Studies on track finding algorithms based on machine learning with GPU and FPGA

F.A. Di Bello on behalf of the ATLAS TDAQ collaboration

CTD, Mini-workshop on Real time Tracking

Università di Genova

INFN
Istituto Nazionale di Fisica Nucleare

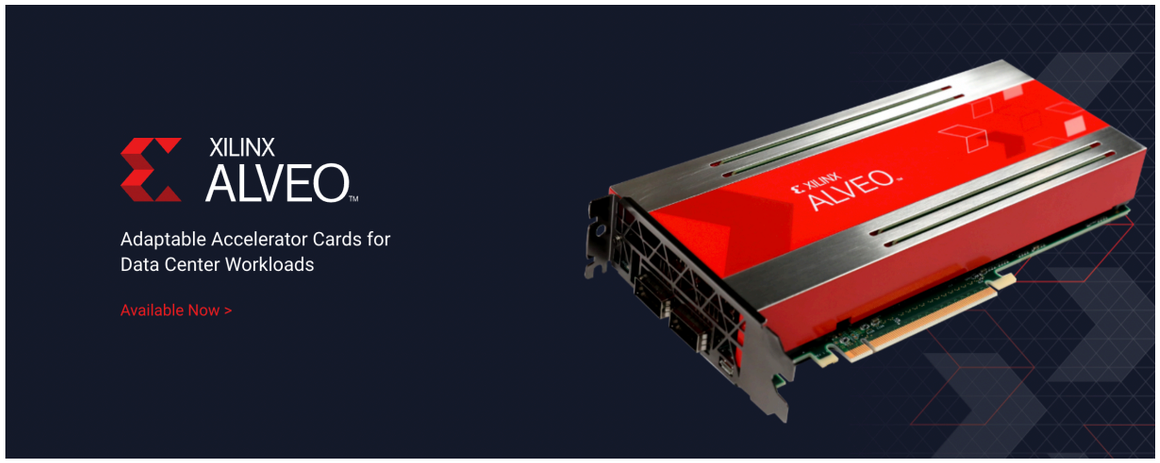
Introduction

We aim to study the performance of ML track finding algorithm in the HLT for muon triggers.

Looking at commercially available FPGA, Xilinx Alveo cards.

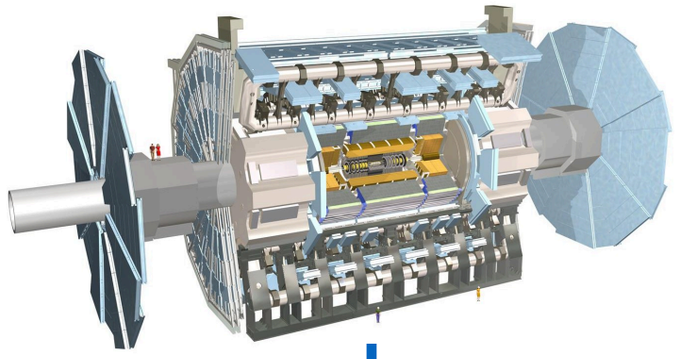
Comparative study between Alveo Cards, GPU and CPU

CPU and GPU Hardware used:
GPU: NVidia RTX A5000 board with 24GB of GDDR6 memory
CPU: single CPU server based on an AMD Epyc 7302 processor running at 2.9 GHz



How do commercial FPGA perform?

Current ATLAS trigger system



100 kHz
~ 2.5 micro-sec

1-1.5 kHz
(nominal)
~ 100 milli sec

2022 rates

Hardware platforms

Two concept for machine learning accelerations

Direct HLS implementation into FPGAs:

Require implementation of a neural network in VHDL or similar.

Significant effort to do so. Platform developed and maintained for HEP community exists: hls4ml

Main advantage is that is fast and suitable for a level-0 trigger.

Use commercial accelerator cards that offer integrated platform for deployment:

Vitis AI

Adaptable & Real-Time AI Inference Acceleration

Commercially available, no had-hoc maintenance.

Dedicated hardware and related software to traslate from high level python codes, into code executable in dedicated hardware.

Not as fast as the other approach, suitable for an HLT trigger

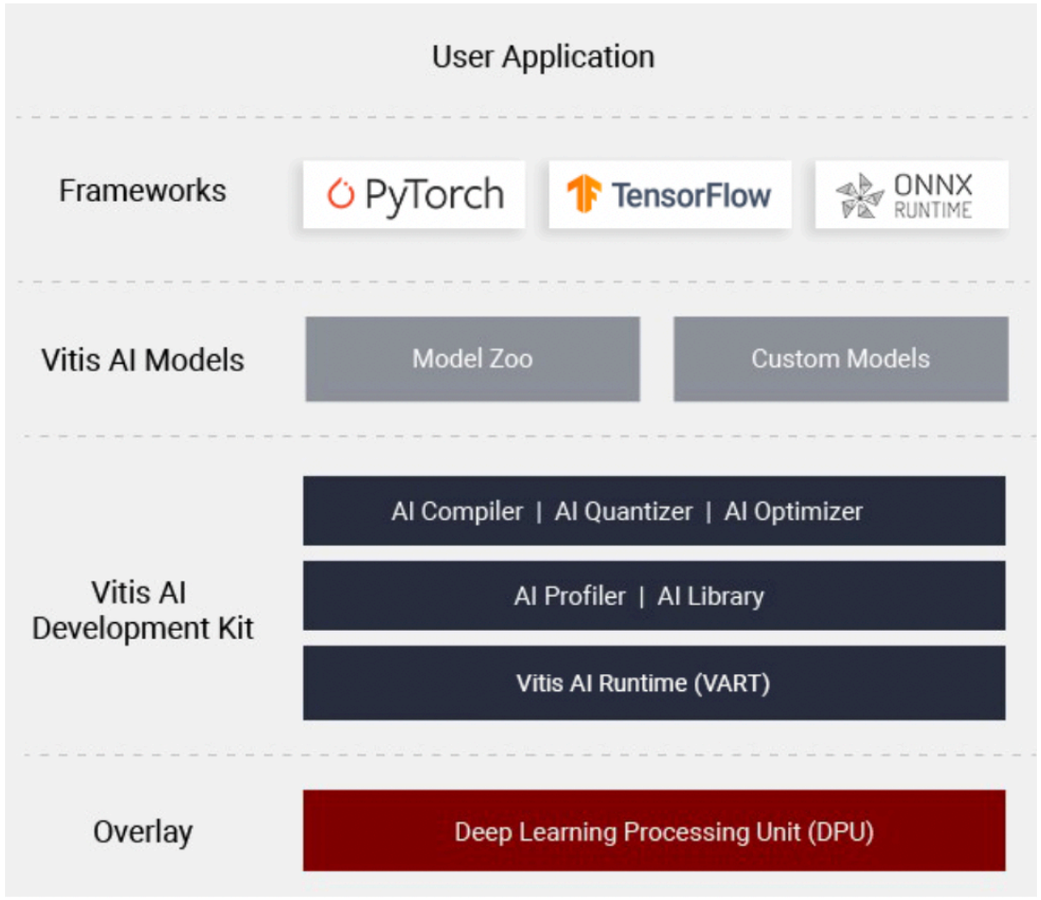
Bounded to the supported architecture

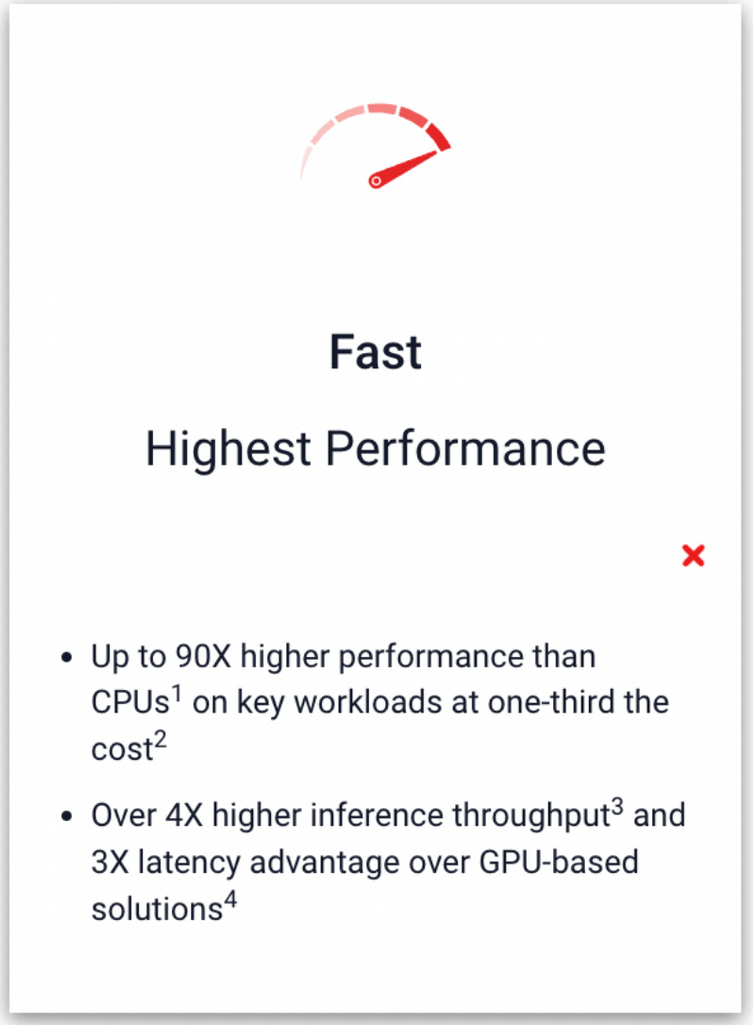
Vitis-AI overview

Xilinx offers several accelerator card designed and built to accelerate ML algorithms (mostly CNN) xilinx

