# Ideas for an Online Data Reduction System for the ePIC dRICH Detector

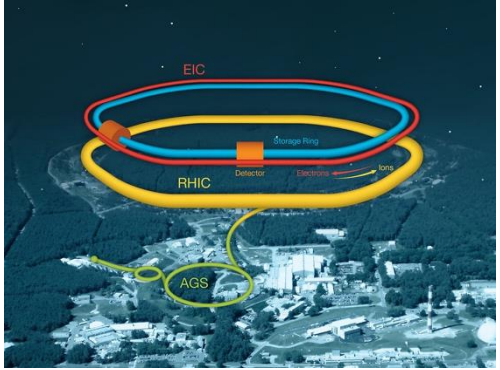
(INFN Sezione di Roma - APE Lab)

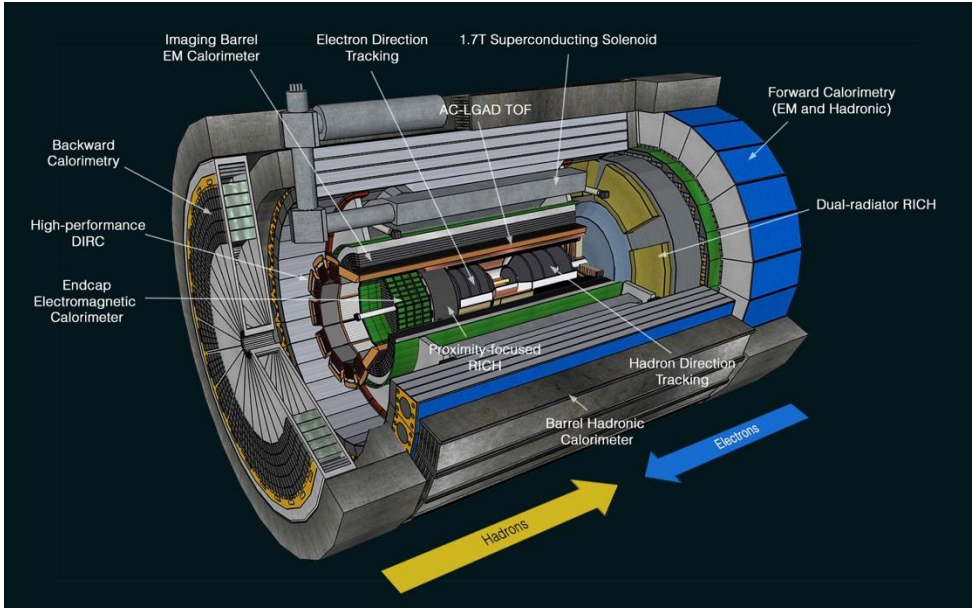
Speaker: Cristian Rossi
(cristian.rossi@roma1.infn.it)

# 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“

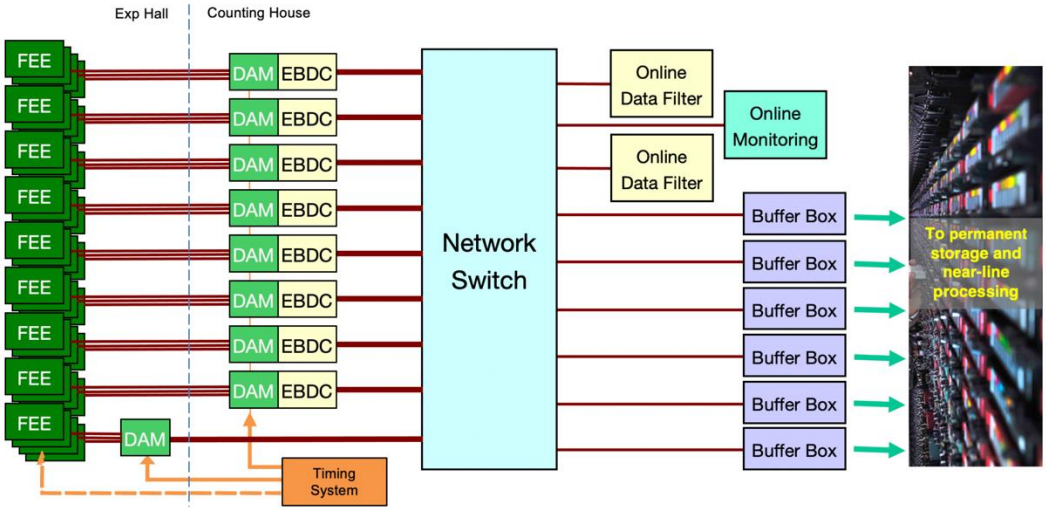


# ePIC: DAQ System

The data from the **Front End Boards (FEBs)** will be aggregated into **Readout Boards (RDOs)** using bidirectional interfaces.

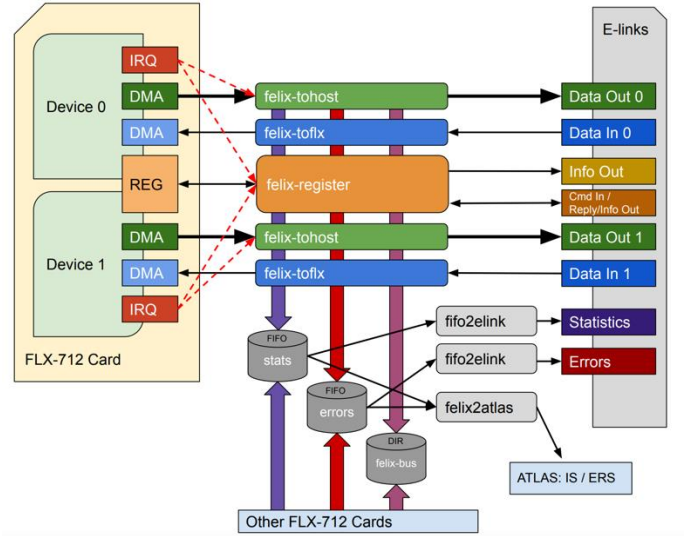
The RDOs will distribute configuration and control information to the FEBs and read hit data as well as monitoring information from the FEBs.
The RDOs will also use a bidirectional optical connection to more powerful FPGA-based hardware, the **Data Aggregation and Manipulation** Board **(DAM)**.

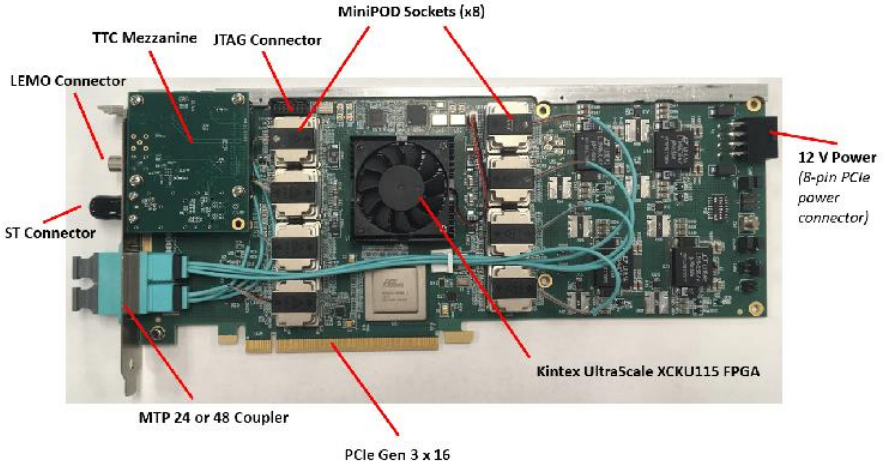


# ePIC: DAM boards (FELIX)

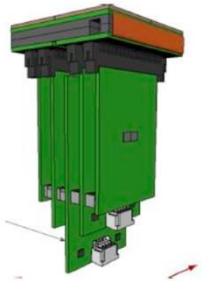
**DAM** boards are envisioned to be a variation of the next generation **FELIX** boards, developed for the ATLAS experiment at LHC.

FELIX FLX-155 board is built around the new **Xilinx Versal FPGA/SoC** family.
It will support up to **48 serial links** running at speeds up to 25Gbps as well as **a 100Gb ethernet** link off the board.
There is a DDR4 16GB RAM slot available to support buffering and they are equipped with a PCIe Gen5x16 bus



# dRICH ➔ RDO and ePIC DAQ (baseline)



1248

- PDU: 1248
- RDO: 1248
- FEB: 4992

1248

I-level DAM (27)

FELIX

- 47 links to PDU
- 1 link to II-level DAM

27

in exp. hall, rack mounted

PC with 4 FELIX each (??)

27

II-level DAM (1)

FELIX

- 27 links to I-level DAM
- link from central ePIC [clock/trigger]

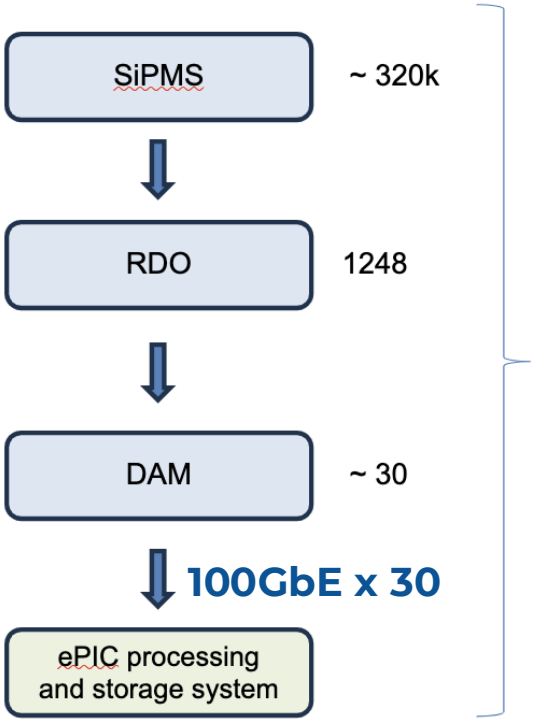
ePIC interaction tagger
able to reach our DAMS in 10 μs!

1

1

# dRICH: Analysis of Output Bandwidth

The dRICH DAQ chain in ePIC ➔ **the throughput issue**




| dRICH DAQ parameters | |
|---|---|
| RDO boards | 1248 |
| ALCOR64 x RDO | 4 |
| dRICH channels (total) | 319488 |
| Number of DAM L1 | 27 |
| Input link in DAM L1 | 47 |
| Output links in DAM L1 | 1 |
| Number of DAM L2 | 1 |
| Input link to DAM L2 | 27 |
| Link bandwidth [ Gb/s] (assumes VTRX+) | 10 |
| Interaction tagger reduction factor | 1 |
| Interaction tagger latency [s] | 2,00E-03 |
| **EIC parameters** | |
| EIC Clock [MHz] | 98,522 |
| Orbit efficiency (takes into account gap) | 0,92 |

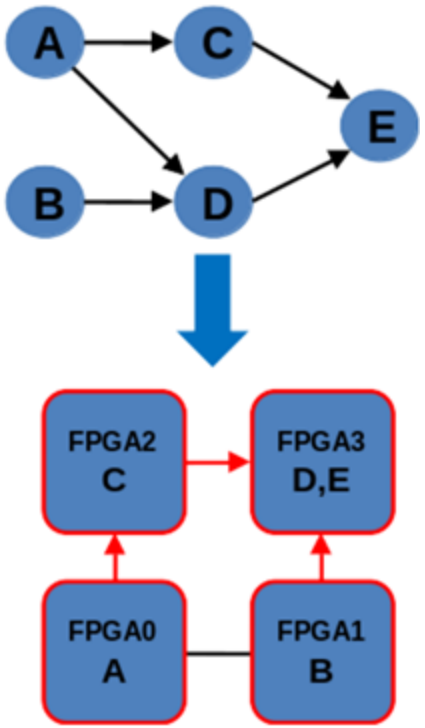
| Bandwidth analysis | | Limit |
|---|---|---|
| Sensor rate per channel [kHz] | 300,00 | 4.000,00 |
| Rate post-shutter [kHz] | 55,20 | 800,00 |
| Throughput to serializer [ Mb/s] | 34,50 | 788,16 |
| Throughput from ALCOR64 [Mb/s] | 276,00 | |
| Throughput from RDO [ Gb/s] | 1,08 | 10,00 |
| Input at each DAM I [Gbps] | 50,67 | 470,00 |
| Buffering capacity at DAM I [MB] | 12,97 | |
| Output from every DAM | 50,67 | 10,00 |
| **Total throughput** | 1.368,14 | 270,00 |

- Sensors DCR: 3-300 kHz (increasing with radiation damage ➔ with experiment lifetime).
- Full detector throughput (FE): 14-1400Gbps
- **A reduction is needed** to cope with 30 channels 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 buches ➔ interaction rate of ~1MHz.
- **A system tagging the (DIS) interacting bunches** can solve the throughput issue (reducing to ~1/100 the data throughput)

# 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()*

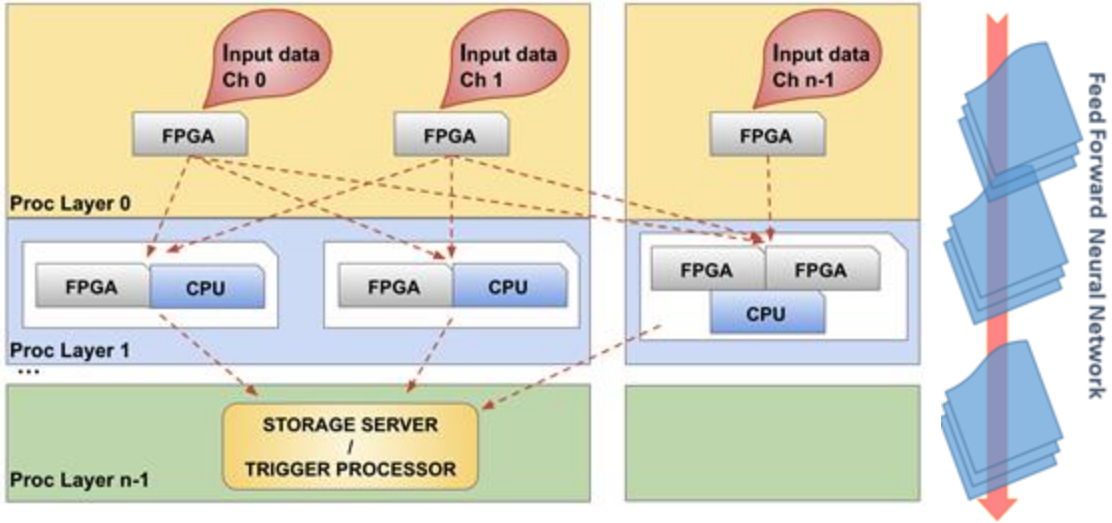



**APEIRON enables the scaling of Xilinx® Vitis High Level Synthesis applications on multiple FPGA interconnected by the INFN communication IP.**

# APEIRON for smart TDAQ Systems

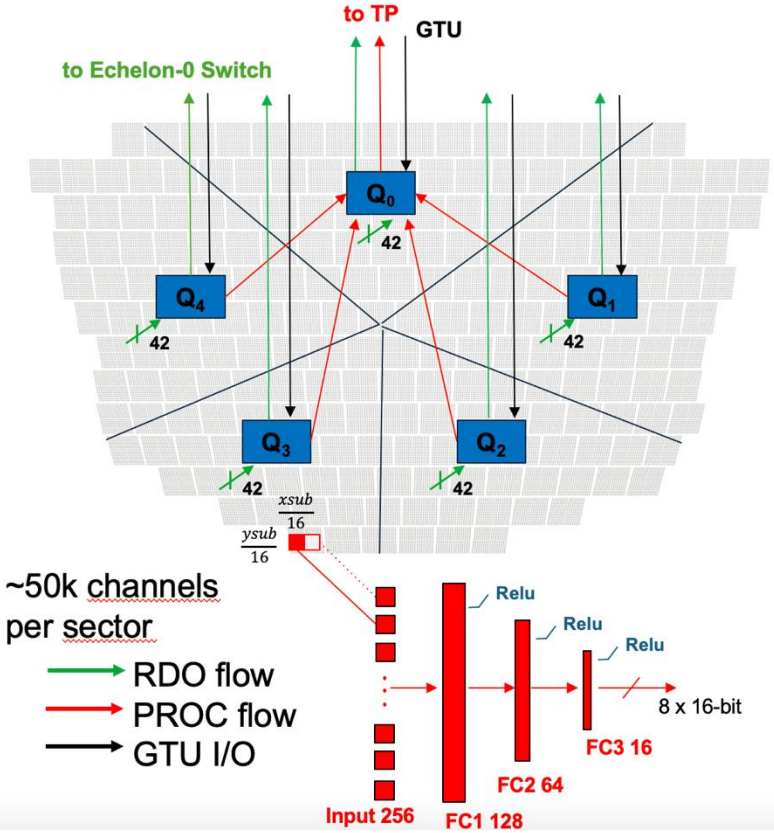
**A**bstract **P**rocessing **E**nvironment for **I**ntelligent **R**ead-**O**ut systems based on **N**eural networks

