[XII International Workshop on Ring Imaging
Cherenkov Detectors - RICH2025 – Mainz (2025)]

Online AI-based distributed data reduction for the dual-radiator RICH detector in the ePIC experiment

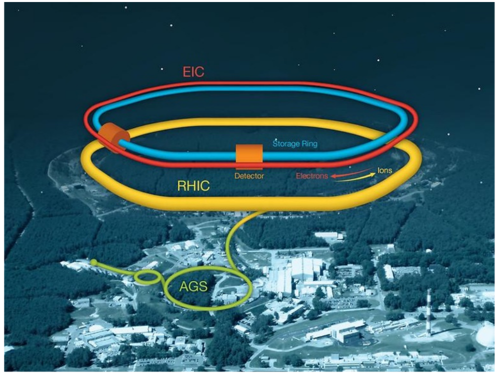
(INFN Sezione di Roma - APE Lab)

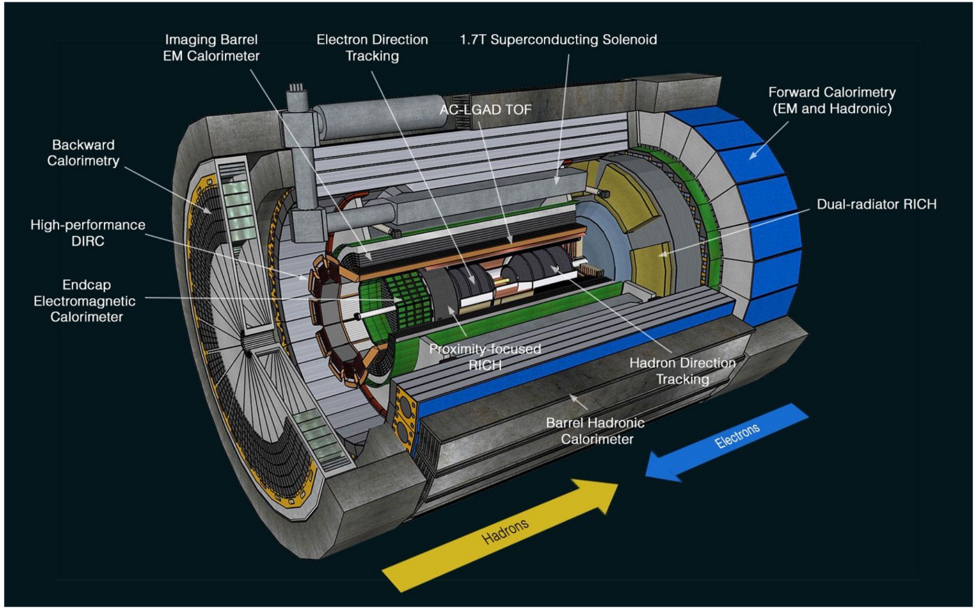
Speaker: Cristian Rossi
(cristian.rossi@roma1.infn.it)

EIC ePIC: overview

The **ePIC collaboration** currently consists of almost 500 members from 171 institutions and is working jointly with the DOE EIC Project to realize the ePIC experiment.
ePIC experiment will be an ~10-meter long cylindrical barrel detector with additional instrumentation that extends to up to 45m in each direction down the EIC beamline.

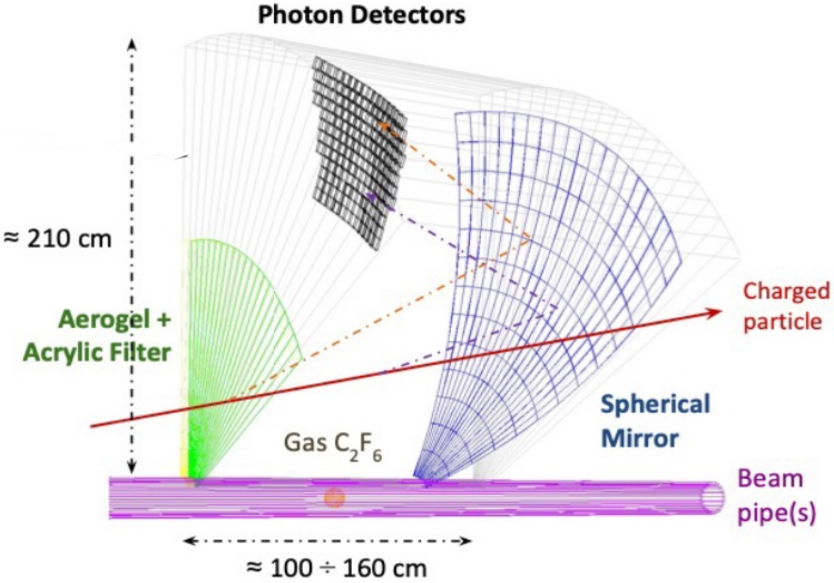
- A 1.7 Tesla superconducting magnet
- High-precision silicon detectors for particle tracking
- Precise calorimeters for measuring particles electromagnetic energy
- A suite of **particle identification (PID)** detectors
- Dense calorimetric detectors to allow the measurement of “jets“

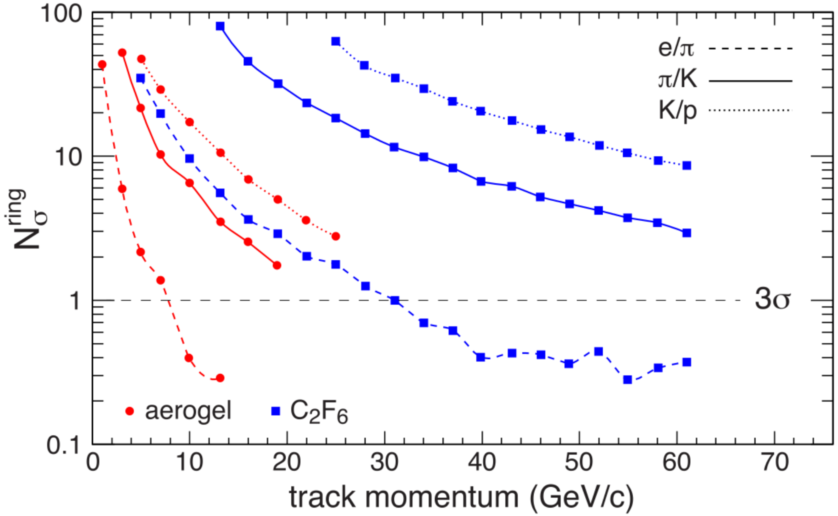



Imaging Barrel EM Calorimeter
Electron Direction Tracking
1.7T Superconducting Solenoid
AC-LGAD TOF
Forward Calorimetry (EM and Hadronic)
Backward Calorimetry
Dual-radiator RICH
High-performance DIRC
Endcap Electromagnetic Calorimeter
Proximity-focused RICH
Hadron Direction Tracking
Barrel Hadronic Calorimeter
Electrons
Hadrons

dRICH ⇒ Design and PID perfomance

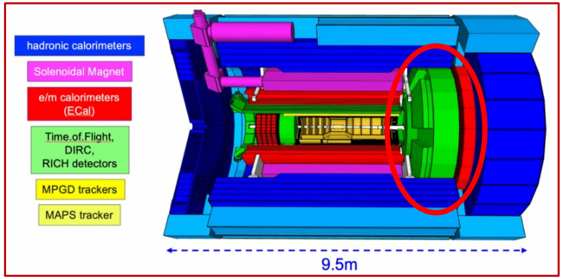
- A **d**ual **R**ing **I**maging **CH**erenkov detector (**dRICH**) will be employed in the forward region ($1.5 < \eta < 3.5$) to provide efficient hadron PID from 3 GeV/c to 50 GeV/c .
- The dRICH comprises two different radiators, aerogel and gas ($C_2 F_6$), to cover the entire momentum range.
- **SiPM based photosensors** are placed in six spherical sectors to detect Cherenkov photons which are focused by six corresponding spherical mirrors.

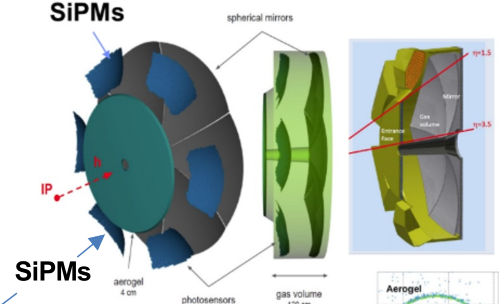




dRICH ⇒ RDO and ePIC DAQ

Forward cap





Aerogel: n = 1. 020 – 1.026

Gas: C_2F_6 (hexafluoroethane)

PDU

FELIX

SERVER

DAM
Data Aggregation Module

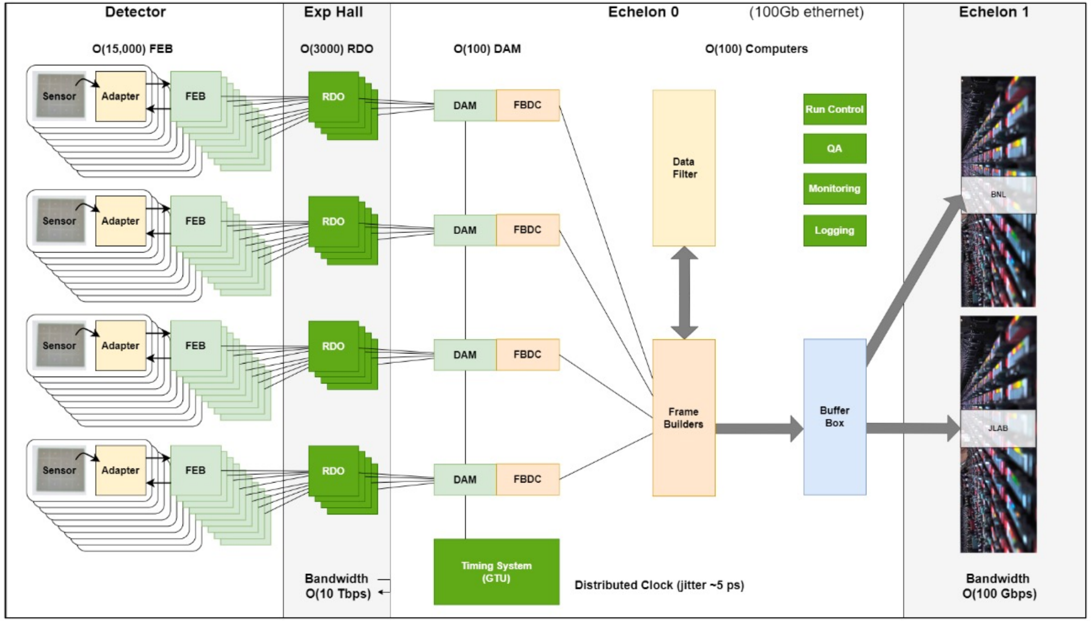
- 42 links from PDUs to Felix-155 board
- 30 Felix-155 boards in total

ePIC processing and storage system

- **1 photodetector unit PDU: 4x64 SiPM array device (256 channels), 4 FEBs, 1 RDO**
- **1248 PDUs for full dRICH readout**
- **319488 readout channels divided in six sectors**

ePIC: DAQ System

- The data from the **Front End Boards (FEBs)** will be aggregated into **Readout Boards (RDOs)** using bidirectional interfaces.
- The RDOs will distribute configuration and control information to the FEBs and read hit data as well as monitoring information from the FEBs.
- The RDOs will also use a bidirectional optical connection to the **Data Aggregation and Manipulation** Board **(DAM)** FPGA-based PCIe cards.

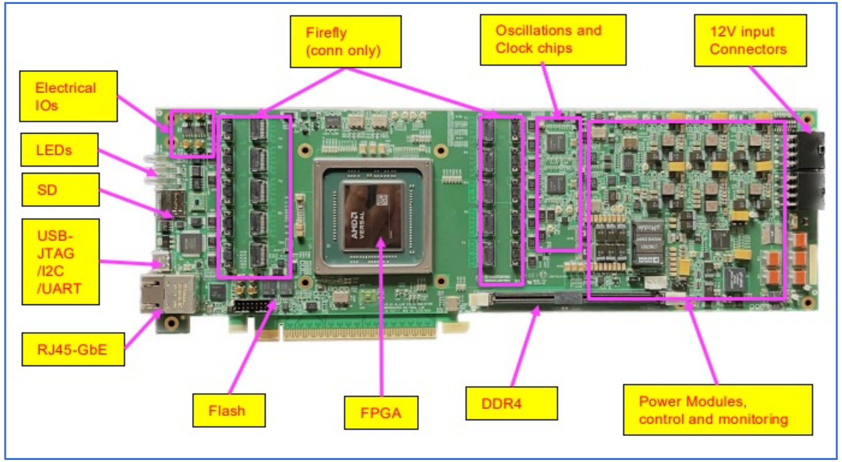


ePIC: DAM boards (FELIX)

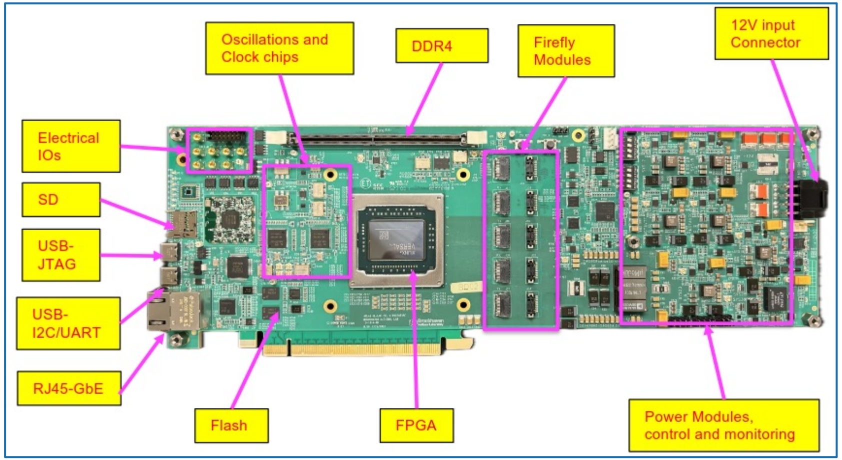
Next generation **FELIX** boards – developed for the Phase-II upgrade of the ATLAS experiment at LHC – adopted as DAM boards.

FELIX FLX-155 board is built around the new **Xilinx Versal FPGA/SoC** family:

- **48 serial links** running at speeds up to 25Gbps
- **100Gb ethernet** link off the board
- **DDR4 16GB RAM** slot available to support buffering
- **PCIe Gen5x16 bus**



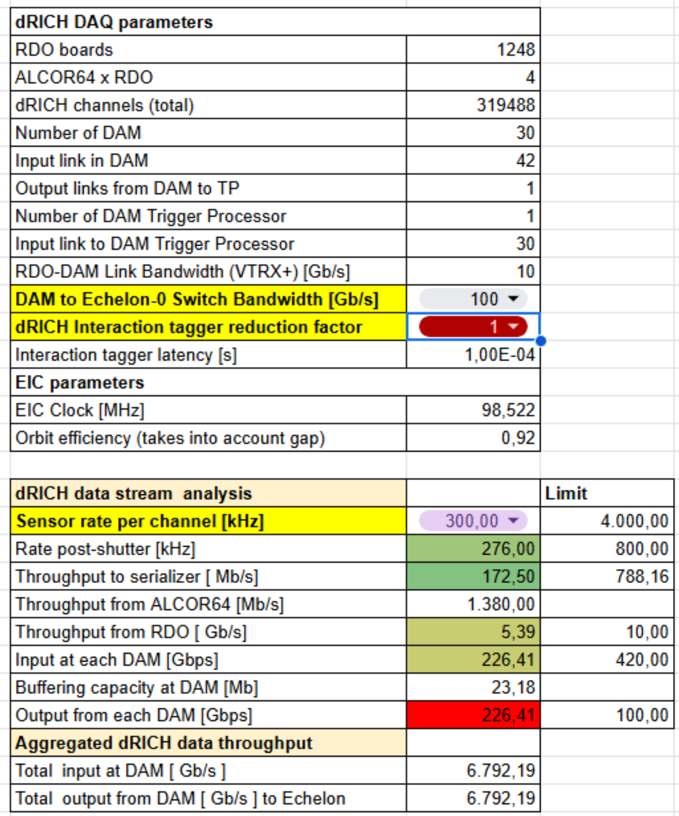
(**FLX-155 :** actual target HW to be used in ePIC dRICH DAQ system)



(**FLX-182** currently available at our lab, used for DAQ development)

dRICH: Analysis of Output Bandwidth

The dRICH DAQ chain in ePIC ⇒ **bandwidth/throughput issue**



dRICH DAQ parameters	
RDO boards	1248
ALCOR64 x RDO	4
dRICH channels (total)	319488
Number of DAM	30
Input link in DAM	42
Output links from DAM to TP	1
Number of DAM Trigger Processor	1
Input link to DAM Trigger Processor	30
RDO-DAM Link Bandwidth (VTRX+) [Gb/s]	10
DAM to Echelon-0 Switch Bandwidth [Gb/s]	100 ▾
dRICH Interaction tagger reduction factor	1 ▾
Interaction tagger latency [s]	1,00E-04
EIC parameters	
EIC Clock [MHz]	98,522
Orbit efficiency (takes into account gap)	0,92

dRICH data stream analysis		Limit
Sensor rate per channel [kHz]	300,00 ▾	4.000,00
Rate post-shutter [kHz]	276,00	800,00
Throughput to serializer [Mb/s]	172,50	788,16
Throughput from ALCOR64 [Mb/s]	1.380,00	
Throughput from RDO [Gb/s]	5,39	10,00
Input at each DAM [Gbps]	226,41	420,00
Buffering capacity at DAM [Mb]	23,18	
Output from each DAM [Gbps]	226,41	100,00
Aggregated dRICH data throughput		
Total input at DAM [Gb/s]	6.792,19	
Total output from DAM [Gb/s] to Echelon	6.792,19	

- Sensors DCR: 3-300 kHz (**increasing with radiation damage** ⇒with experiment lifetime).
- Considering planned techinques to manage SiPMs irradiation (e.g. annealing):
  - **worst DCR case**: 300 kHz

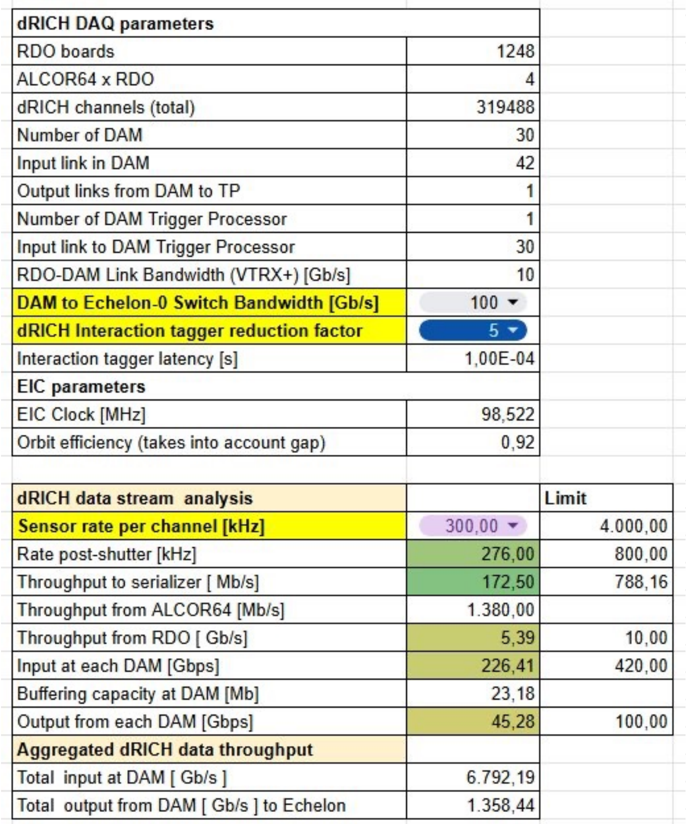
  ⇒ Full detector throughput (FE): 6.792,19 Gbps

  ⇒ a **reduction is needed** to cope with 30 channels (30x100GbE) bandwidth availability

⇒ Single **DAM output bandwidth** : **226,41 Gbps (!)**

dRICH: Analysis of Output Bandwidth

The dRICH DAQ chain in ePIC ⇒ **bandwidth/throughput issue**



dRICH DAQ parameters	
RDO boards	1248
ALCOR64 x RDO	4
dRICH channels (total)	319488
Number of DAM	30
Input link in DAM	42
Output links from DAM to TP	1
Number of DAM Trigger Processor	1
Input link to DAM Trigger Processor	30
RDO-DAM Link Bandwidth (VTRX+) [Gb/s]	10
DAM to Echelon-0 Switch Bandwidth [Gb/s]	100 ▾
dRICH Interaction tagger reduction factor	5 ▾
Interaction tagger latency [s]	1,00E-04
EIC parameters	
EIC Clock [MHz]	98,522
Orbit efficiency (takes into account gap)	0,92

dRICH data stream analysis		Limit
Sensor rate per channel [kHz]	300,00 ▾	4.000,00
Rate post-shutter [kHz]	276,00	800,00
Throughput to serializer [Mb/s]	172,50	788,16
Throughput from ALCOR64 [Mb/s]	1.380,00	
Throughput from RDO [Gb/s]	5,39	10,00
Input at each DAM [Gbps]	226,41	420,00
Buffering capacity at DAM [Mb]	23,18	
Output from each DAM [Gbps]	45,28	100,00
Aggregated dRICH data throughput		
Total input at DAM [Gb/s]	6.792,19	
Total output from DAM [Gb/s] to Echelon	1.358,44	

- Sensors DCR: 3-300 kHz (**increasing with radiation damage** ⇒with experiment lifetime).
- Considering planned techinques to manage SiPMs irradiation (e.g. annealing):
  - **worst DCR case**: 300 kHz

  ⇒ Full detector throughput (FE): 6.792,19 Gbps

  ⇒ a **reduction is needed** to cope with 30 channels (30x100GbE) bandwidth availability

- EIC beams bunch spacing: ~10 ns ⇒ bunch crossing rate of 100 MHz
- For the low interaction cross-section (DIS) ⇒ one interaction every ~100 bunches ⇒ interaction rate of ~1MHz.