The claim is that inference and throughput are improved over standard CPU and GPU

Improvements also expected in terms of power consumption





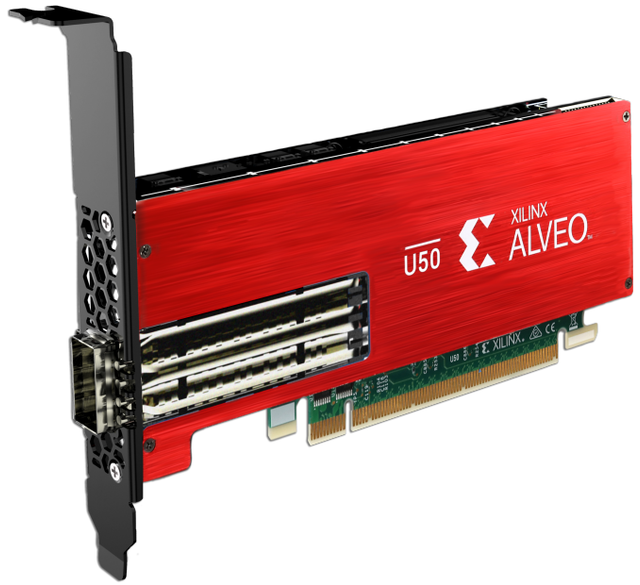
The hardware tested

Several accelerator cards are commercially available: cards

U250

Designed for machine learning inference, video transcoding, and database search & analytics

U50



Designed for financial computing, machine learning, computational storage, and data search and analytics

VCK5000

It is an AI development card, more versatile than the other two

The toy model used in this study

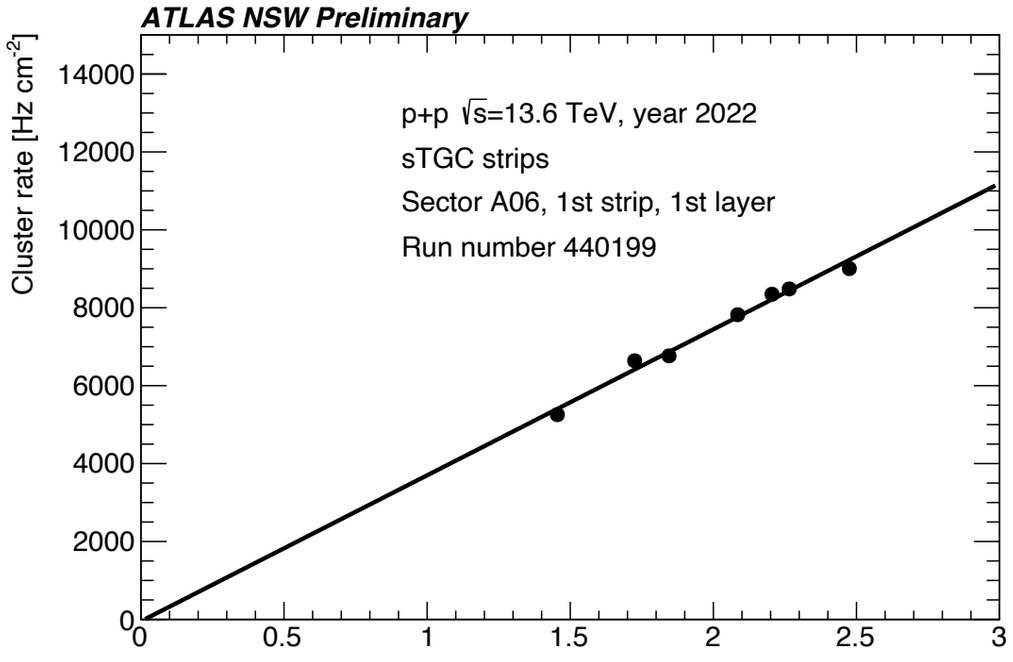
To speed up R&D part of the study, a toy model is simulated

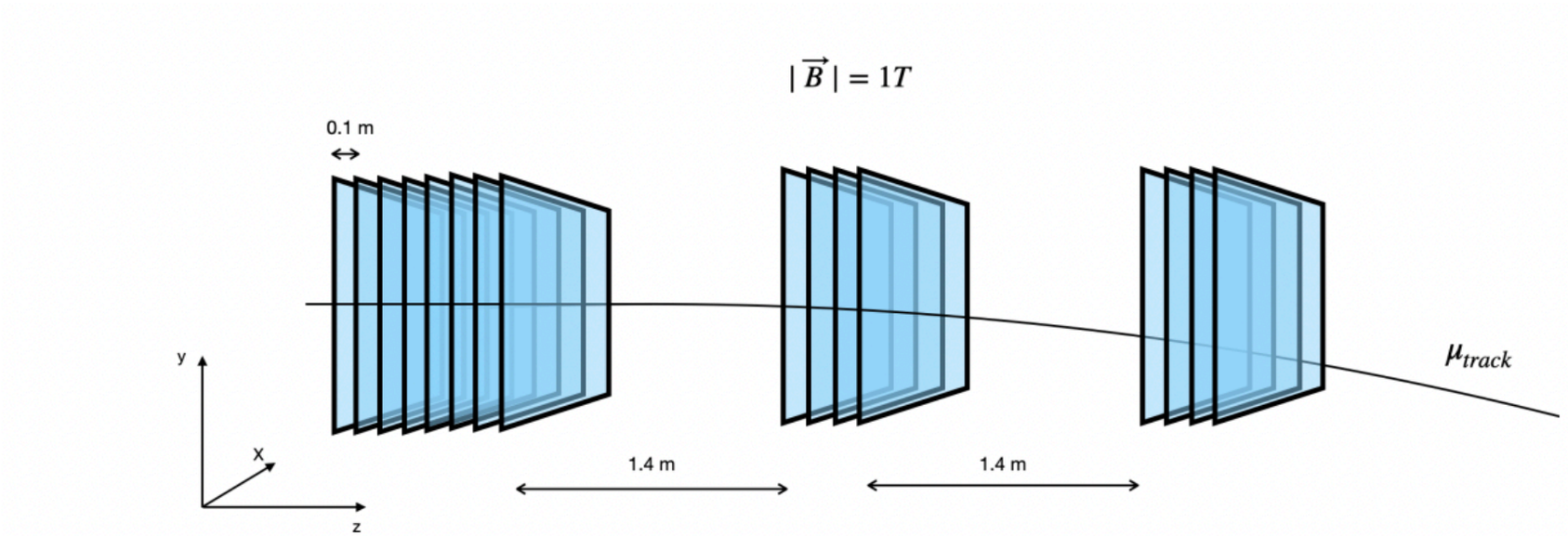
Toy model is inspired by a muon system

4 samples produced with different noise rates: 2, 5, 10, 15 kHz/cm**2

Effect from correlated background is also emulated

Will now discuss the main reco steps for tracking: clustering and pattern reco and their performance on CPU/GPU and FPGAs





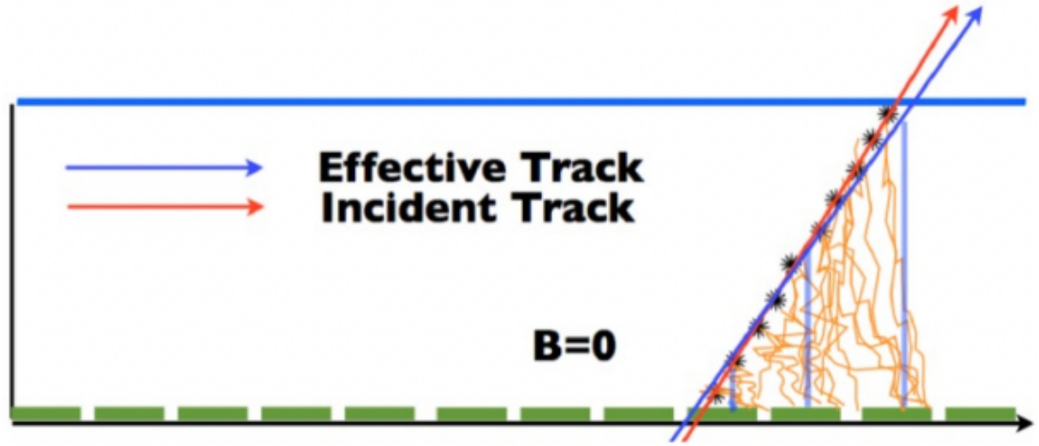
Cluster reconstruction

A cluster is formed from neighbouring hits

Typically, the weighted centroid of the cluster is used

$$x_c = \frac{\sum_i x_i q_i}{\sum_i q_i}$$

Collected charge: q_i
Strip position: x_i



The known challenges with the standard approach are:

1. Depending on the incidence angle of the muon, a degradation is expected

2. “Correlated” background that originates from interaction with material prior to the active layers

ML is good candidate to improve the clustering performance

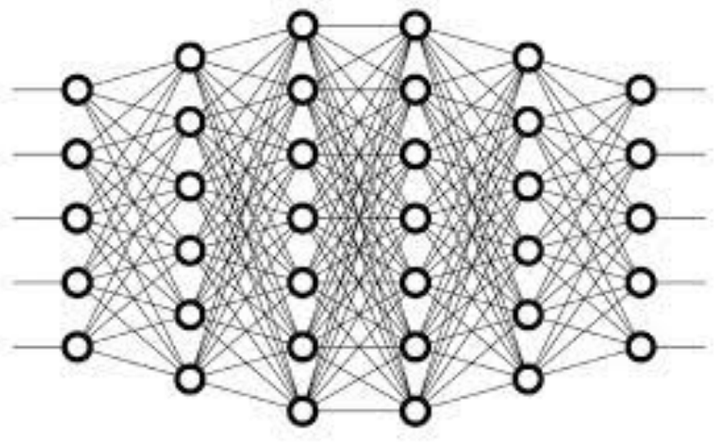
The Deep Neural Net approach

A deep neural network is used (similar to what done in the inner silicon tracker ref)

