Deep Learning fast inference on FPGA for CMS Muon Level-1 Trigger studies

International Symposium on Grids & Clouds 2021

26th March 2021

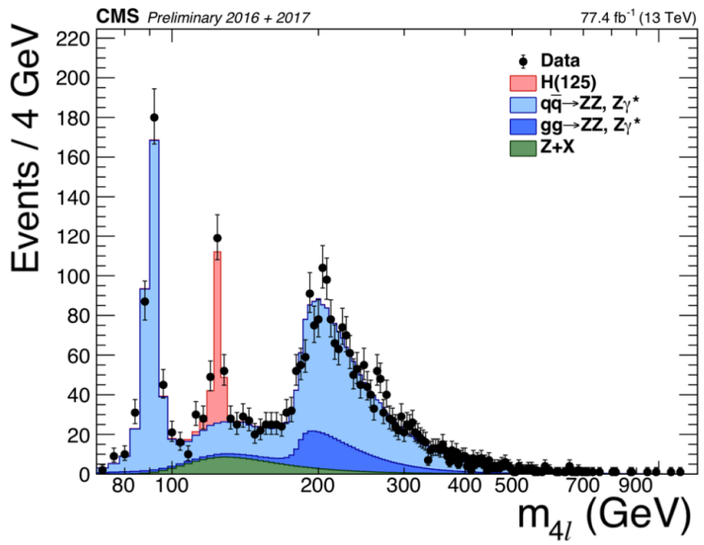
Tommaso Diotalevi
(INFN and University of Bologna)

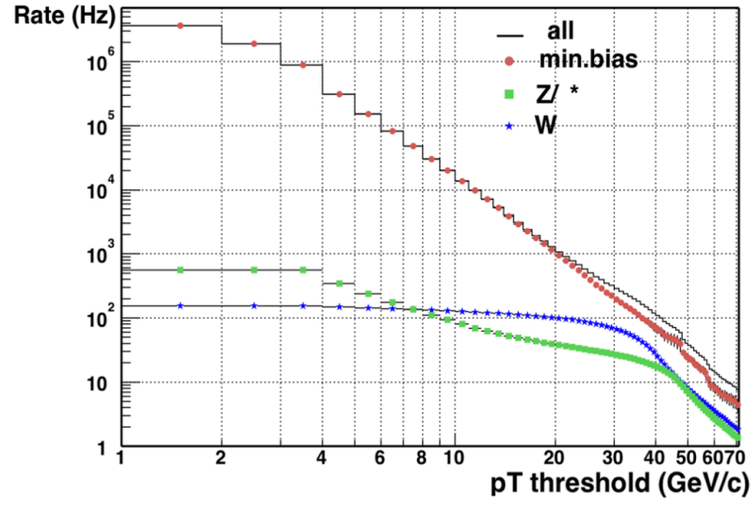
Co-authors:

Marco Lorusso (University of Bologna)
Riccardo Travaglini (INFN Bologna)
Carlo Battilana (INFN and University of Bologna)
Daniele Bonacorsi (University of Bologna)

Introduction

- Muons detection has a fundamental role in CMS: such particles are produced, in fact, by an high amount of physical processes fundamental for the High Energy Physics (e.g. Higgs Boson);
- The CMS trigger system selects events with muons, by operating cuts in *transverse momentum* (p_T), to maintain the rate of acquisition under control;