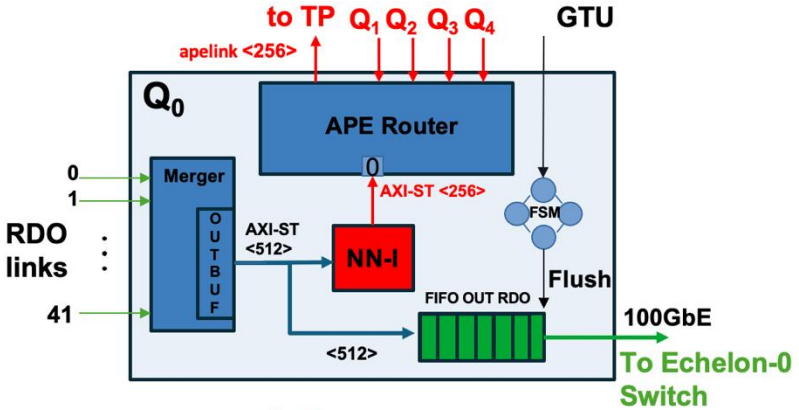
- Input **data streams** from several different channels (data sources, detectors subsectors) recombined through the processing layers using a **low-latency, modular and scalable network infrastructure**

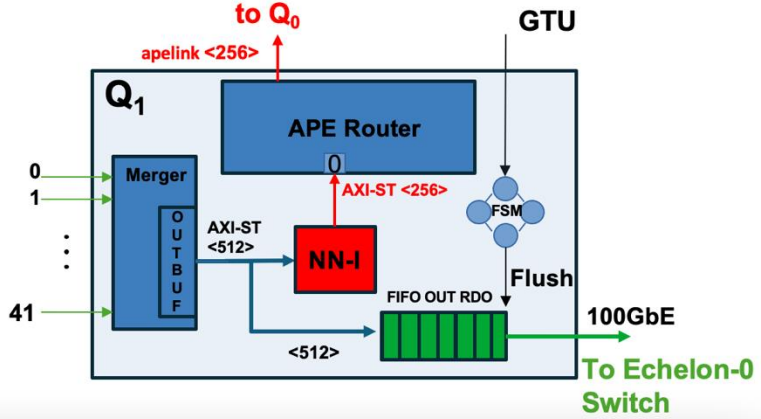



- More resource-demanding NN layers can be implemented in subsequent processing layers.
- Classification produced by the NN in last processing layer (e.g. pid) will be input for the **trigger processor/storage online data reduction stage for triggerless systems**.

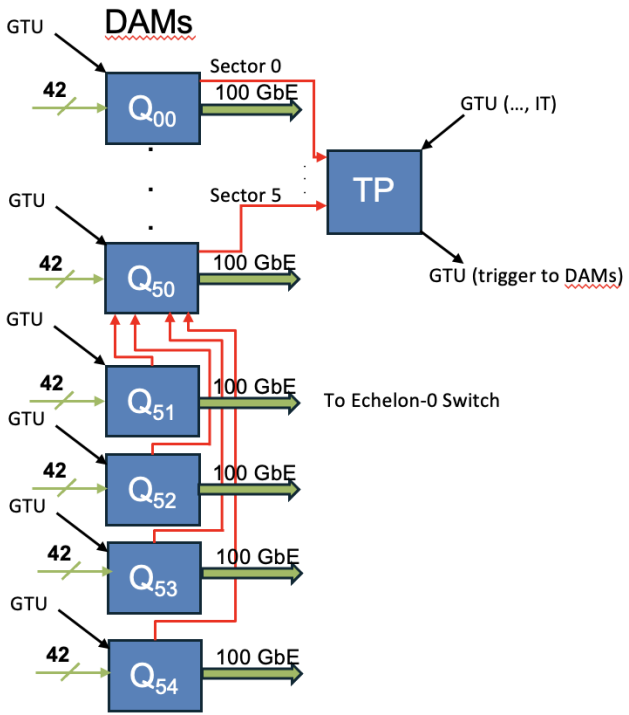
# dRICH Data Reduction Stage on FPGA: example deployment

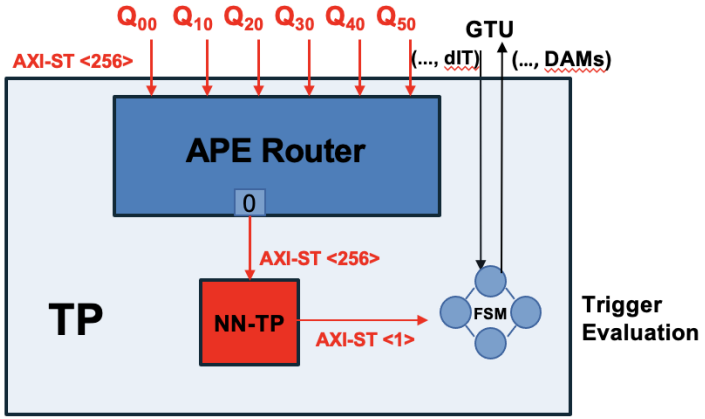


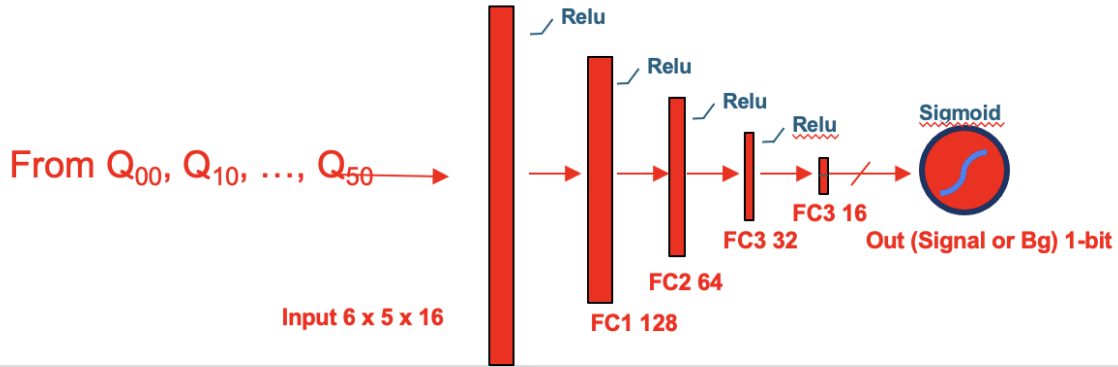



# dRICH Data Reduction Stage on FPGA: example deployment



# dRICH: Data reduction (features)

**Online Signal/Noise discrimination using ML:**

- Collecting datasets using data available from simulation campaigns

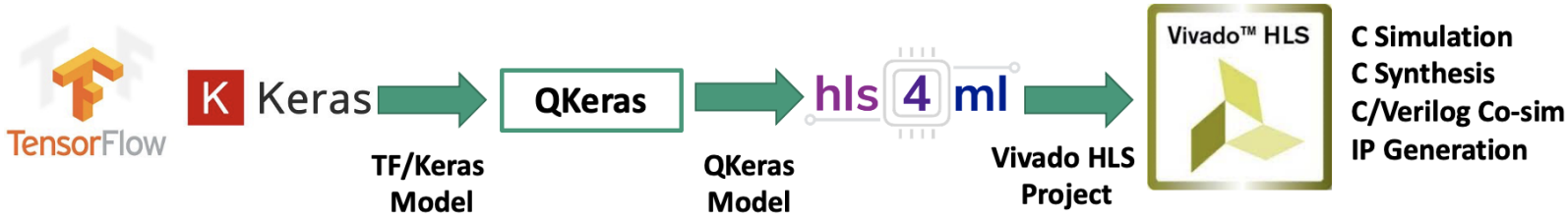
**Background:**

- e/p with beam pipe gas
- Synchrotron radiation (MC only, it would be useful to have it reconstructed)

- **Merged (i.e. the Signal)**:
  - physics signal + e/p with beam pipe gas background (full)

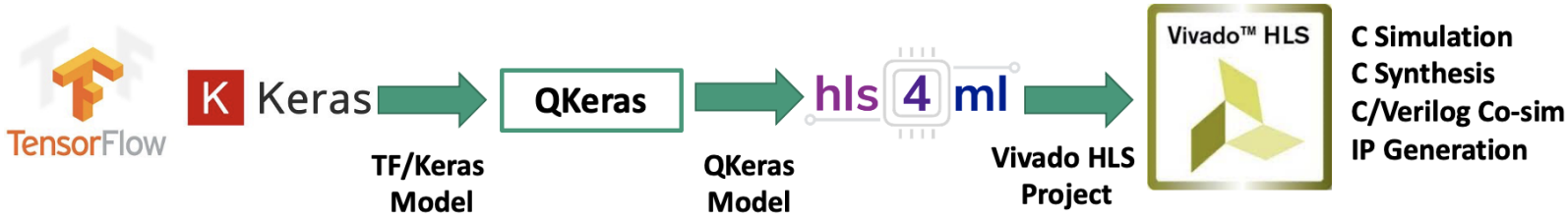
- **SiPM Noise:**
  - Dark current rate (DCR) modelled in the reconstruction stage (*recon.rb* eic-shell method)

# 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:

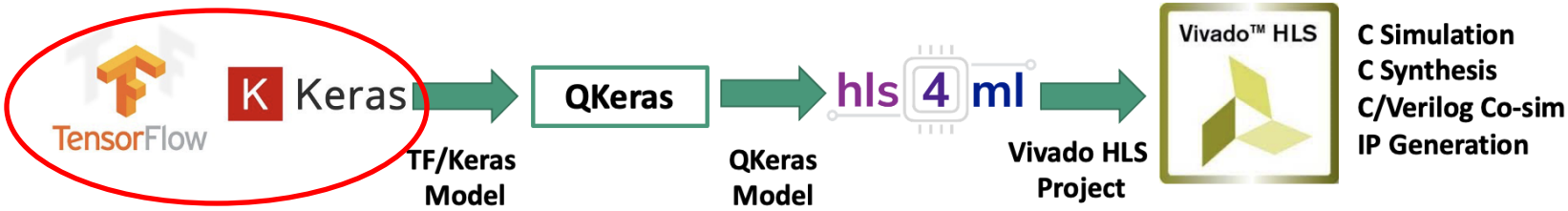
# dRICH Data reduction: How?
# ➔ Design and Implementation Workflow




**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.**

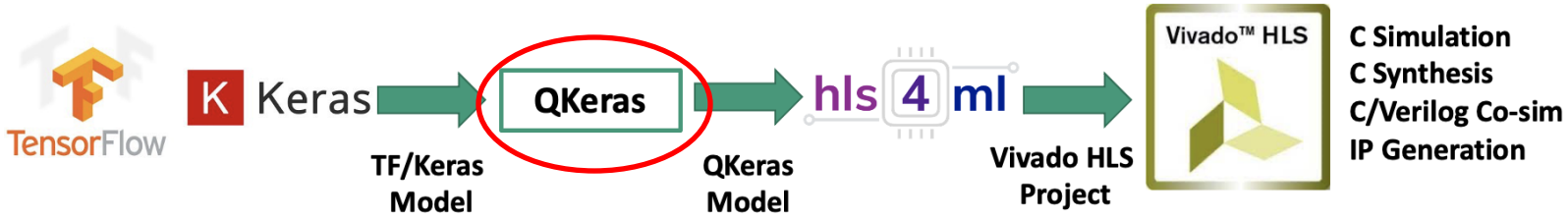
# dRICH Data reduction: How?
# ➔ Design and Implementation Workflow




**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,…).

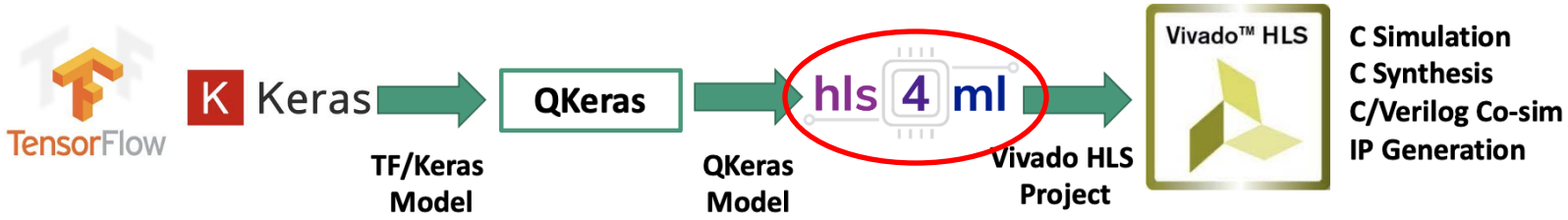
# dRICH Data reduction: How?
# ➔ Design and Implementation Workflow




**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.

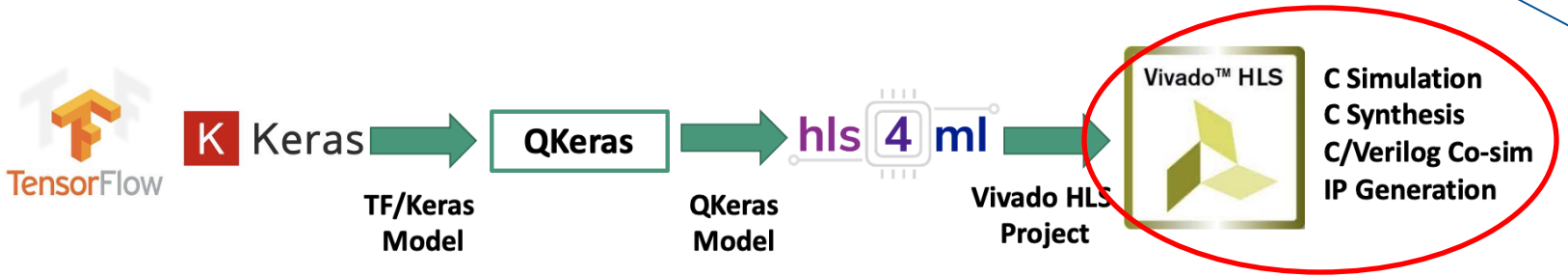
# dRICH Data reduction: How?
# ➔ Design and Implementation Workflow