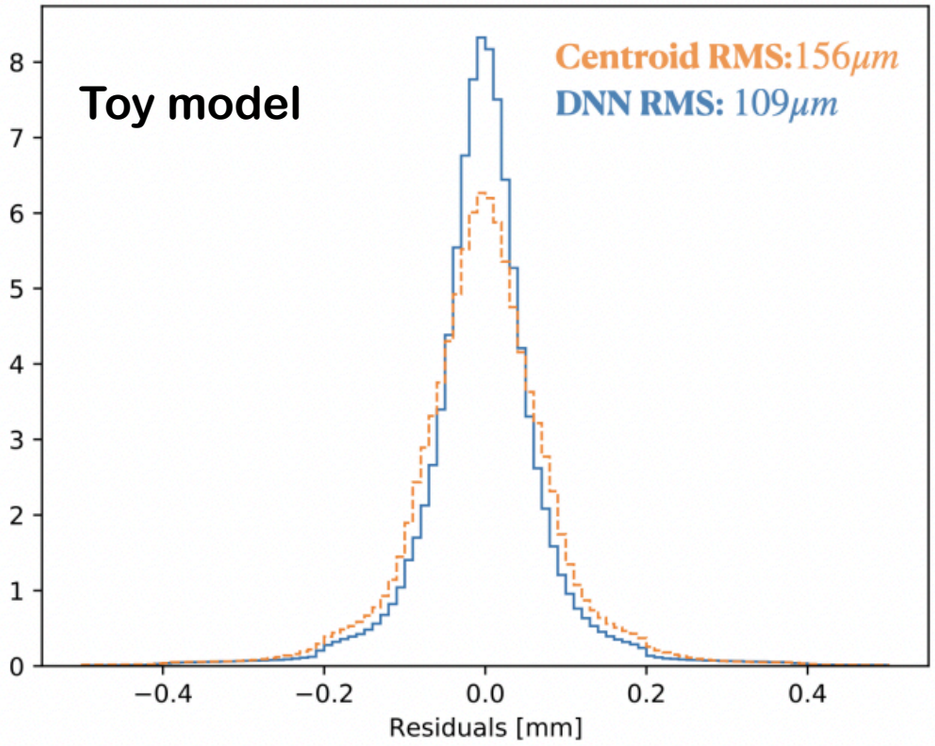
Inputs are:

1. The total number of hits belonging to the cluster
2. The charge of the strip with highest charge
3. The charge of its two left-right closest neighbours
4. The position of the strip with highest charge
5. The Position of its two left-right closest neighbours

NB: if the cluster has less than 5 strips, zero-padding is employed

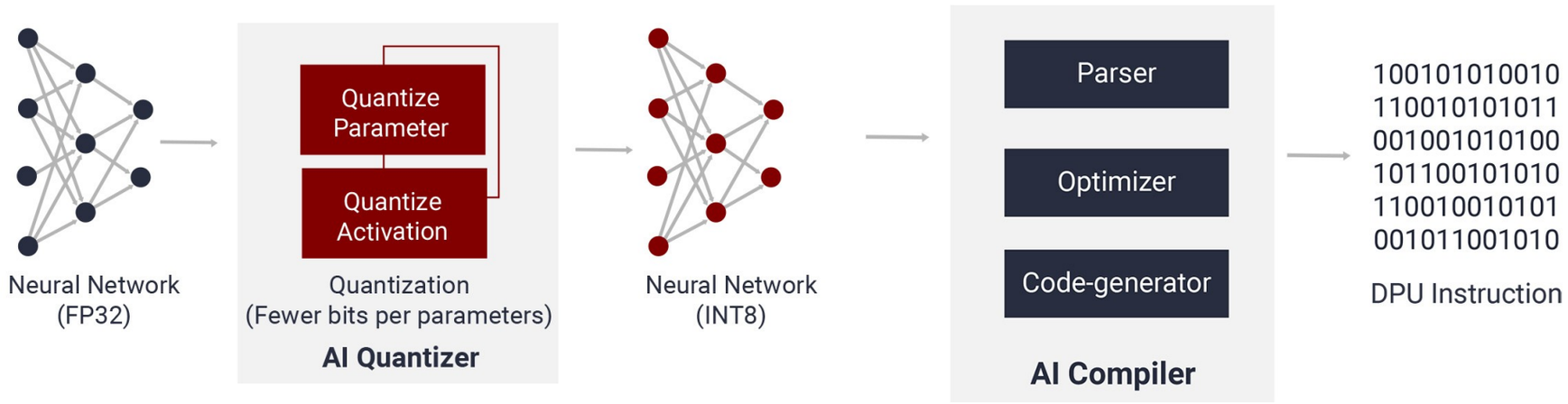


Standard regression using as target the true crossing position of the muon



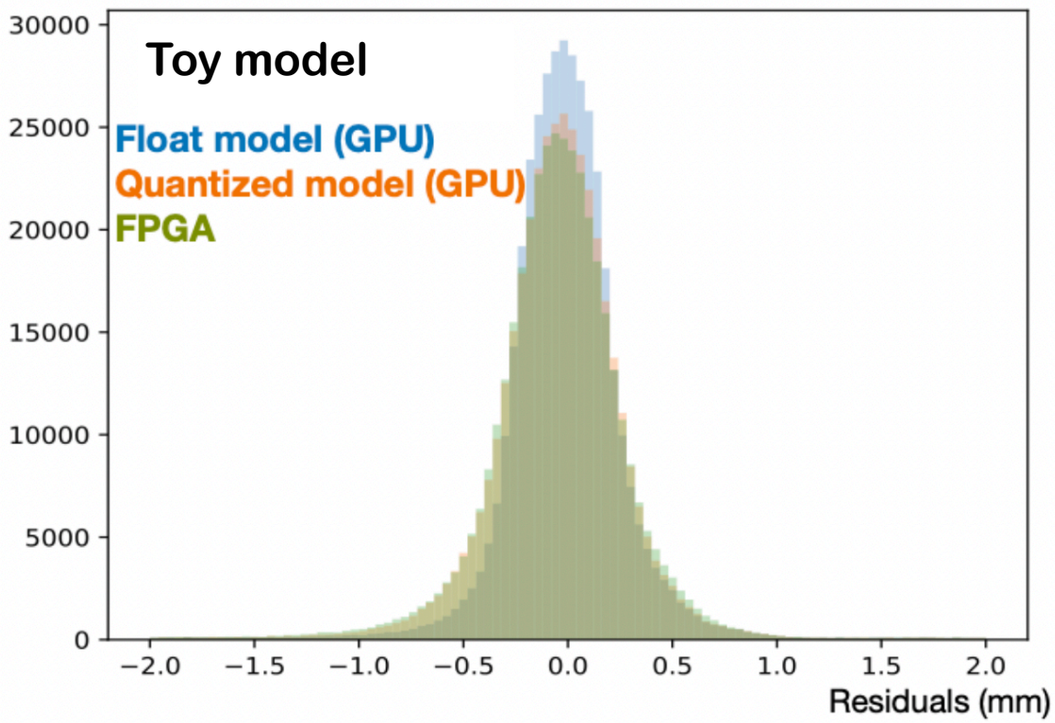
Run the DNN inference with Alveo cards

Standard workflow for deployment into FPGA. Pruning tools also available but not tested



Quantization convert 32-bit floating-point weights and activations to fixed-point INT8.

Many quantisation models available, here no retraining was performed, accepting a small degradation of performance .