⇒ **A system tagging dark current noise-only events** can solve the throughput issue (reducing down to 1/5 the data throughput)

⇒ Single **DAM output bandwidth** : **45,64 Gbps**

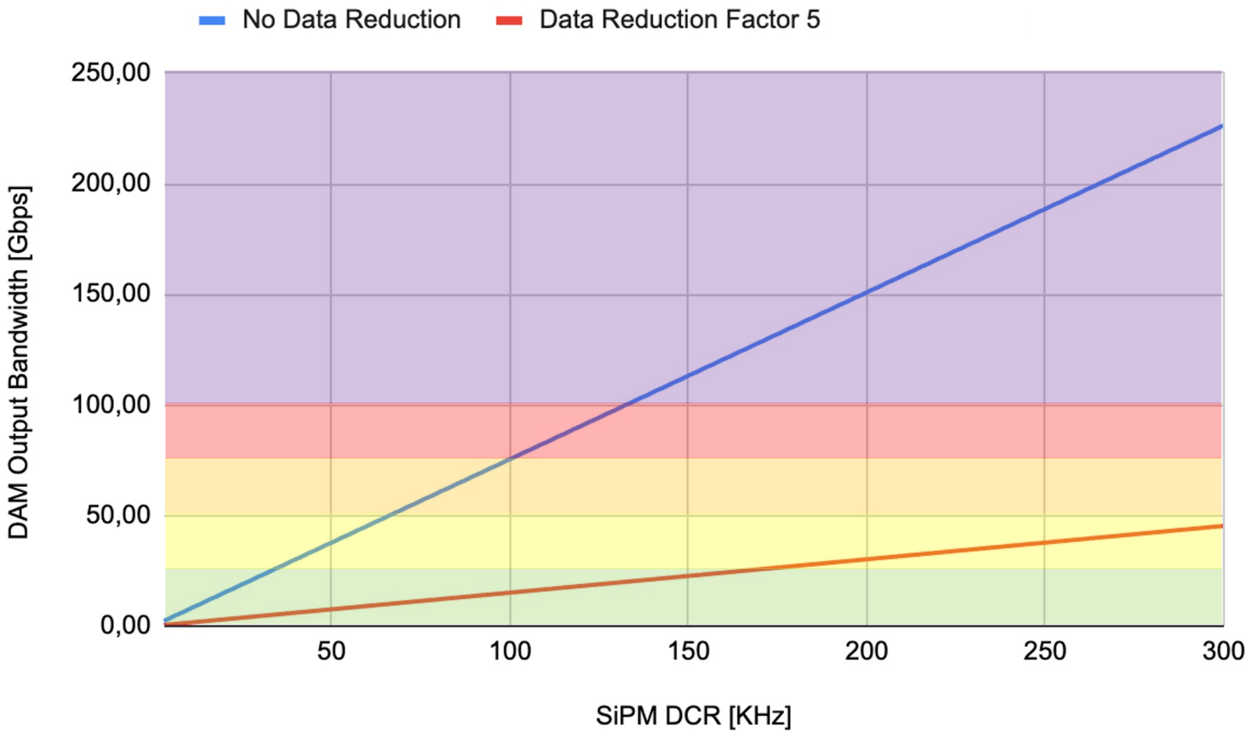
dRICH: Analysis of Output Bandwidth

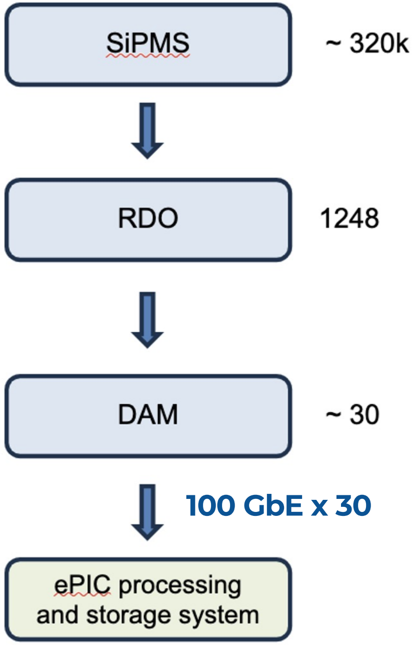
Net payload BW < ¼ Max

< Net payload BW < 1/2 Max

< Net payload BW < 3/4 Max

< Net payload BW < Max





SiPMS ~ 320k

RDO 1248

DAM ~ 30

100 GbE x 30

ePIC processing and storage system

dRICH: Data reduction with ML Classifier

Online Signal/ Noise discrimination using ML

- **Signal (i.e. Merged Phys Signal + Bkg):**
  - **Physics Signal:**
    - e.g DIS
  - **Physics Background:**
    - e/p with beam pipe
    - Synchrotron radiation (not included yet)
- **SiPM Noise:**
  - Dark count rate (DCR) modelled in the reconstruction stage

ML task:

Discriminate between **Noise-Only** and **Signal+Noise** events

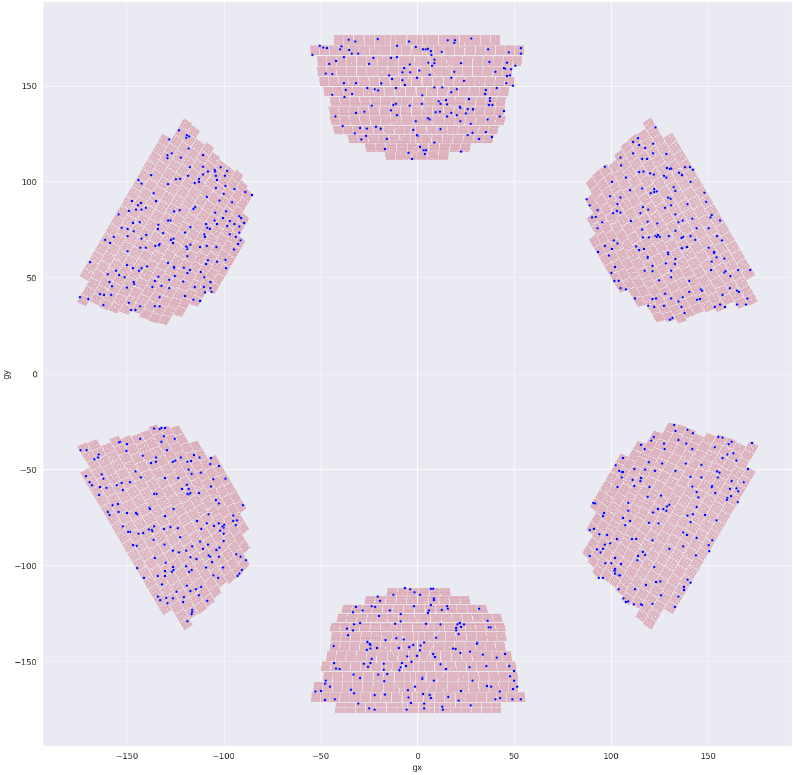
Noise-Classifier:

- **Positive: Noise-Only** event
- **Negative: Signal+Noise** event

dRICH Data reduction: Classes Def.

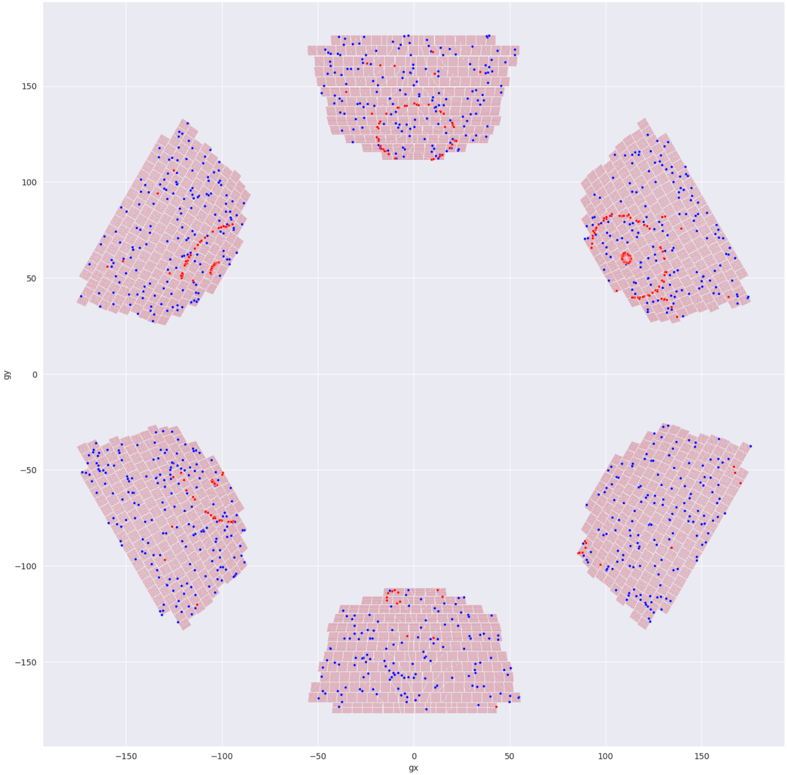
(Positive)

Noise-Only



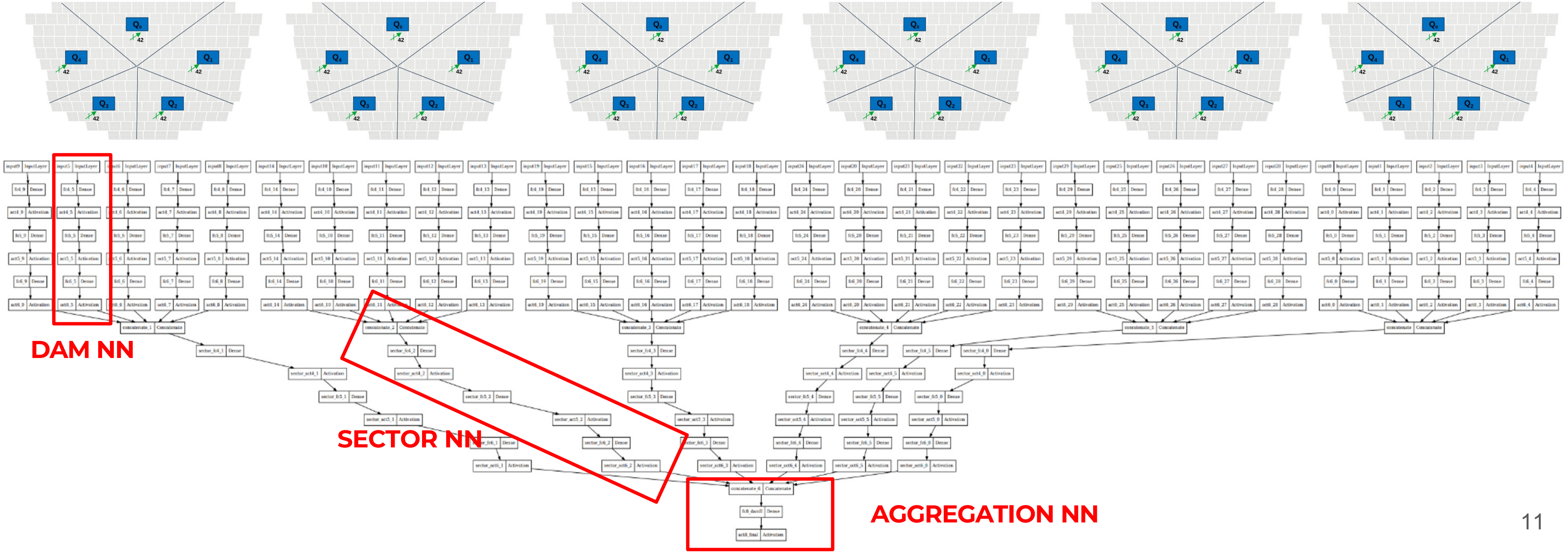
(Negative)

Phys Signal+Phys Background+Noise



dRICH Data reduction: Tensorflow-Keras Model definition

- The NN model mimics the DAQ system architecture
  - **30** (# of subsectors x # of sectors) **MLP networks** deployed on 30 DAM FPGAs.
  - **6 sector MLP networks + 1 aggregation network** deployed on the TP FPGA.



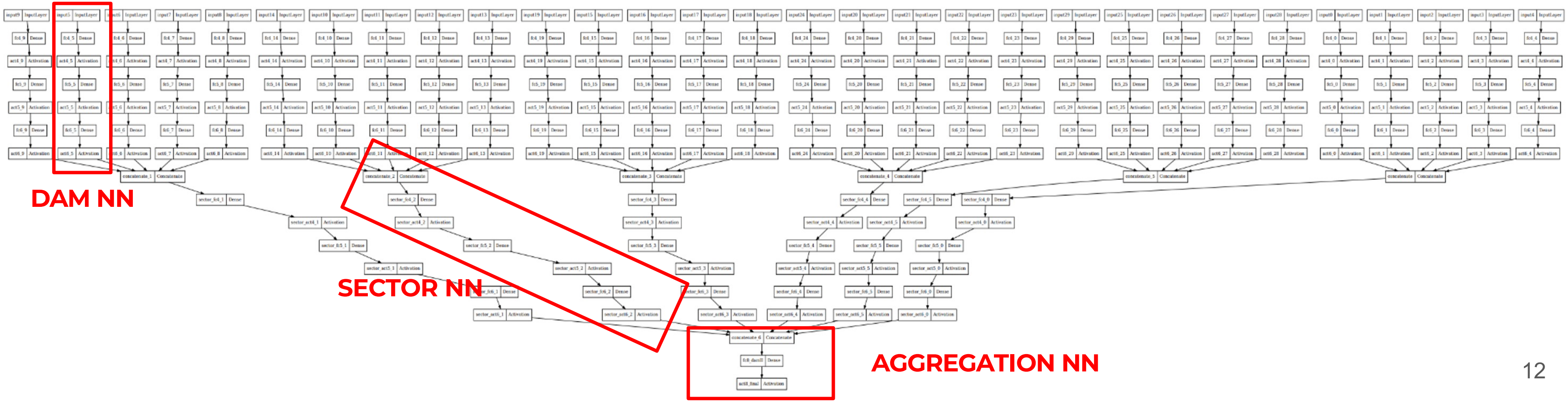
DAM NN

SECTOR NN

AGGREGATION NN

dRICH Data reduction: Tensorflow-Keras Model definition

- The 30 DAM networks are concatenated to feed 6 intermediate model (called **Sector NN**) to be deployed on an additional **Trigger Processor (TP) FPGA**.
- Each Sector NN work on the aggregated information of a single sector (5 DAMs)
- The 6 outputs from Sector NNs are then aggregated and processed in a **lightweight TP NN** (single layer, 5 neurons), deployed on the same TP FPGA



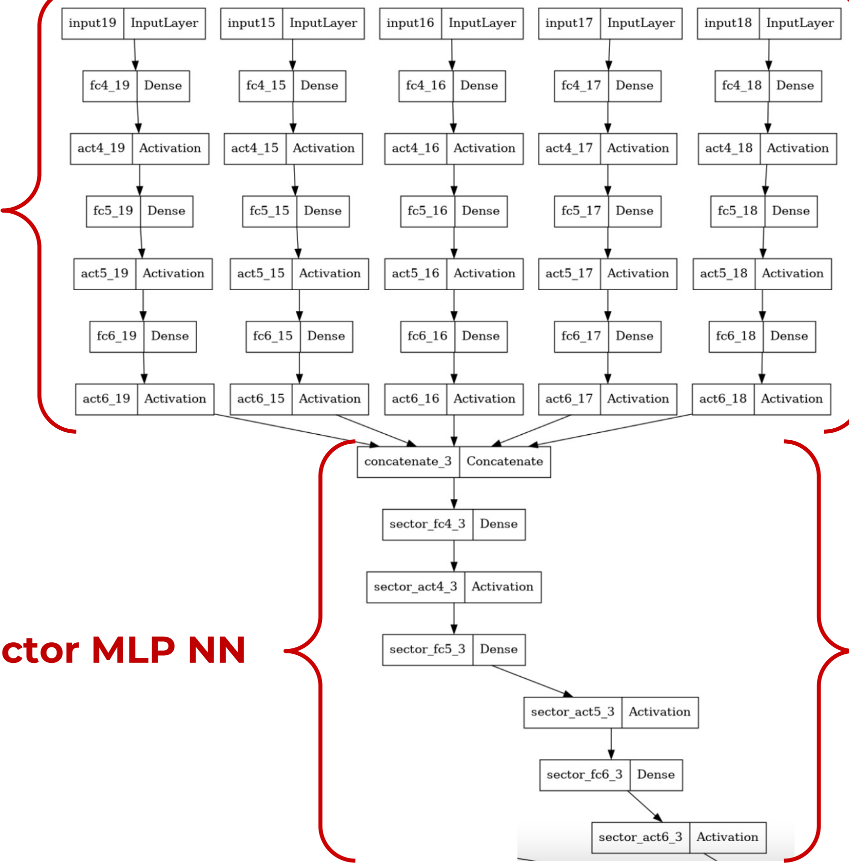
DAM NN

SECTOR NN

AGGREGATION NN

dRICH Data reduction: Tensorflow-Keras Model definition

5 MLP DAM NNs (same sector)

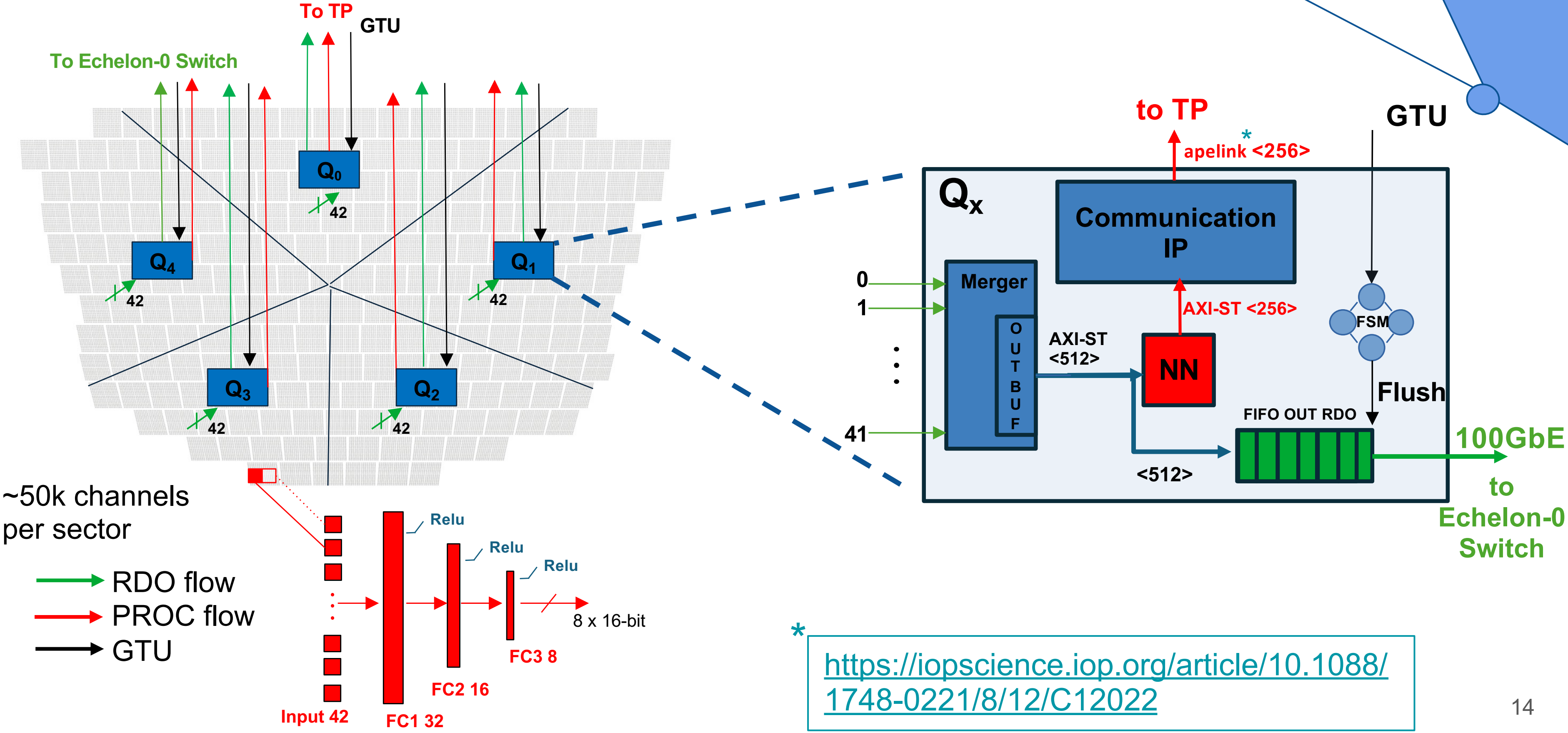


Sector MLP NN

For each sector, 5 MLP DAM output (**embedding**) are concatenated and then used to feed the Sector MLP model