**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.

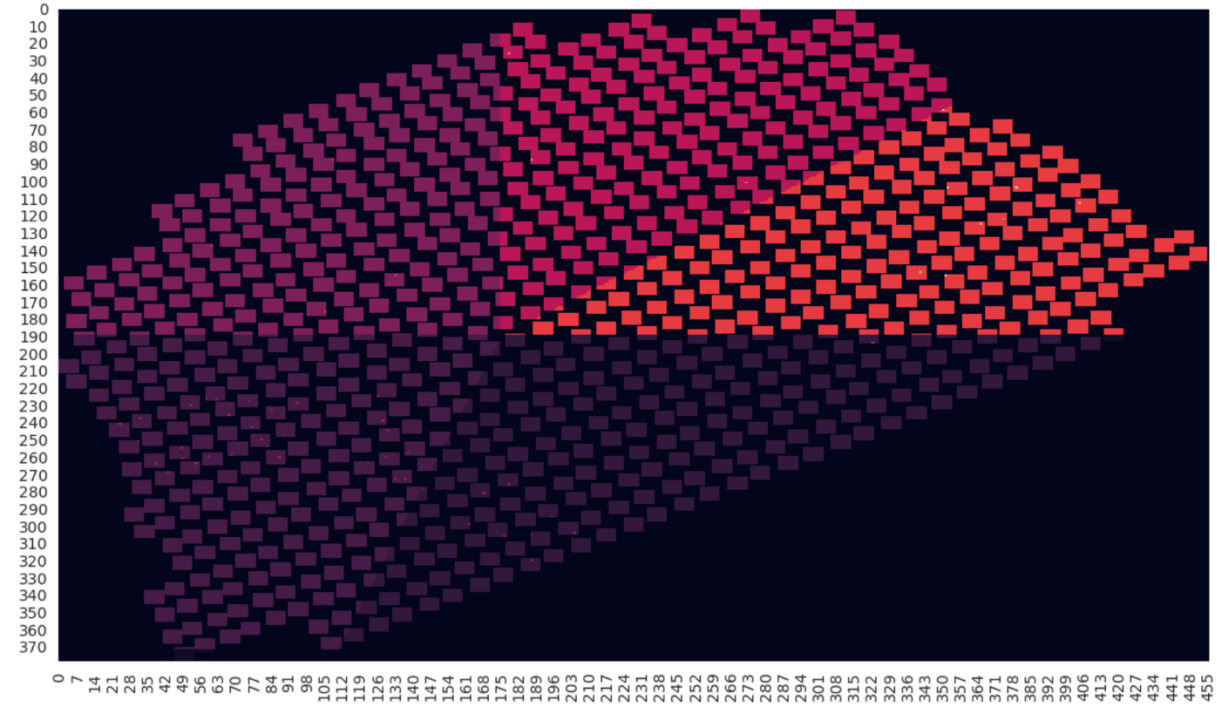
# dRICH Data reduction: How?
# ➔ Design and Implementation Workflow




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

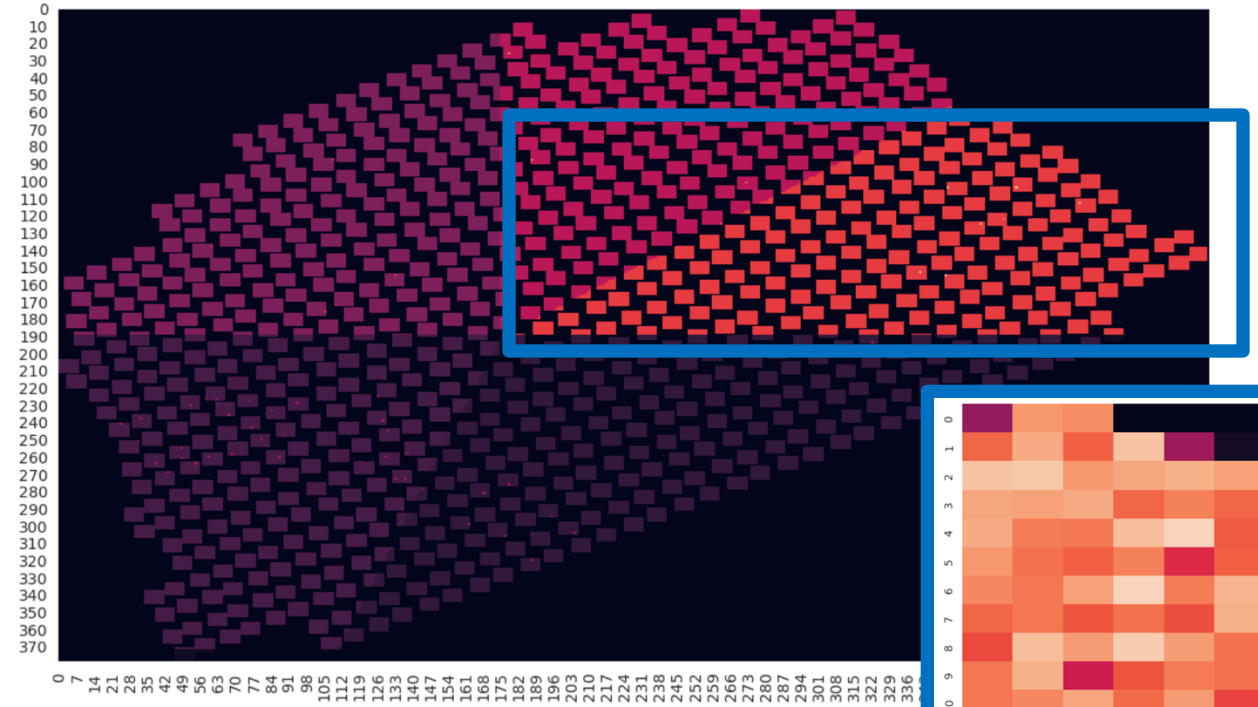
- **Vivado HLS ➔** co-simulation for verification of performance (experimented very good agreement with QKeras Model)

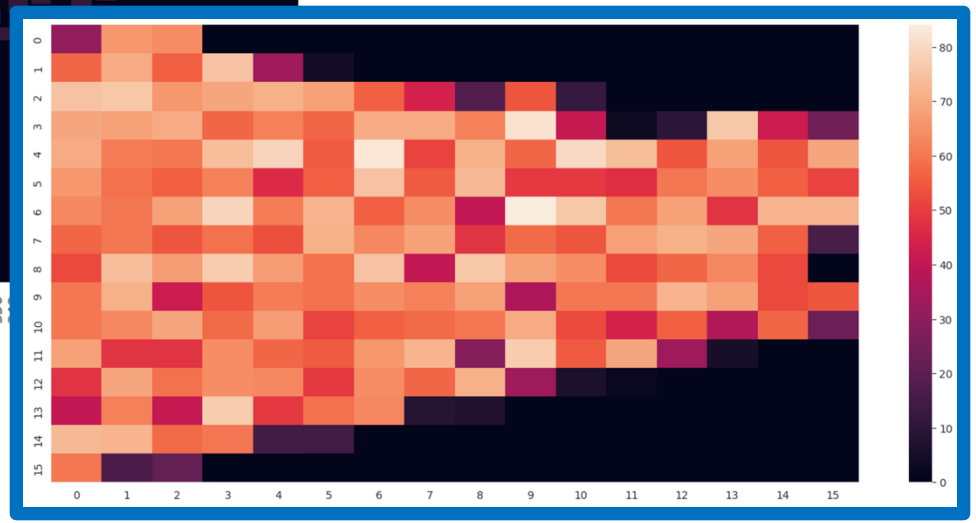
# dRICH: Data reduction ➔ Subsectors



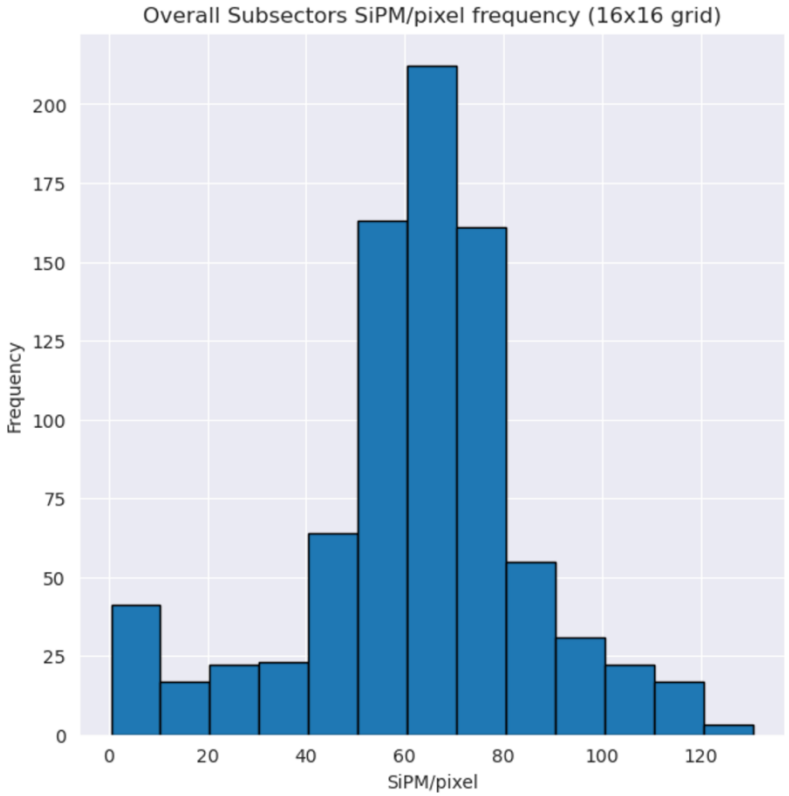

This is an example of a possible division of a dRICH sector into **5 subsectors ➔ To feed the NN input layer,** each subsector readout information should be converted to a **16x16 grid ➔ 256 inputs**

# dRICH: Data reduction ➔ Subsectors

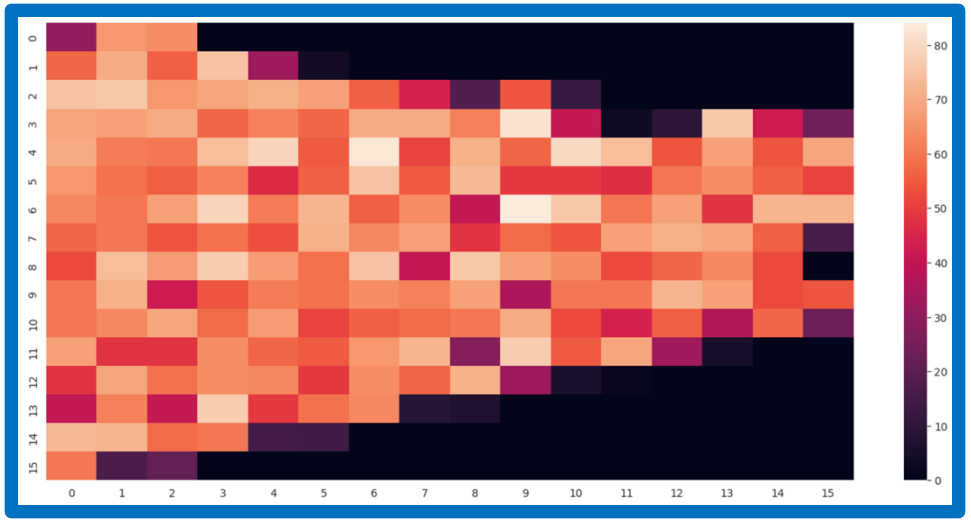




# dRICH: Data reduction ➔ Grid Definition

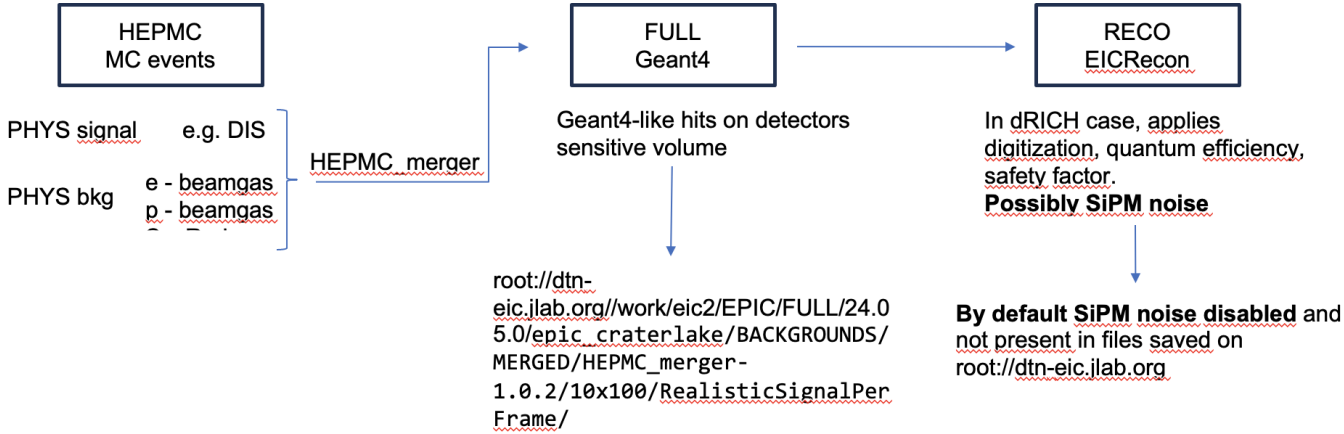



- A single grid pixel sums up informations of **~70 SiPMs**
- Edge subsectors  pixels contain less infomations ➔ this feature can be learnt by the NN



# dRICH Data reduction ➔ Dataset

**EPIC simulation pipeline:**



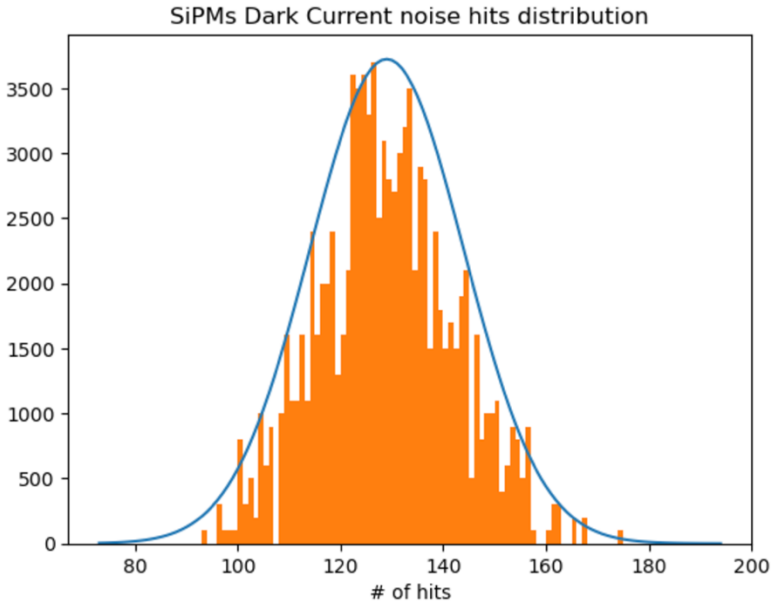

1. Run dRICH EICrecon algorithms on available FULL root files ourselves, one time with noise enabled, one time without it
   ➔ Easy, just use **drich-dev/recon.rb with nois configs.**
   BUT currently only few FULL files are available (29), corresponding to about 7k events.