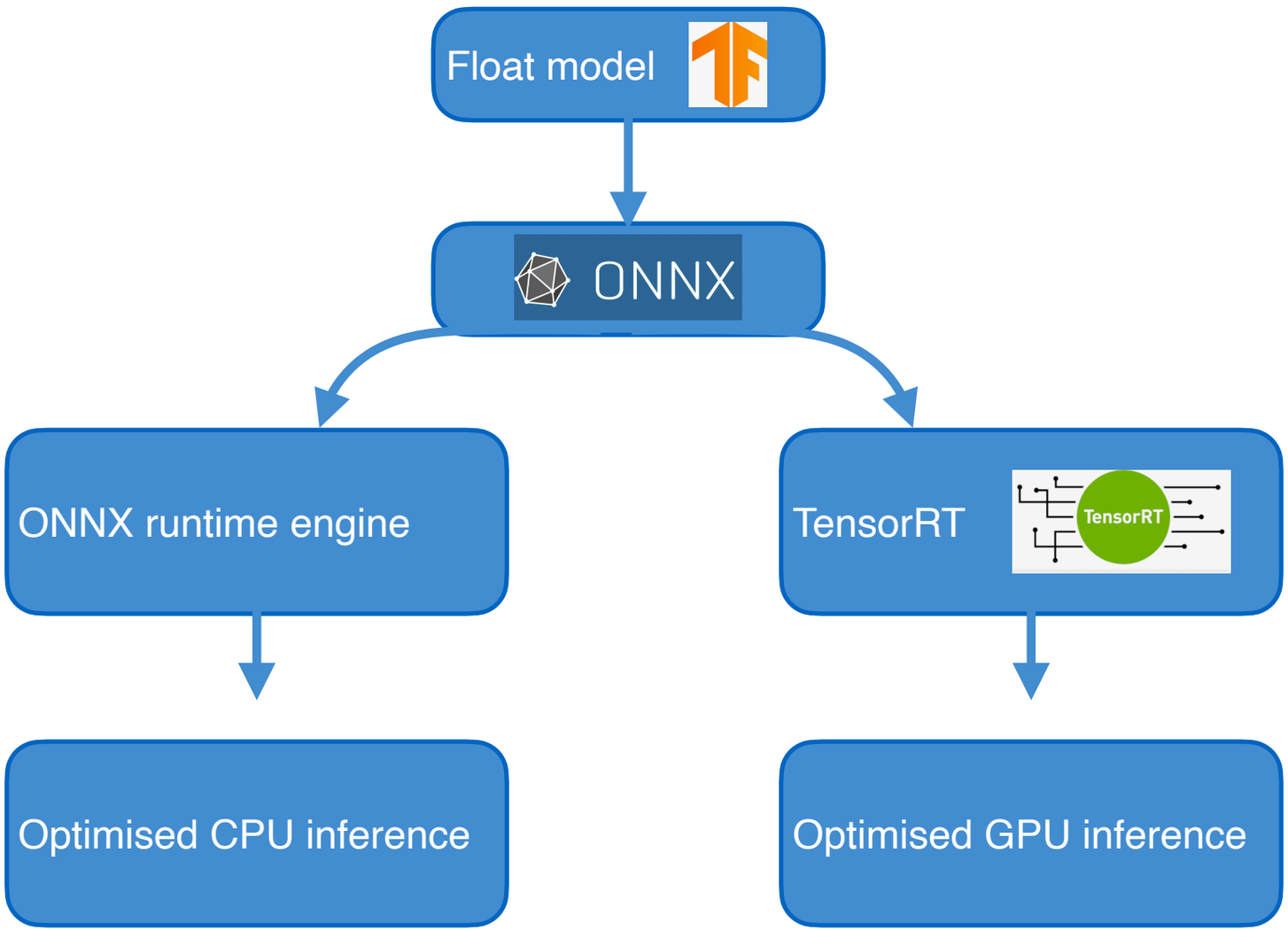


Run performance evaluation with GPU/CPU

Usage of ONNX on CPU link

Usage of TensorRT link on GPU

Both offer high-performance deep learning inference, includes a deep learning inference optimizer and runtime that delivers low latency and high throughput for inference applications.

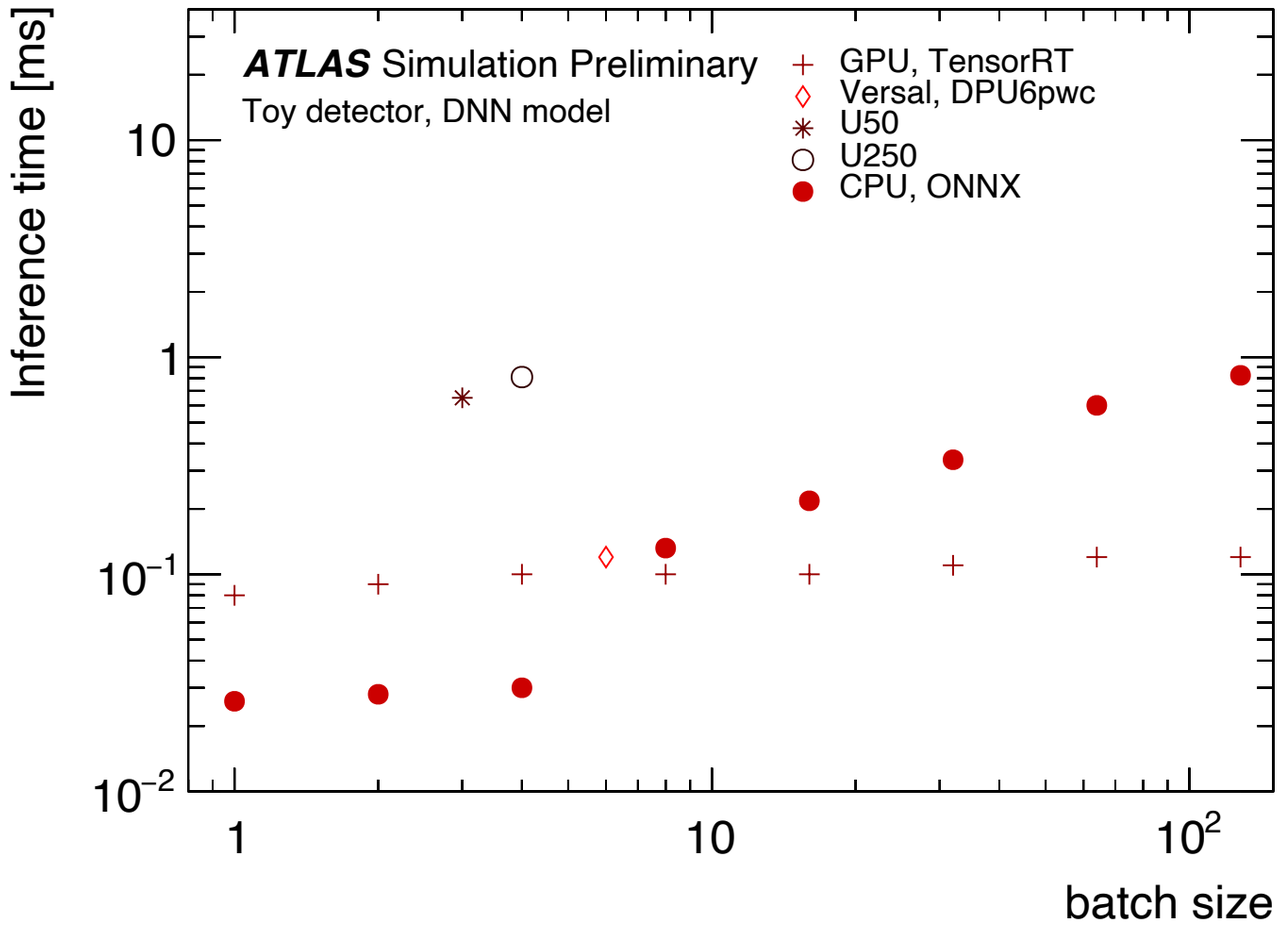


Inference time results

CPU is already well within the latency requirements

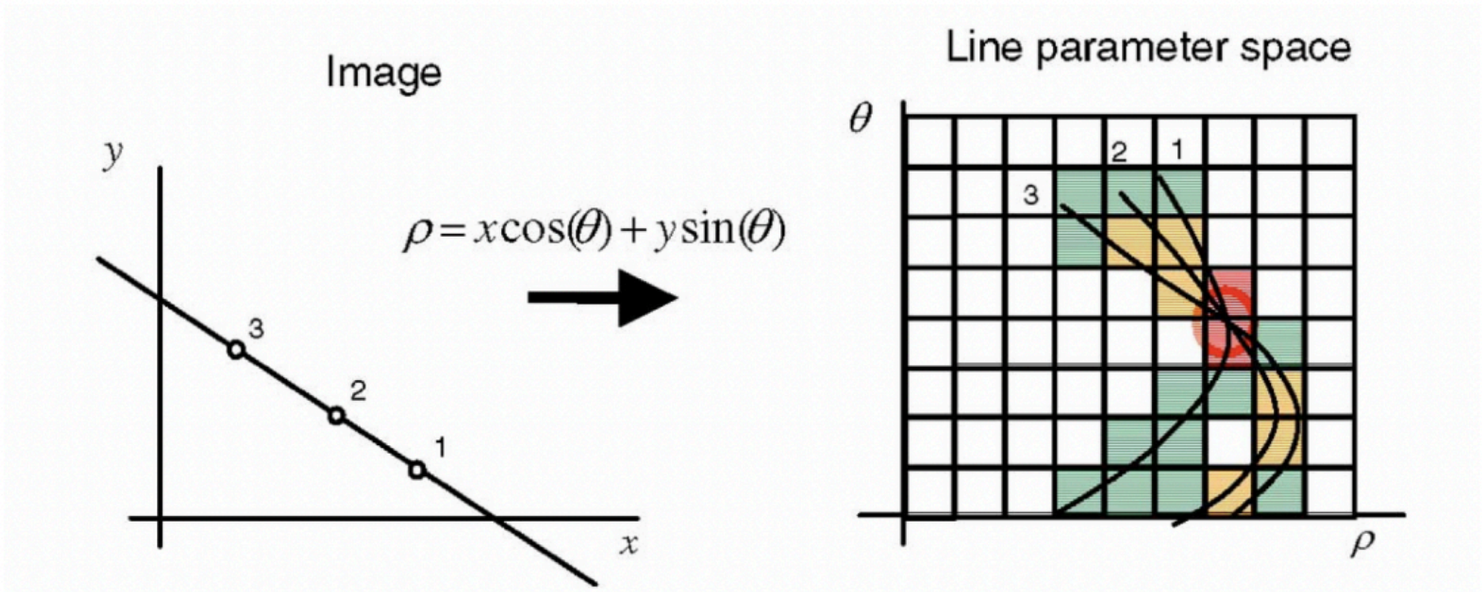
GPU with TensorRT improves a bit further