⇒ **sector local information** extracted from the incoming data to perform the final prediction

dRICH Data Reduction Stage on FPGA: subsectors’ design



~50k channels per sector

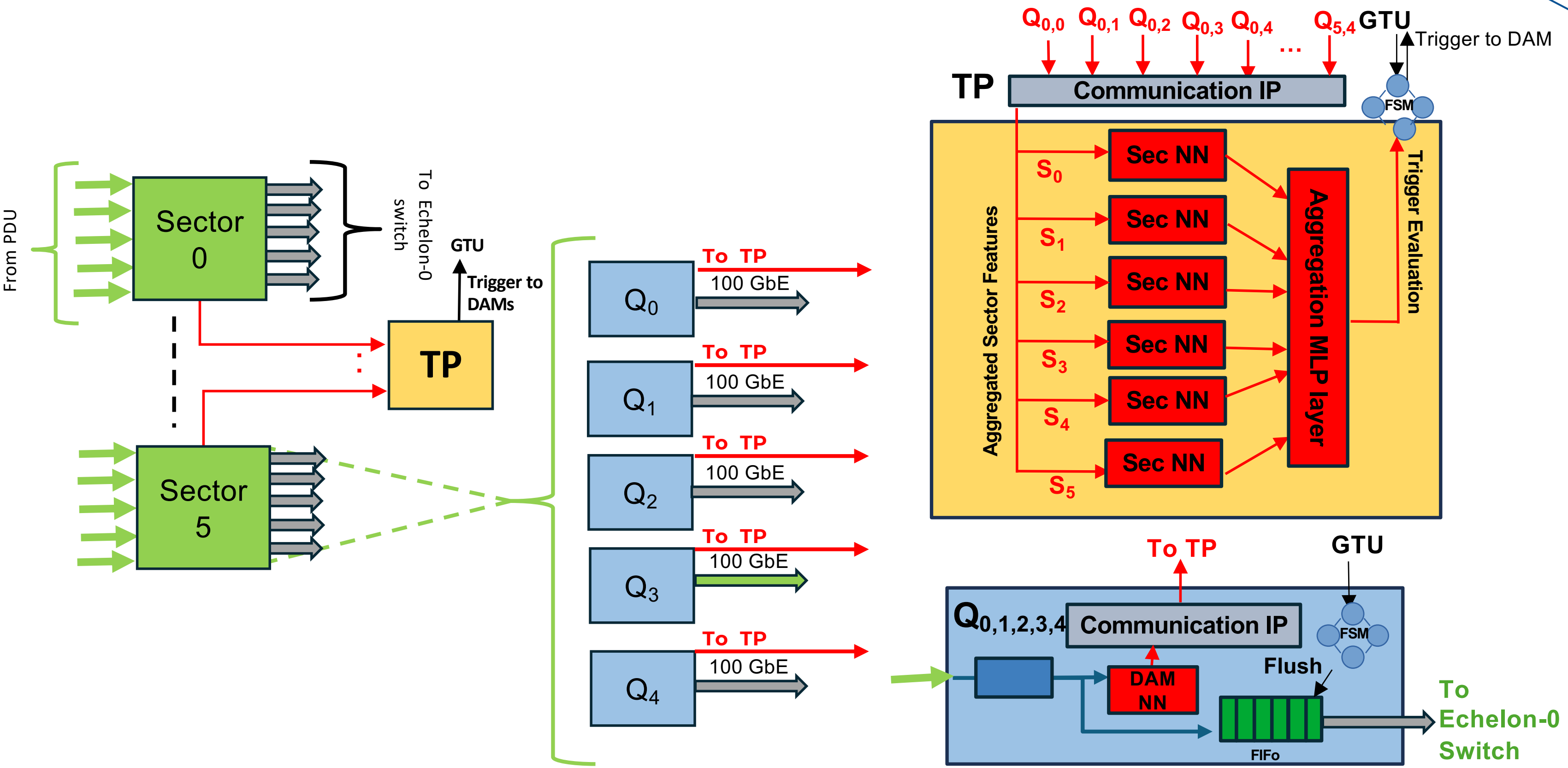
→ RDO flow

→ PROC flow

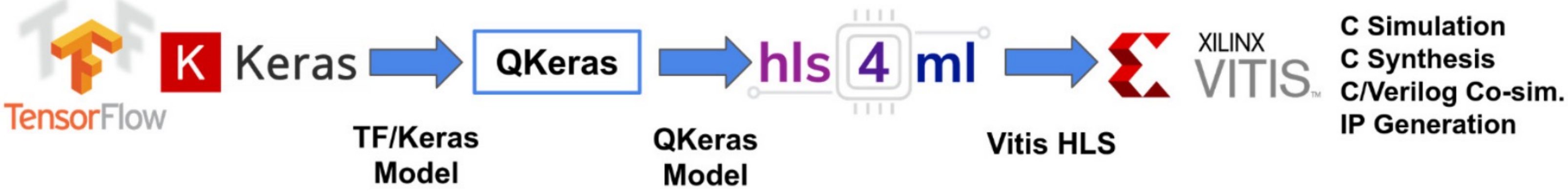
→ GTU

* https://iopscience.iop.org/article/10.1088/1748-0221/8/12/C12022

dRICH Data Reduction Stage on FPGA: subsectors' design



dRICH Data reduction: How?
⇒ Design and Implementation Workflow



TF/Keras Model

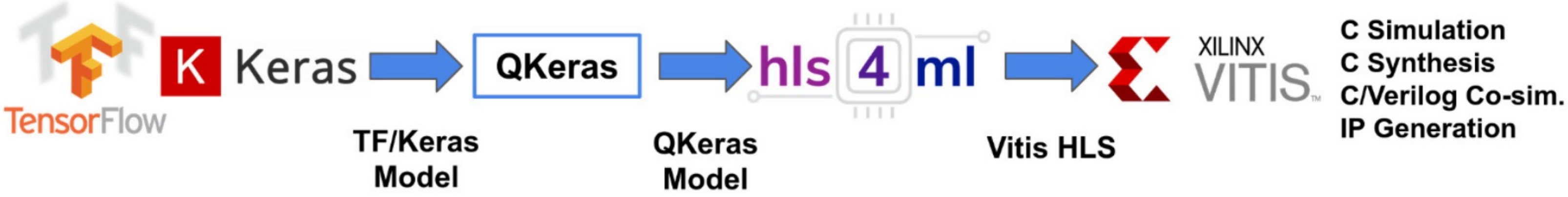
QKeras Model

Vitis HLS

C Simulation
C Synthesis
C/Verilog Co-sim.
IP Generation

Design targets (efficiency, purity, throughput, latency) and hardware constraints (mainly FPGA resource usage) must be taken into account and verified at any stage:

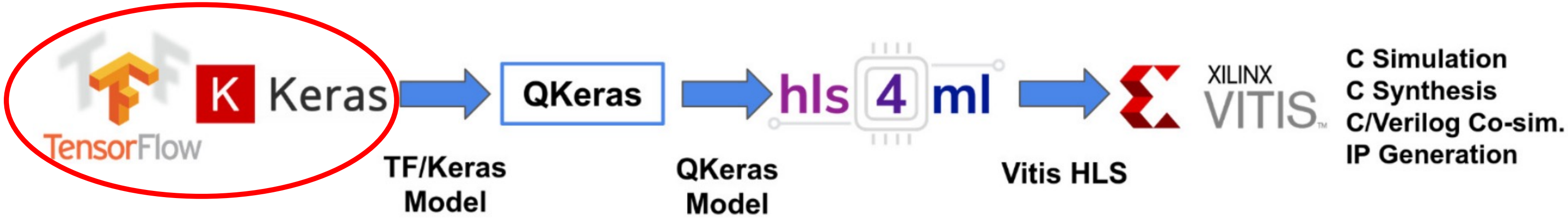
dRICH Data reduction: How?
⇒ Design and Implementation Workflow



Design targets (efficiency, purity, throughput, latency) and hardware constraints (mainly FPGA resource usage) must be taken into account and verified at any stage:

- **Generation strategy of training and validation data sets.**

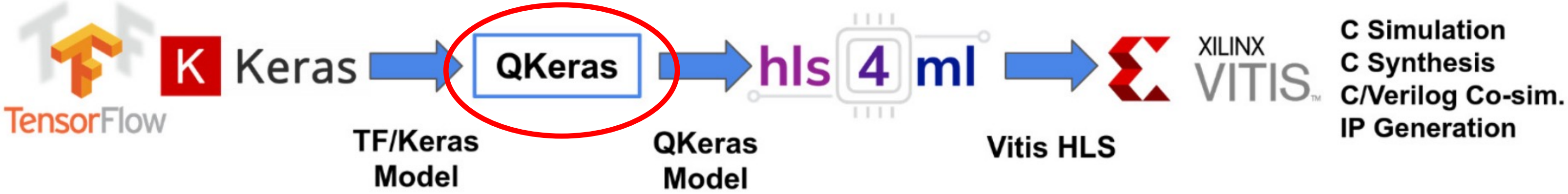
dRICH Data reduction: How?
⇒ Design and Implementation Workflow



Design targets (efficiency, purity, throughput, latency) and hardware constraints (mainly FPGA resource usage) must be taken into account and verified at any stage:

- **TensorFlow/Keras**
  ⇒ NN architecture (number and kind of layers) and **representation of the input**
  ⇒ Training strategy (class balancing, batch sizes, optimizer choice, learning rate,...).

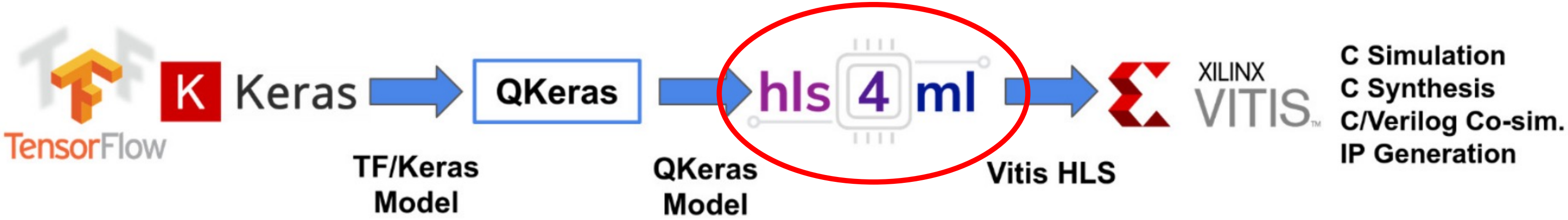
dRICH Data reduction: How?
⇒ Design and Implementation Workflow



Design targets (efficiency, purity, throughput, latency) and hardware constraints (mainly FPGA resource usage) must be taken into account and verified at any stage:

- **Qkeras** ⇒ Search iteratively the minimal representation size in **bits** of weights, biases and activations.

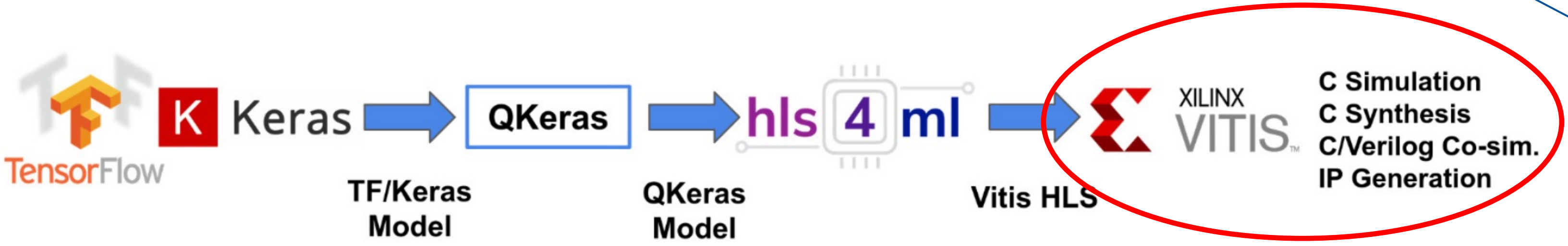
dRICH Data reduction: How?
⇒ Design and Implementation Workflow



Design targets (efficiency, purity, throughput, latency) and hardware constraints (mainly FPGA resource usage) must be taken into account and verified at any stage:

- **hls4ml** ⇒ Tuning of REUSE FACTOR config param (low values ⇒ low latency, high throughput, high resource usage), clock frequency.

dRICH Data reduction: How?
⇒ Design and Implementation Workflow



Design targets (efficiency, purity, throughput, latency) and hardware constraints (mainly FPGA resource usage) must be taken into account and verified at any stage:

- **Vitis HLS** ⇒ co-simulation for verification of performance (experimented very good agreement with QKeras Model)

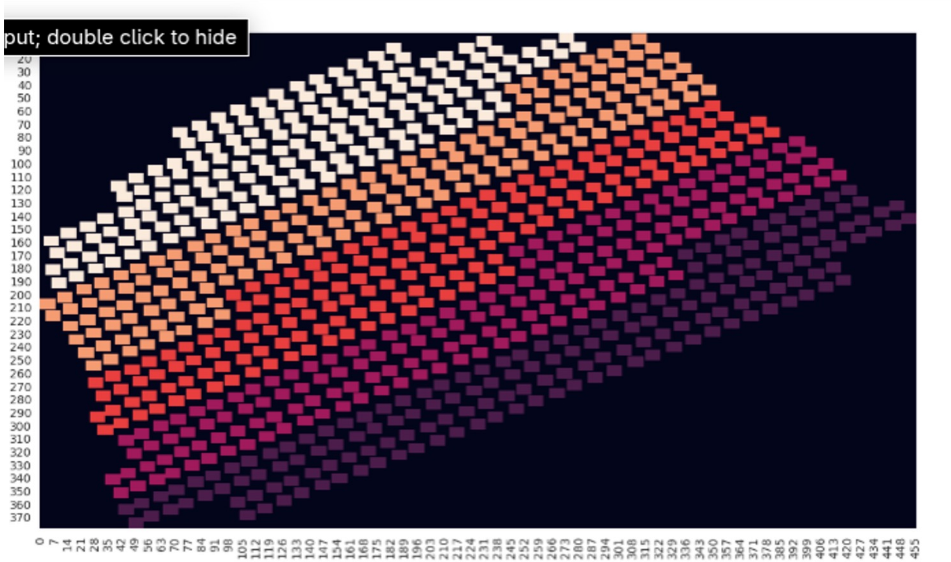
dRICH: Data reduction ⇒ Mapping

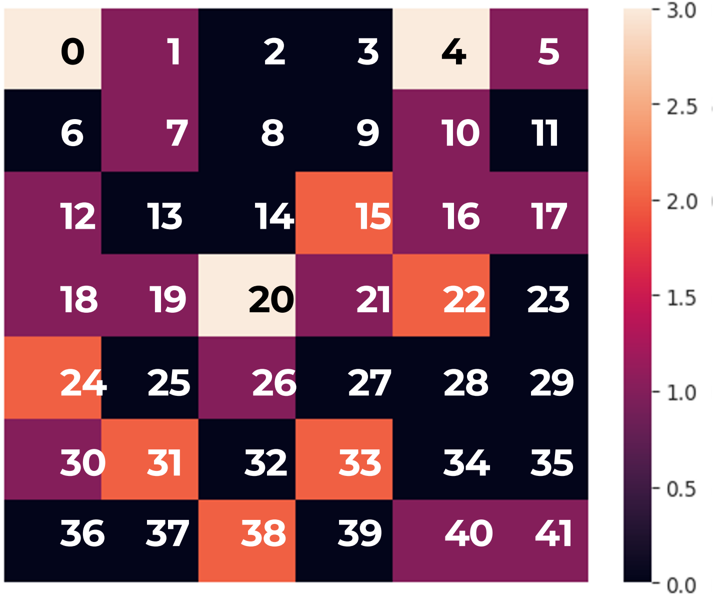
- **42 input links for each DAM,** corresponding to the number of expected PDUs per subsector (~210/5).

⇒ **Each PDU is input** to a neuron of the input layer of the MLP NN

⇒ **42 input neuron** for the input layer of the MLP NN

(“Answer to the Ultimate Question of Life, the Universe, and Everything”)

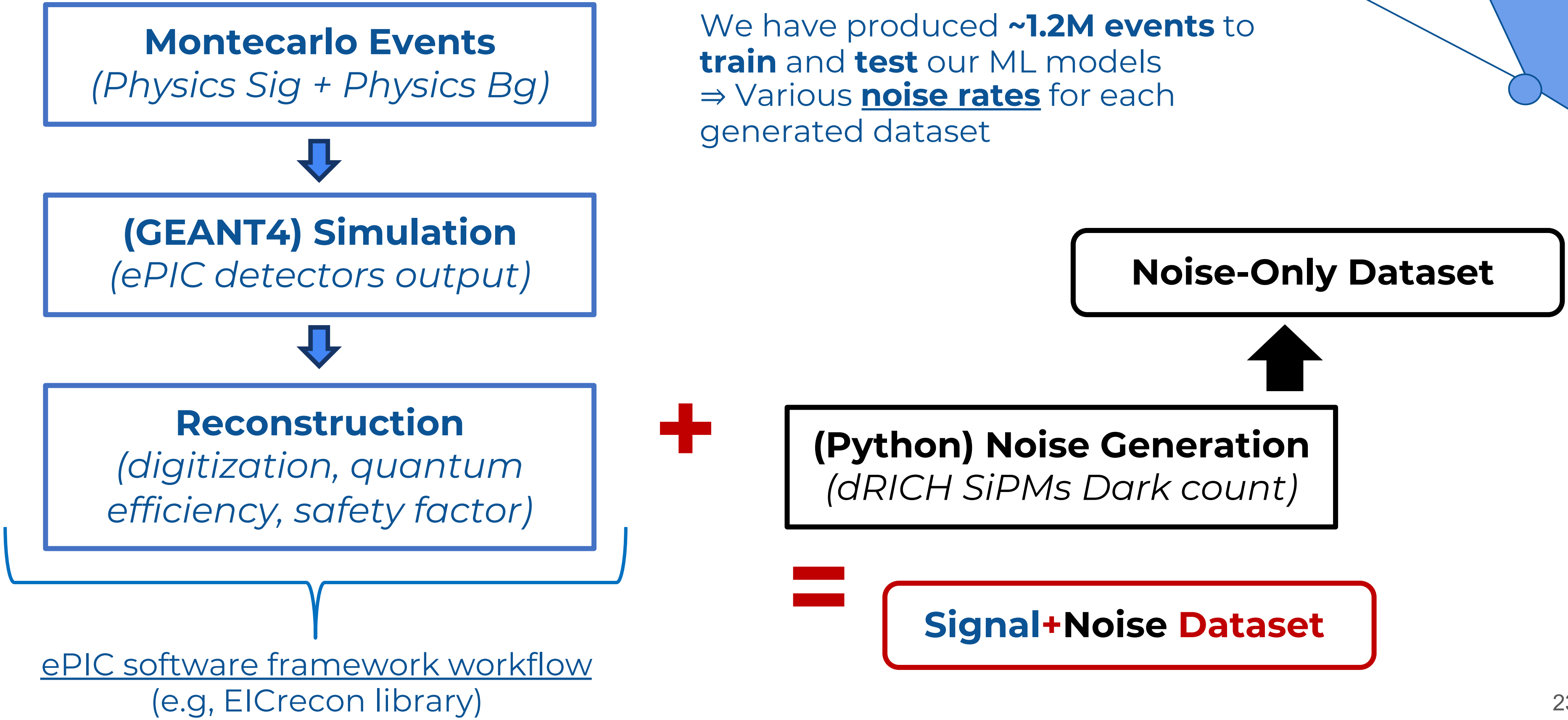




0	1	2	3	4	5
6	7	8	9	10	11
12	13	14	15	16	17
18	19	20	21	22	23
24	25	26	27	28	29
30	31	32	33	34	35
36	37	38	39	40	41

dRICH Data reduction ⇒ Dataset

We have produced **~1.2M events** to **train** and **test** our ML models ⇒ Various **noise rates** for each generated dataset



Montecarlo Events
(Physics Sig + Physics Bg)

⇩

(GEANT4) Simulation
(ePIC detectors output)

⇩

Reconstruction
(digitization, quantum efficiency, safety factor)

ePIC software framework workflow
(e.g, EICrecon library)

+

(Python) Noise Generation
(dRICH SiPMs Dark count)

⬆

Noise-Only Dataset

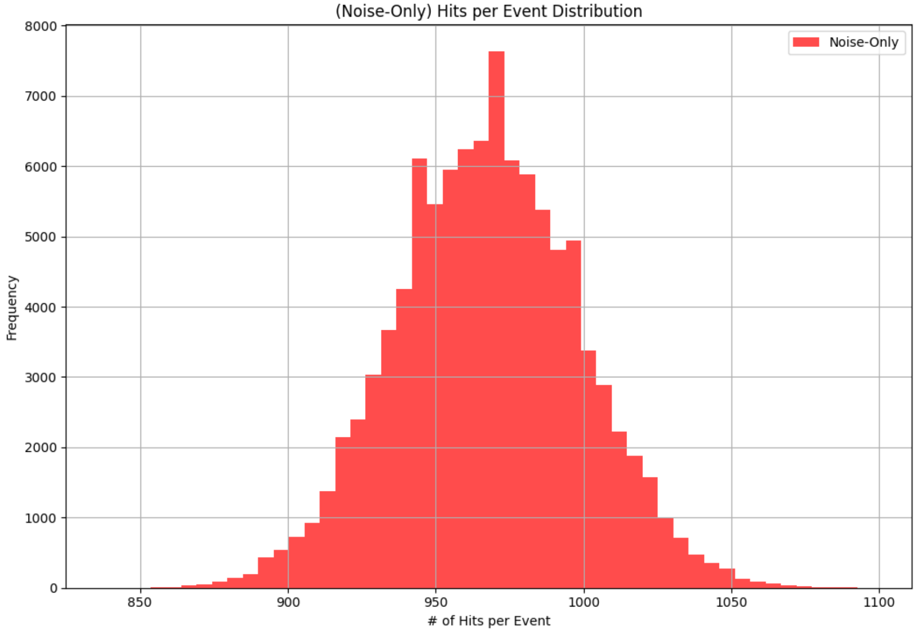
=

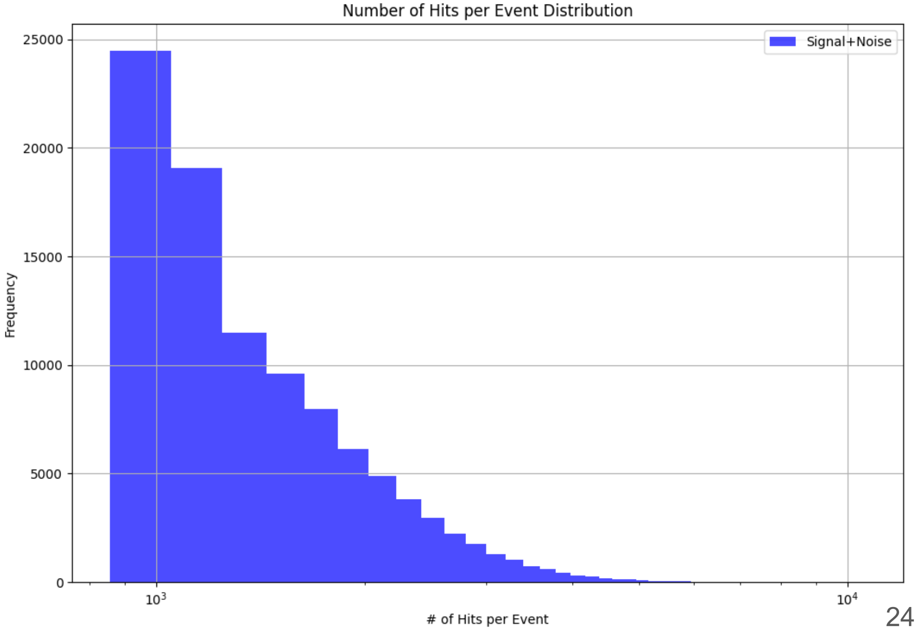
Signal+Noise Dataset

dRICH Data reduction: Noise Distribution

- **Gaussian** dark current SiPM **noise hits distribution**:
  - mean = noiseRate*noiseTimeWindow*NumberOfSiPMsDRICH
  - sigma = 0.1*avg
  - noiseTimeWindow = 10 ns

noiseRate = 300 kHz





dRICH Data reduction:
Tensorflow training and evaluation

⇒ We trained the 30 MLP DAM models concatenated to the single MLP TP model by using 100k Signal+Noise and 100k Noise Only events.
⇒ **200k balanced dataset** (90% training set, 8% testing set, 2% validation set) varying the Dark Count Rate parameter:

