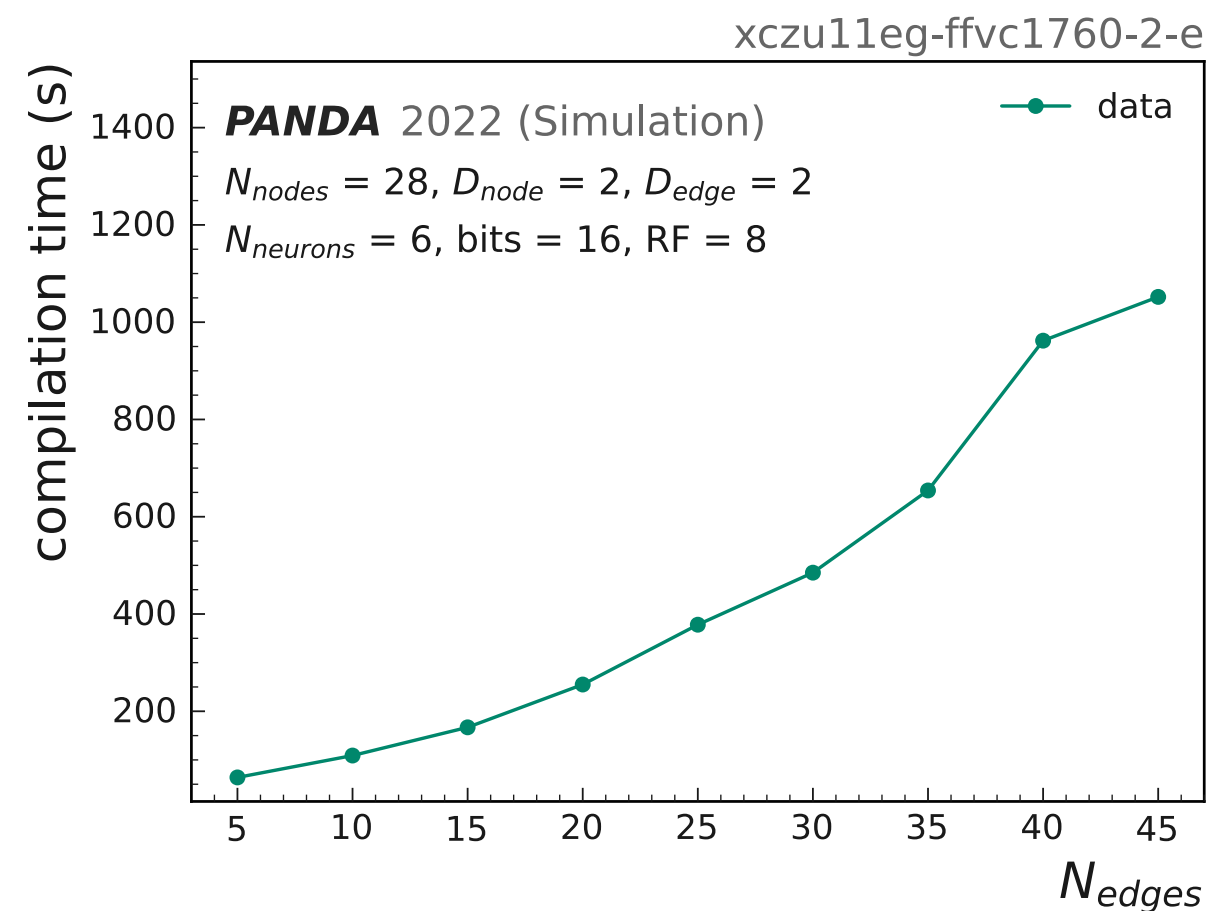
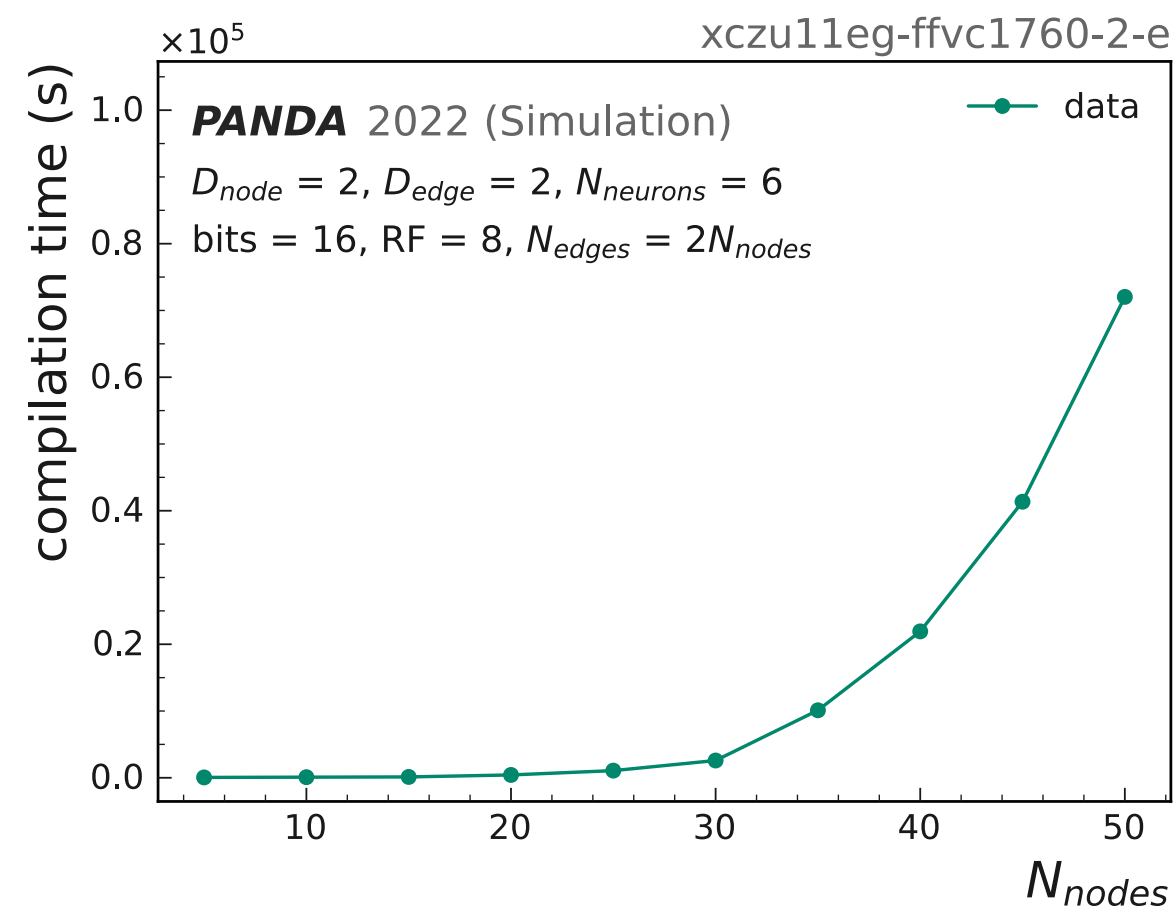