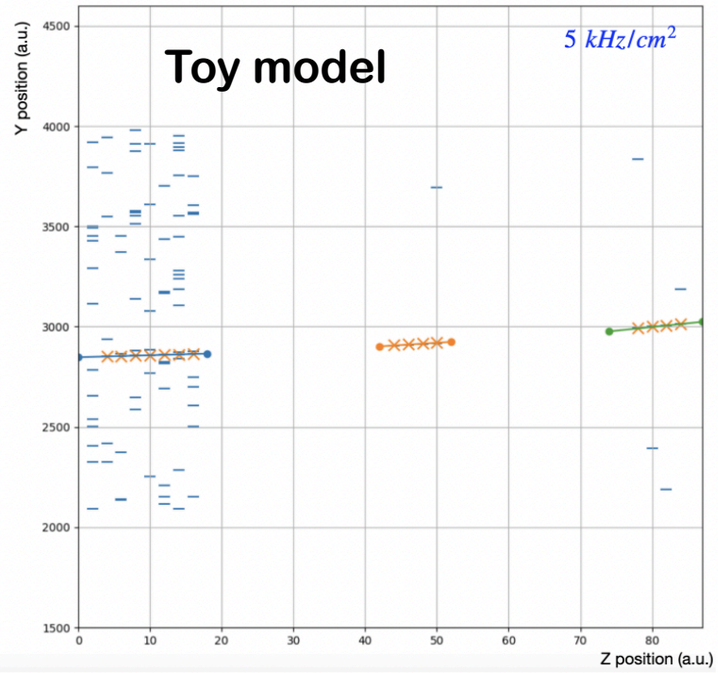
No significant gain observed over both architectures



Pattern recognition

Pattern reco. Is currently based on “Hough transform”





Within each sector, it is possible to approximate the muon with a straight line and run 3 HT

In a second step, a functional fit is run to extract the pT of the crossing muon

ML can help here as well, treating all layers simultaneously, profiting from their inter-correlations

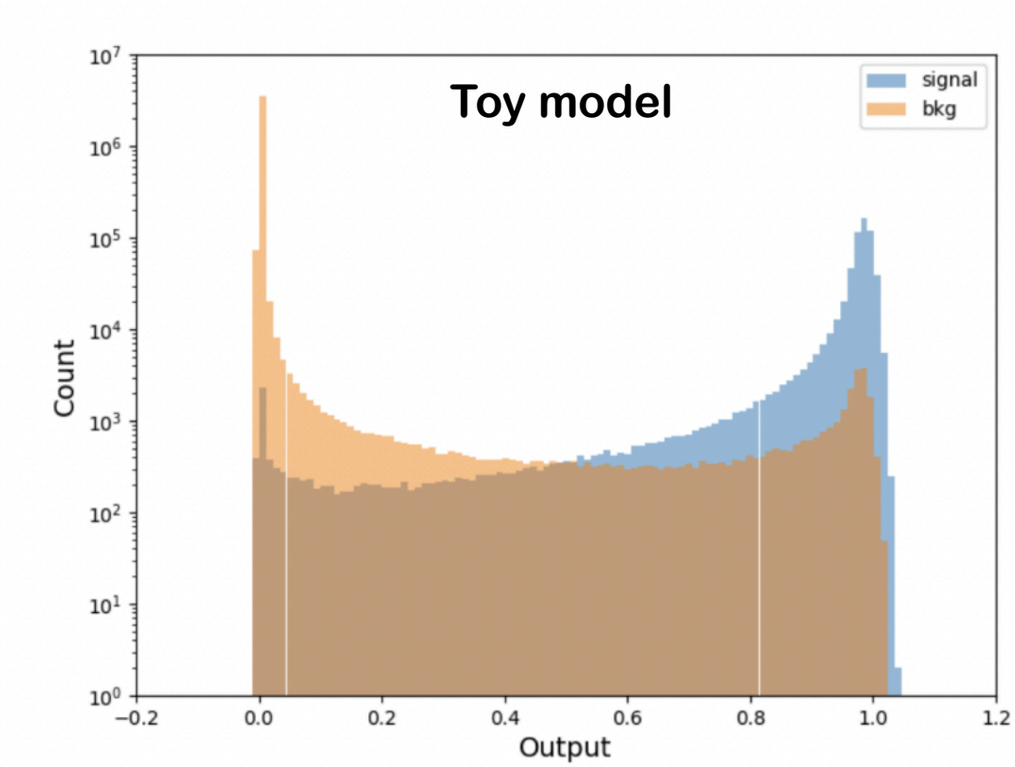
Alternative: a CNN approach

In order to test the algorithm with Alveo cards, a CNN was also developed

A CNN is not an optimal approach for pattern reco tasks but it is useful for testing FPGA performance

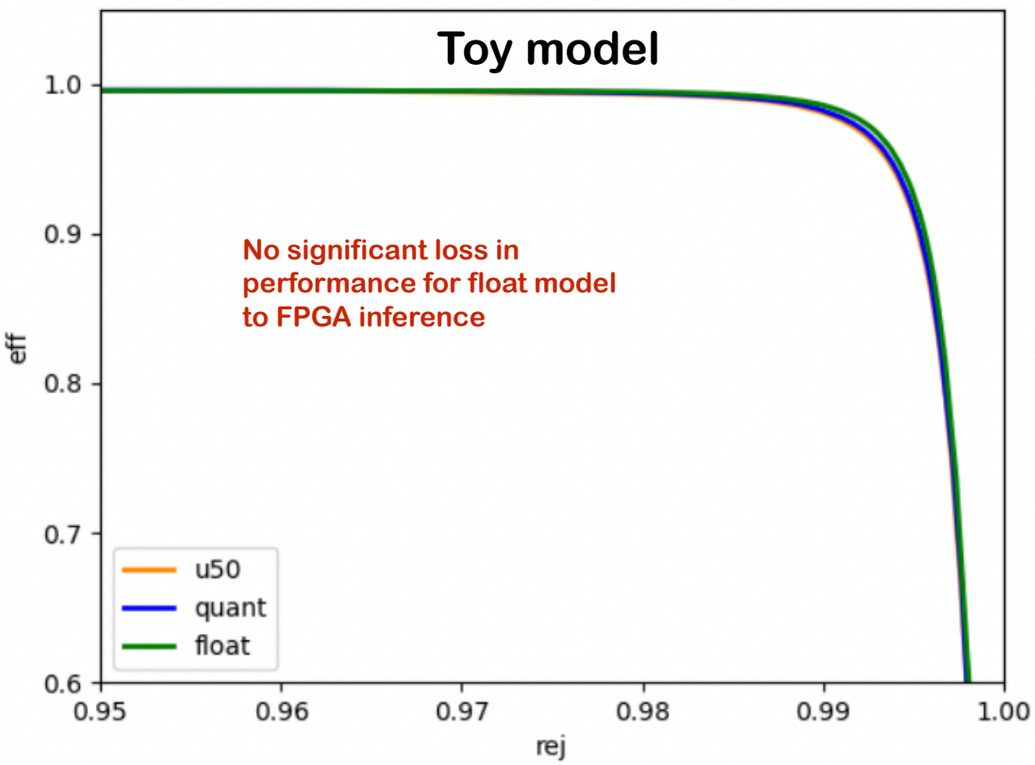
An event display is translated into a 3000x16 pixel 2D image, and convolution/deconvolution operation are used

The output is an image whose intensity indicate the probability of the hits being associated to the muon



Layer (type)	Output Shape	Param #
input_1 (InputLayer)	[(None, 3000, 16, 1)]	0
conv1 (Conv2D)	(None, 3000, 16, 2)	194
pool1 (MaxPooling2D)	(None, 1500, 16, 2)	0
conv2 (Conv2D)	(None, 1500, 16, 4)	100
pool2 (MaxPooling2D)	(None, 750, 16, 4)	0
conv3 (Conv2D)	(None, 750, 16, 8)	392
pool3 (MaxPooling2D)	(None, 375, 16, 8)	0
conv4 (Conv2D)	(None, 375, 16, 16)	6160
pool4 (MaxPooling2D)	(None, 375, 8, 16)	0
conv5 (Conv2D)	(None, 375, 8, 16)	32784
Tconv0 (Conv2DTranspose)	(None, 375, 16, 2)	258
Tconv1 (Conv2DTranspose)	(None, 750, 16, 4)	68
Tconv2 (Conv2DTranspose)	(None, 1500, 16, 8)	264
Tconv3 (Conv2DTranspose)	(None, 3000, 16, 16)	1040
output (Conv2D)	(None, 3000, 16, 1)	49