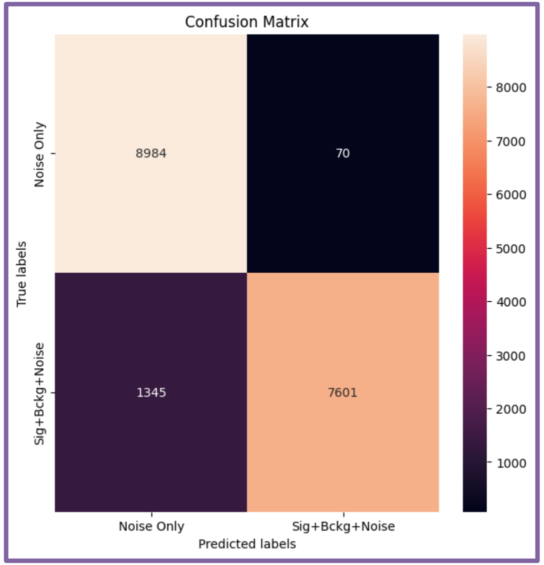
◆ **Gaussian Noise Hits Distribution model:**

- noiseRate = **25 kHz**, noisetimeWindow = 10ns;
- noiseRate = **50 kHz**, noisetimeWindow = 10ns;
- noiseRate = **100 kHz**, noisetimeWindow = 10ns;
- noiseRate = **150 kHz**, noisetimeWindow = 10ns;
- noiseRate = **200 kHz**, noisetimeWindow = 10ns;
- noiseRate = **300 kHz**, noisetimeWindow = 10ns;

Preliminary results

NN Model performance (100 kHz & 10ns)

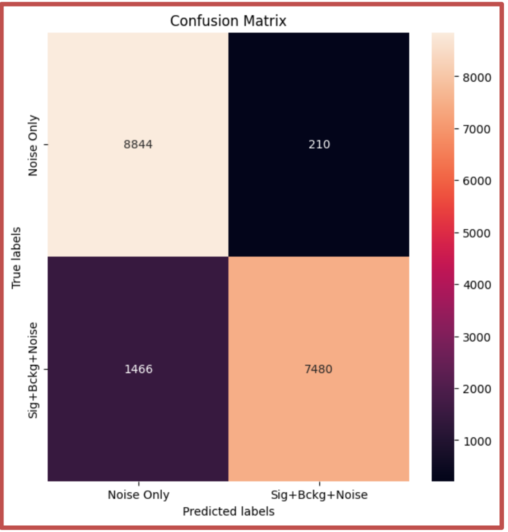
Keras model



- ❑ Accuracy = (TP+TN)/(TP+TN+FP+FN) = 0.921
- ❑ Purity = TP/(TP+FP) = 0.870
- ❑ Recall = TP/(TP+FN) = 0.992

Model Quantization

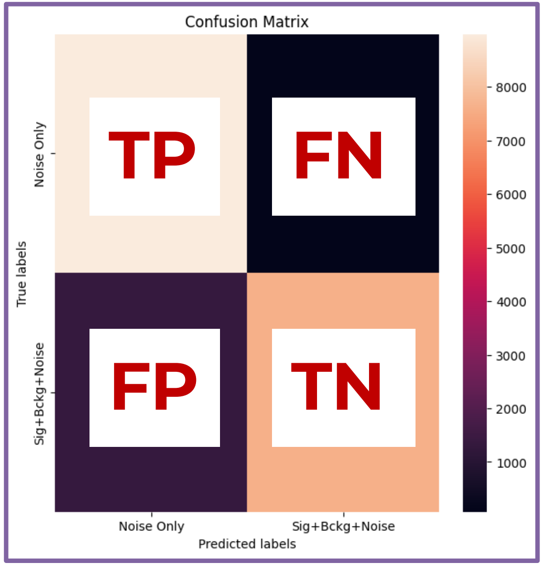
- Inputs, Activations: fixed point<16,6>
- Weights, Biases: fixed point<8,1>



- ❑ Accuracy = (TP+TN)/(TP+TN+FP+FN) = 0.906
- ❑ Purity = TP/(TP+FP) = 0.858
- ❑ Recall = TP/(TP+FN) = 0.977

NN Model performance (100 KHz & 10ns)

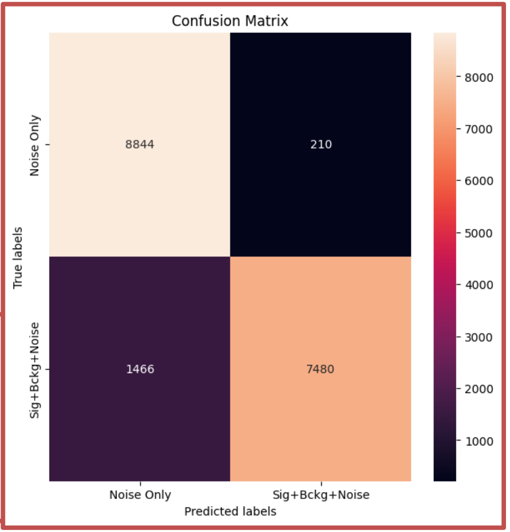
Keras model



- ❑ Accuracy = (TP+TN)/(TP+TN+FP+FN) = 0.921
- ❑ Purity = TP/(TP+FP) = 0.870
- ❑ Recall = TP/(TP+FN) = 0.992

Model Quantization

- Inputs, Activations: fixed point<16,6>
- Weights, Biases: fixed point<8,1>

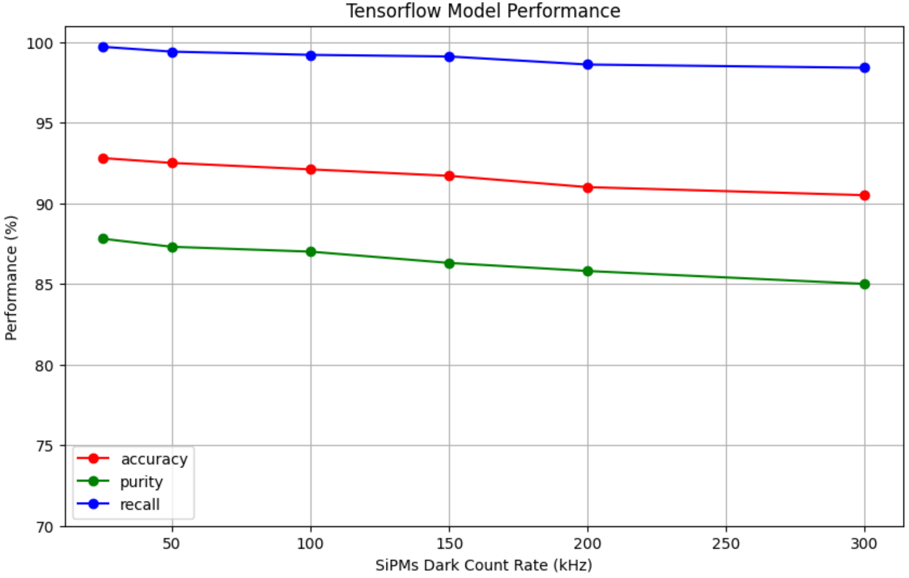


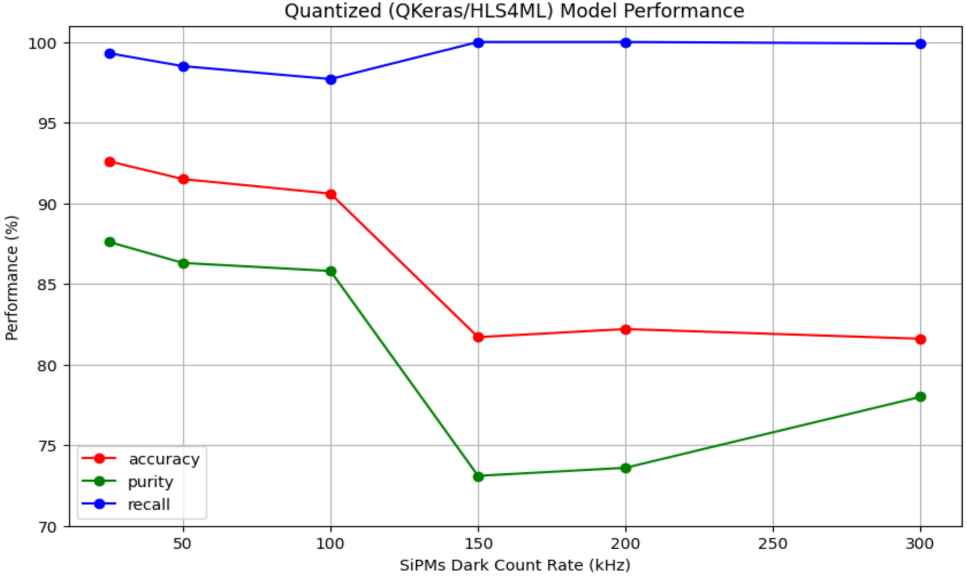
- ❑ Accuracy = (TP+TN)/(TP+TN+FP+FN) = 0.906
- ❑ Purity = TP/(TP+FP) = 0.858
- ❑ Recall = TP/(TP+FN) = 0.977

NN Model performance scaling

- We noticed a drop of classification performance with increasing dark count rate (e.g. increasing number of noise hits per event), but still *purity > 85%* for noisiest case (DCR = 300 kHz).
- As expected, performance drop after quantization step

Preliminary results

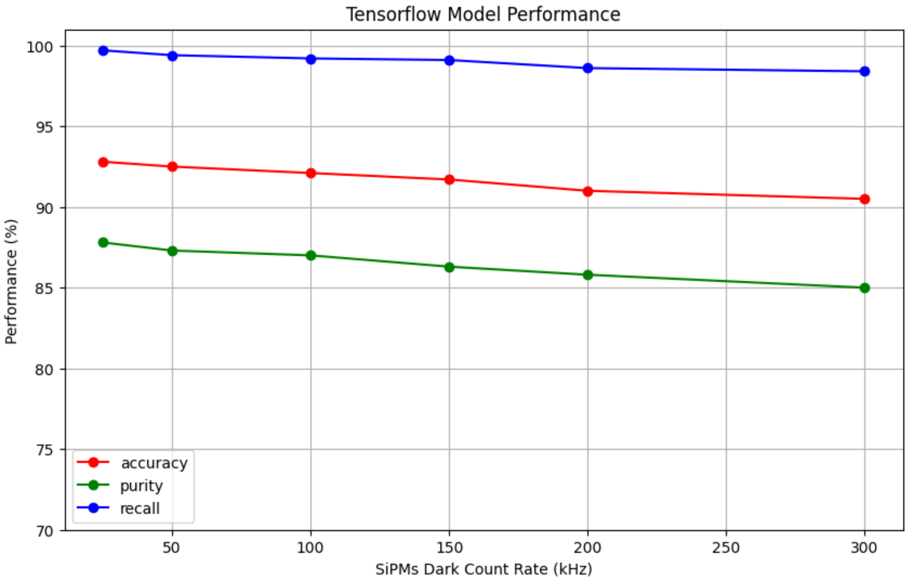

Tensorflow Model Performance

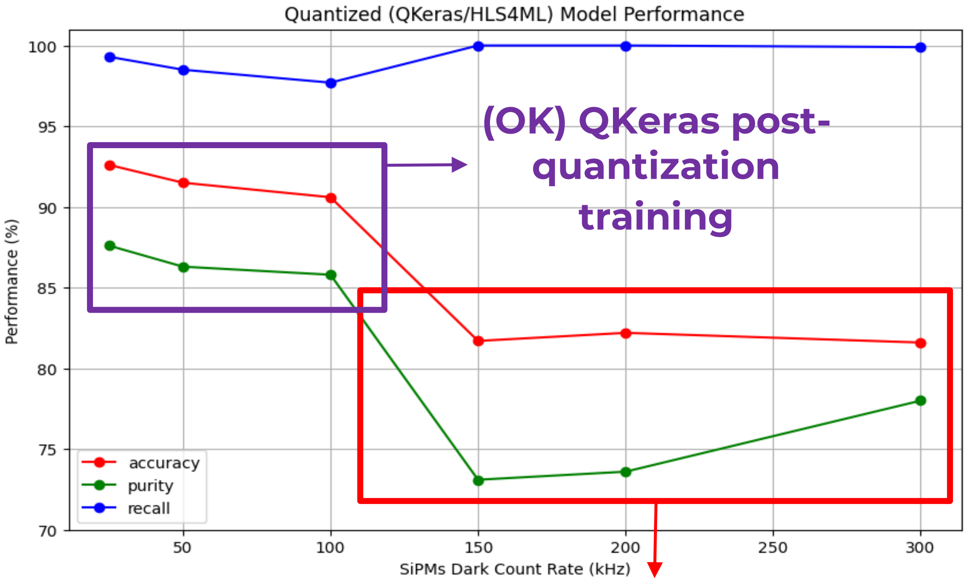

Quantized (QKeras/HLS4ML) Model Performance

NN Model performance scaling

- We noticed a drop of classification performance with increasing dark count rate (e.g. increasing number of noise hits per event), but still *purity > 85%* for noisiest case (DCR = 300 kHz).
- As expected, performance drop after quantization step

Preliminary results


Tensorflow Model Performance

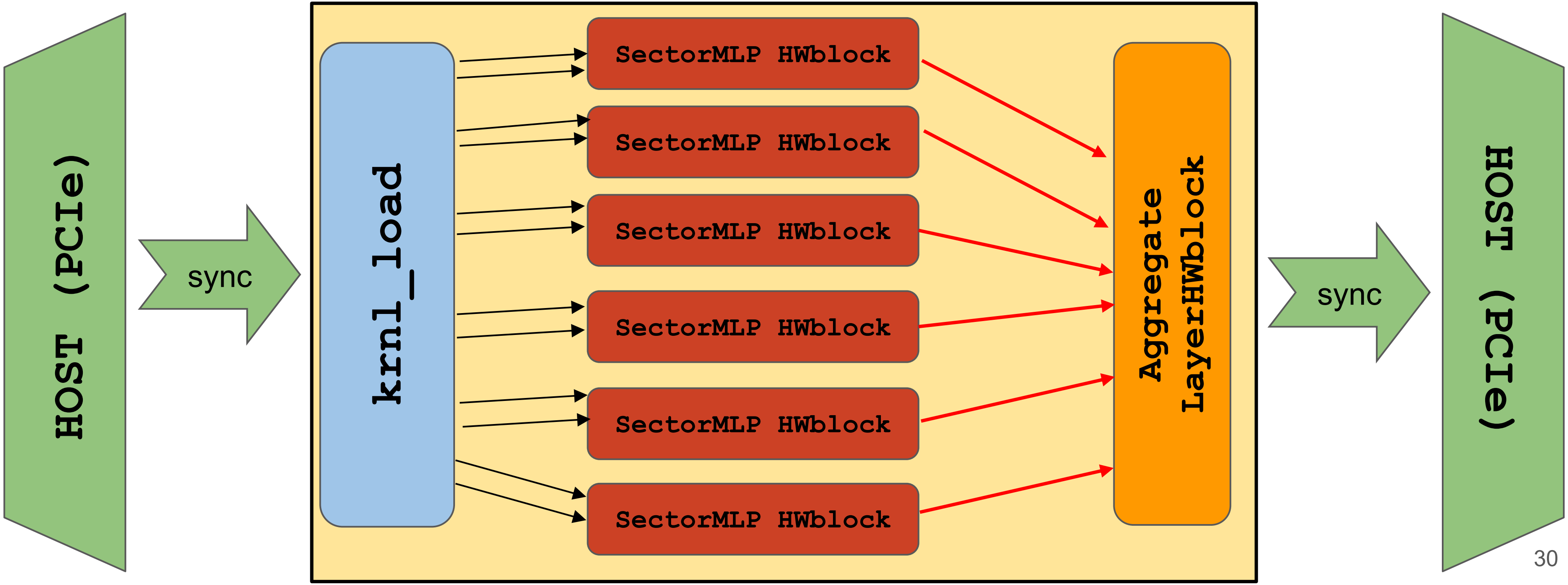

Quantized (QKeras/HLS4ML) Model Performance

(OK) QKeras post-quantization training

(NOT OK) no QKeras training, only quantization

dRICH Data reduction stage on FPGA: HLS4ML ⇒ HW implementation

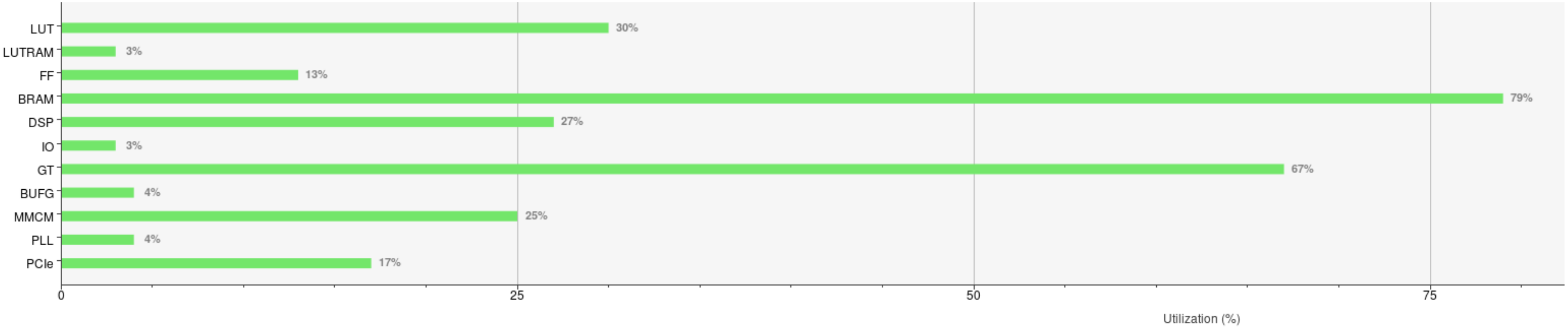
Stripped-down HW design (on Xilinx Alveo U280) used to validate the **TP NN model (6 Sector MLP + Aggregate Layer)** implementation and to assess system performance.



dRICH Data reduction:
HLS4ML ⇒ (FPGA) HW Synthesis

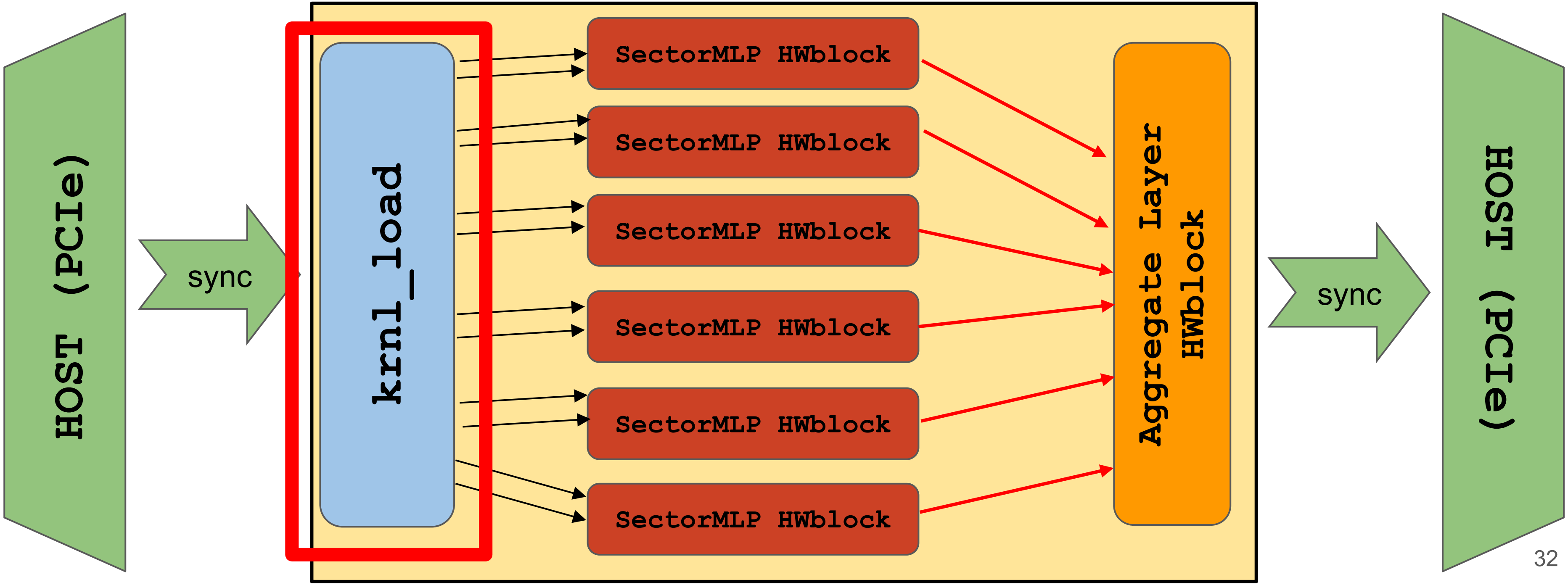
- TP NN design (6 Sector NN + Aggregation MLP NN) fits into the available FPGA resources of the Xilinx Alveo U280 board.
- Post-synthesis Vitis reports ⇒ **high BRAM utilization** due to allocation of 6 different sets of weights and biases for the 6 Sector NNs

⇒ **occupation percentage** to take into account when moving to the target HW (FELIX-155 Xilinx Versal Prime) and integrating with the standard DAQ firmware.