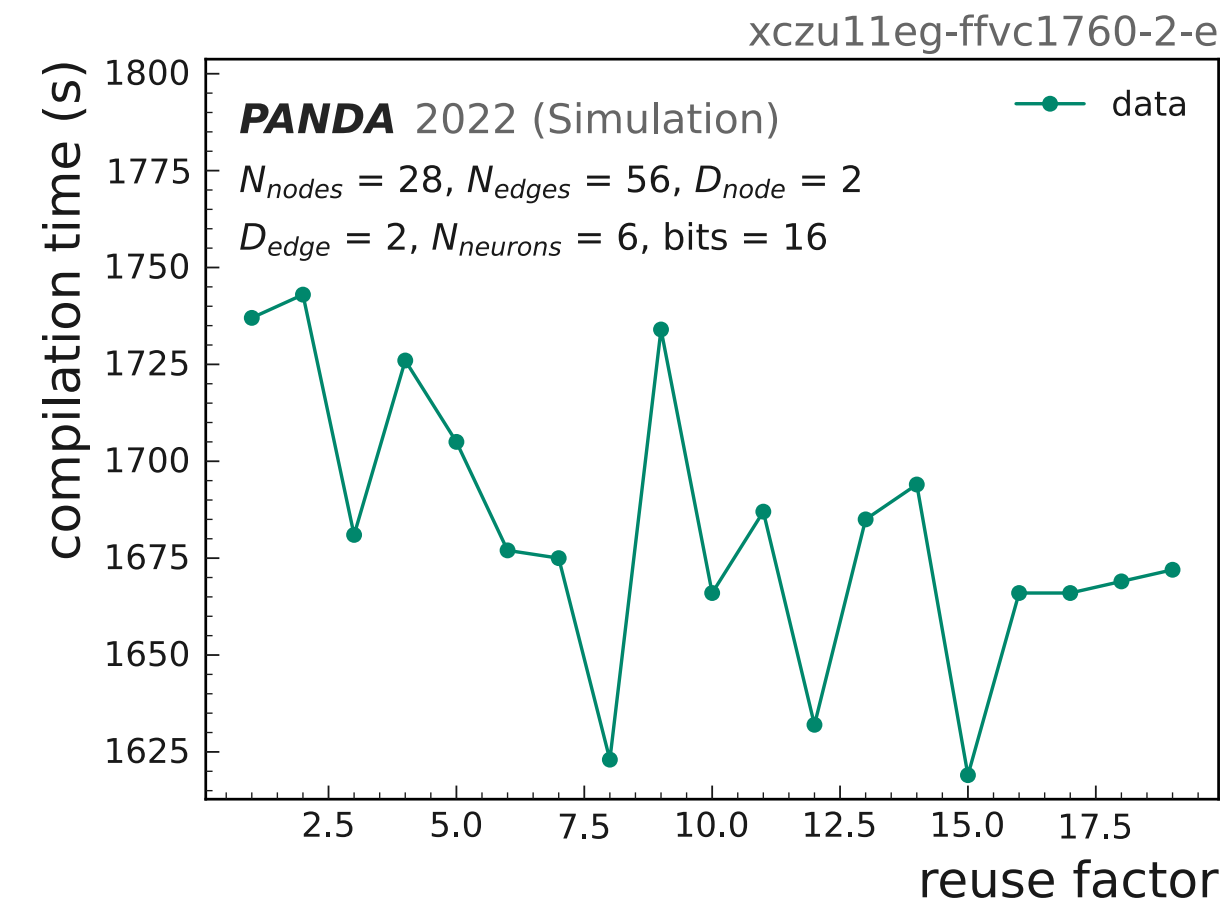