2. Use RECO files, more easily available on server (much smaller) but have to add noise ourselves (current model seems to be just white noise). Still, currently only about 40k events are saved on server, summing different months of simulation campaigns

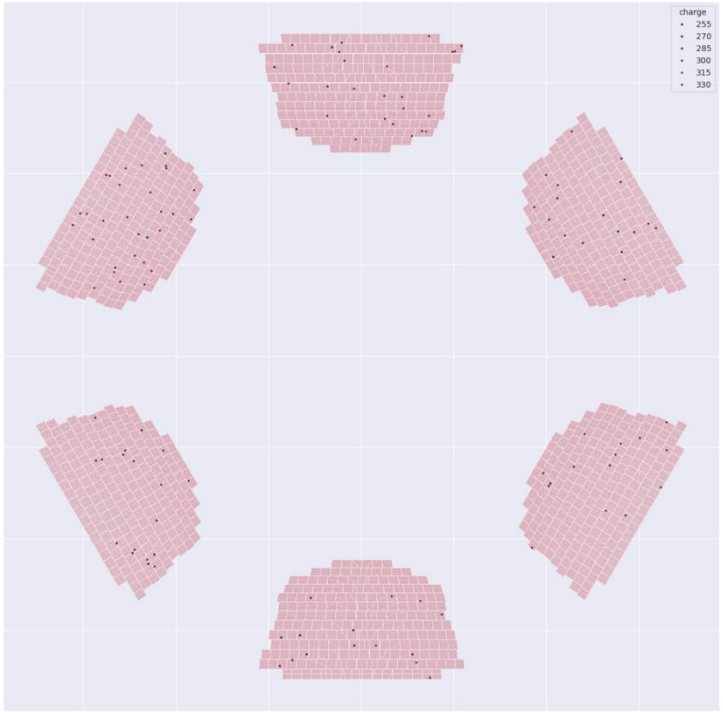
3. **Run the entire simulation pipeline ourselves, starting from HEPMC files.**

# dRICH Data reduction: Input Data (Features Definition)

**Gaussian** dark current SiPM **noise hits distribution**
(obtained by modifying directly EICRecon config files ➔ default fixed number of 129 noise hits)

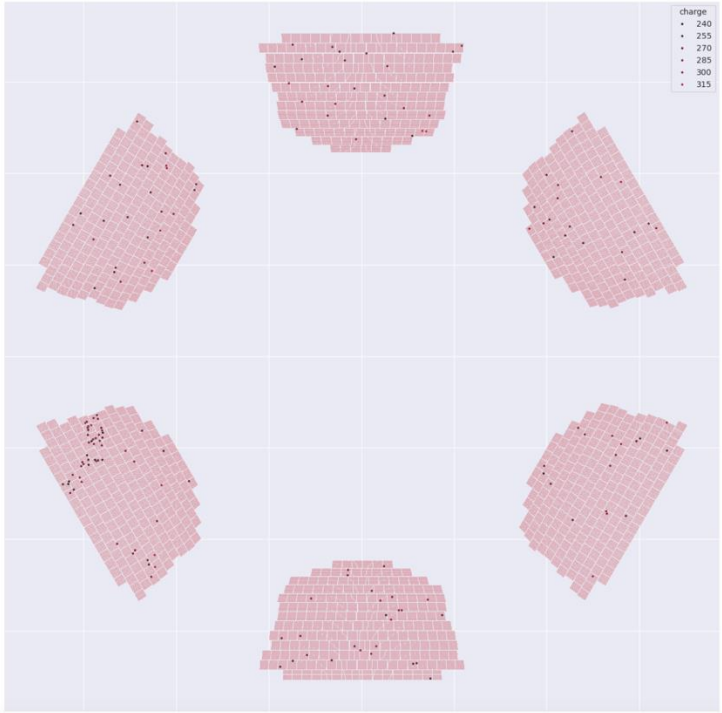



➤ **Noise Only**

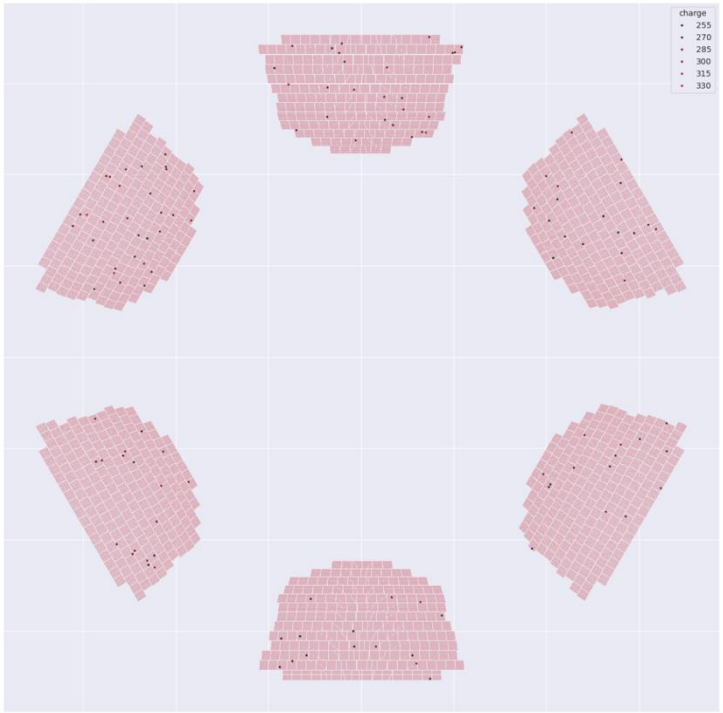


# dRICH Data reduction: Input Data (Features Definition)

- **Signal+Background+Noise**

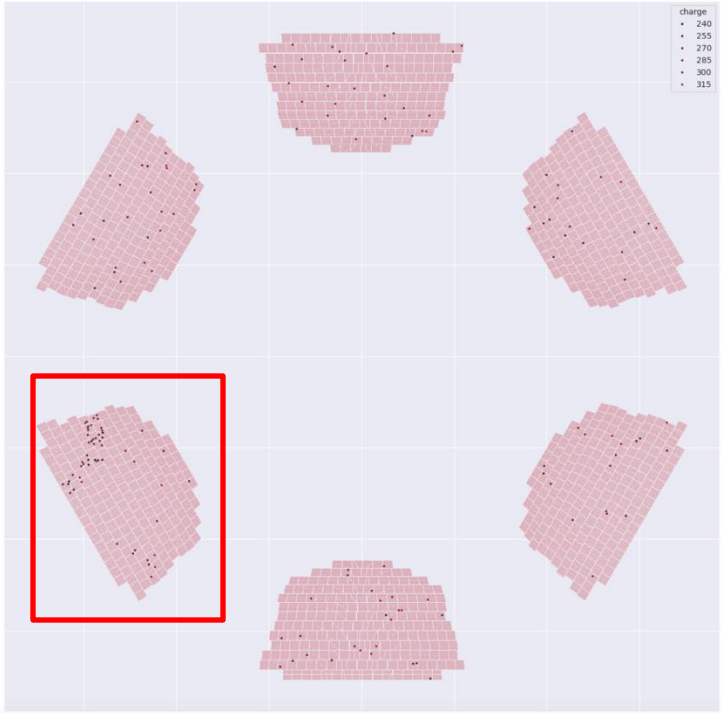



- **Noise Only**

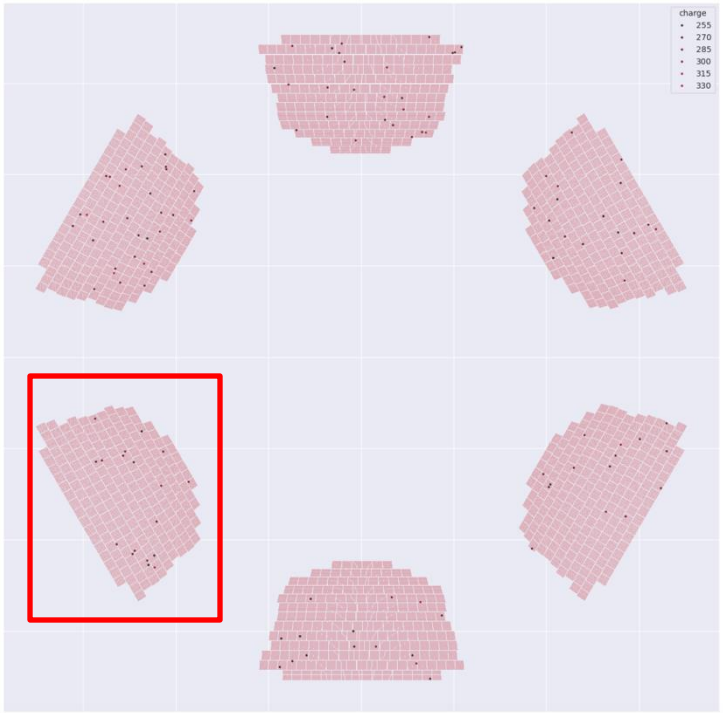


# dRICH Data reduction: Input Data (Features Definition)

- **Signal+Background+Noise**

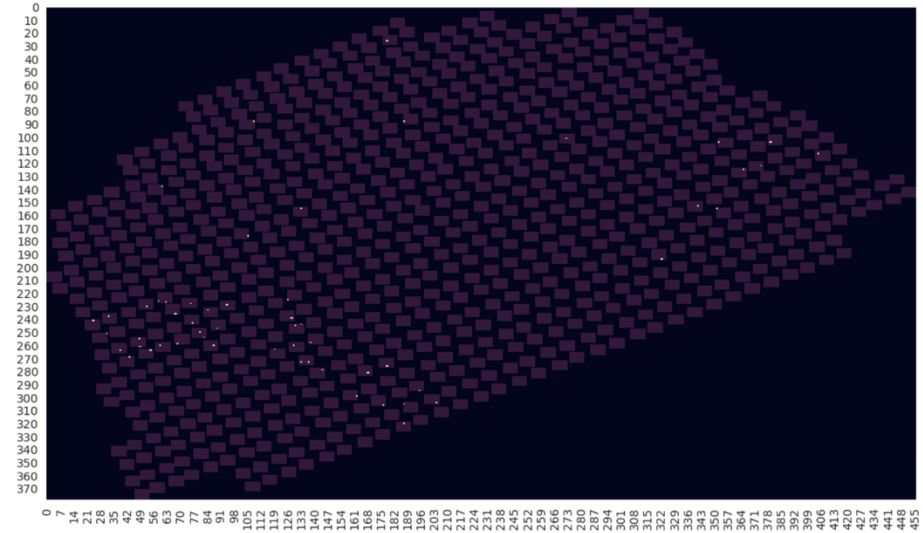



- **Noise Only**

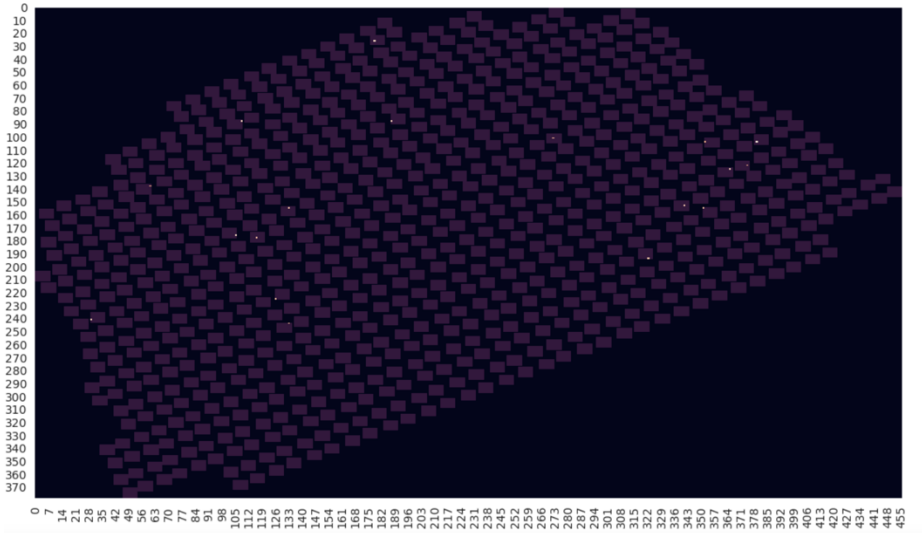


# dRICH Data reduction: Input Data

- Signal+Background+Noise

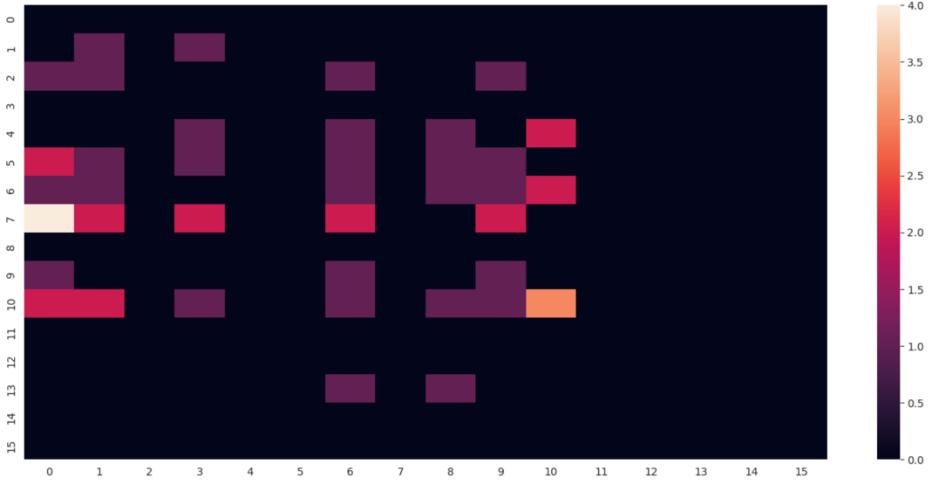


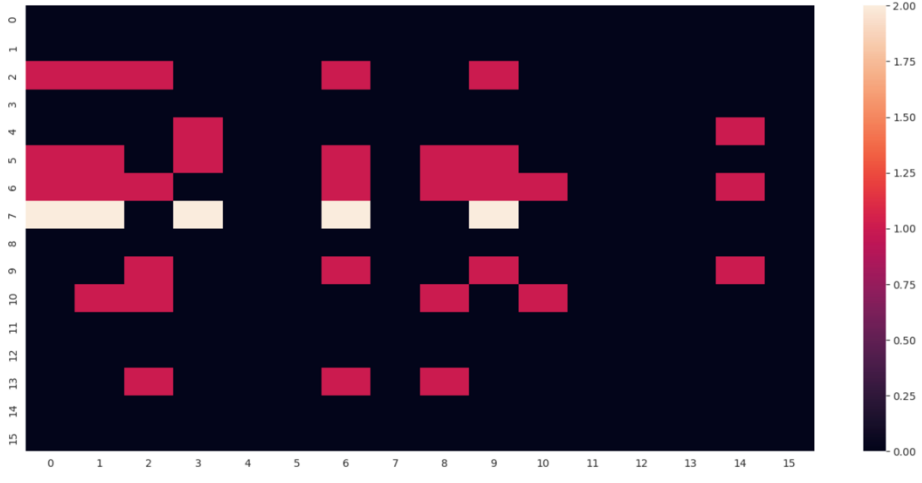
- Noise Only



# dRICH Data reduction: Input Grids

- Signal+Background+Noise
- Noise Only



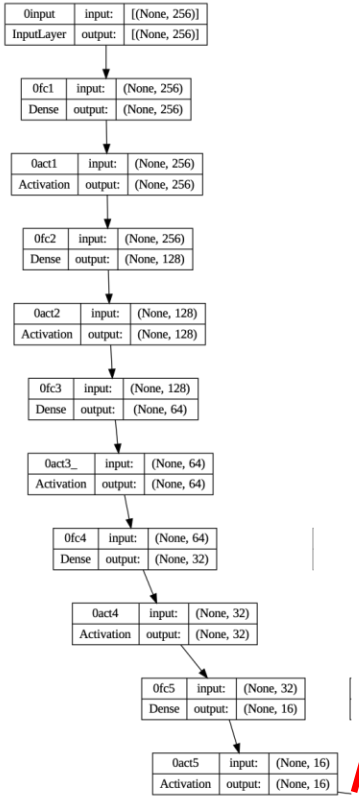




**16x16 Grid ➔ 256 input NN neurons**

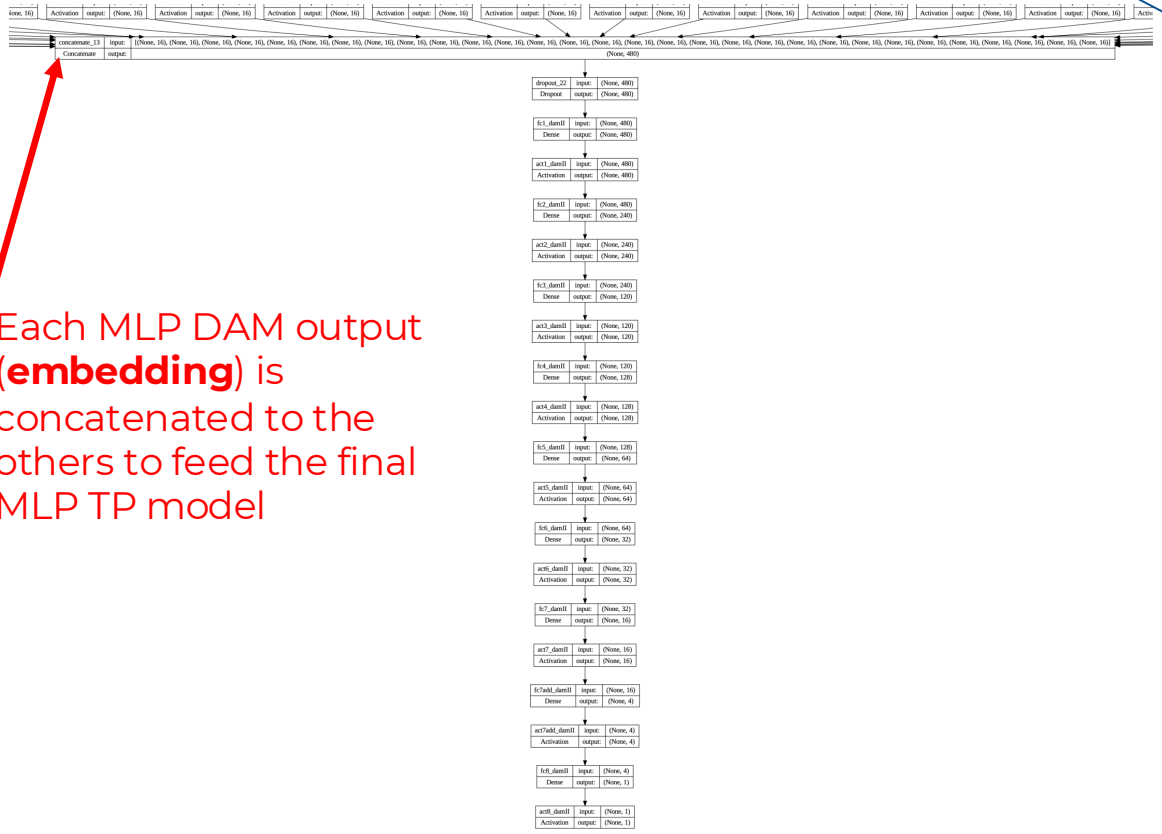
# dRICH Data reduction: Tensorflow-Keras Model definition