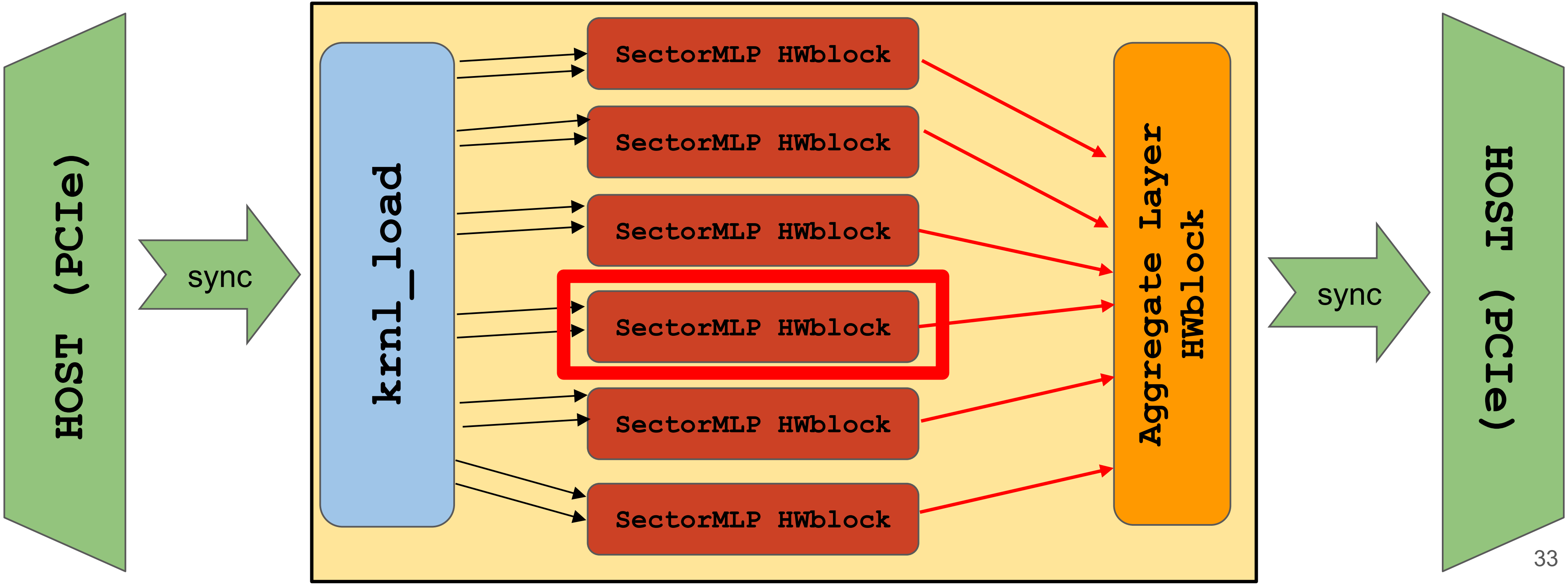
dRICH Data reduction stage on FPGA: HLS4ML ⇒ HW implementation

Stripped-down HW design (on Xilinx Alveo U280) used to validate the **TP NN model (6 Sector MLP + Aggregate Layer)** implementation and to assess system performance.



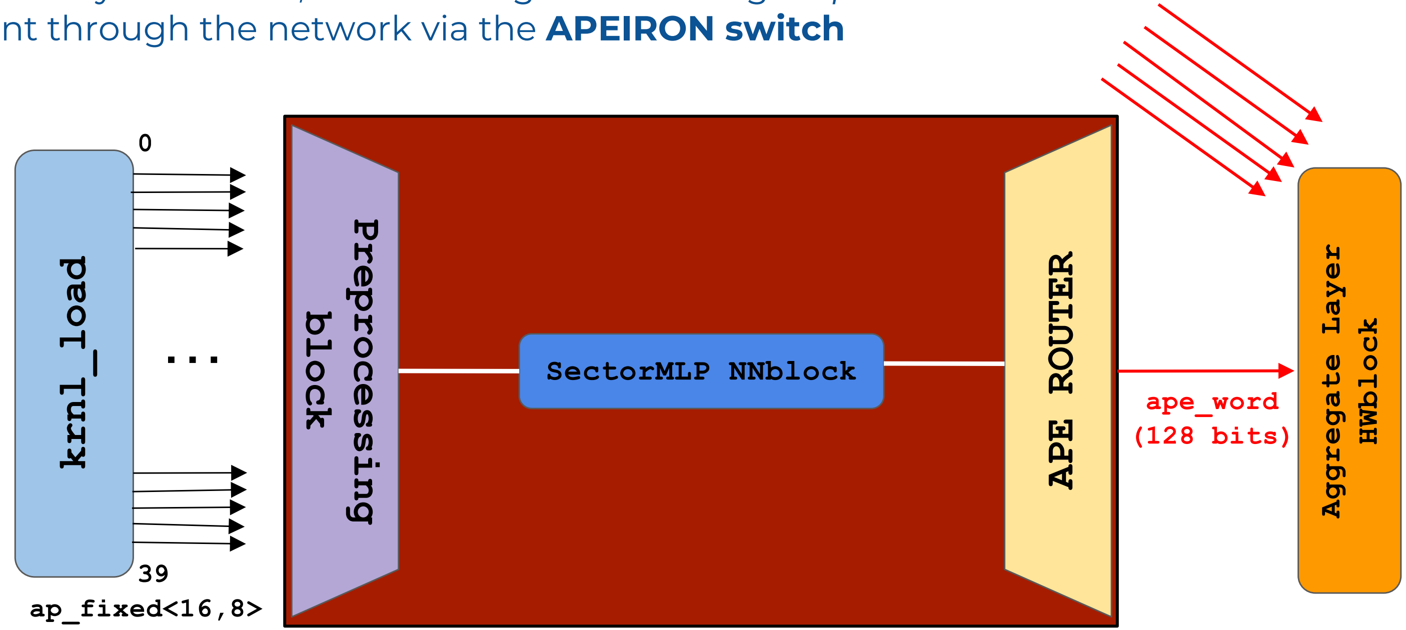
dRICH Data reduction stage on FPGA: HLS4ML ⇒ HW implementation

Stripped-down HW design (on Xilinx Alveo U280) used to validate the **TP NN model (6 Sector MLP + Aggregate Layer)** implementation and to assess system performance.



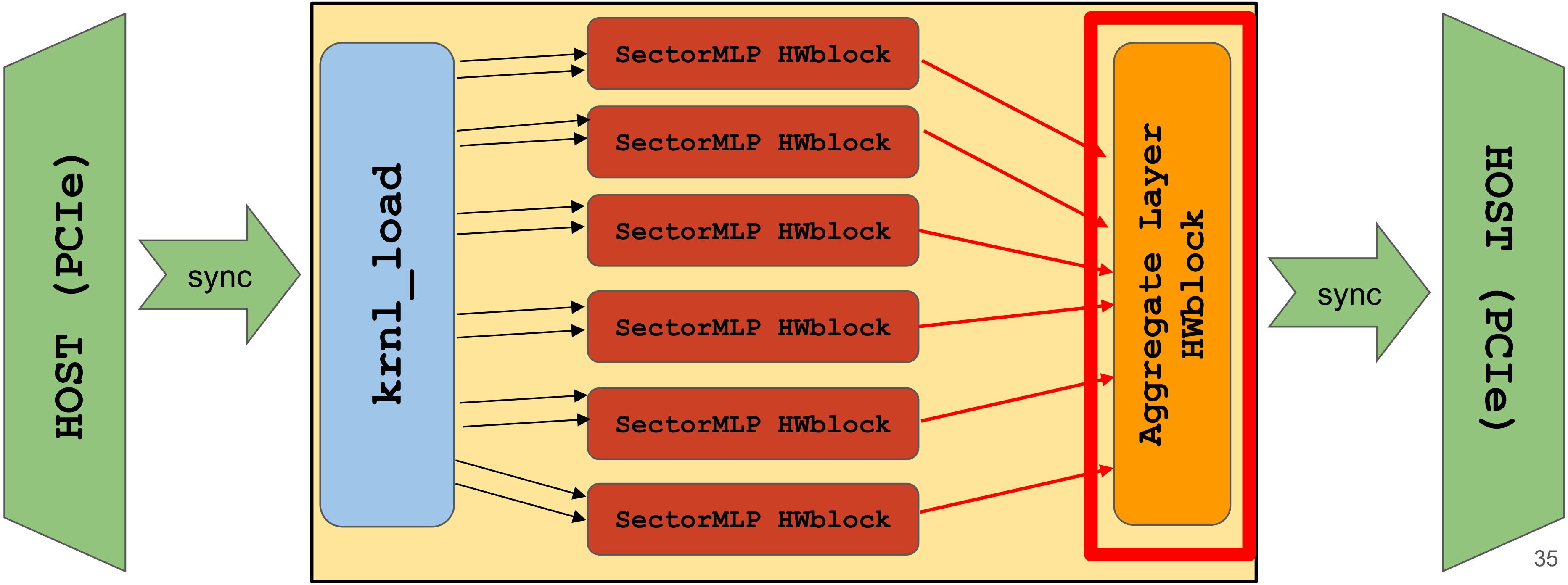
dRICH Data reduction stage on FPGA: HLS4ML ⇒ HW implementation

- The 40 input ap_fixed<16,8> are connected to the **preprocessing block**, which merges the whole set of input in order to feed the **MLP HLS4ML block**.
- The NN block computes its output by using ap_fixed<8,0> weights and biases.
- The output, composed by 4 features, is then merged into a single *ape_word* of 128bits and then sent through the network via the **APEIRON switch**



dRICH Data reduction stage on FPGA: HLS4ML ⇒ HW implementation

Stripped-down HW design (on Xilinx Alveo U280) used to validate the **TP NN model (6 Sector MLP + Aggregate Layer)** implementation and to assess system performance.



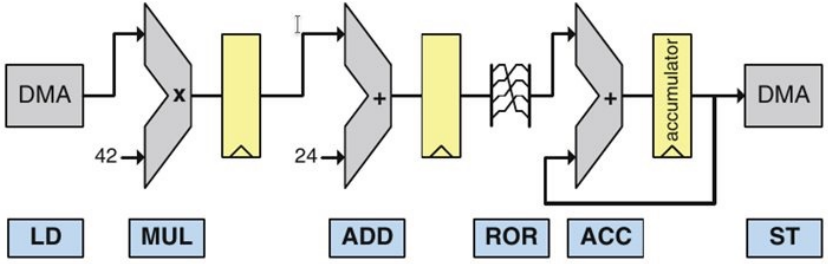
dRICH Data reduction stage on FPGA: HW challenges and targets

⇒ Why FPGA are good for real-time inference?

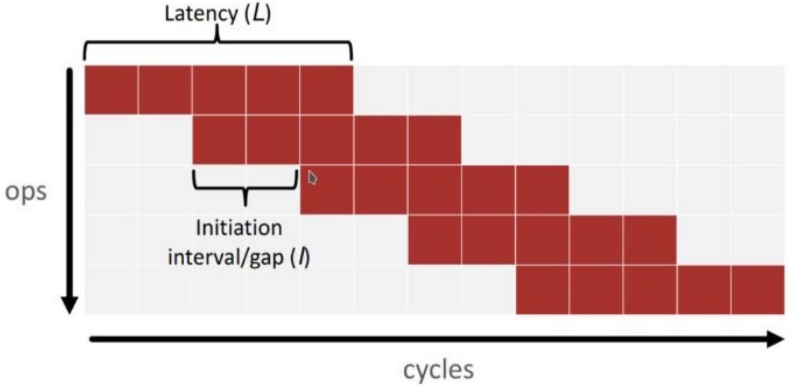
- Customizable I/O and deterministic latency make them well suited for TDAQ systems.
- Improvements to silicon manufacturing process made them very interesting for heavy computation as well.

- In our case, the challenge is the **processing throughput**
  ⇒ a pipelined design can potentially produce a new output at each clock cycle.
- ***Initiation interval (II)***: Number of clock cycles before the function can accept new input data.
  ⇒ ***the lower the II, the higher the throughput***
- The greater the number of pipeline stages, the greater the latency.

- High level synthesis tools allows to describe datapaths in FPGA using high level software languages (C/C++, OpenCL, SYCL,...).



FPGA datapath

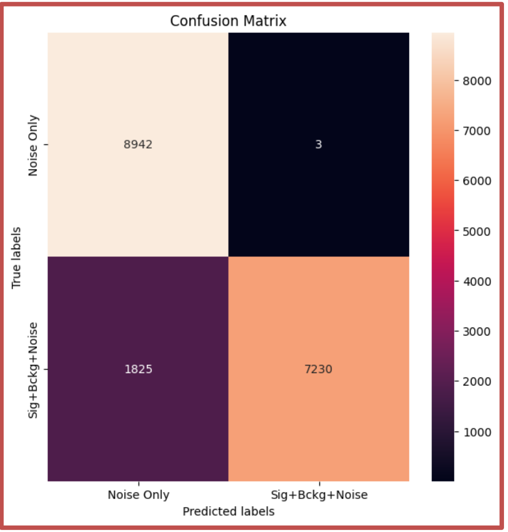


HLS4ML FPGA performance (100kHz & 10ns)

- ❑ Throughput (DDR) = 2.065 MHz
  ➔ instantiation interval II~97 cycles (@200 MHz)
- ❑ Throughput (BRAM) = 10.867 MHz
  ➔ instantiation interval II~19 cycles (@200 MHz)

Model Quantization

- Inputs, Activations: fixed point<16,6>
- Weights, Biases: fixed point<8,1>



- ❑ Accuracy = (TP+TN)/(TP+TN+FP+FN) = 0.898
- ❑ Purity = TP/(TP+FP) = 0.831
- ❑ Recall = TP/(TP+FN) = 0.999

HLS4ML FPGA performance (100kHz & 10ns)

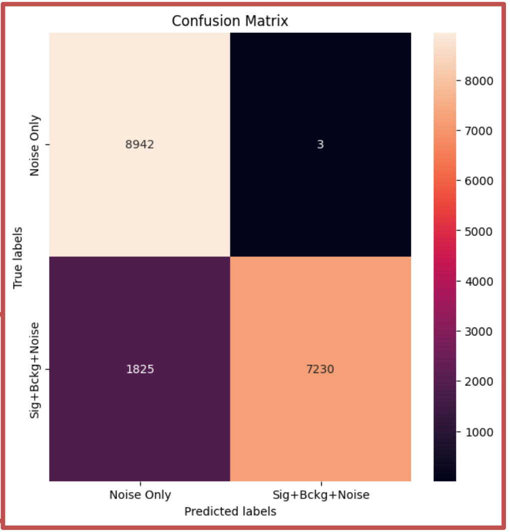
❑ Throughput (DDR) = 2.065 MHz
➔ instantiation interval II~97 cycles (@200 MHz)

❑ Throughput (BRAM) = 10.867 MHz
➔ instantiation interval II~19 cycles (@200 MHz)

Throughput issue! ==> evaluation ongoing on whole HW design instantiation interval!

Model Quantization

- Inputs, Activations: fixed point<16,6>
- Weights, Biases: fixed point<8,1>



❑ Accuracy = (TP+TN)/(TP+TN+FP+FN) = 0.898
❑ Purity = TP/(TP+FP) = 0.831
❑ Recall = TP/(TP+FN) = 0.999

Conclusions

- Implementation of a simplified version of the distributed MLP NN model
- Assessed its performance in terms of **accuracy/purity/recall (ML classification metrics)** and **resources/throughput (HW implementation metrics)**
- Working to improve:
  - **purity** (reduce at minimum the number of signal events classified as noise)
  - **Post-quantization** performance beyond 100kHz noise rate
  - **Throughput** of the full pipeline
- Development of a simplified distributed MLP on two FPGAs including all the architectural blocks (5 DAM NNs and a full TP) is ongoing ⇒ validation of the **DAM to TP communication**
- Sector NN model fully validated ⇒ Xilinx Versal design for the target FELIX implementation is ongoing

Thanks for your attention!

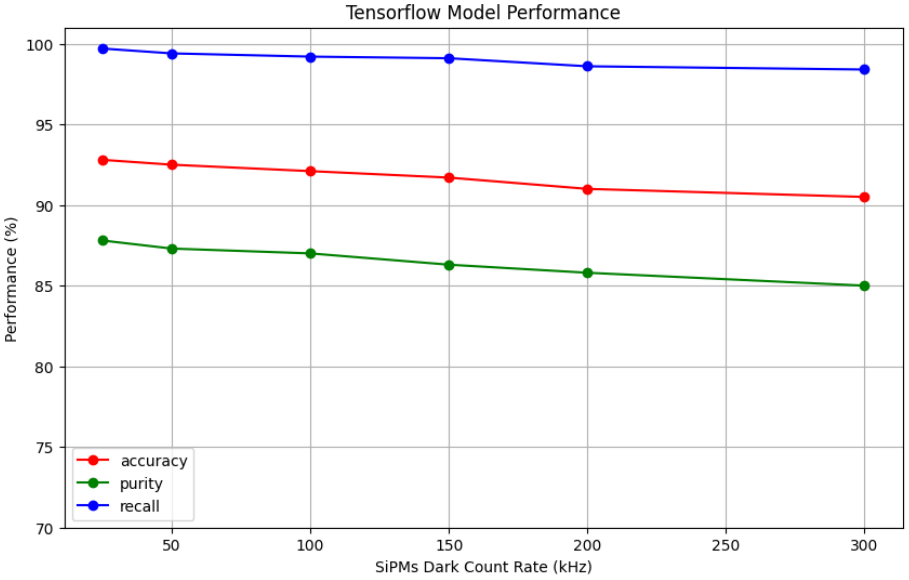
Contacts:

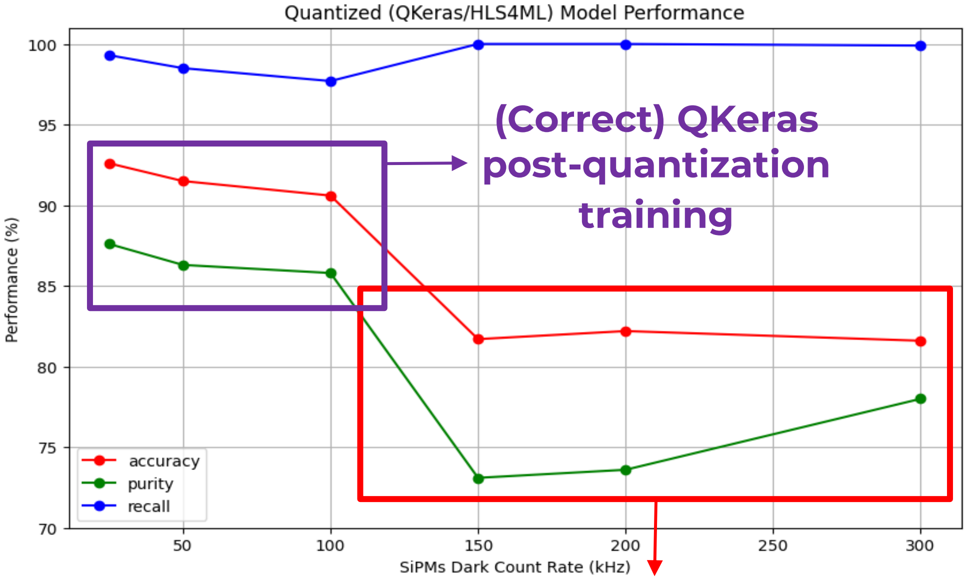
- cristian.rossi@roma1.infn.it
- alessandro.lonardo@roma1.infn.it
- https://apegate.roma1.infn.it

BACKUP SLIDES

NN Model performance scaling

We noticed a drop of prediction performance with increasing dark count rate (e.g. increasing number of noise hits per event), but still *purity > 85%* for noisiest case (DCR = 300 kHz).
As expected, prediction performance drop after quantization step



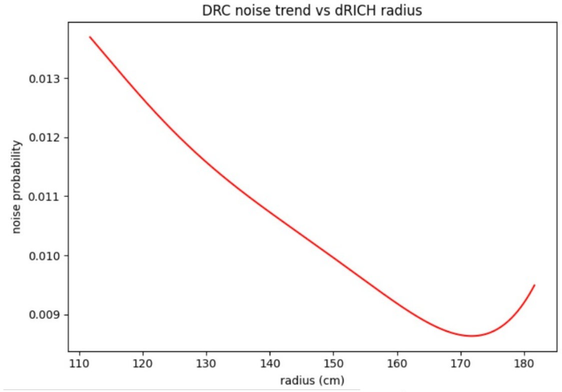


(Correct) QKeras post-quantization training

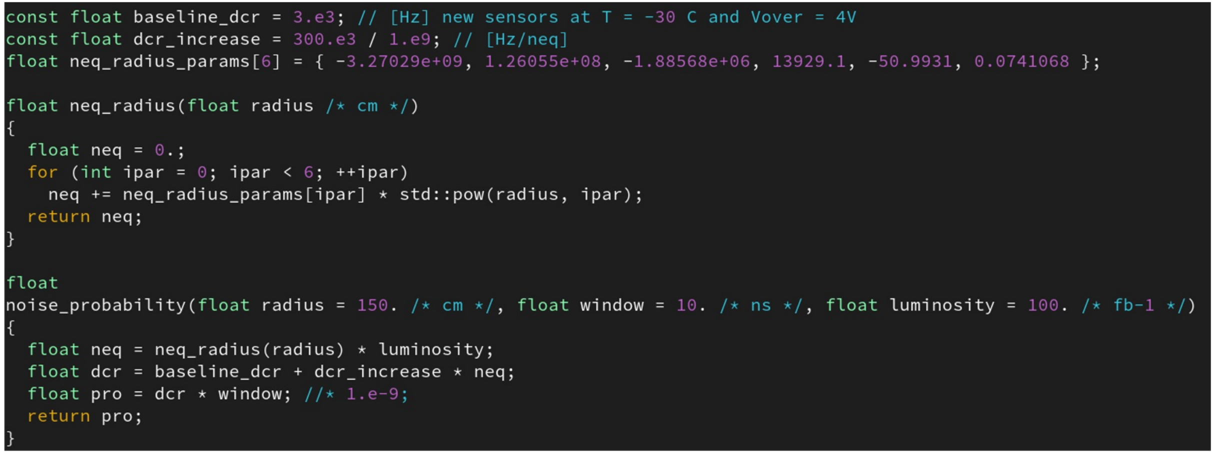
No QKeras training, only HLS4ML quantization

dRICH Data reduction: Noise Distribution

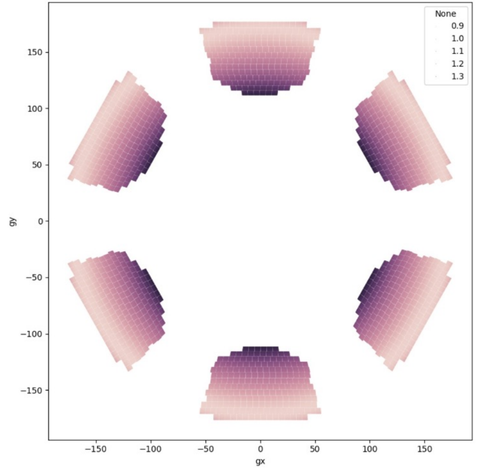
- Dark current SiPM **noise hits distribution,** obtained by introducing Dark Count probability of single dRICH SiPM with a dependence on its **radial distance from the detector z-axis** and on the **integrated luminosity**
  ⇒ Implemented in EICRecon digitization step (new flag to enable new model noise)