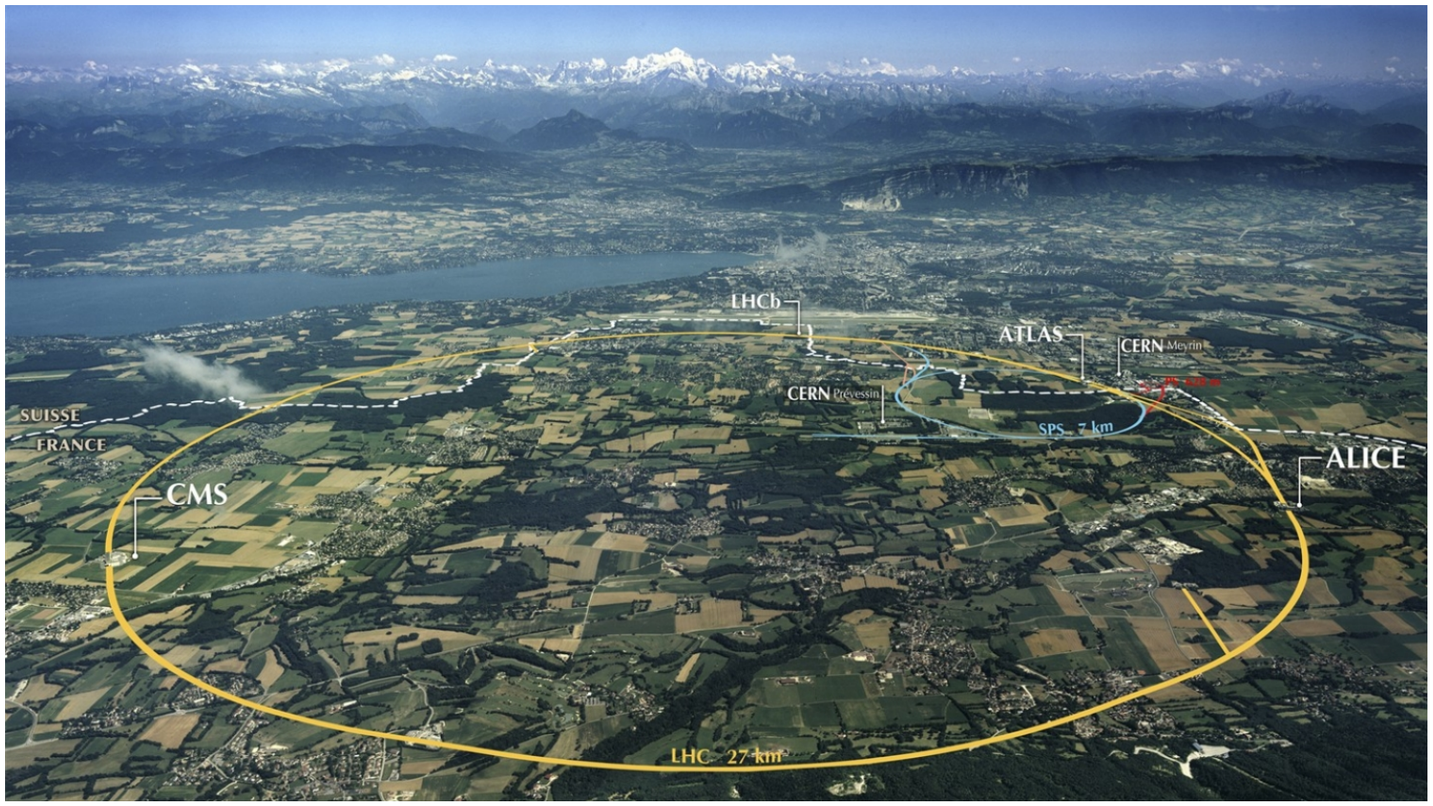




- Looking forward at the next phase of operation of the LHC, HL-LHC, new software/hardware approaches are needed to face an increase in the particle detection rate;
- Machine/Deep Learning allows to exploit all the information available at trigger level, in contrast with the limited ones used by the current algorithms.

The CMS experiment

The CMS experiment is one of the four major particle detector operating at the LHC (Large Hadron Collider) particle accelerator, at the CERN laboratories in Geneva.