- To be coherent with the hardware design composition of the proposed system, we trained **30** (# of subsectors x #number of sectors) **concatenated MLP networks** into a single MLP final model

# dRICH Data reduction: Tensorflow-Keras Model definition



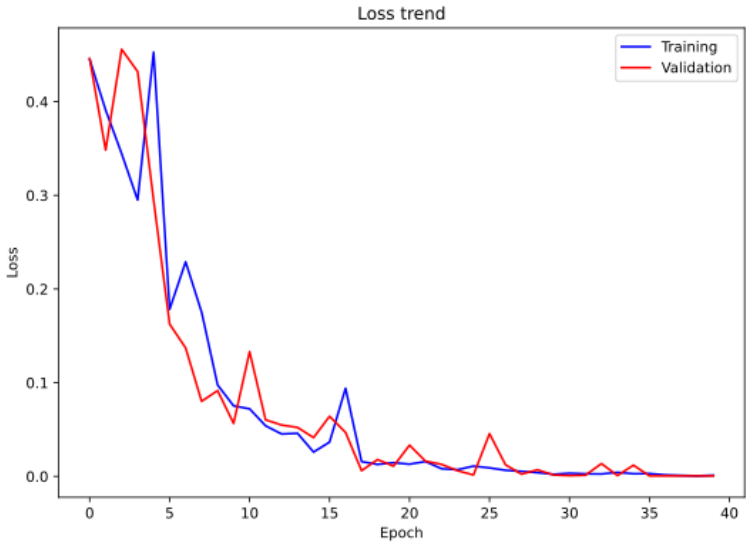

Each MLP DAM output (**embedding**) is concatenated to the others to feed the final MLP TP model



# dRICH Data reduction: Tensorflow training and evaluation

- We trained (offline) the 30 MLP DAM models concatenated to the single MLP TP model by using 100k Signal+Background+Noise and 100k Noise Only events ➔ **200k balanced dataset** (90% training set, 10% validation set)
- We minimize a typical Binary CrossEntropy loss function in 40 epochs, **backpropagating** the result to all the input models

➔ in this way, trained 30 MLP DAM models result are **uncorrelated**, coherently with the target design in which each susbector NN is blind wrt to the others NN

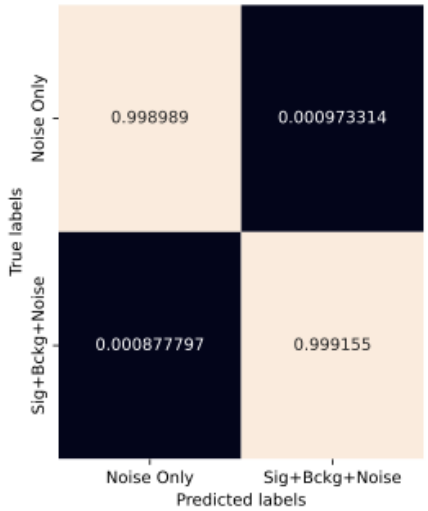


# dRICH Data reduction: Tensorflow training and evaluation

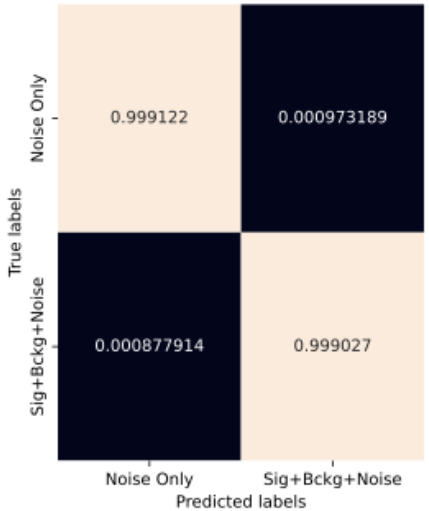
- We trained (offline) the 30 MLP DAM models concatenated to the single MLP TP model by using 100k Signal+Background+Noise and 100k Noise Only events ➔ **200k balanced dataset** (90% training set, 10% validation set)
- We minimize a typical Binary CrossEntropy loss function in 40 epochs, **backpropagating** the result to all the input models

➔ Overall model accuracy seems to scale with the dataset lenght. However, with these events, we obtained **a ~99,9% accurate «noise only» classifier**

**Purity**




**Accuracy**

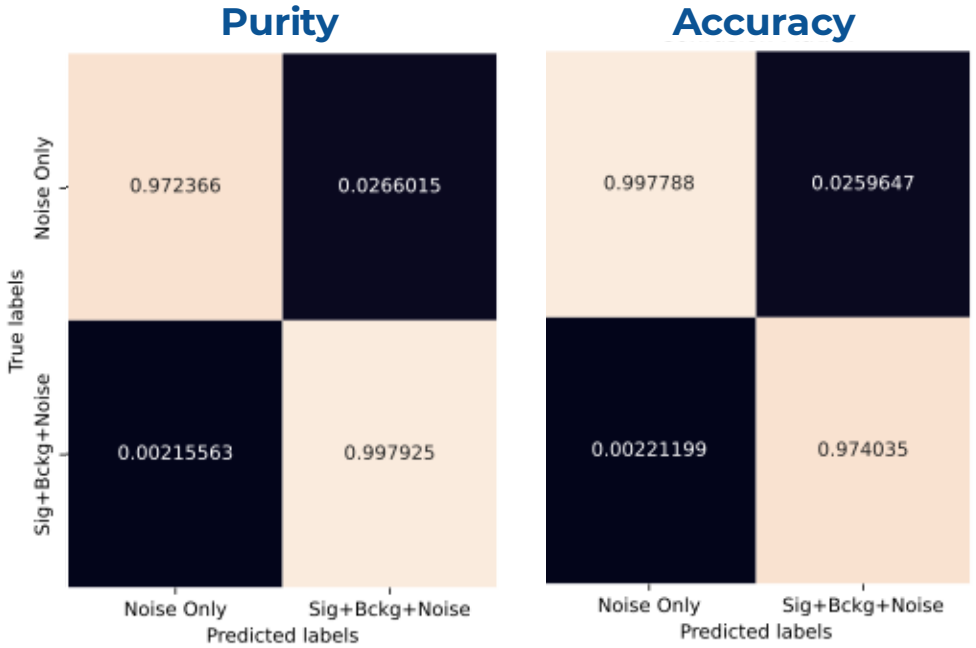


# dRICH Data reduction: Qkeras quantization step

- Starting from previous models weights, we trained (offline) 30 MLP DAM **quantized** models concatenated to a single MLP TP **quantized** model by using 100k Signal+Background+Noise and 100k Noise Only events ➔ **200k balanced dataset** (90% training set, 10% validation set)

➔Through **quantization**, we defined**:**
**quantized fixed point<16,6> inputs**
**quantized fixed point<8,1> weights**
**quantized fixed point<8,1> biases**

Obtaining a
**~97.2% accurate «noise only» classifier**



# dRICH Data reduction: HLS4ML ➔ (FPGA) HW Synthesis

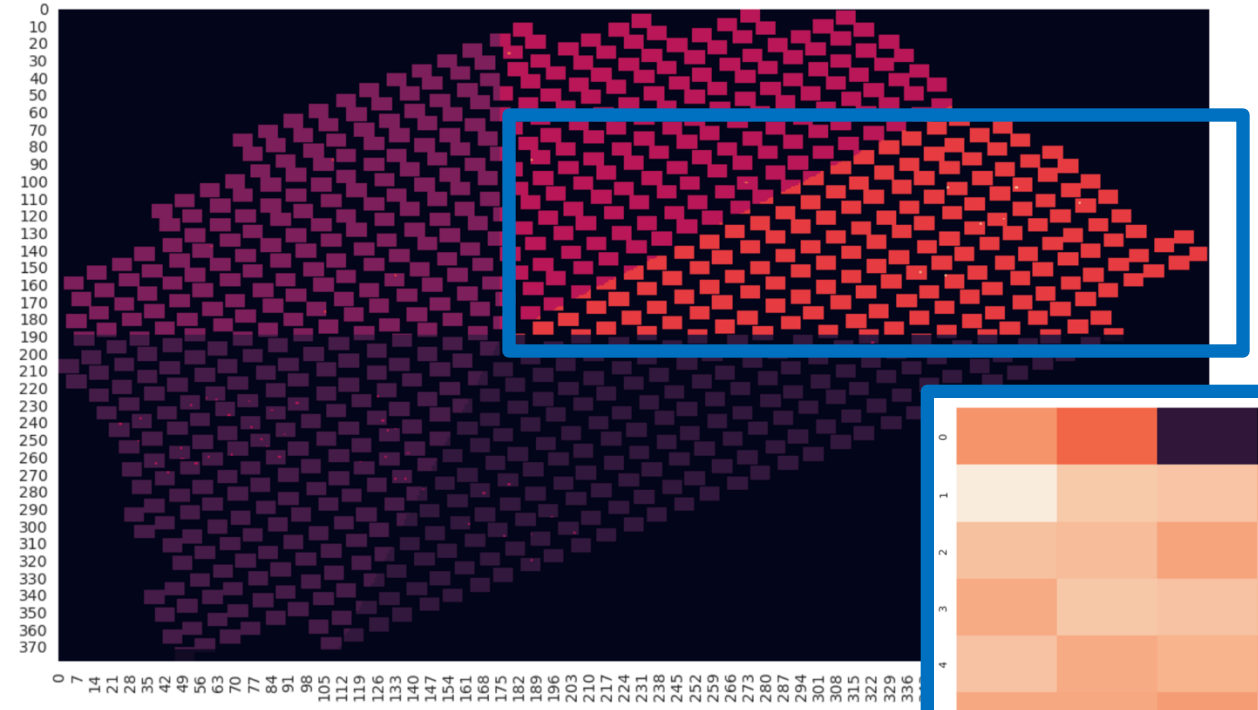
**Unluckily** (at this first stage of development), when moving to FPGA by using the **HLS4ML framework**, we were NOT able to produce a MLP DAM model able to cope with the **experiment expected streaming data rate**
➔ too many computation resources to unroll (in particular due to the matrix multiplication between the first two 256 and 128 neurons layers)

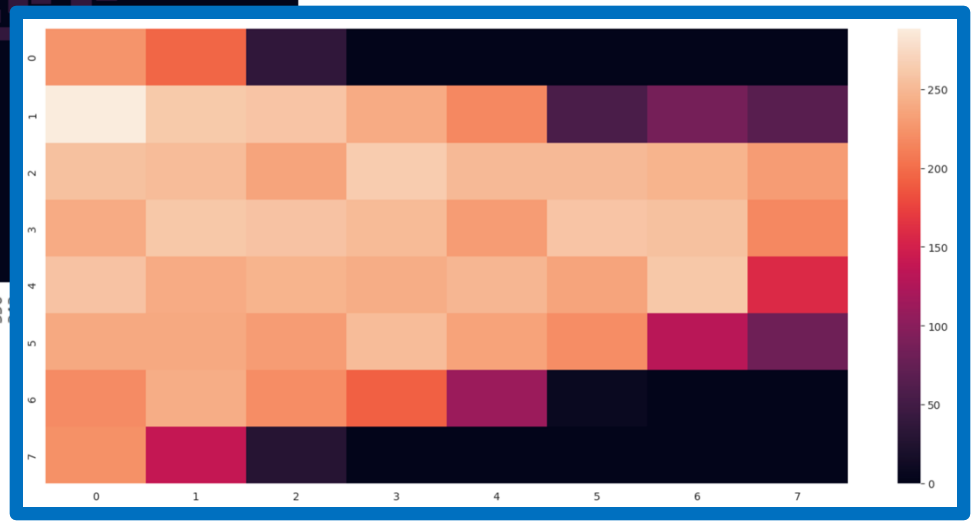
➔ To correctly synthetize the model at 200 MHz of operational clock, we used a **REUSE FACTOR = 64**, obtaining an instantiation interval **II = 68 clock cycles**

➔ **Throughput = 2,94 MHz (<< 100 MHz)**

```
+ Timing:
    * Summary:
    +--------+---------+----------+------------+
    |  Clock |  Target | Estimated| Uncertainty|
    +--------+---------+----------+------------+
    |ap_clk  | 5.00 ns | 4.228 ns |   0.62 ns  |
    +--------+---------+----------+------------+

+ Latency:
    * Summary:
    +---------+---------+----------+----------+-----+-----+----------+
    |  Latency (cycles) |  Latency (absolute) |  Interval | Pipeline |
    |   min   |   max   |    min   |    max   | min | max |   Type   |
    +---------+---------+----------+----------+-----+-----+----------+
    |       73|       74| 0.365 us | 0.370 us |   67|   68| dataflow |
    +---------+---------+----------+----------+-----+-----+----------+
```

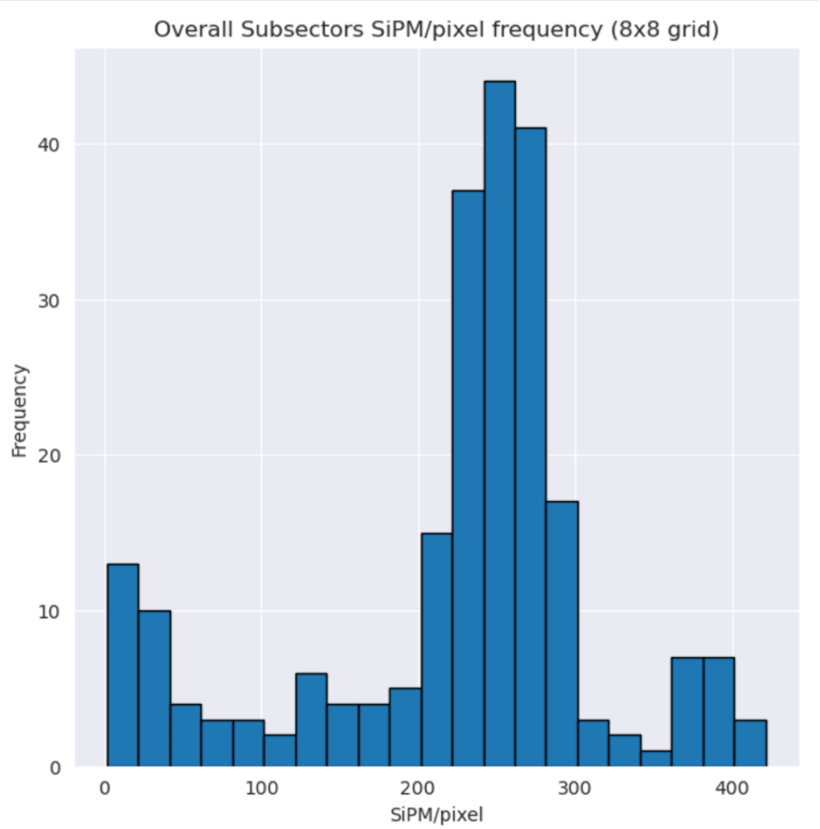
# dRICH: Data reduction ➔ Subsectors



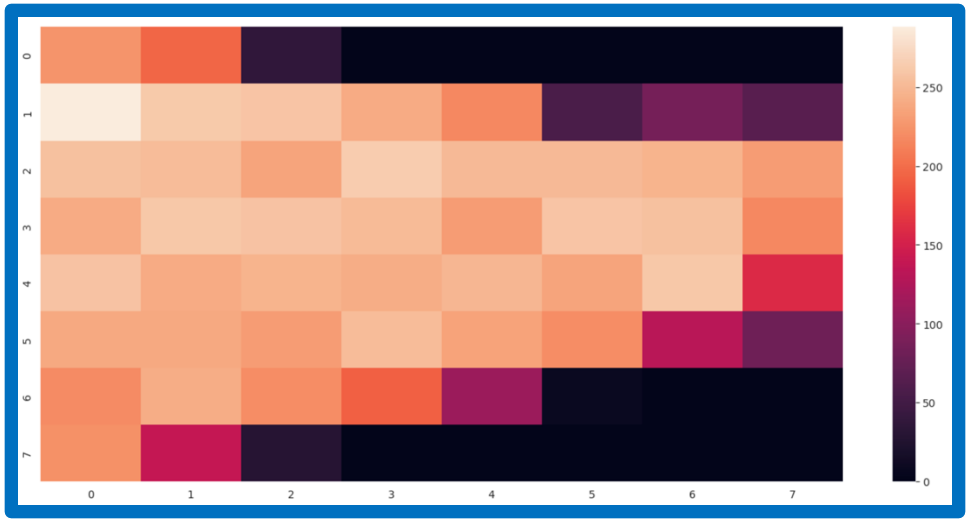


Let’s evaluate a **8x8 Grid** as input!