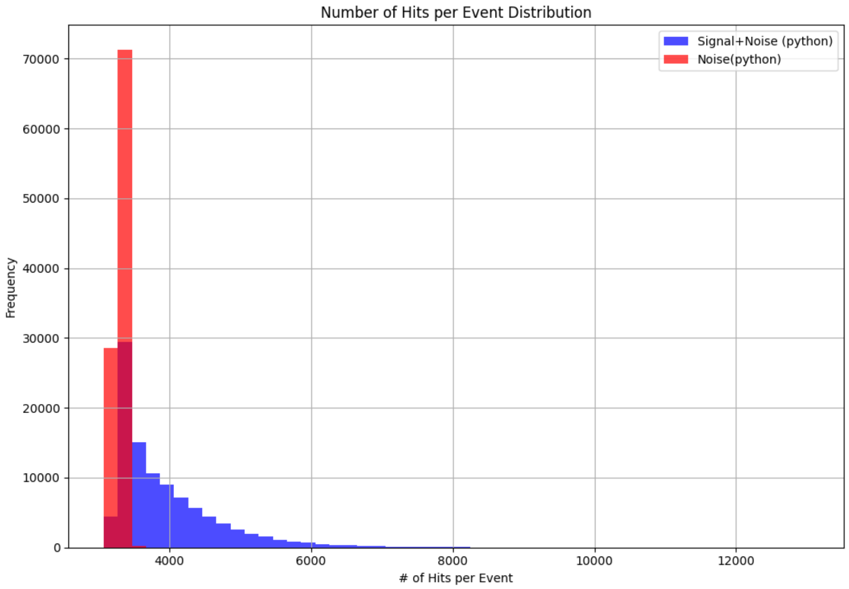


(R. Preghenella's contribution)

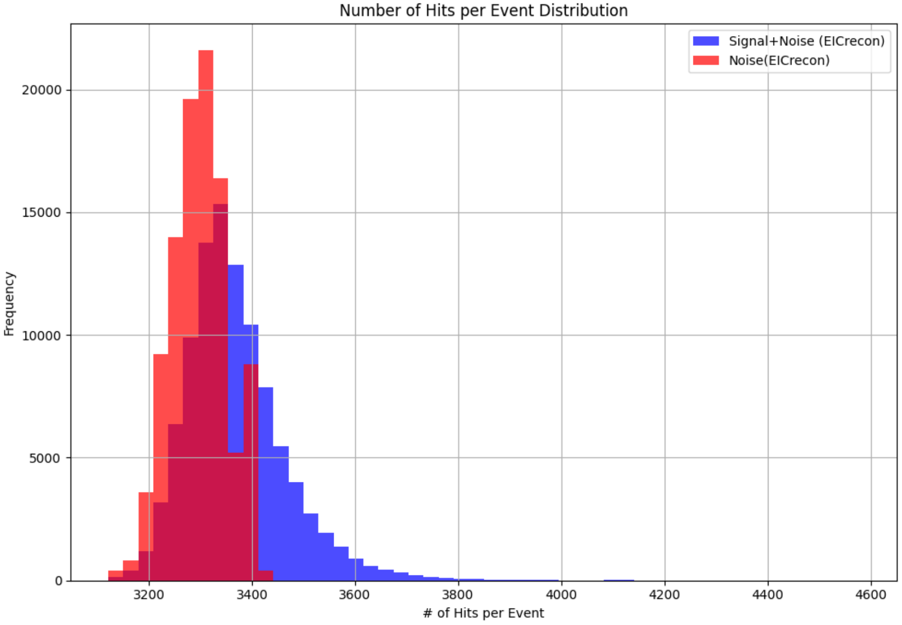


```
const float baseline_dcr = 3.e3; // [Hz] new sensors at T = -30 C and Vover = 4V
const float dcr_increase = 300.e3 / 1.e9; // [Hz/neq]
float neq_radius_params[6] = { -3.27029e+09, 1.26055e+08, -1.88568e+06, 13929.1, -50.9931, 0.0741068 };

float neq_radius(float radius /* cm */)
{
  float neq = 0.;
  for (int ipar = 0; ipar < 6; ++ipar)
    neq += neq_radius_params[ipar] * std::pow(radius, ipar);
  return neq;
}

float
noise_probability(float radius = 150. /* cm */, float window = 10. /* ns */, float luminosity = 100. /* fb-1 */)
{
  float neq = neq_radius(radius) * luminosity;
  float dcr = baseline_dcr + dcr_increase * neq;
  float pro = dcr * window; //* 1.e-9;
  return pro;
}
```

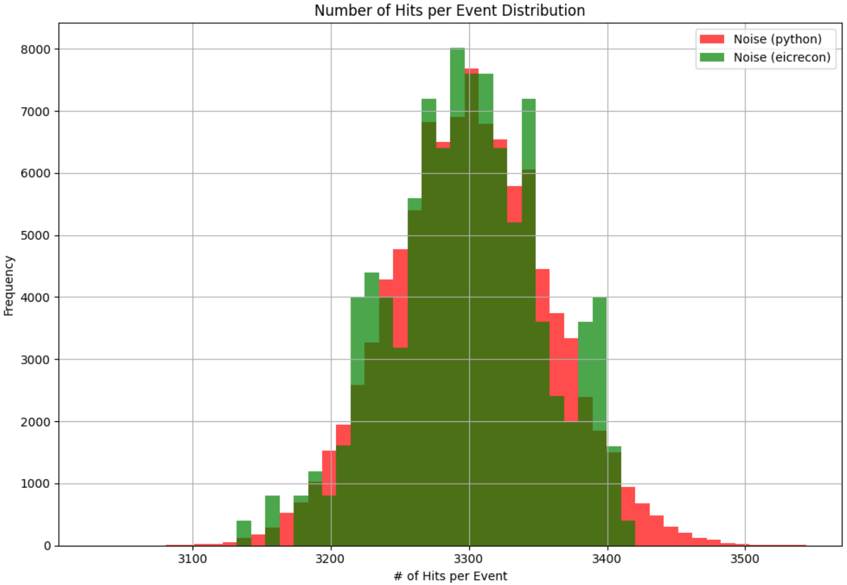




Performance
>80%
(@100fb-1)
Eic-shell
version=25.06



Performance ~99%
(@100fb-1)
Eic-shell
version=24.12



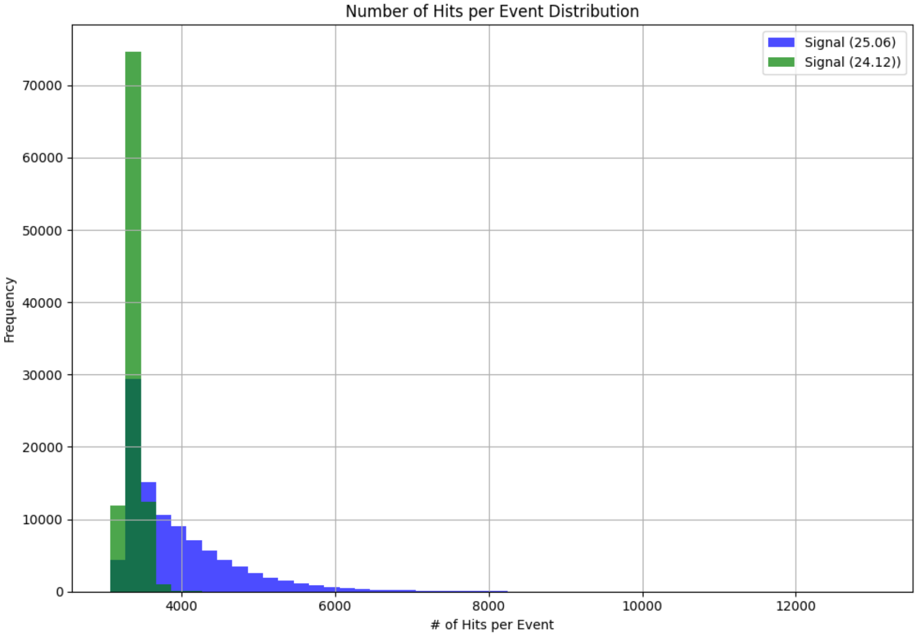
Noise comparison (@100fb-1)

==> same distro mean

Signal Comparison (@100fb-1)

Eic-shell version=24.12 vs 25.06

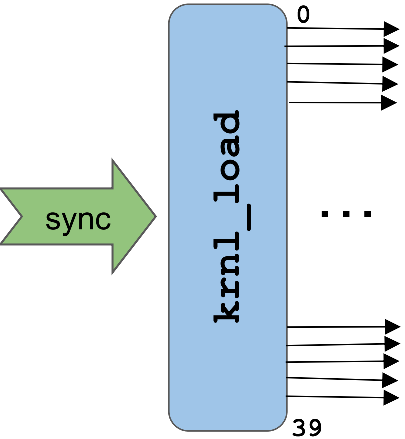
==> same starting MC files

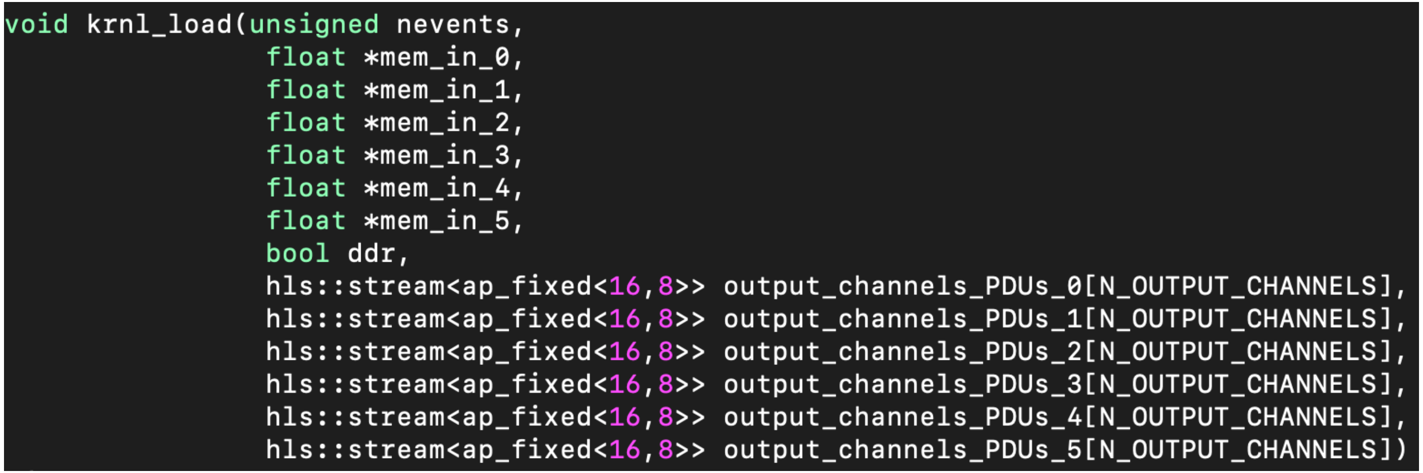


dRICH Data reduction stage on FPGA: HLS4ML ⇒ HW implementation

- **krnl_load** is connected to the Host CPU via PCIe bus, allowing to load events data on the FPGA DDR. Corresponding input data are sent to each of the 6 **Sector MLP blocks** through 40 input hls::stream<ap_fixed<16,8>>.

- By disabling the *ddr* kernel flag, **krnl_load** can send through the system few events data (O(10)) already loaded on the FPGA BRAM during firmware synthesis. In this way, **throughput** measurements can be performed without **DDR reading bottleneck**

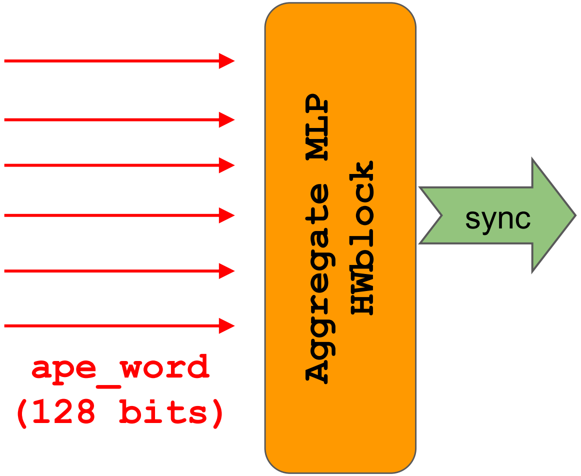


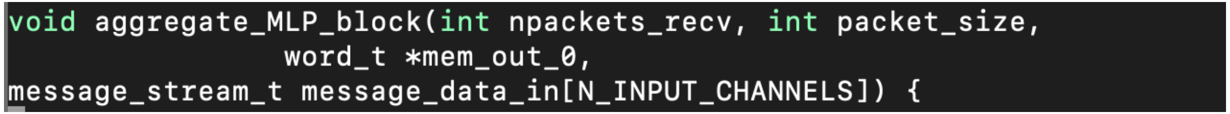


```
void krnl_load(unsigned nevents,
                float *mem_in_0,
                float *mem_in_1,
                float *mem_in_2,
                float *mem_in_3,
                float *mem_in_4,
                float *mem_in_5,
                bool ddr,
                hls::stream<ap_fixed<16,8>> output_channels_PDUs_0[N_OUTPUT_CHANNELS],
                hls::stream<ap_fixed<16,8>> output_channels_PDUs_1[N_OUTPUT_CHANNELS],
                hls::stream<ap_fixed<16,8>> output_channels_PDUs_2[N_OUTPUT_CHANNELS],
                hls::stream<ap_fixed<16,8>> output_channels_PDUs_3[N_OUTPUT_CHANNELS],
                hls::stream<ap_fixed<16,8>> output_channels_PDUs_4[N_OUTPUT_CHANNELS],
                hls::stream<ap_fixed<16,8>> output_channels_PDUs_5[N_OUTPUT_CHANNELS])
```

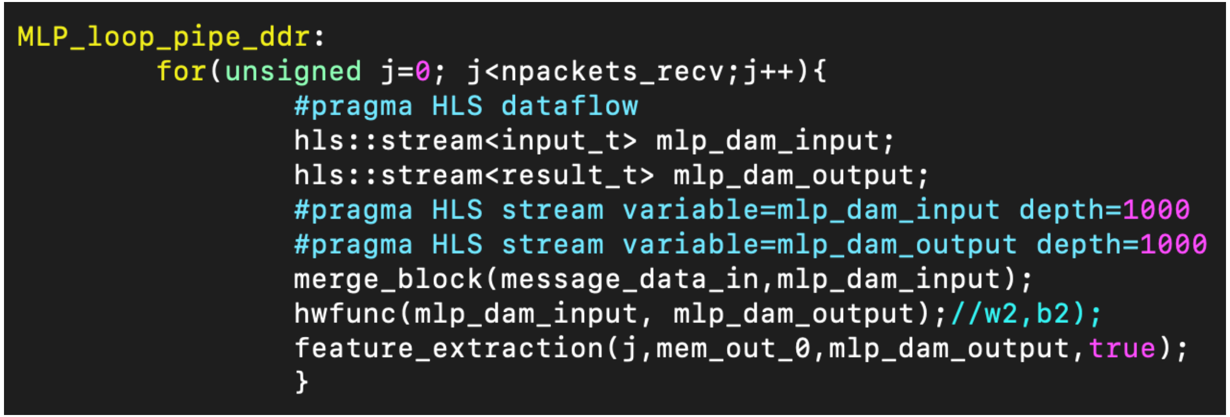
dRICH Data reduction stage on FPGA: HLS4ML ⇒ HW implementation

- ➔ **Aggregate MLP HWblock** receives as input 6 *ape_word* from the 6 **Sector MLP blocks**, each containing 4 features corresponding to the information extracted from a single dRICH sector. Here, incoming data are merged to feed the last MLP layer of the NN model, which finally computes the prediction. This last output is then loaded back to the Host CPU via PCIe in order to compare prediction with the true label of the processed event





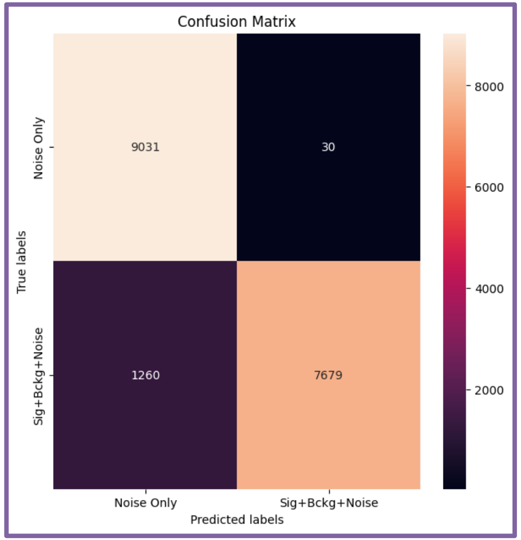
```
void aggregate_MLP_block(int npackets_recv, int packet_size,
                word_t *mem_out_0,
message_stream_t message_data_in[N_INPUT_CHANNELS]) {
```



```
MLP_loop_pipe_ddr:
        for(unsigned j=0; j<npackets_recv;j++){
                #pragma HLS dataflow
                hls::stream<input_t> mlp_dam_input;
                hls::stream<result_t> mlp_dam_output;
                #pragma HLS stream variable=mlp_dam_input depth=1000
                #pragma HLS stream variable=mlp_dam_output depth=1000
                merge_block(message_data_in,mlp_dam_input);
                hwfunc(mlp_dam_input, mlp_dam_output);//w2,b2);
                feature_extraction(j,mem_out_0,mlp_dam_output,true);
                }
```

NN Model performance (25 KHz & 10ns)

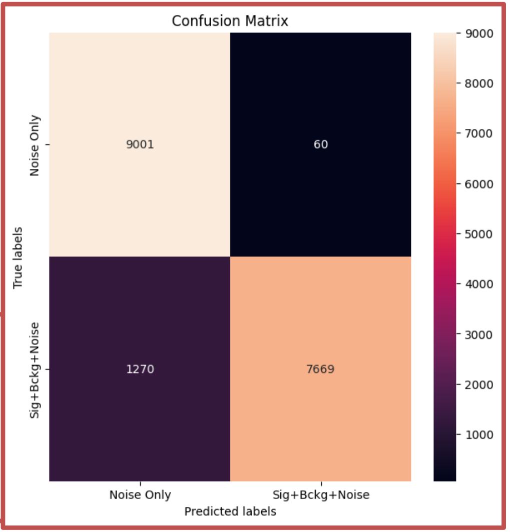
Keras model



- □ Accuracy = (TP+TN)/(TP+TN+FP+FN) = 0.928
- □ Purity = TP/(TP+FP) = 0.878
- □ Recall = TP/(TP+FN) = 0.997

Model Quantization

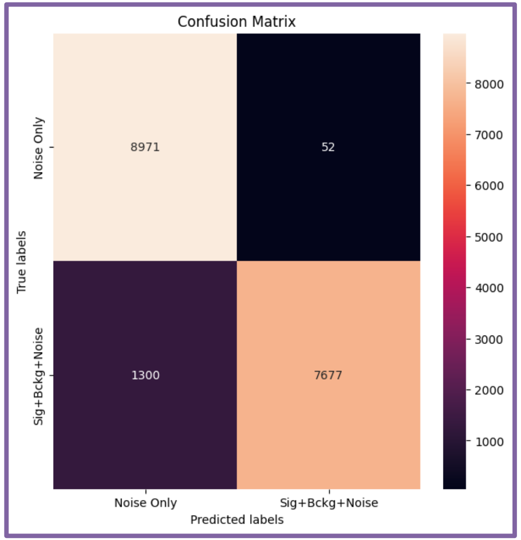
- Inputs, Activations: fixed point<16,6>
- Weights, Biases: fixed point<8,1>



- □ Accuracy = (TP+TN)/(TP+TN+FP+FN) = 0.926
- □ Purity = TP/(TP+FP) = 0.876
- □ Recall = TP/(TP+FN) = 0.993

NN Model performance (50 KHz & 10ns)

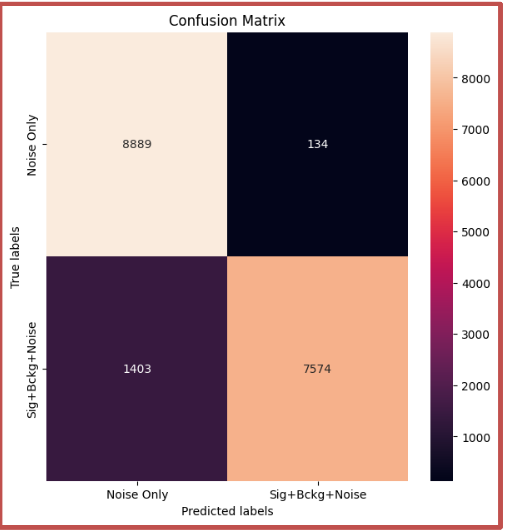
Keras model



- □ Accuracy = (TP+TN)/(TP+TN+FP+FN) = 0.925
- □ Purity = TP/(TP+FP) = 0.873
- □ Recall = TP/(TP+FN) = 0.994

Model Quantization

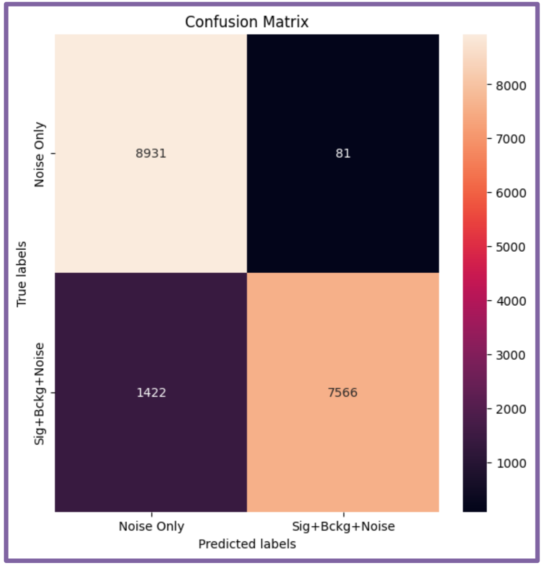
- Inputs, Activations: fixed point<16,6>
- Weights, Biases: fixed point<8,1>



- □ Accuracy = (TP+TN)/(TP+TN+FP+FN) = 0.915
- □ Purity = TP/(TP+FP) = 0.863
- □ Recall = TP/(TP+FN) = 0.985