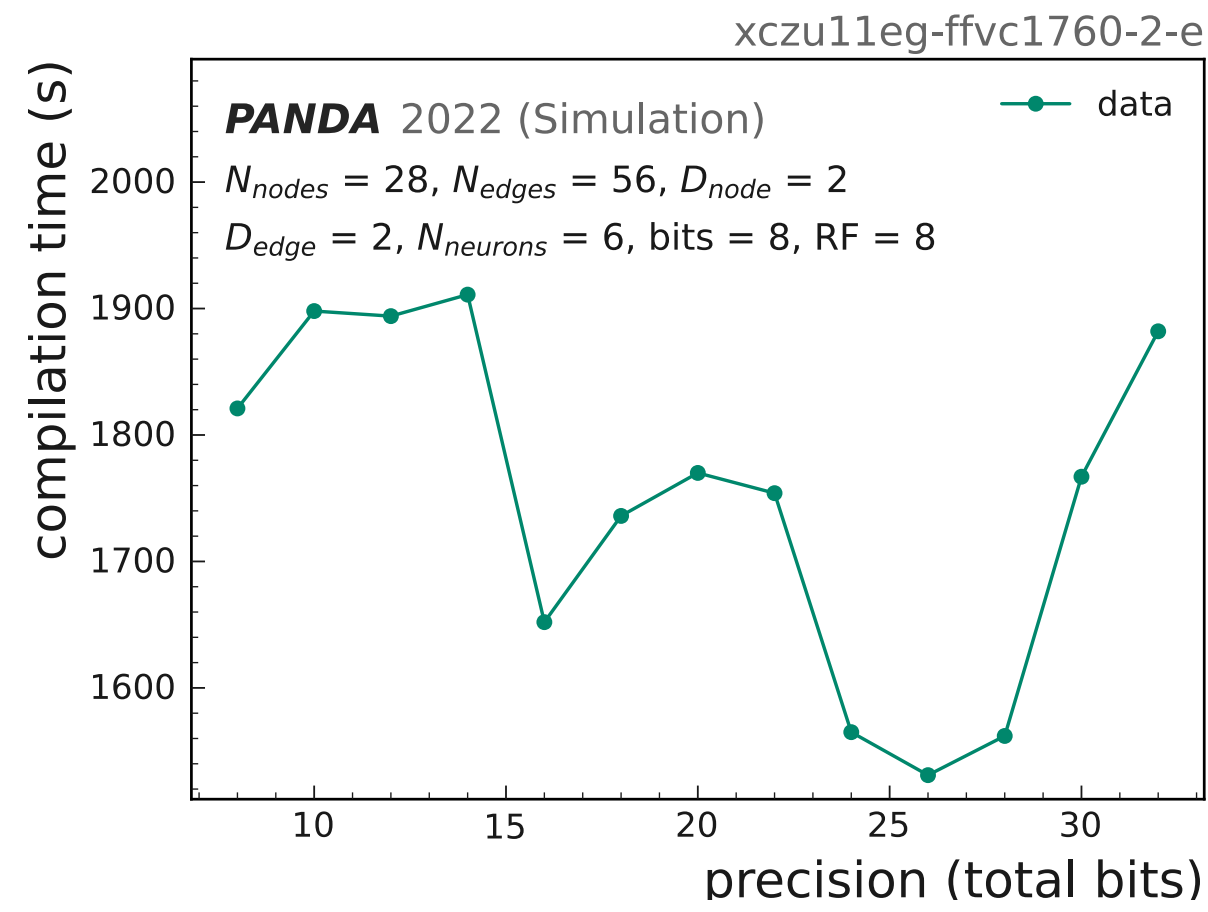
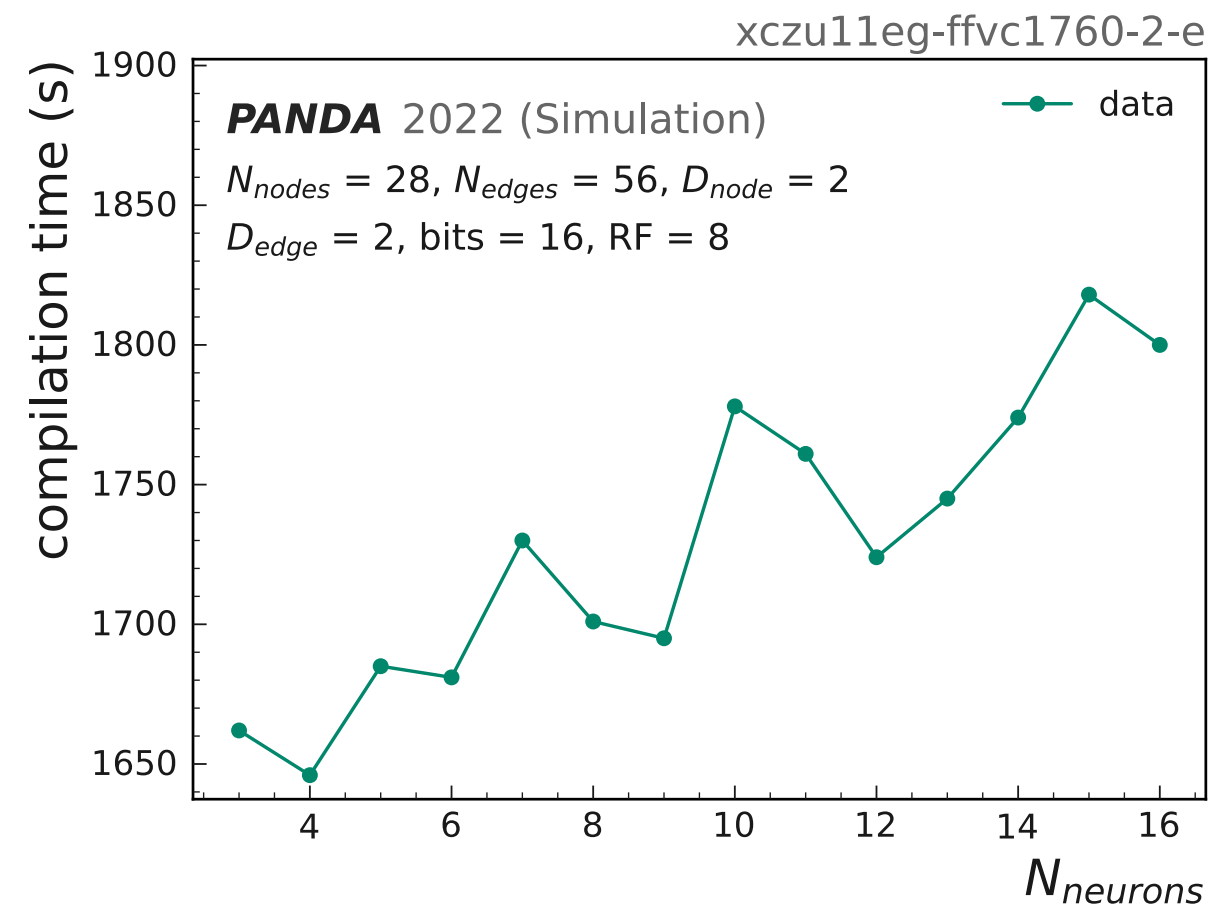
InteractionNetwork(node_dim: 2, edge_dim: 2, hidden_size: 8)