# dRICH: Data reduction ➔ Grid Definition

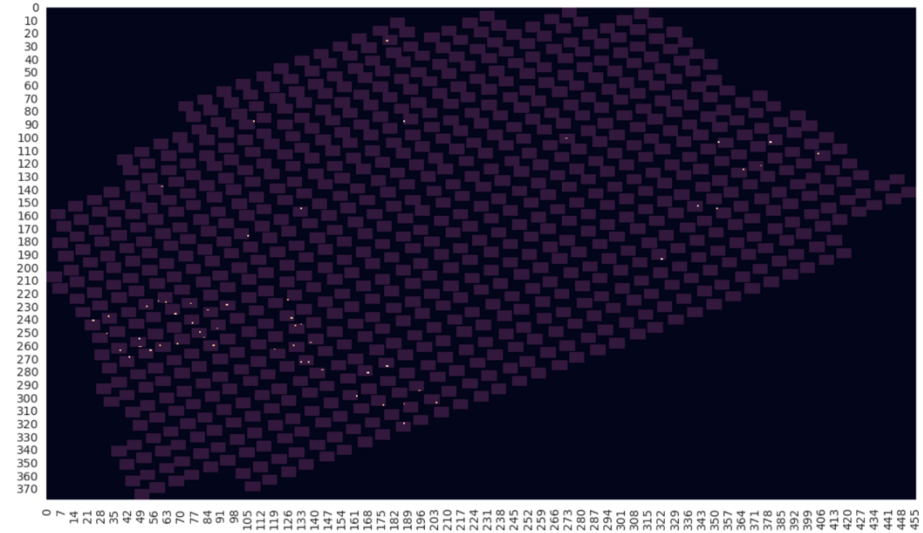



- A single grid pixel sums up informations of **~250 SiPMs** (wrt previous ~70 )
- Edge subsectors pixels contain less infomations ➔ this feature can be learnt by the NN (even in this case?)

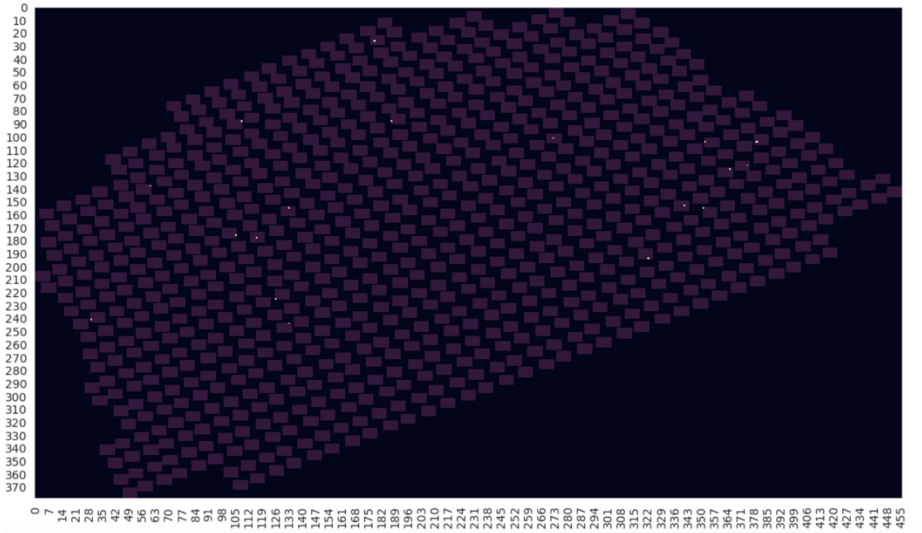


# dRICH Data reduction: Input Data

- Signal+Background+Noise

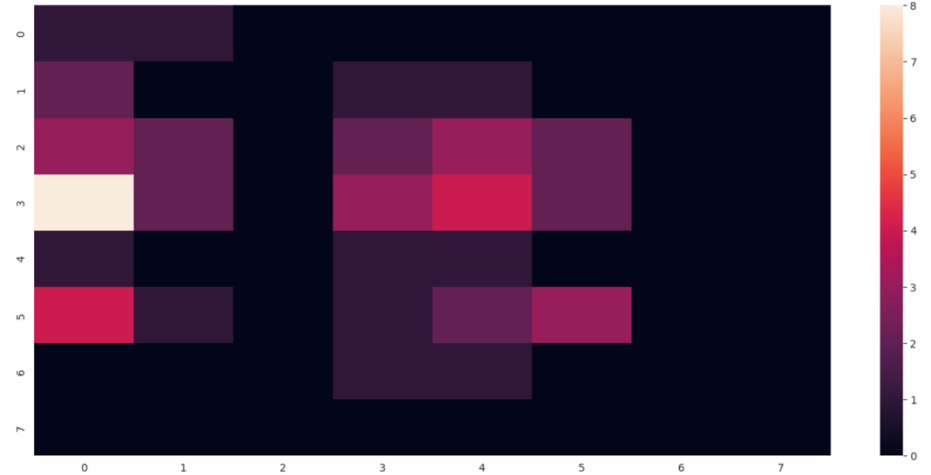


- «Solo» Noise

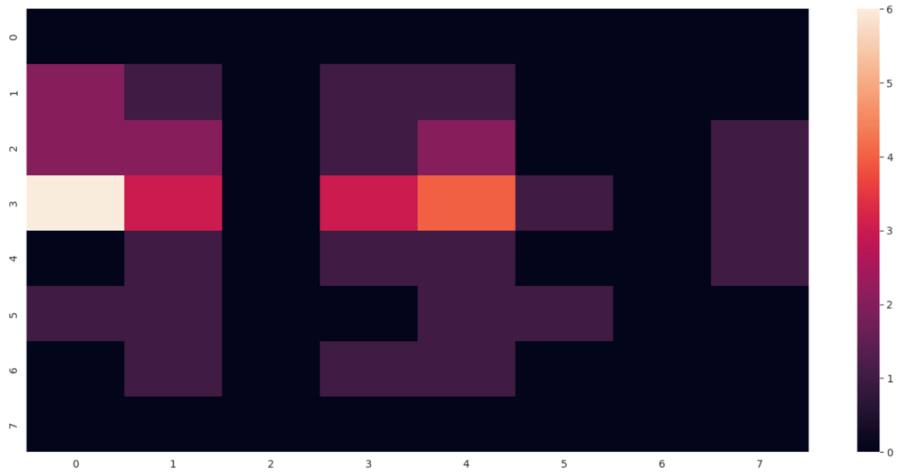


# dRICH Data reduction: 8x8 Grid Test

- Signal+Background+Noise

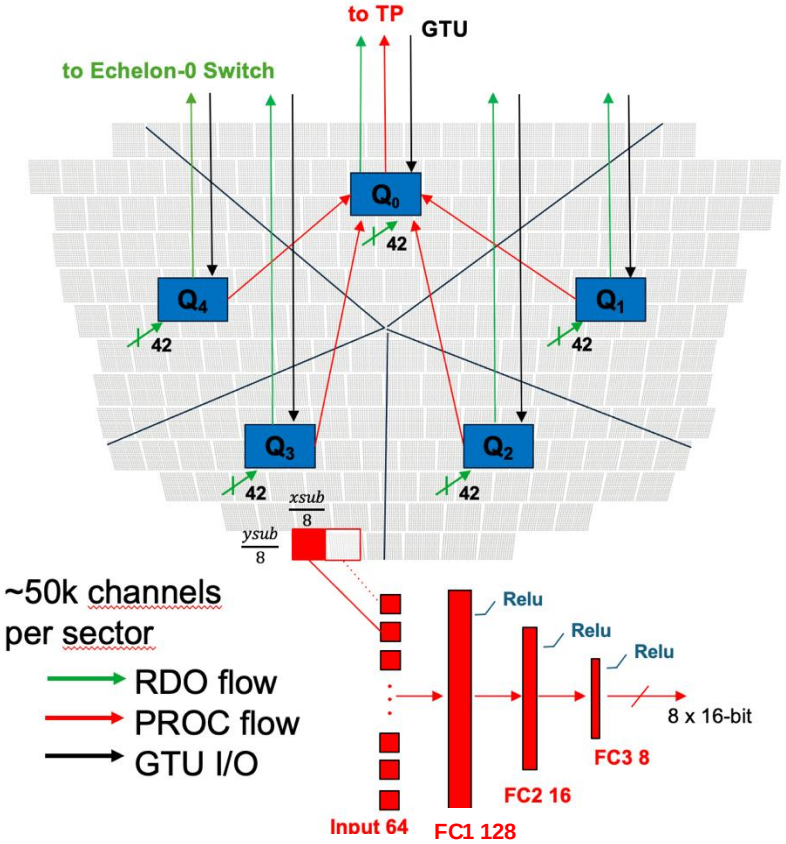


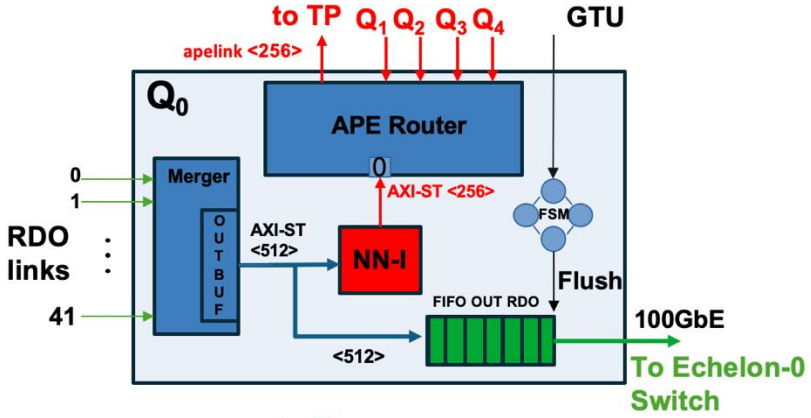
- «Solo» Noise

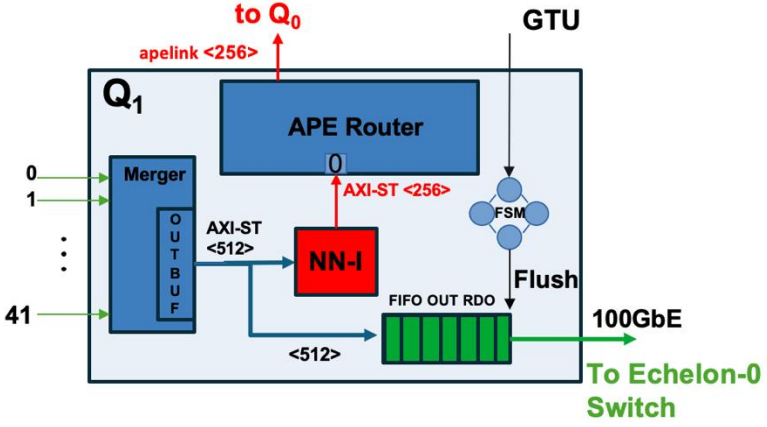


**8x8 Grid ➔ 64 input NN neurons**

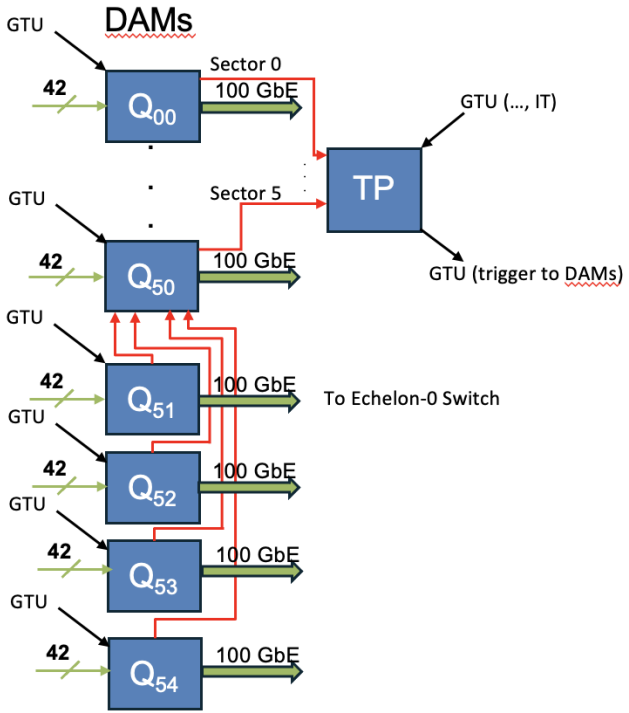
# dRICH Data Reduction Stage on FPGA: example deployment

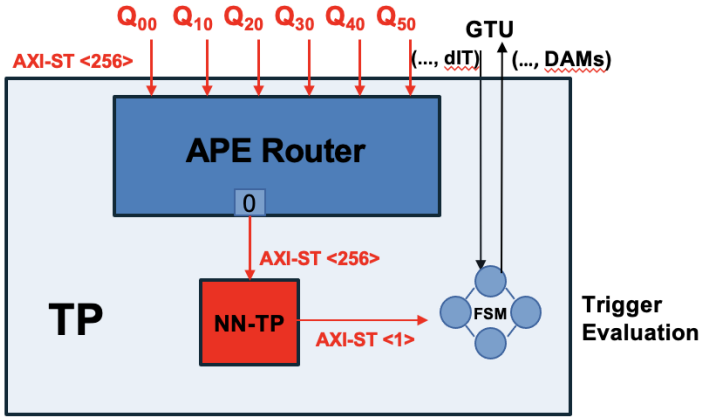


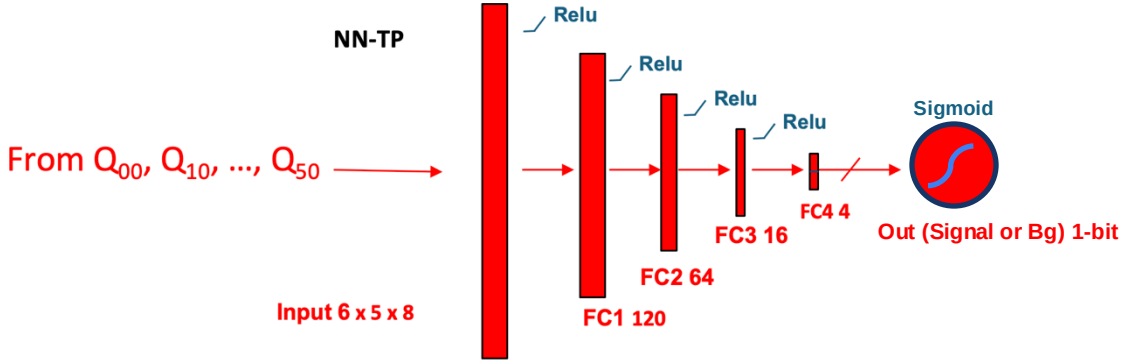



# dRICH Data Reduction Stage on FPGA: example deployment



# dRICH Data reduction: HLS4ML ➔ HW Synthesis 8x8 Grid

➔ To correctly synthetize the model at 200 MHz of operational clock, we used a **REUSE FACTOR = 1**, obtaining an instantiation interval **II = 5 clock cycles**