NN Model performance (150 KHz & 10ns)

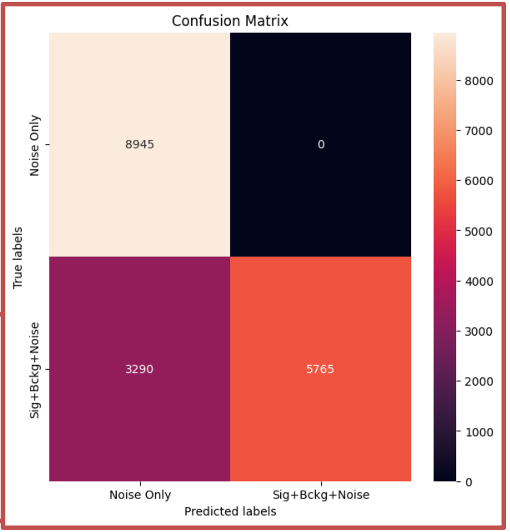
Keras model



- □ Accuracy = (TP+TN)/(TP+TN+FP+FN) = 0.917
- □ Purity = TP/(TP+FP) = 0.863
- □ Recall = TP/(TP+FN) = 0.991

Model Quantization

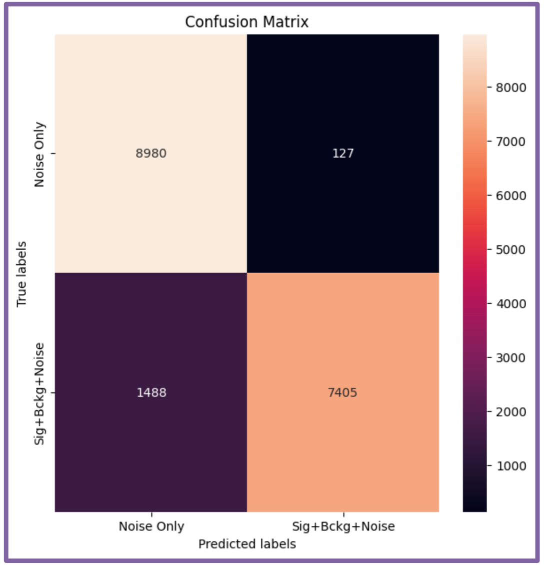
- Inputs, Activations: fixed point<16,6>
- Weights, Biases: fixed point<8,1>



- □ Accuracy = (TP+TN)/(TP+TN+FP+FN) = 0.817
- □ Purity = TP/(TP+FP) = 0.731
- □ Recall = TP/(TP+FN) = 1.000

NN Model performance (200 KHz & 10ns)

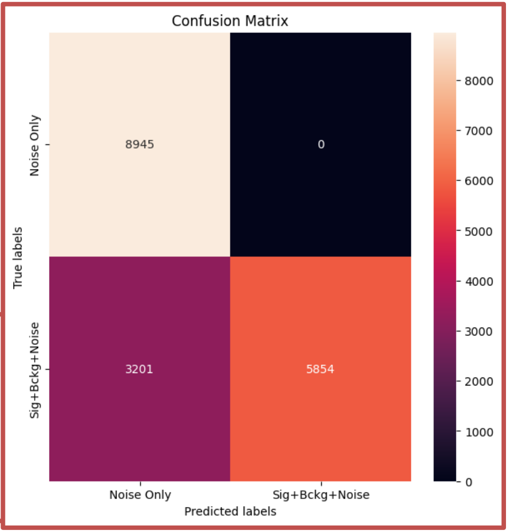
Keras model



- ❑ Accuracy = (TP+TN)/(TP+TN+FP+FN) = 0.910
- ❑ Purity = TP/(TP+FP) = 0.858
- ❑ Recall = TP/(TP+FN) = 0.986

Model Quantization

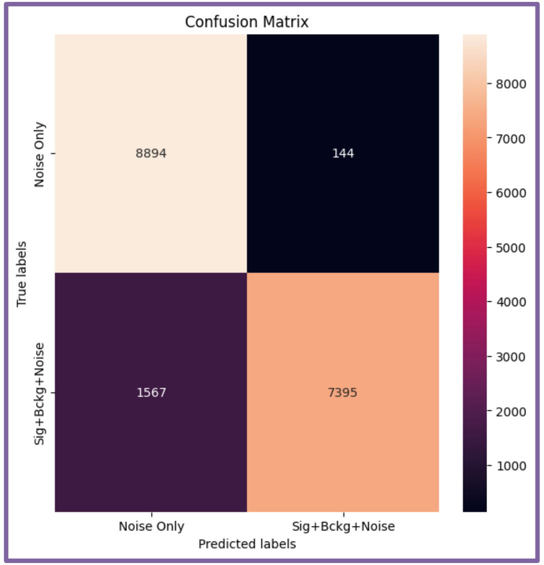
- Inputs, Activations: fixed point<16,6>
- Weights, Biases: fixed point<8,1>



- ❑ Accuracy = (TP+TN)/(TP+TN+FP+FN) = 0.822
- ❑ Purity = TP/(TP+FP) = 0.736
- ❑ Recall = TP/(TP+FN) = 1.000

NN Model performance (300 KHz & 10ns)

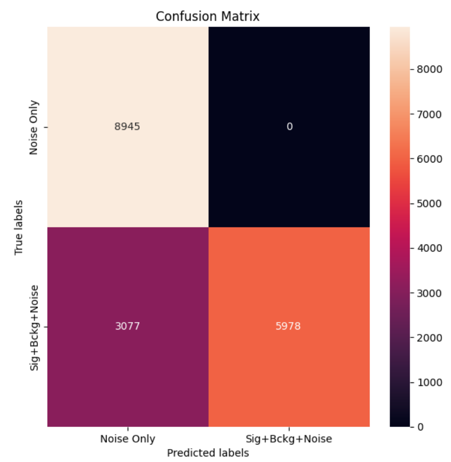
Keras model



- □ Accuracy = (TP+TN)/(TP+TN+FP+FN) = 0.905
- □ Purity = TP/(TP+FP) = 0.850
- □ Recall = TP/(TP+FN) = 0.984

Model Quantization

- Inputs, Activations: fixed point<16,6>
- Weights, Biases: fixed point<8,1>



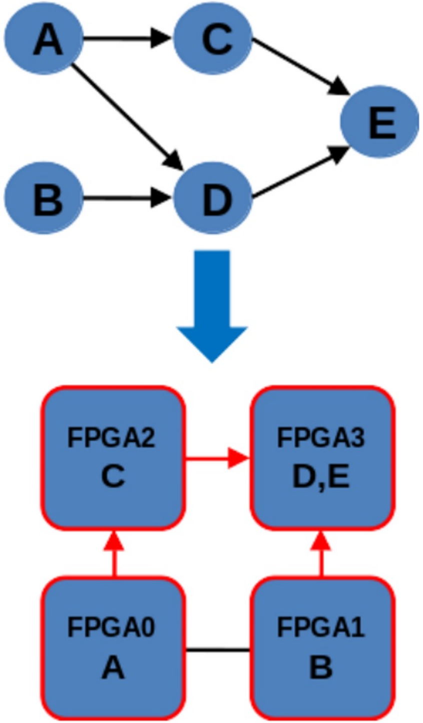
- □ Accuracy = (TP+TN)/(TP+TN+FP+FN) = 0.829
- □ Purity = TP/(TP+FP) = 0.744
- □ Recall = TP/(TP+FN) = 1.000

BACKUP^2 SLIDES

APEIRON: overview

APEIRON is a framework developed to offer hardware and software support for the execution of real-time dataflow applications on a system composed by interconnected FPGAs

- Enabling the mapping the dataflow graph of the application on the distributed FPGA system and offering runtime support for the execution.
- Allowing users, with no (or little) experience in hardware design tools, to develop their applications on such distributed FPGA-based platforms:
  - Tasks are implemented in C++ using High Level Synthesis tools (Xilinx® Vitis).
  - Lightweight C++ communication API (HAPECOM)
    - Non-blocking *send()*
    - Blocking *receive()*

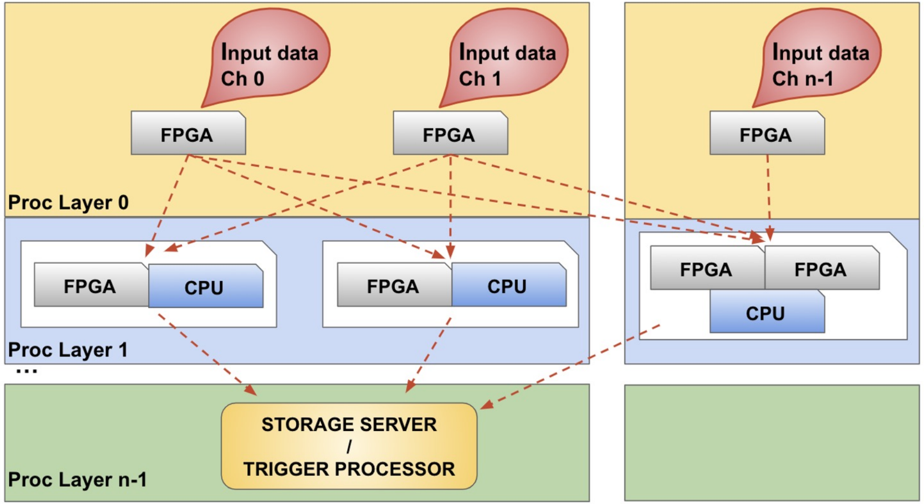


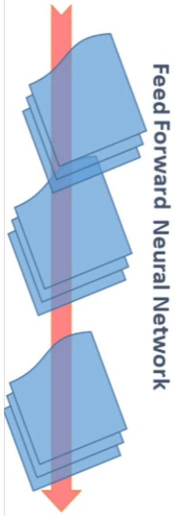
APEIRON enables the scaling of Xilinx® Vitis High Level Synthesis applications on multiple FPGA interconnected by the INFN communication IP.

APEIRON for smart TDAQ Systems

Abstract **P**rocessing **E**nvironment for **I**ntelligent **R**ead-**O**ut systems based on **N**eural networks

- Input **data streams** from several different channels (data sources, detectors/sub-detectors) recombined through the processing layers using a **low-latency, modular and scalable network infrastructure**

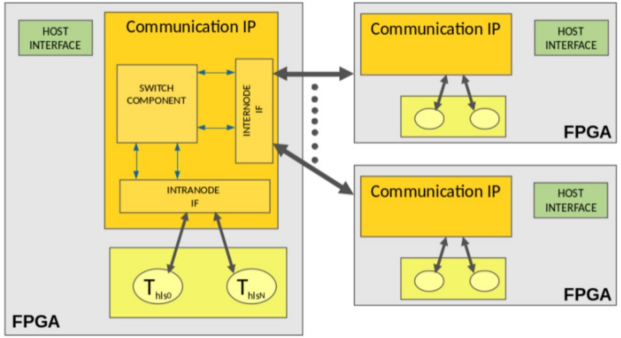


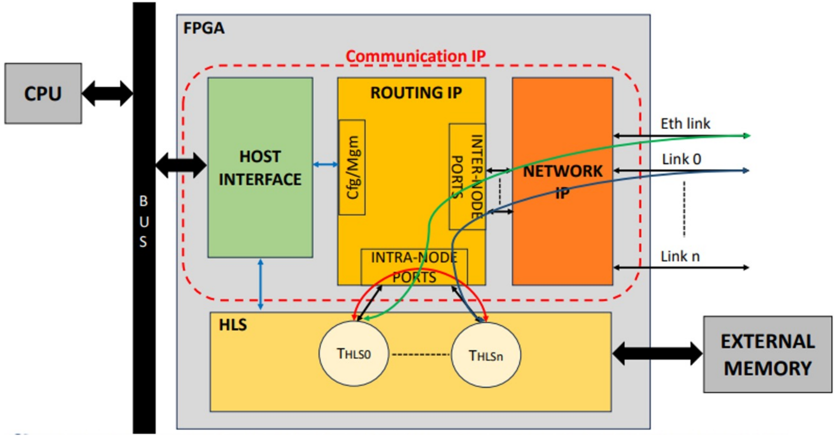


- More resource-demanding NN layers can be implemented in subsequent processing layers.
- Classification produced by the NN in last processing layer (e.g. pid) will be input for the **trigger processor/storage online data reduction stage for triggerless systems**.

APEIRON building blocks:

• INFN Communication IP



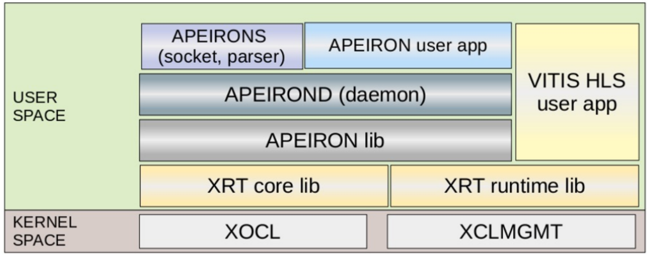


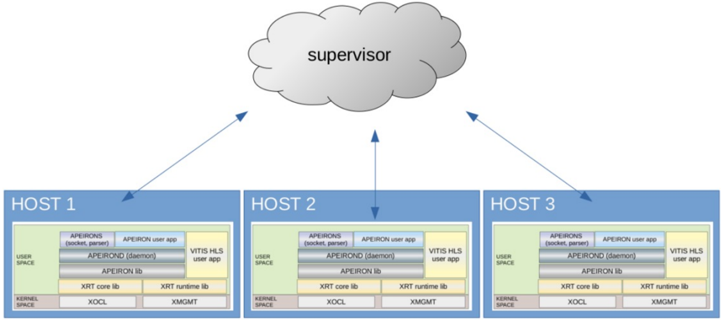
INFN is developing the IPs implementing a direct network that allows **low-latency data transfer** between processing tasks deployed on the same FPGA (**intra-node communication**) and on different FPGAs (inter-node communication)

- **Host Interface IP:** Interface the FPGA logic with the host through the system bus.
- **Routing IP:** Routing of intra-node and inter-node messages between processing tasks on FPGA. •
- **Network IP:** Network channels and Application-dependent I/O
    - **APElink** 20 Gbps → 40 Gbps
    - UDP/IP over 1/10 GbE → 25/40/100 GbE
    - **ETH port** → Xilinx® 10G/25G High Speed Ethernet Subsystem

APEIRON building blocks:

- Software Stack





The APEIRON runtime software stack is built on top of the Xilinx® XRT one adding three layers to:

- add the functionalities required to manage multiple FPGA execution platforms (e.g., **program** the devices, **configure** the IPs, start/stop **execution**, **monitor** the status of IPs, …);
- reduce the impact of changes in XRT API introduced with any new version of Vitis on the APEIRON host-side applications;
- decouple the APEIRON software stack from the specific platform, easing the future porting of the framework to different platforms/vendors.

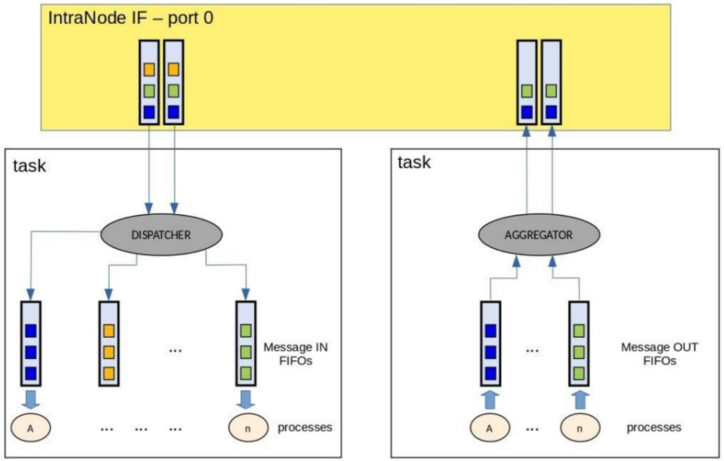
Apeirond is a persistent daemon used to manage multiple access request from user apps to the board.
Using the network socket exposed by apeirond modules, the **supervisor** can write commands and read status of the different instances of the APEIRON framework running in each node, allowing the user to have a complete overview of the multiple FPGA execution platform

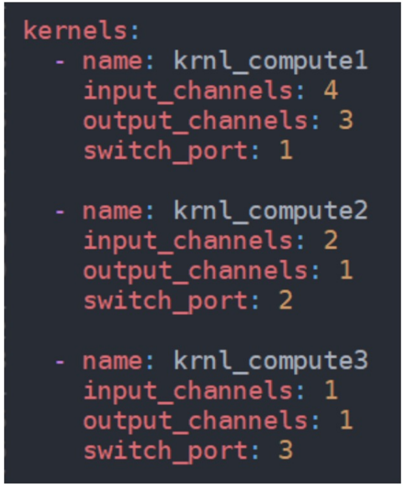
APEIRON: FPGA bitstream generation

- The **HLS task** must have a generic interface, implementation is free
- A **YAML configuration file** is used to describe the kernels interconnection topology, specifying how many input/output channels they have

```
void example_task(
[list of optional kernel specific
parameters], message_stream_t
message_data_in[N_INPUT_CHANNELS],
message_stream_t
message_data_out[N_OUTPUT_CHANNELS])
```

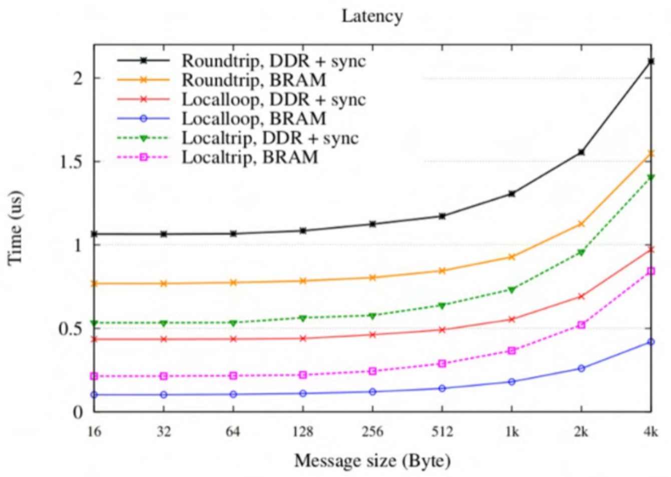
Adaptation toward/from IntraNode ports of the Routing IP is done by the automatically generated **Aggregator** and **Dispatcher** kernel templates.

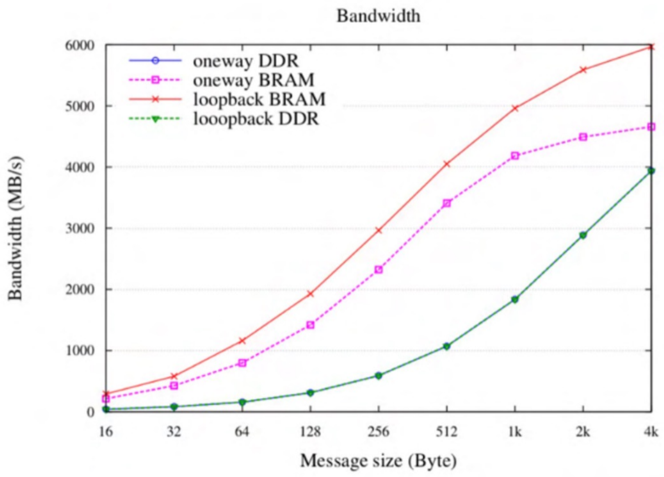




```yaml
kernels:
  - name: krnl_compute1
    input_channels: 4
    output_channels: 3
    switch_port: 1

  - name: krnl_compute2
    input_channels: 2
    output_channels: 1
    switch_port: 2

  - name: krnl_compute3
    input_channels: 1
    output_channels: 1
    switch_port: 3
```

APEIRON performance
(Communication IP: 256 bit datapath @200MHz)





Latency

	DDR+sync(ns)	BRAM(ns)
Intra-node (localtrip)	533	213
Inter-node (roundtrip)	1065	768

Bandwidth

	DDR+sync(MB/s)	BRAM(MB/s)
Intra-node (loopback)	3938	5967

APEIRON applications:

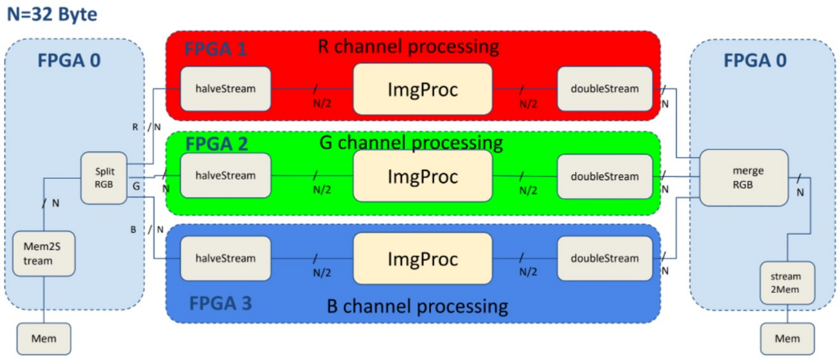
- ## FIPLib-multiFPGA

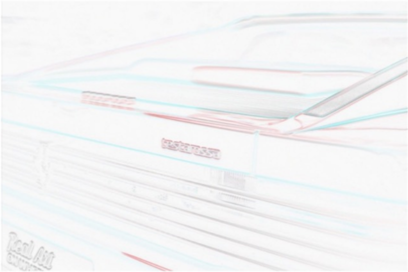
FPGA **I**mage **P**rocessing **Lib**rary ⇒ multi-FPGA implementation via APEIRON

- Developed by ENEA in C++, it employs the **Vitis HLS flow** to construct the library's kernels for the execution of image processing algorithms.