Total params: 41,309
Trainable params: 41,309
Non-trainable params: 0

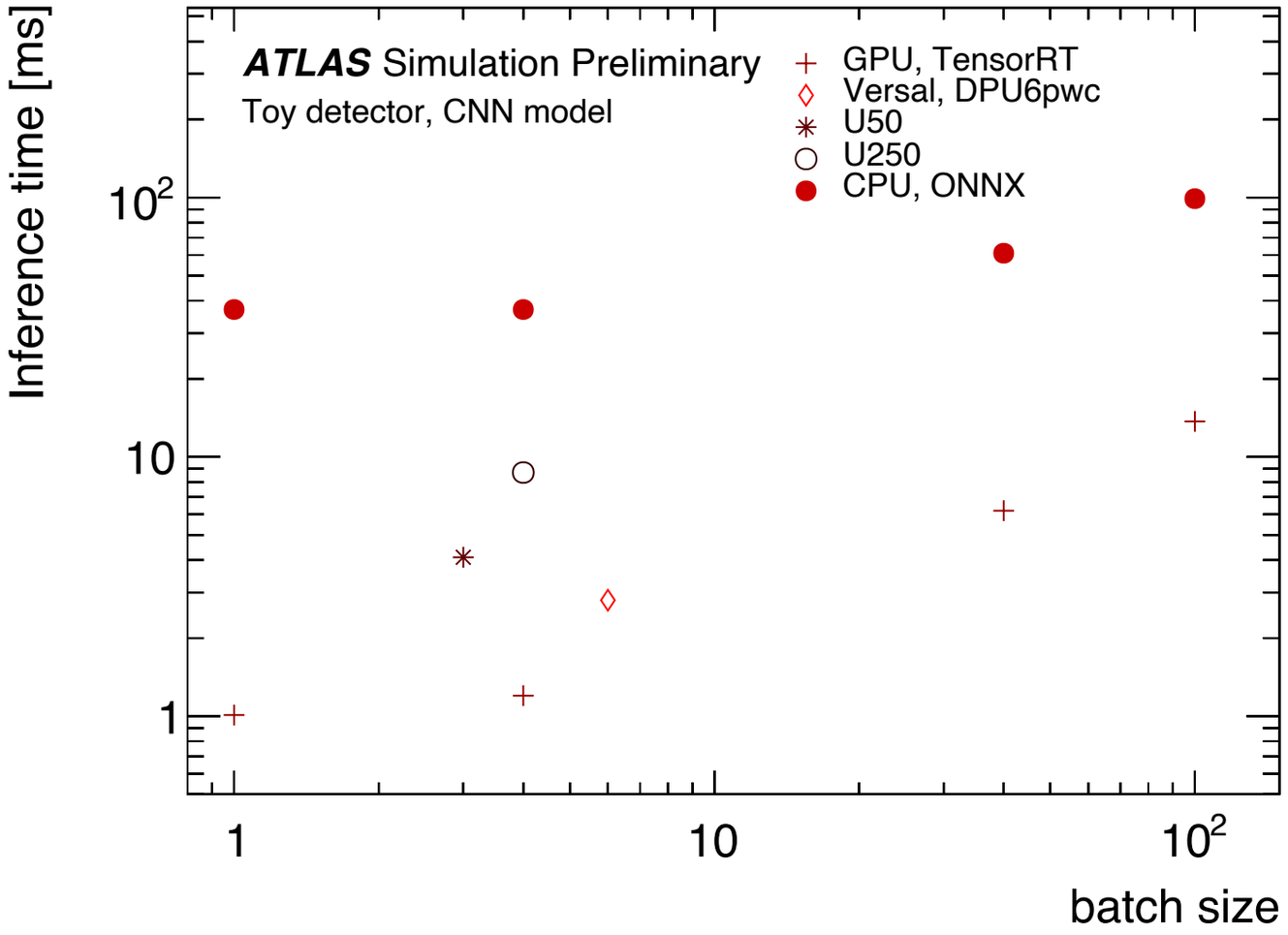


Comparison CNN

CNN model successfully tested on CPU, GPU and several FPGAs

Overall CPU already meets the requirement imposed by the HLT latency

Largest improvement is seen with TensorRT on GPU. Study on CPU load needs to be studied, as well as the power dissipations



Pattern recognition with an RNN

ML based on what Xilinx commercially releases

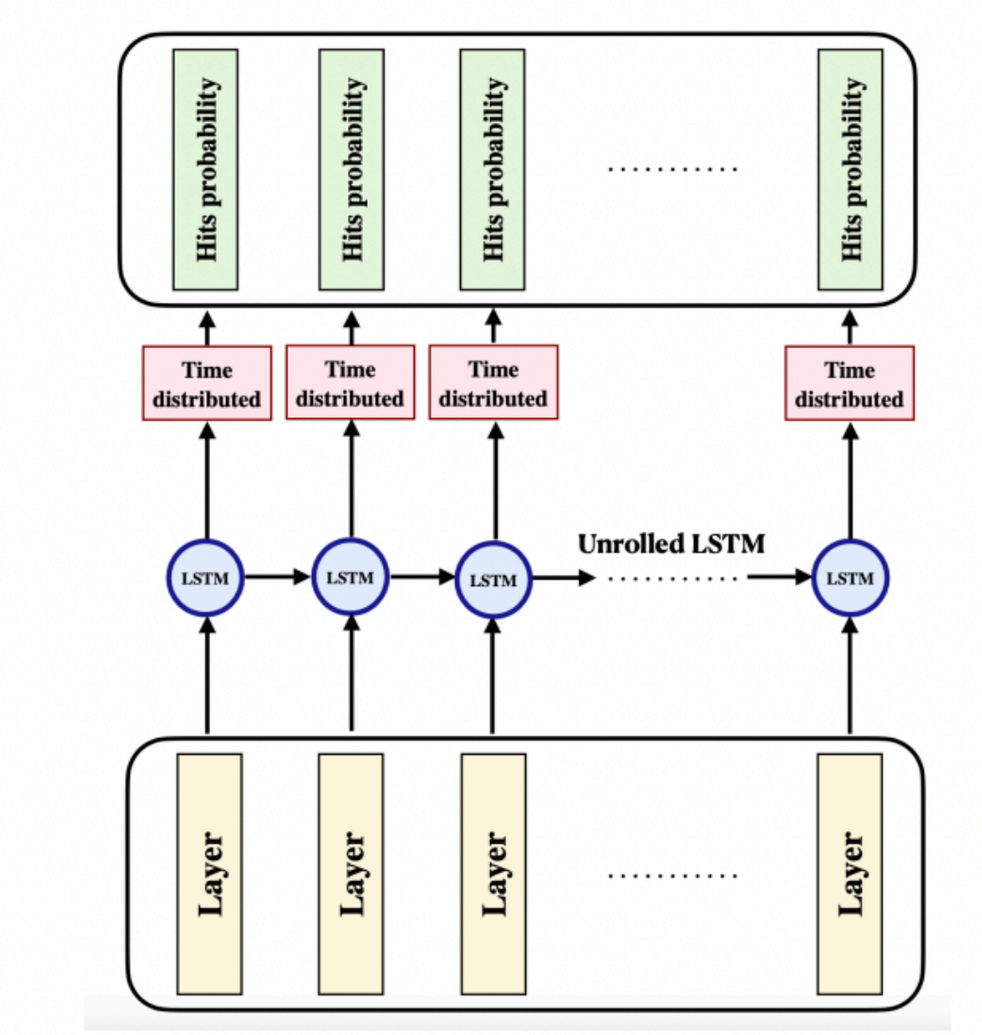
An RNN layer is expected to become available, but yet not possible

Inputs are output of the cluster DNN

More sophisticated ML approaches such as GNN and/ or transformers, are not yet supported

In the RNN approach, consequent layers are ordered based on their position.

Two possibilities: outside-in or, inside-out



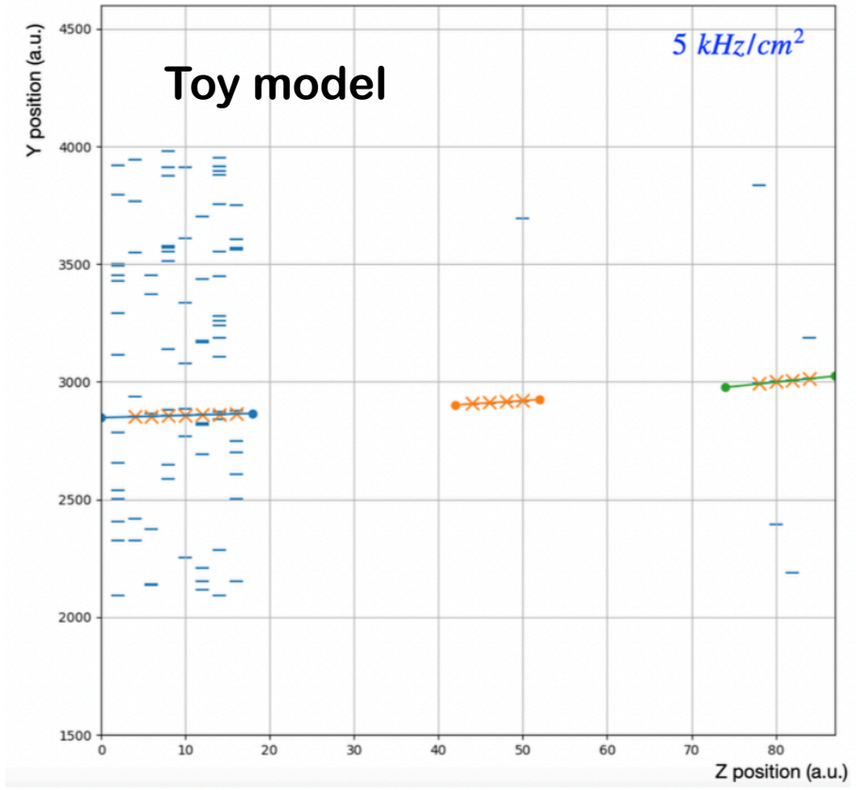
RNN performance results

A simplified Hough transform is implemented. Detector is split into three sectors, pattern recognition is performed in each of those singularly.