➔ **Throughput = 40MHz (< 100 MHz)**

```
+ Timing:
    * Summary:
    +--------+---------+----------+------------+
    |  Clock |  Target | Estimated| Uncertainty|
    +--------+---------+----------+------------+
    |ap_clk  | 5.00 ns | 4.374 ns |   0.62 ns  |
    +--------+---------+----------+------------+

+ Latency:
    * Summary:
    +---------+---------+-----------+-----------+-----+-----+----------+
    |  Latency (cycles) |   Latency (absolute)  |  Interval | Pipeline |
    |   min   |   max   |    min    |    max    | min | max |   Type   |
    +---------+---------+-----------+-----------+-----+-----+----------+
    |       14|       14| 70.000 ns | 70.000 ns |    5|    5| dataflow |
    +---------+---------+-----------+-----------+-----+-----+----------+
```

# dRICH Data reduction: HLS4ML ➔ HW Synthesis 8x8 Grid

➔ The possible **overhead** in the full II pipepline **introduced by the communication between DAMs and TP** will be considered **in further developments**

**STILL LOW, BUT PROMISING! (can be improved via modifying part of HLS4ML code)**

➔ To correctly synthetize the model at 200 MHz of operational clock, we used a **REUSE FACTOR = 1**, obtaining an instantiation interval **II = 5 clock cycles**

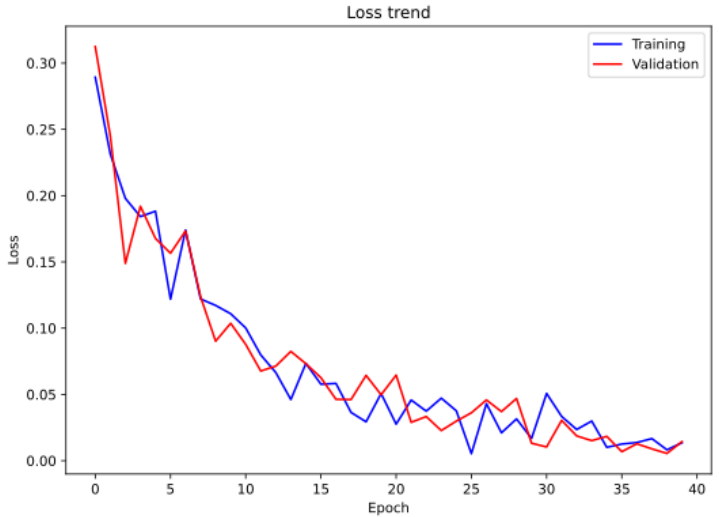
➔ **Throughput = 40MHz (< 100 MHz)**

```
+ Timing:
    * Summary:
    +--------+---------+----------+------------+
    |  Clock |  Target | Estimated| Uncertainty|
    +--------+---------+----------+------------+
    |ap_clk  | 5.00 ns | 4.374 ns |   0.62 ns  |
    +--------+---------+----------+------------+

+ Latency:
    * Summary:
    +---------+---------+-----------+-----------+-----+-----+----------+
    |  Latency (cycles) |   Latency (absolute)  |  Interval | Pipeline |
    |   min   |   max   |    min    |    max    | min | max |   Type   |
    +---------+---------+-----------+-----------+-----+-----+----------+
    |       14|       14| 70.000 ns | 70.000 ns |    5|    5| dataflow |
    +---------+---------+-----------+-----------+-----+-----+----------+
```

# dRICH Data reduction (8x8 Grid): Tensorflow training and evaluation

- We trained (offline) the 30 MLP DAM models concatenated to the single MLP TP model by using 100k Signal+Background+Noise and 100k Noise Only events ➔ **200k balanced dataset** (90% training set, 10% validation set)
- We minimize a typical Binary CrossEntropy loss function in 40 epoch, **backpropagating** the result to all the input models

➔ in this way, trained 30 MLP DAM models result are **uncorrelated**, coherently with the target design in which each susbector NN is blind wrt to the others NN

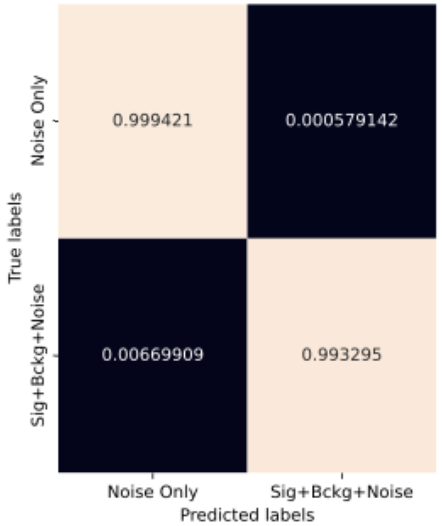


# dRICH Data reduction (8x8 Grid): Tensorflow training and evaluation

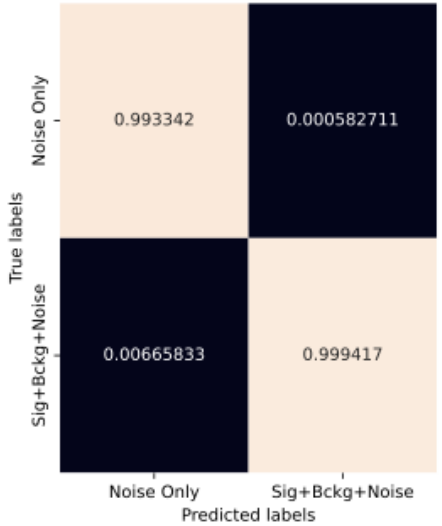
- We trained (offline) the 30 MLP DAM models concatenated to the single MLP TP model by using 100k Signal+Background+Noise and 100k Noise Only events ➔ **200k balanced dataset** (90% training set, 10% validation set)
- We minimize a typical Binary CrossEntropy loss function in 40 epoch, **backpropagating** the result to all the input models

➔ Overall model accuracy seems to scale with the dataset lenght. However, with these events, we manage to obtain **a ~99,3% accurate «noise only» classifier (**wrt 16x16 grid input ~99,9% accurate one**)**

**Purity**




**Accuracy**



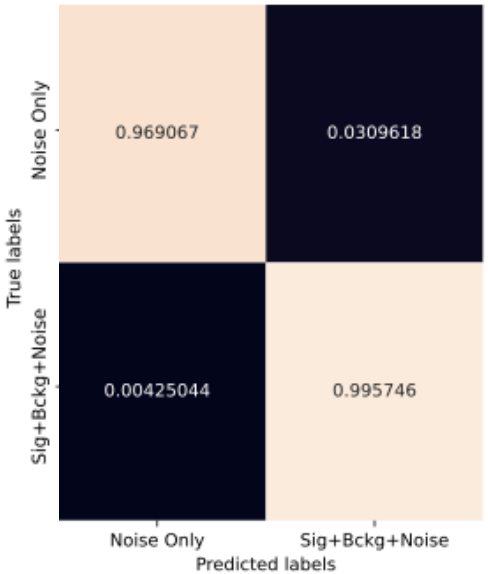
# dRICH Data reduction (8x8 Grid): Qkeras quantization step

- Starting from the previous model weights, we trained (offline) 30 MLP DAM **quantized** models concatenated to a single MLP TP **quantized** model by using 100k Signal+Background+Noise and 100k Noise Only events ➔ **200k balanced dataset** (90% training set, 10% validation set)

➔Through **quantization**, we defined**:**
**quantized fixed point<16,6> inputs**
**quantized fixed point<16,6> weights**
**quantized fixed point<16,6> biases**

Obtaining a **~96,9% accurate noise classifie**
**(**wrt 16x16 grid input ~97,2% accurate one**)**

**Purity**




**Accuracy**

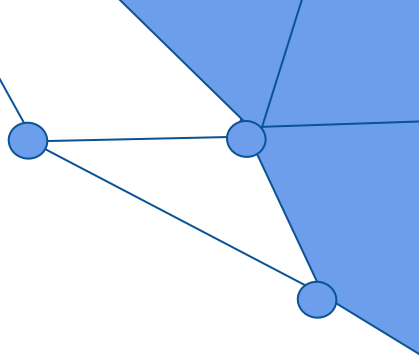
# Conclusions

- 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 MLP 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)

# Thanks for your attention!

**Contacts:**

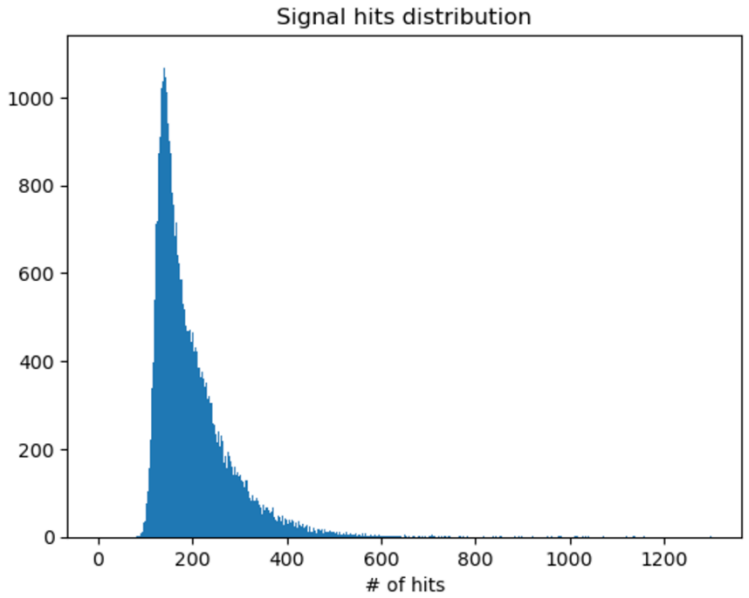
- cristian.rossi@roma1.infn.it
- alessandro.lonardo@roma1.infn.it



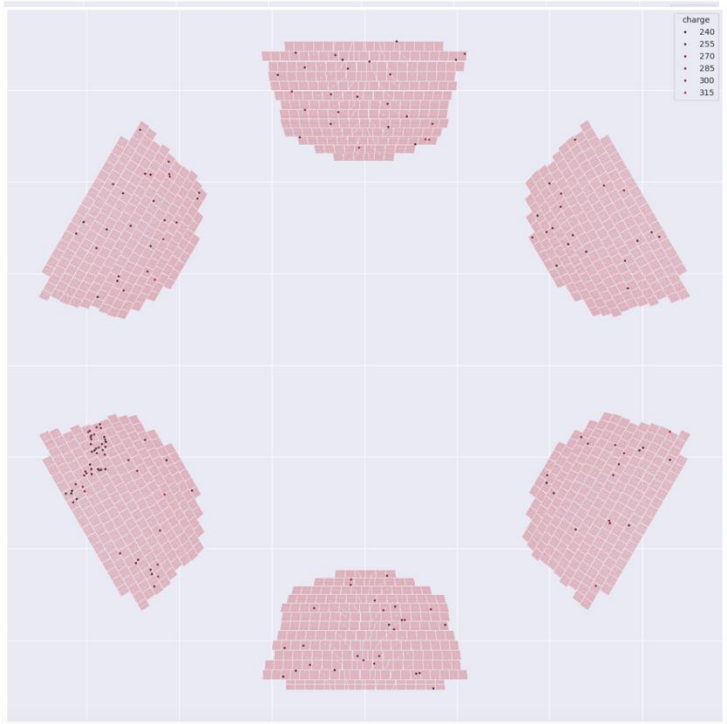
# BACKUP SLIDES

# dRICH Data reduction: Input Data (Features Definition)

**PHYS SIGNAL + PHYS BACKGROUNG +** dark current SiPM **NOISE hits distribution**



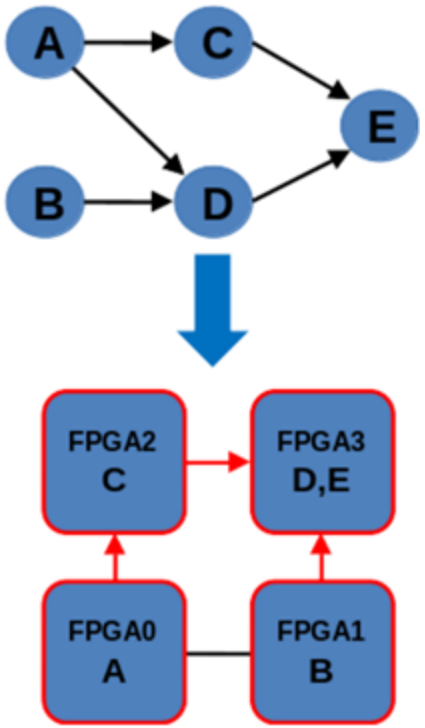

➢ **Sig + Bckg + Noise**



# 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()*

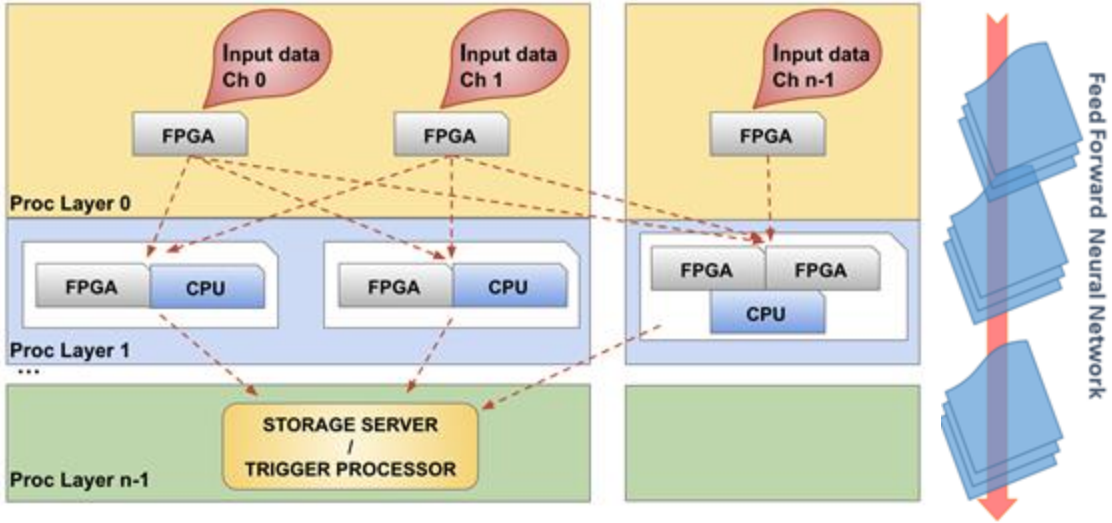



**APEIRON enables the scaling of Xilinx® Vitis High Level Synthesis applications on multiple FPGA interconnected by the INFN communication IP.**

# APEIRON for smart TDAQ Systems

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

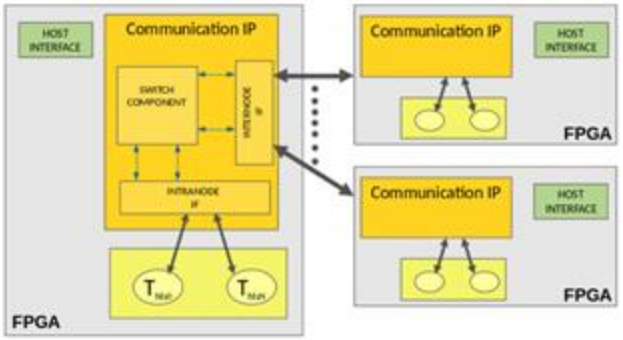
- Input **data streams** from several different channels (data sources, detectors/sub-detectors) recombined through the processing layers using a **low-latency, modular and scalable network infrastructure**

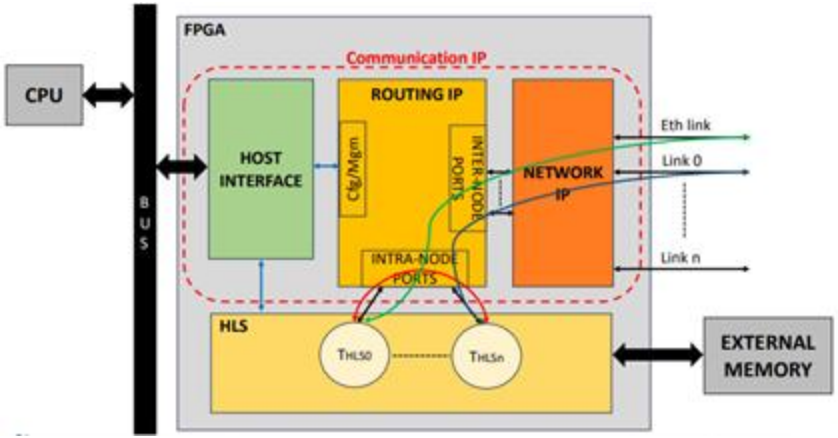



- More resource-demanding NN layers can be implemented in subsequent processing layers.
- Classification produced by the NN in last processing layer (e.g. pid) will be input for the **trigger processor/storage online data reduction stage for triggerless systems**.