- FIPLib encompasses nearly 70 functionalities, conceived with a **streaming behavior**

- On a multi-FPGA setup, we were able to split the overall image processing by implementing a single RGB kernel on each node
 ⇒ **increased internal datapath to 32B**, avoiding FPGA resource limitation

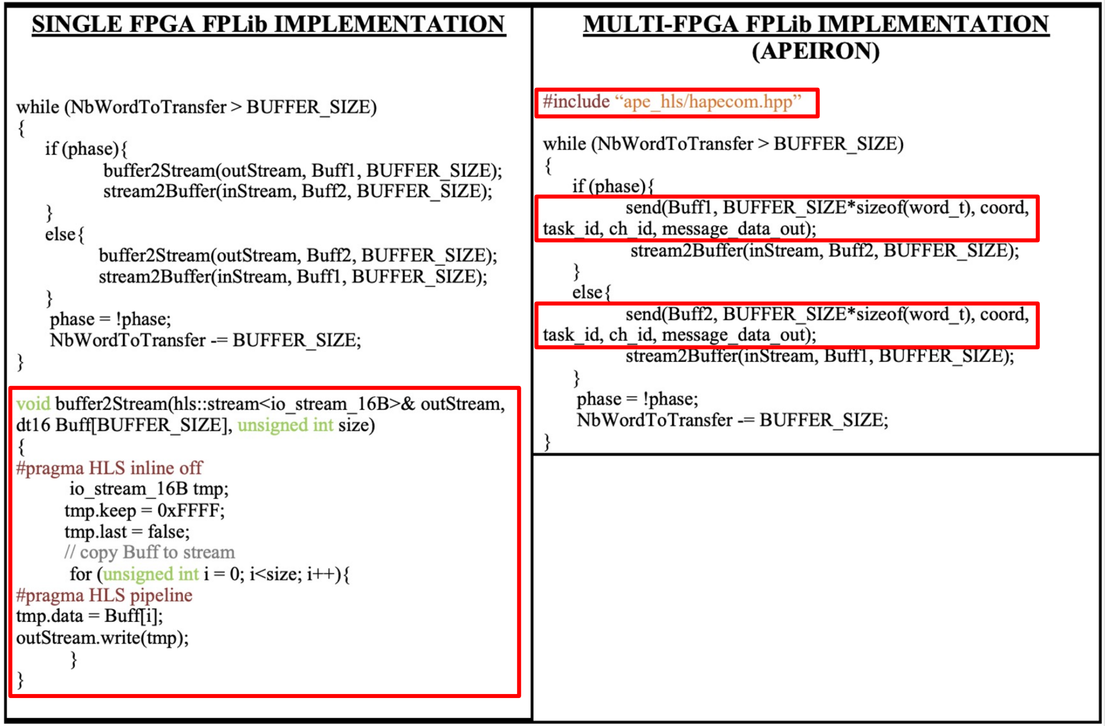




APEIRON applications:
- # FIPLib-multiFPGA

FPGA **I**mage **P**rocessing **Lib**rary ⇒ multi-FPGA implementation via APEIRON

- Implementing **FIPLib HLS kernels as APEIRON tasks** means changing the interface of each of them to cope with the standard required by the framework to compile the entire project and to generate the bitstream
  ⇒ use of HAPECOM C++ communication API



SINGLE FPGA FPLib IMPLEMENTATION

```
while (NbWordToTransfer > BUFFER_SIZE)
{
    if (phase){
            buffer2Stream(outStream, Buff1, BUFFER_SIZE);
            stream2Buffer(inStream, Buff2, BUFFER_SIZE);
    }
    else{
            buffer2Stream(outStream, Buff2, BUFFER_SIZE);
            stream2Buffer(inStream, Buff1, BUFFER_SIZE);
    }
    phase = !phase;
    NbWordToTransfer -= BUFFER_SIZE;
}
```

```
void buffer2Stream(hls::stream<io_stream_16B>& outStream,
dt16 Buff[BUFFER_SIZE], unsigned int size)
{
#pragma HLS inline off
        io_stream_16B tmp;
        tmp.keep = 0xFFFF;
        tmp.last = false;
        // copy Buff to stream
        for (unsigned int i = 0; i<size; i++){
#pragma HLS pipeline
tmp.data = Buff[i];
outStream.write(tmp);
        }
}
```

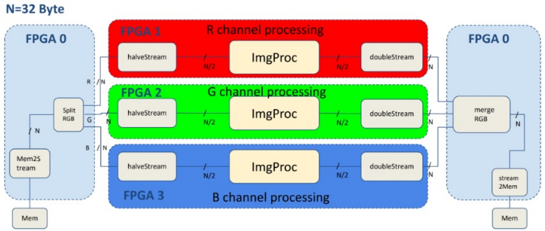
MULTI-FPGA FPLib IMPLEMENTATION (APEIRON)

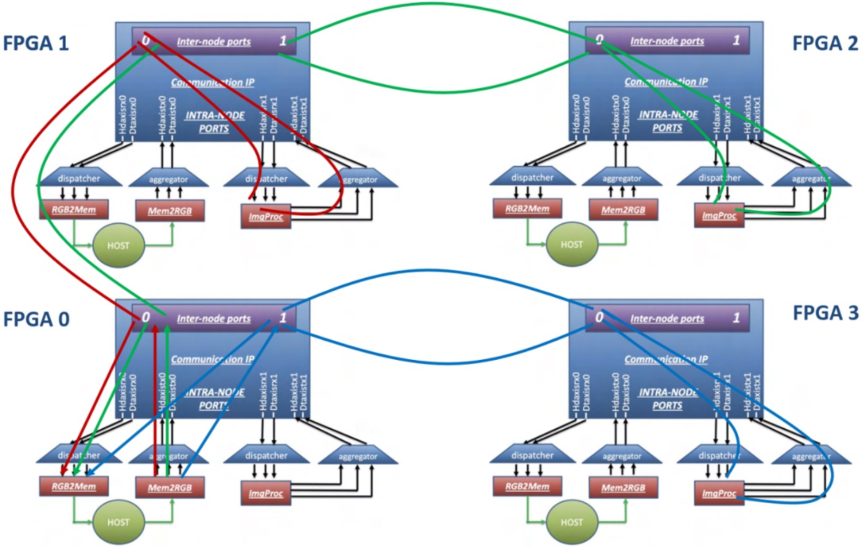
```
#include "ape_hls/hapecom.hpp"

while (NbWordToTransfer > BUFFER_SIZE)
{
    if (phase){
            send(Buff1, BUFFER_SIZE*sizeof(word_t), coord,
task_id, ch_id, message_data_out);
            stream2Buffer(inStream, Buff2, BUFFER_SIZE);
    }
    else{
            send(Buff2, BUFFER_SIZE*sizeof(word_t), coord,
task_id, ch_id, message_data_out);
            stream2Buffer(inStream, Buff1, BUFFER_SIZE);
    }
    phase = !phase;
    NbWordToTransfer -= BUFFER_SIZE;
}
```

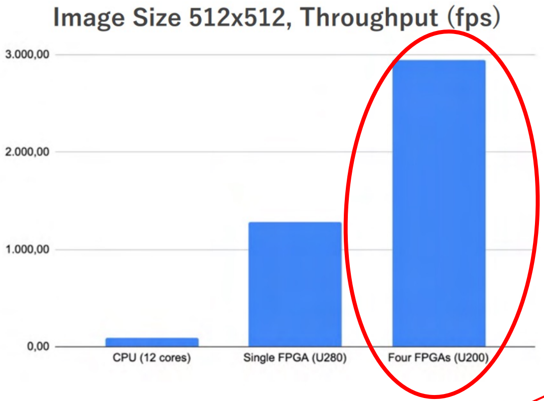
APEIRON applications:

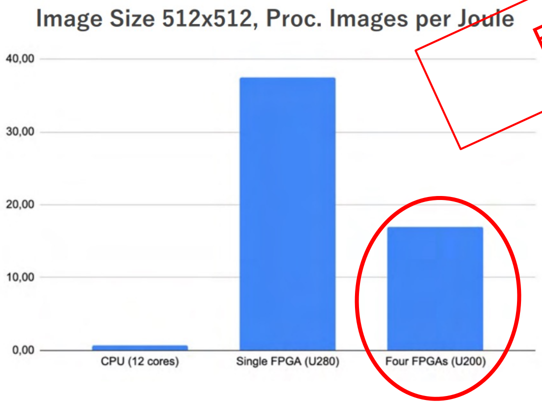
- # FIPLib-multiFPGA

textarossa









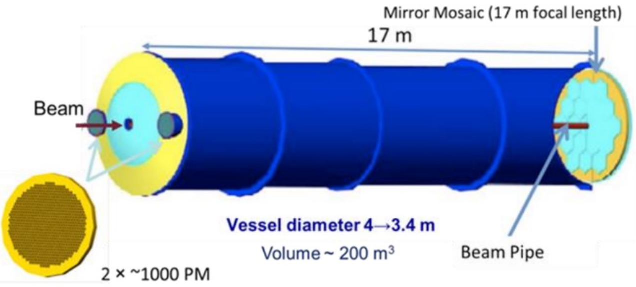
Energy Efficiency - Throughput trade-off

APEIRON applications:

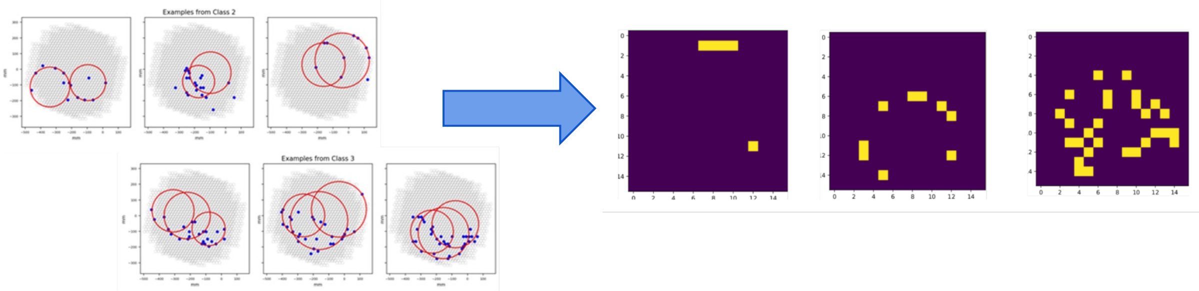
● RAIDER

Real-time **AI**-based **D**ata analytics on h**E**te**R**ogeneous distributed systems

- **High throughput online streaming processing** on multi-FPGA ⇒ **number of Cherenkov rings prediction** on the stream of events generated by the RICH detector in the CERN NA62 experiment at a rate of about 10 MHz, using multiple *CNN_kernel* replica.

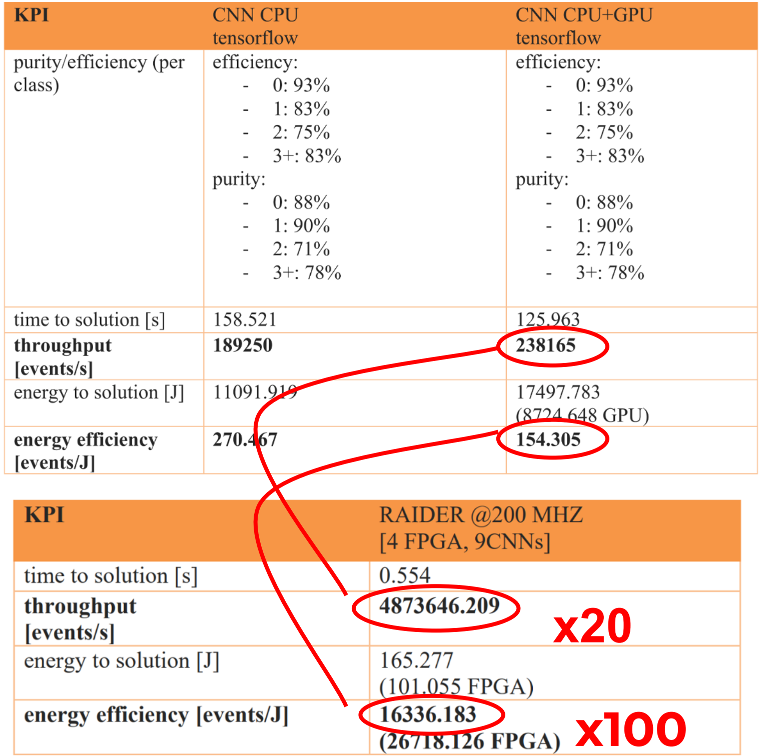


- **Lightweight CNN model** deployed on Xilinx Alveo U280 FPGA (limited resource usage) ⇒ receives as input compressed representation of the original event in form of B&W 16x16 image (via *imagifier* kernel)



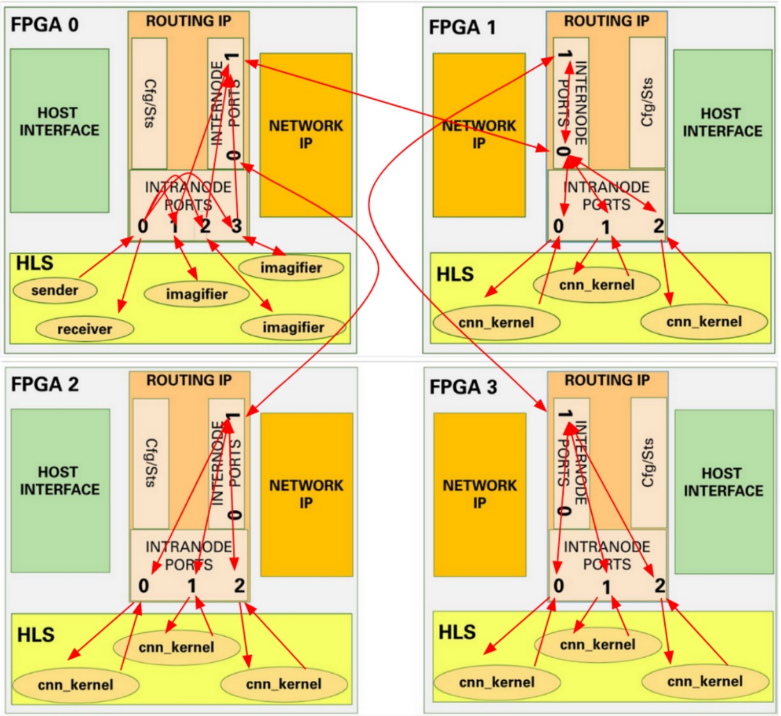
APEIRON applications:

- ## RAIDER



KPI	CNN CPU tensorflow	CNN CPU+GPU tensorflow
purity/efficiency (per class)	efficiency: - 0: 93% - 1: 83% - 2: 75% - 3+: 83% purity: - 0: 88% - 1: 90% - 2: 71% - 3+: 78%	efficiency: - 0: 93% - 1: 83% - 2: 75% - 3+: 83% purity: - 0: 88% - 1: 90% - 2: 71% - 3+: 78%
time to solution [s]	158.521	125.963
throughput [events/s]	**189250**	**238165**
energy to solution [J]	11091.919	17497.783 (8724.648 GPU)
energy efficiency [events/J]	**270.467**	**154.305**

KPI	RAIDER @200 MHZ [4 FPGA, 9CNNs]
time to solution [s]	0.554
throughput [events/s]	**4873646.209**
energy to solution [J]	165.277 (101.055 FPGA)
energy efficiency [events/J]	**16336.183 (26718.126 FPGA)**



Conclusions

- The APEIRON framework enables the **development and deployment of Vitis HLS dataflow applications distributed on multiple-FPGA systems,** leading to increased performance in terms of **throughput** and **energy efficiency**

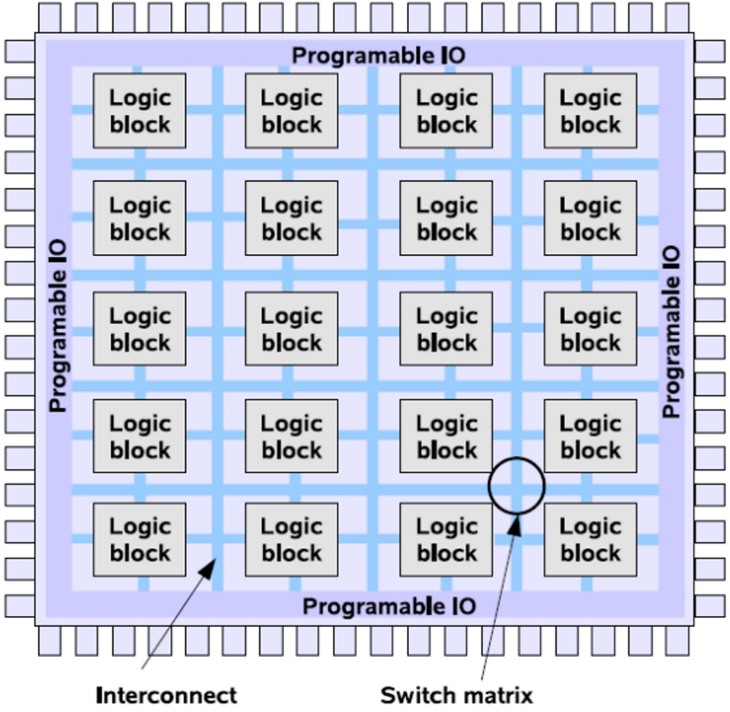
- The co-design of its software stack and of the Communication IP allowed to reach **very low and deterministic latency and a high fraction of the channel's raw bandwidth** for communications between FPGAs, addressing fundamental bottlenecks for real-time distributed dataflow applications.

- We control the workflow for the implementation of **real-time/high throughput classifiers on FPGA** using limited resources,
  This hints for applying the methodology also to:
  - less capable (i.e. front-end) FPGAs
  - complex design making use of a large fraction of FPGA resource
    ⇒ multi-node setup/user-defined topology

FPGA overview

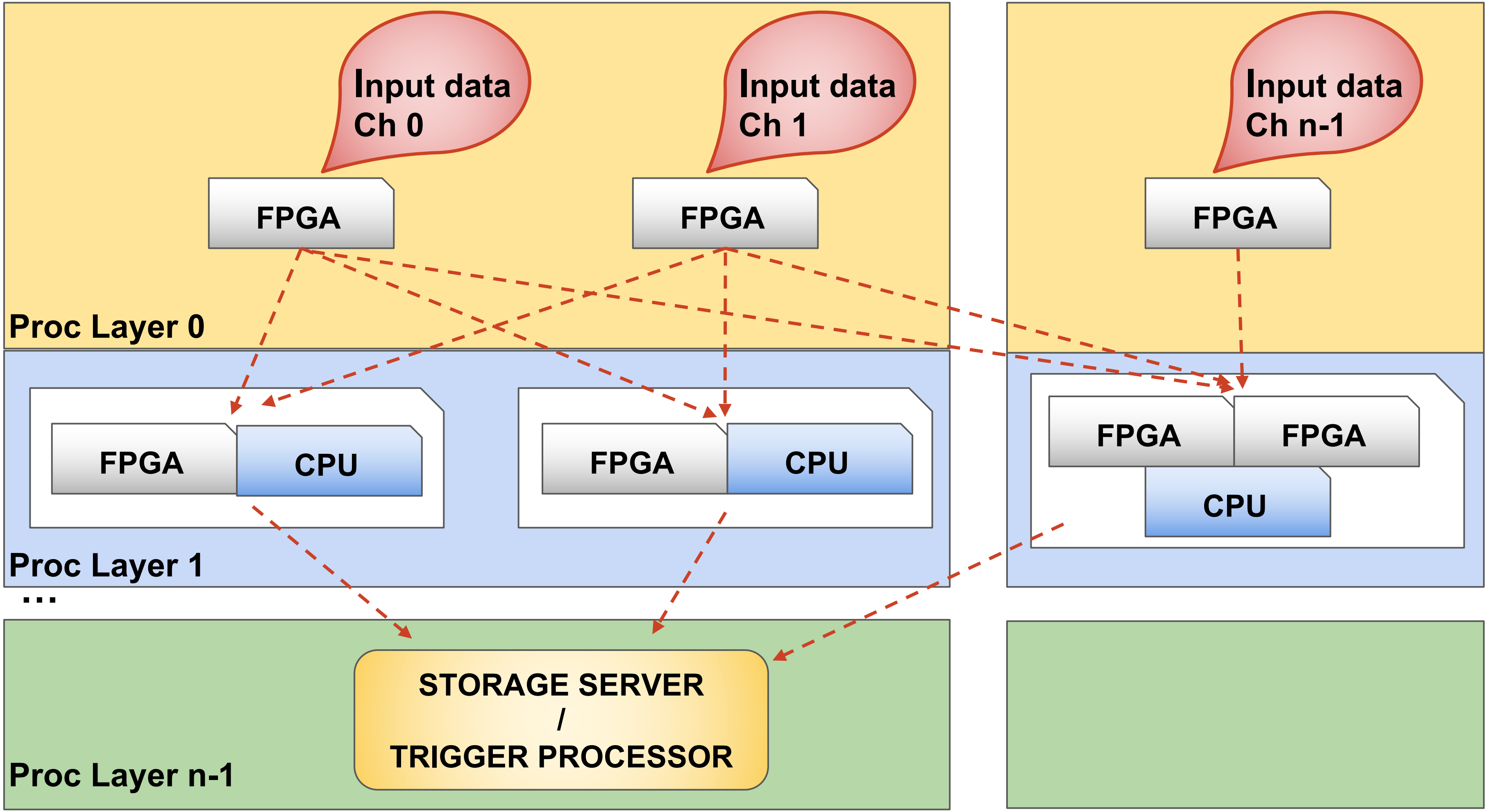
The basic structure of an FPGA is composed of the following elements:

- **Look-up table (LUT):** This element performs logic operations
- **Flip-Flop (FF):** This register element stores the result of the LUT
- **Wires:** These elements connect elements to one another, both logic and clock
- **Input/Output (I/O) pads:** These physically available ports get signals in and out of the FPGA



Conclusions

- We sketched a data reduction system designed based on DAM's FPGAs as a risk-mitigation action to the possible problem of an excessive data bandwidth requirement from the dRICH to Echelon-0 due to SiPM DCR.
- We showed results of the initial activities we made to proof the design concept.
- The design is based on a **distributed Dense NN** model, that can reach **near-optimal performance in terms of accuracy (using simulated data), and promising performance in terms of pipeline throughput.**
- Next steps:
  - Deploy the distributed NN on two FPGAs already available in our lab (Xilinx Alveo U200) representing a DAM and the TP, integrating the communication in the pipeline and assessing its impact on pipeline throughput (and latency).
  - Become familiar with the FELIX board HW and FW (we are receiving a FLX-182 on loan from JLab) to start devising the integration of our design in its FW.
  - In addition different NN models (CNNs, GNN,...) and data reduction tasks/ideas (Cherenkov ring detection...) can be explored, taking into account ePIC DAQ parameters and without altering its data streaming design ("parasitic" mode)



Input data Ch 0

Input data Ch 1

Input data Ch n-1

FPGA

FPGA

FPGA

Proc Layer 0

FPGA

CPU

FPGA

CPU

FPGA

FPGA

CPU

Proc Layer 1

...

STORAGE SERVER / TRIGGER PROCESSOR

Proc Layer n-1