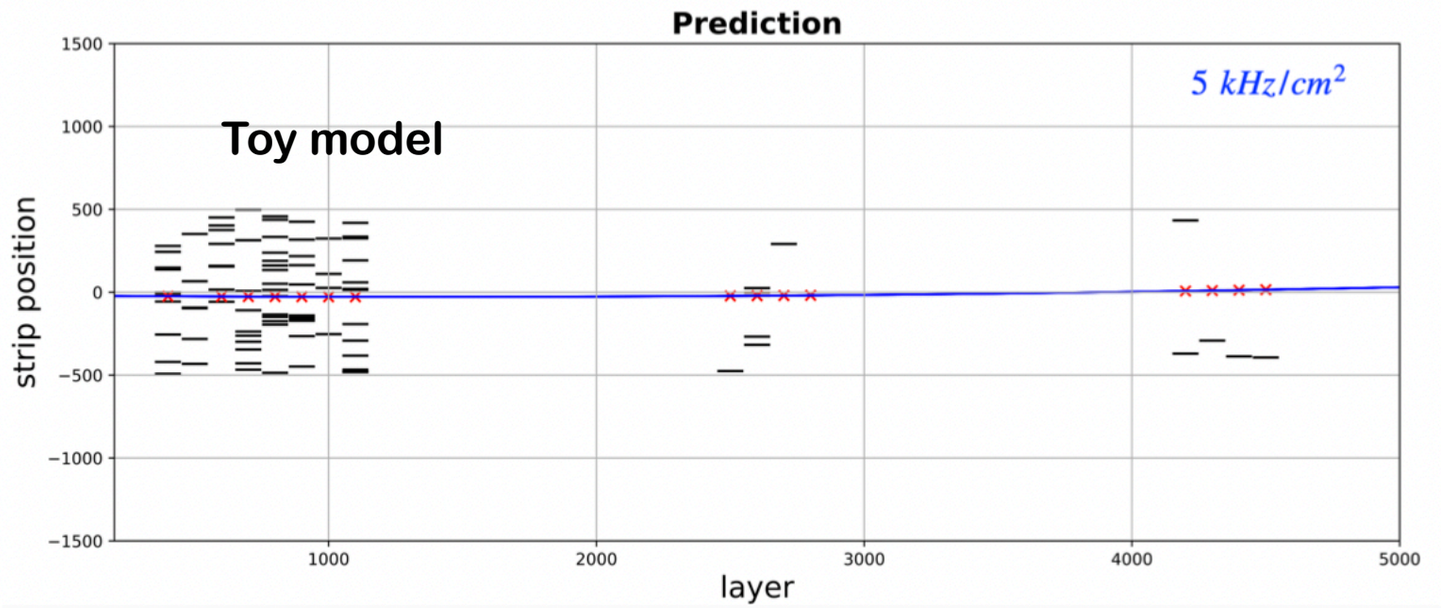
The RNN model instead works for the whole detector.

The HT inference time was estimated to be around 1 ms

Hough Transform



RNN



RNN performance results

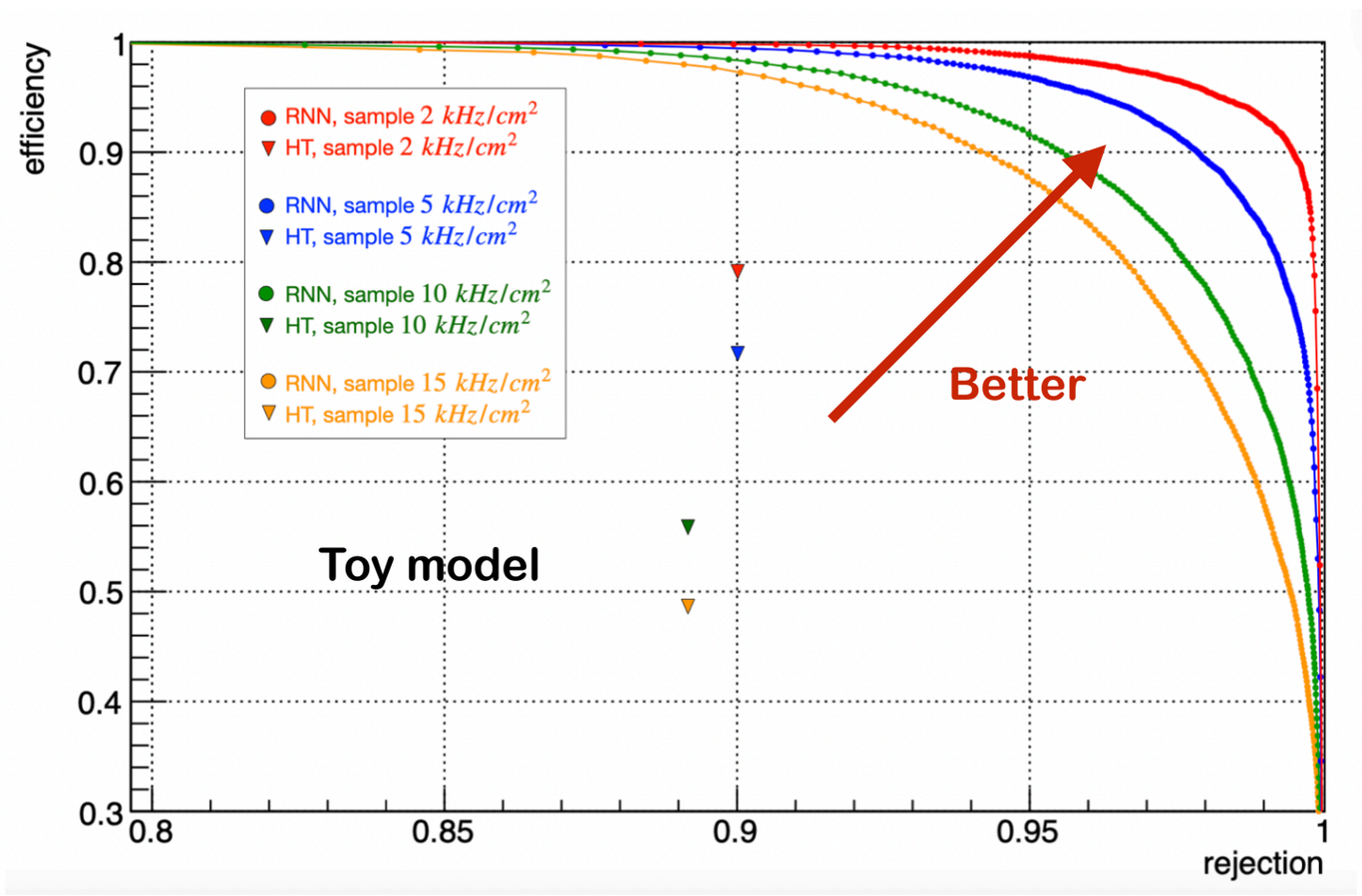
M. Carnesale
PhD Thesis

Performance evaluated for different rates

Generally, a decrease of performance is seen at higher rates, as expected

Model is evaluated on CPU only. Versal boards do not support RNN yet (supported is expected). On GPU tensorRT did not support timing layer.

Using ONNX on CPU, the performance show an inference for a single event is O(1 ms), well within the latency requirement for a HLT trigger



Conclusions

DNN model for cluster position successfully tested on CPU, GPU and several Alveo cards.

A CNN model has been implemented to study the FPGA performance.

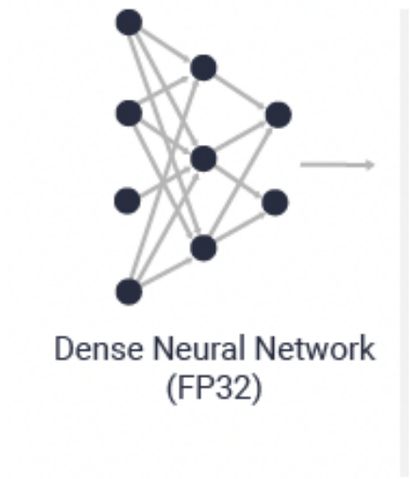
RNN model proposed as an alternative to HT methods. Inference time already on CPU is well within design constraint.

RNN model are yet to be supported by VitisAI, expected support in the next Vitis-AI versions.

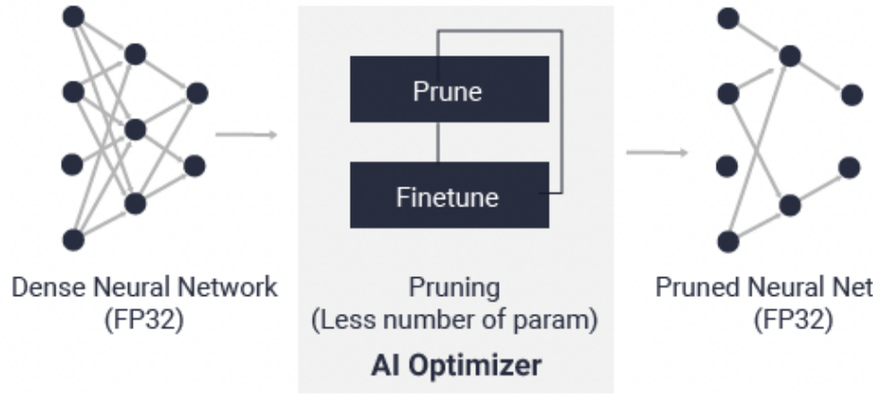
CPU load and power consumption shall still be studied… as well as cost

Workflow to use Alveo cards

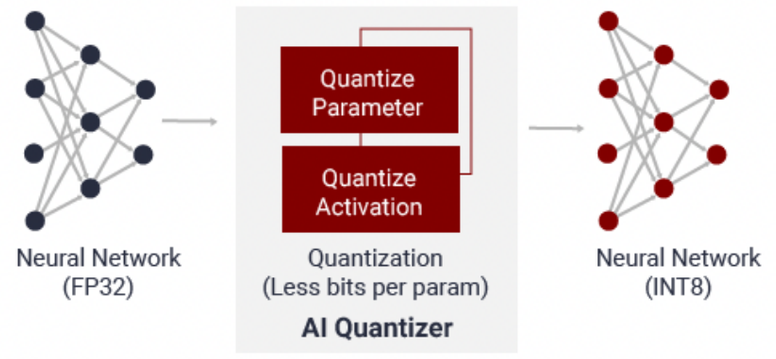
Develop your favourite model with your favourite framework