# APEIRON building blocks:

- **INFN Communication IP**



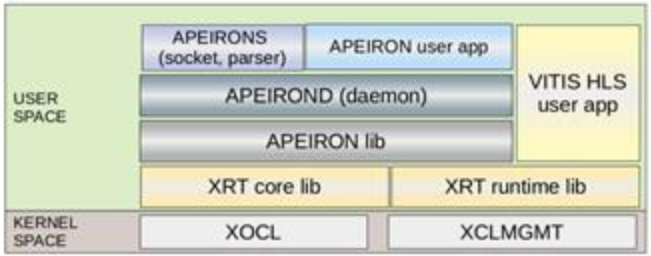



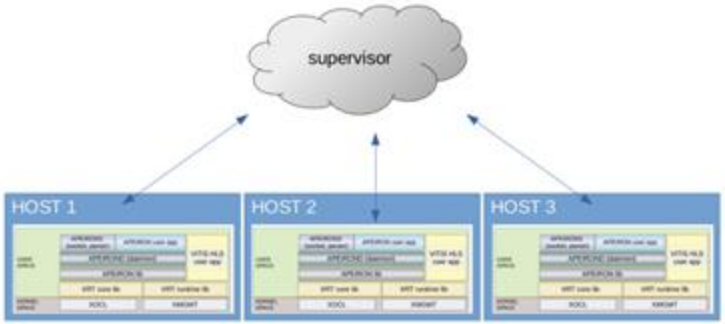

INFN is developing the IPs implementing a direct network that allows **low-latency data transfer** between processing tasks deployed on the same FPGA (**intra-node communication**) and on different FPGAs (inter-node communication)

- **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.

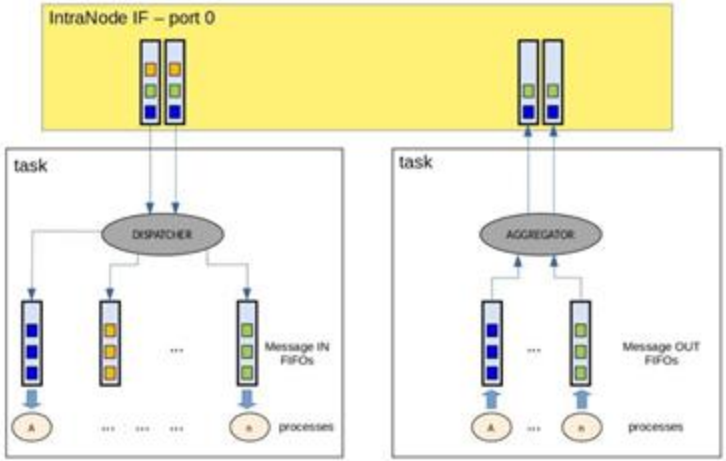
**Apeirond** is a persistent daemon used to manage multiple access request from user apps to the board.
Using the network socket exposed by apeirond modules, the **supervisor** can write commands and read status of the different instances of the APEIRON framework running in each node, allowing the user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

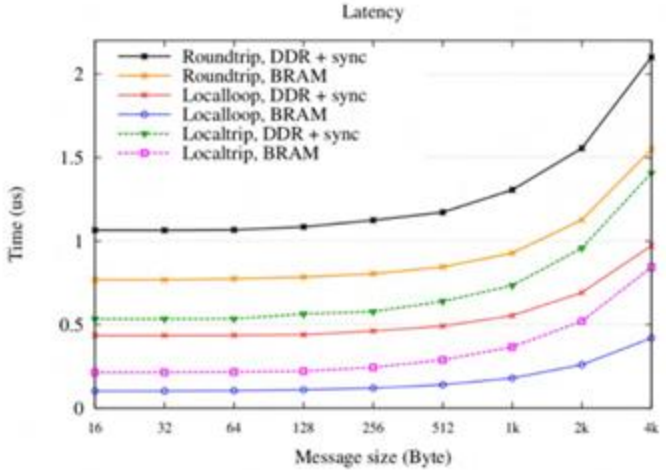
Adaptation toward/from IntraNode ports of the Routing IP is done by the automatically generated **Aggregator** and **Dispatcher** kernel templates.

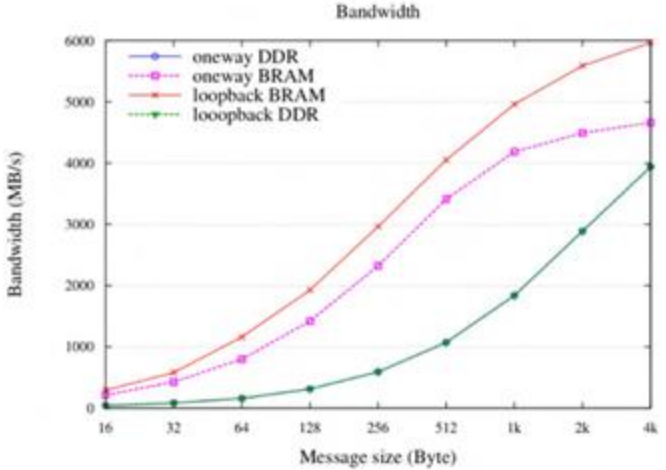
```
void example_task(
[list of optional kernel specific
parameters], message_stream_t
message_data_in[N_INPUT_CHANNELS],
message_stream_t
message_data_out[N_OUTPUT_CHANNELS])
```




```
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) | 213 | 533 |
| Inter-node (roundtrip) | 768 | 1065 |

**Bandwidth**

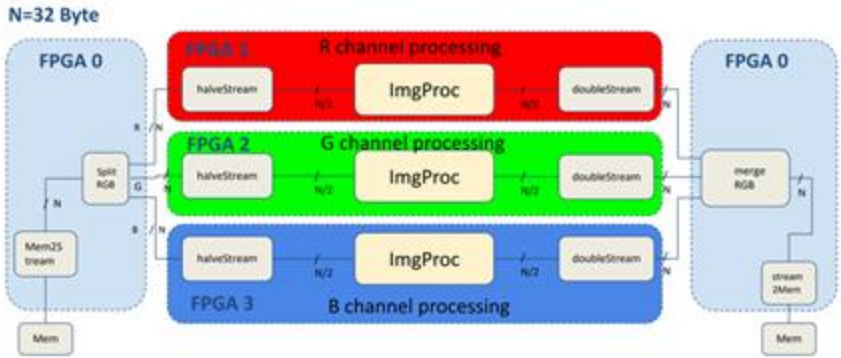
| | DDR+sync(MB/s) | BRAM(MB/s) |
|---|---|---|
| Intra-node (loopback) | 5967 | 3938 |

# APEIRON applications:

- **FIPLib-multiFPGA**

**F**PGA **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**

**F**PGA **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**

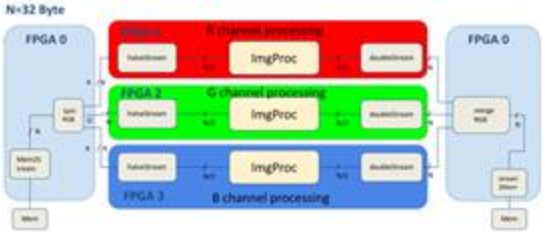
```
while (NbWordToTransfer > BUFFER_SIZE)
{
    if (phase){
            buffer2Stream(outStream, Buff1, BUFFER_SIZE);
            stream2Buffer(inStream, Buff2, BUFFER_SIZE);
    }
    else{
            buffer2Stream(outStream, Buff2, BUFFER_SIZE);
            stream2Buffer(inStream, Buff1, BUFFER_SIZE);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

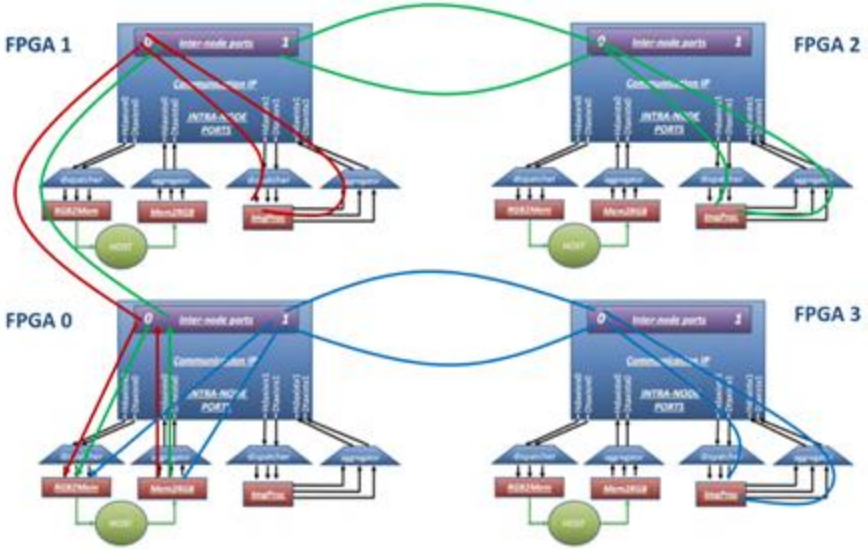
**MULTI-FPGA FPLib IMPLEMENTATION (APEIRON)**

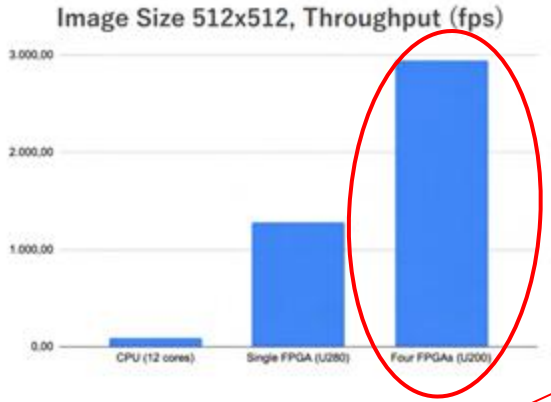
```
#include "ape_hls/hapecom.hpp"

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;
}
```

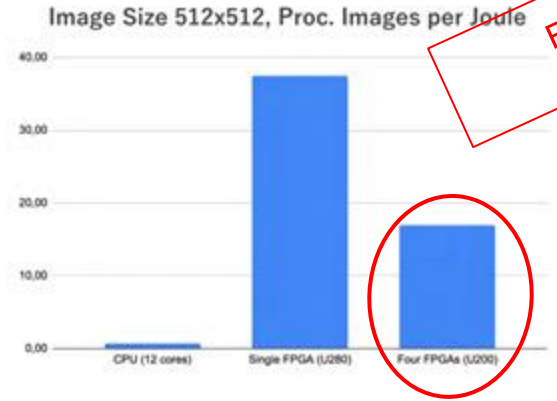
# APEIRON applications:

- FIPLib-multiFPGA



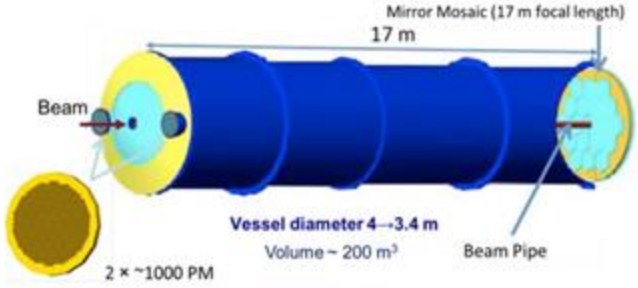

Energy Efficiency - Throughput trade-off

# APEIRON applications:

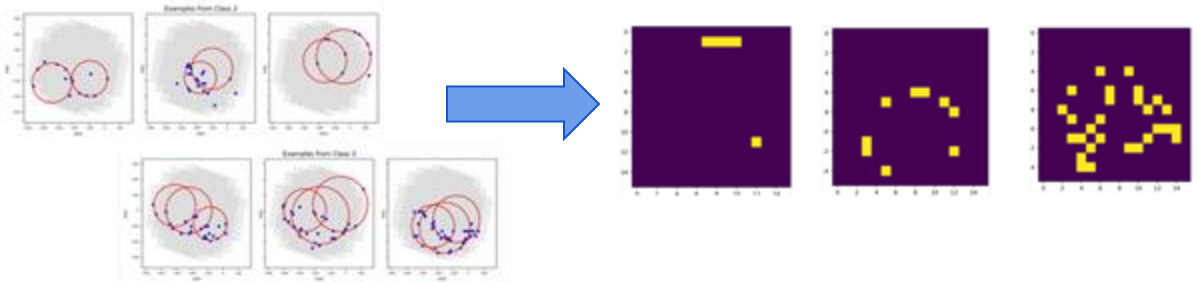
- **RAIDER**

**R**eal-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)

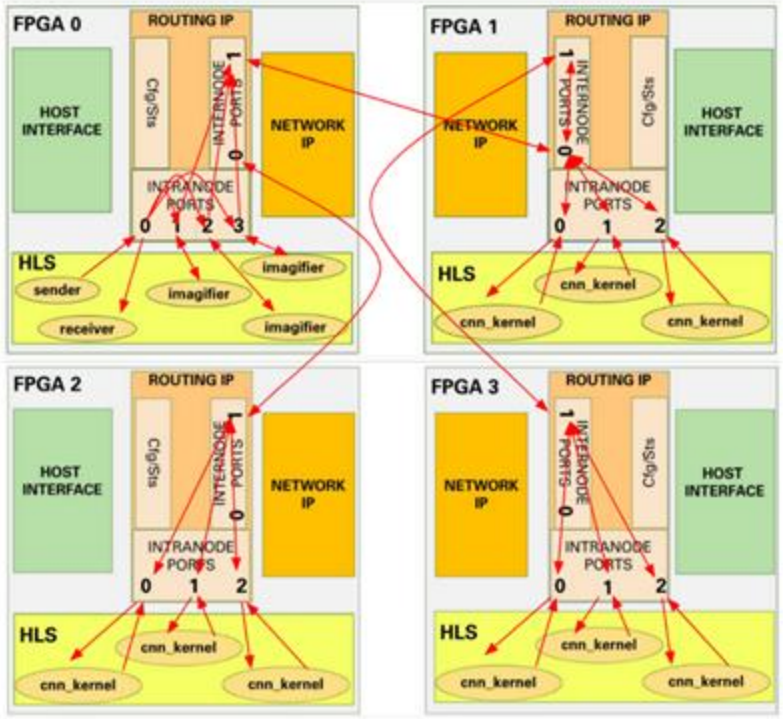


# APEIRON applications:

- RAIDER

| KPI | CNN CPU tensorflow | CNN CPU+GPU tensorflow |
|---|---|---|
| purity/efficiency (per class) | efficiency:<br>- 0: 93%<br>- 1: 83%<br>- 2: 75%<br>- 3+: 83%<br>purity:<br>- 0: 88%<br>- 1: 90%<br>- 2: 71%<br>- 3+: 78% | efficiency:<br>- 0: 93%<br>- 1: 83%<br>- 2: 75%<br>- 3+: 83%<br>purity:<br>- 0: 88%<br>- 1: 90%<br>- 2: 71%<br>- 3+: 78% |
| time to solution [s] | 158.521 | 125.963 |
| **throughput [events/s]** | **189250** | **238165** |
| energy to solution [J] | 11091.919 | 17497.783 (8224.648 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** x20 |
| energy to solution [J] | 165.277 (101.055 FPGA) |
| **energy efficiency [events/J]** | **16336.183 (26718.126 FPGA)** x100 |



# 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