Modules	Parameters
R1.layers.0.weight	48
R1.layers.0.bias	8
R1.layers.2.weight	64
R1.layers.2.bias	8
R1.layers.4.weight	16
R1.layers.4.bias	2
O.layers.0.weight	32
O.layers.0.bias	8
O.layers.2.weight	64
O.layers.2.bias	8
O.layers.4.weight	16
O.layers.4.bias	2
R2.layers.0.weight	48
R2.layers.0.bias	8
R2.layers.2.weight	64
R2.layers.2.bias	8
R2.layers.4.weight	8
R2.layers.4.bias	1

Total Trainable Params: 413

hls4ml Compilation Times


challenge: long compilation times



Code available in Git Repo

 `git@github.com:grheine/IN_hls4ml.git`




README.md 

IN_hls4ml

Installation of packages

- Download the environment file `deep_tracking_env.yml`

```
conda env create -f deep_tracking_env.yml python=3.9
conda activate deep_tracking
pip install -e .
```



- included are also 2 style files for the matplotlib figures in the directory files. These can also be installed manually in your matplotlib installation.

To test the code run the pipeline:

```
git clone https://github.com/grheine/IN_hls4ml.git
cd IN_hls4ml
jupyter notebook IN_pipeline.ipynb
```

A larger data sample of 30.000 events can be downloaded here <https://etpwww.etp.kit.edu/~grheine/raw.h5>