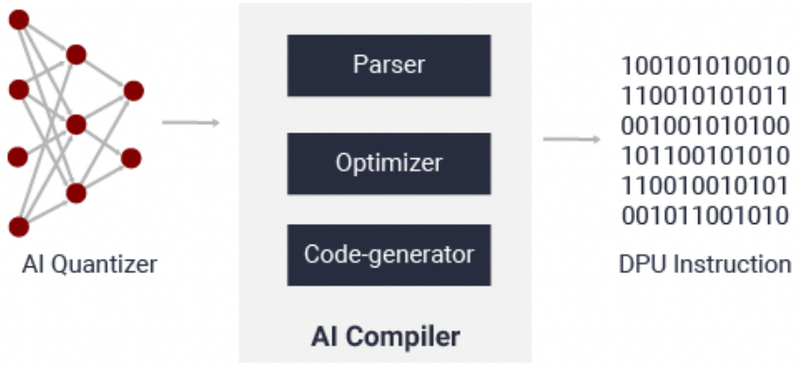
Pruning (optional)



Quantize



Compile



Run the inference

Varying the batch size

Main point is that tensorRT does not work with dynamic batch sizes


```python
import onnx
onnx_model = onnx.load_model('model_singleLoss.onnx')

BATCH_SIZE = 1
inputs = onnx_model.graph.input
for input in inputs:
    dim1 = input.type.tensor_type.shape.dim[0]
    dim1.dim_value = BATCH_SIZE

model_name = "model_singleLoss_mod.onnx"
onnx.save_model(onnx_model, model_name)
```

Models we are interested in

RNN:

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 16, 100)]         0
_________________________________________________________________
lstm (LSTM)                  [(None, 16, 200), (None,  240800
_________________________________________________________________
lstm_1 (LSTM)                [(None, 16, 20), (None, 2 17680
_________________________________________________________________
time_distributed (TimeDistri (None, 16, 51)            1071
=================================================================
Total params: 259,551
Trainable params: 259,551
Non-trainable params: 0
```

CNN:

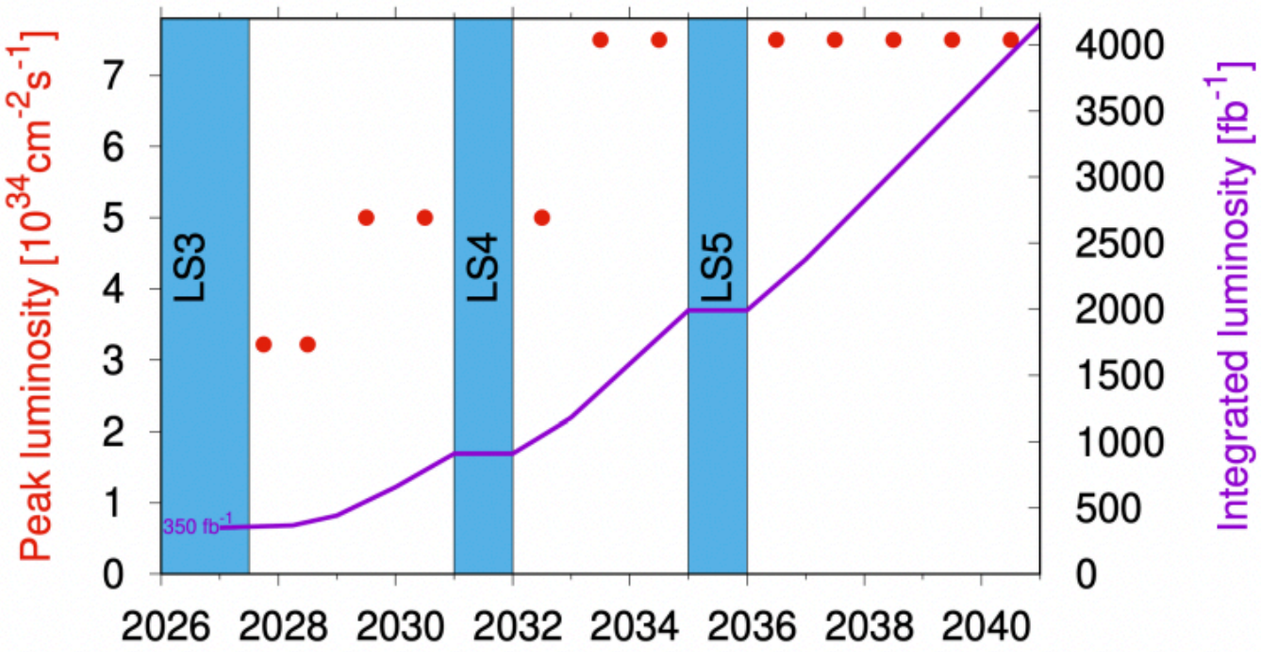
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 3000, 16, 1)]     0
_________________________________________________________________
conv1 (Conv2D)               (None, 3000, 16, 2)       194
_________________________________________________________________
pool1 (MaxPooling2D)         (None, 1500, 16, 2)       0
_________________________________________________________________
conv2 (Conv2D)               (None, 1500, 16, 4)       100
_________________________________________________________________
pool2 (MaxPooling2D)         (None, 750, 16, 4)        0
_________________________________________________________________
conv3 (Conv2D)               (None, 750, 16, 8)        392
_________________________________________________________________
pool3 (MaxPooling2D)         (None, 375, 16, 8)        0
_________________________________________________________________
conv4 (Conv2D)               (None, 375, 16, 16)       6160
_________________________________________________________________
pool4 (MaxPooling2D)         (None, 375, 8, 16)        0
_________________________________________________________________
conv5 (Conv2D)               (None, 375, 8, 16)        32784
_________________________________________________________________
Tconv0 (Conv2DTranspose)     (None, 375, 16, 2)        258
_________________________________________________________________
Tconv1 (Conv2DTranspose)     (None, 750, 16, 4)        68
_________________________________________________________________
Tconv2 (Conv2DTranspose)     (None, 1500, 16, 8)       264
_________________________________________________________________
Tconv3 (Conv2DTranspose)     (None, 3000, 16, 16)      1040
_________________________________________________________________
output (Conv2D)              (None, 3000, 16, 1)       49
=================================================================
Total params: 41,309
Trainable params: 41,309
Non-trainable params: 0
```

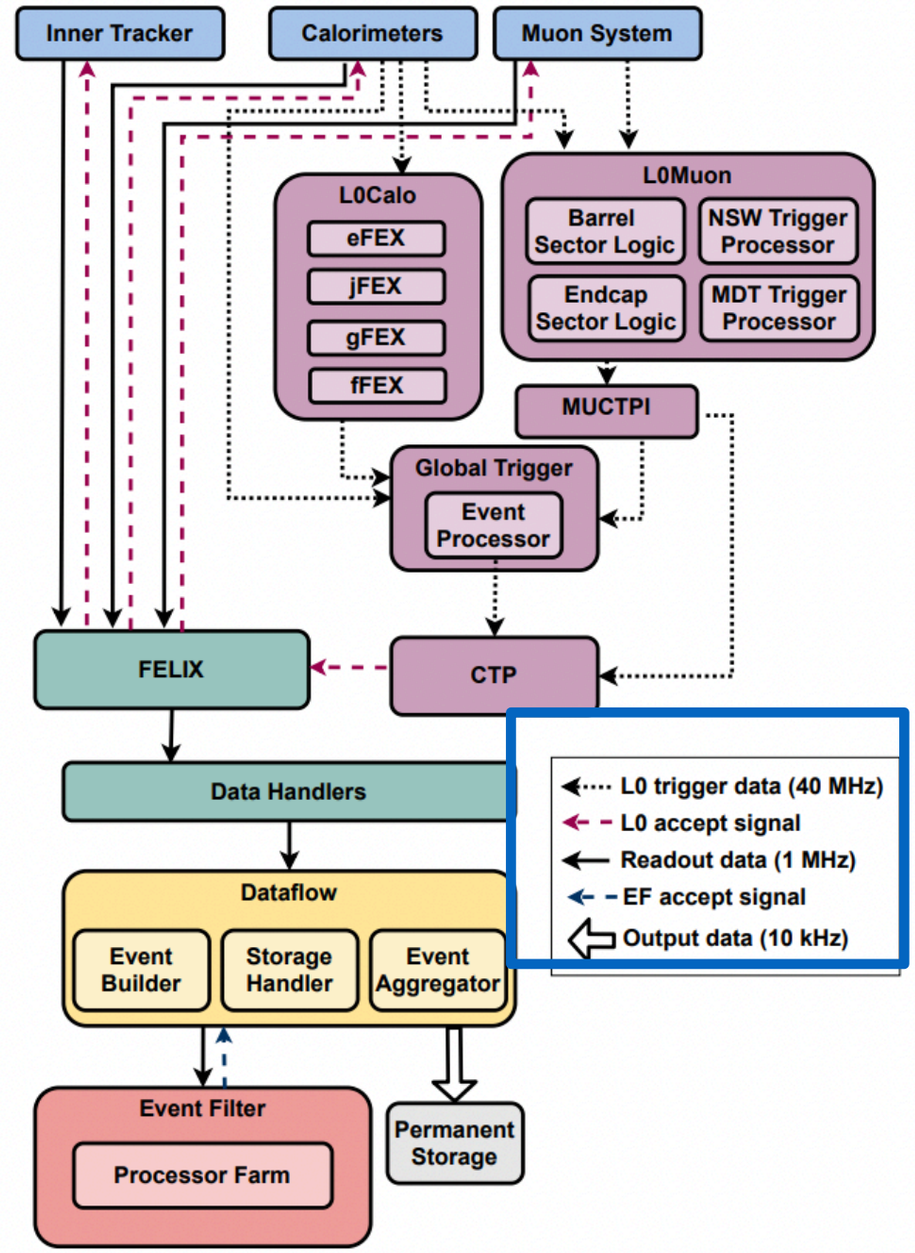
ATLAS HL-LHC trigger system

The work here is relevant for future RUN3 operations, but most importantly for triggering at HL-LHC

High luminosity and pile-up makes trigger decisions much more challenging

We will mostly consider as use-case muon system, for future applications to muon tracking





[TDR trigger HL-LHC]