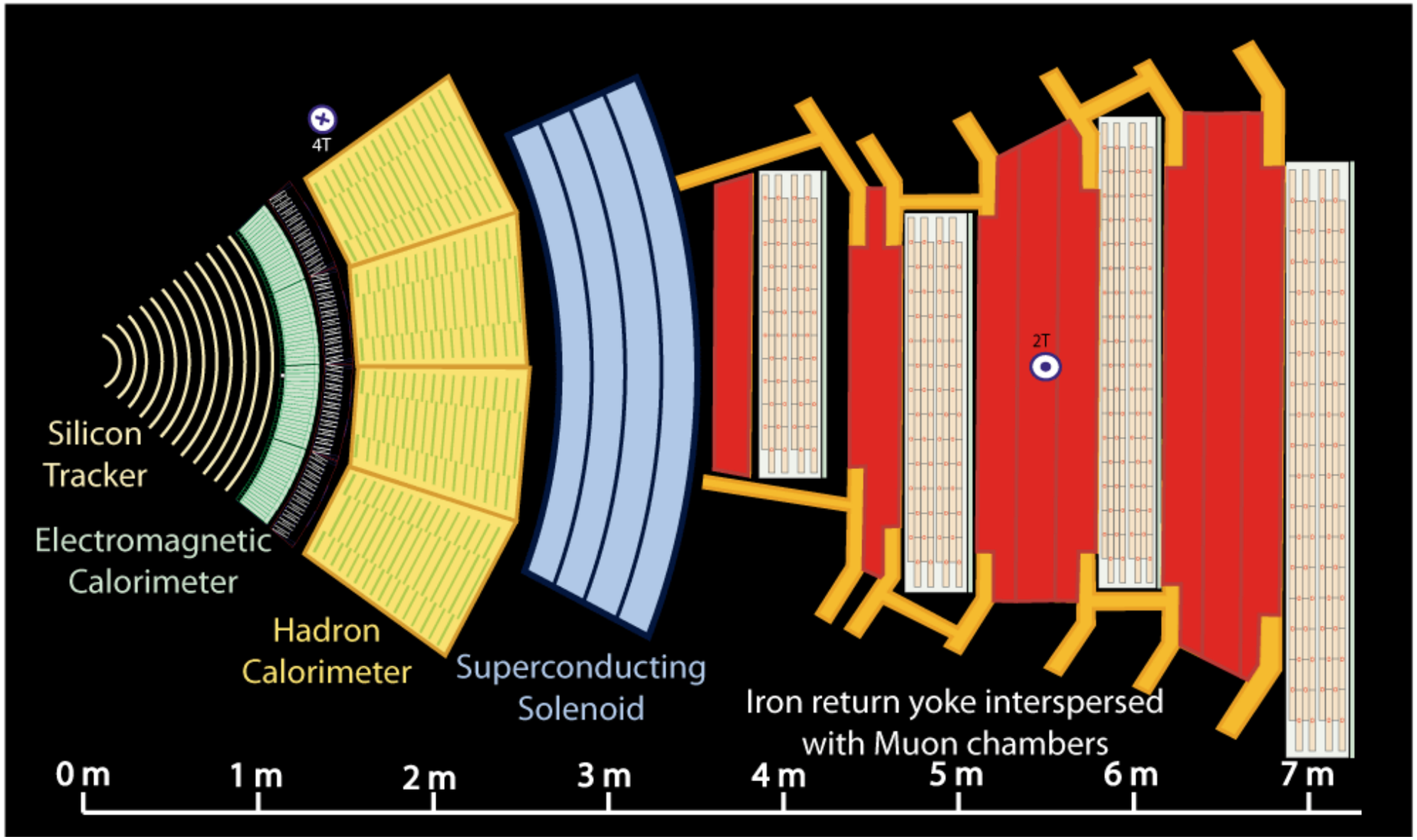


LHC accelerator complex

Length: ~21.6m
Diameter: ~15m
Weight: ~13000 tons

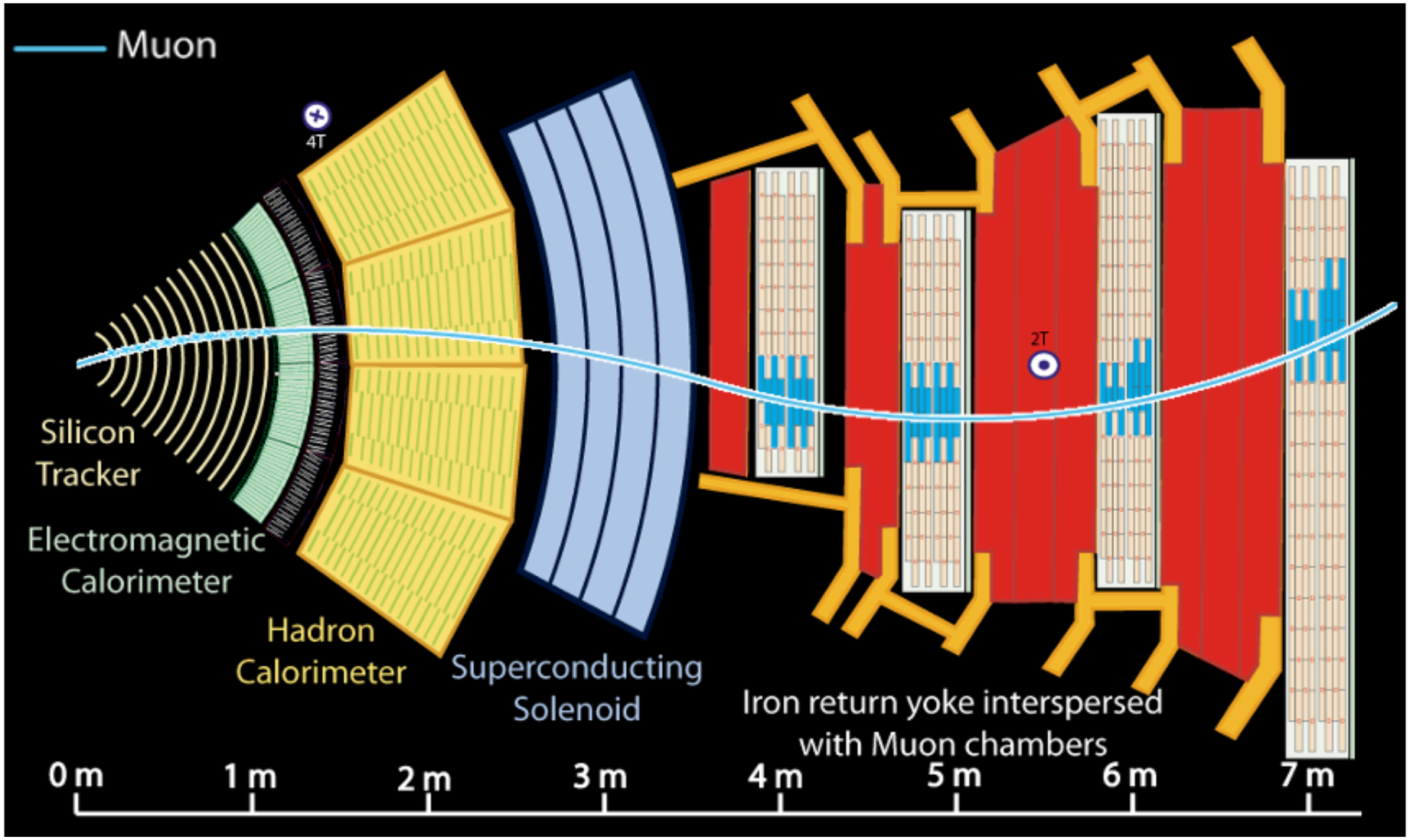
CMS transversal section

The CMS experiment



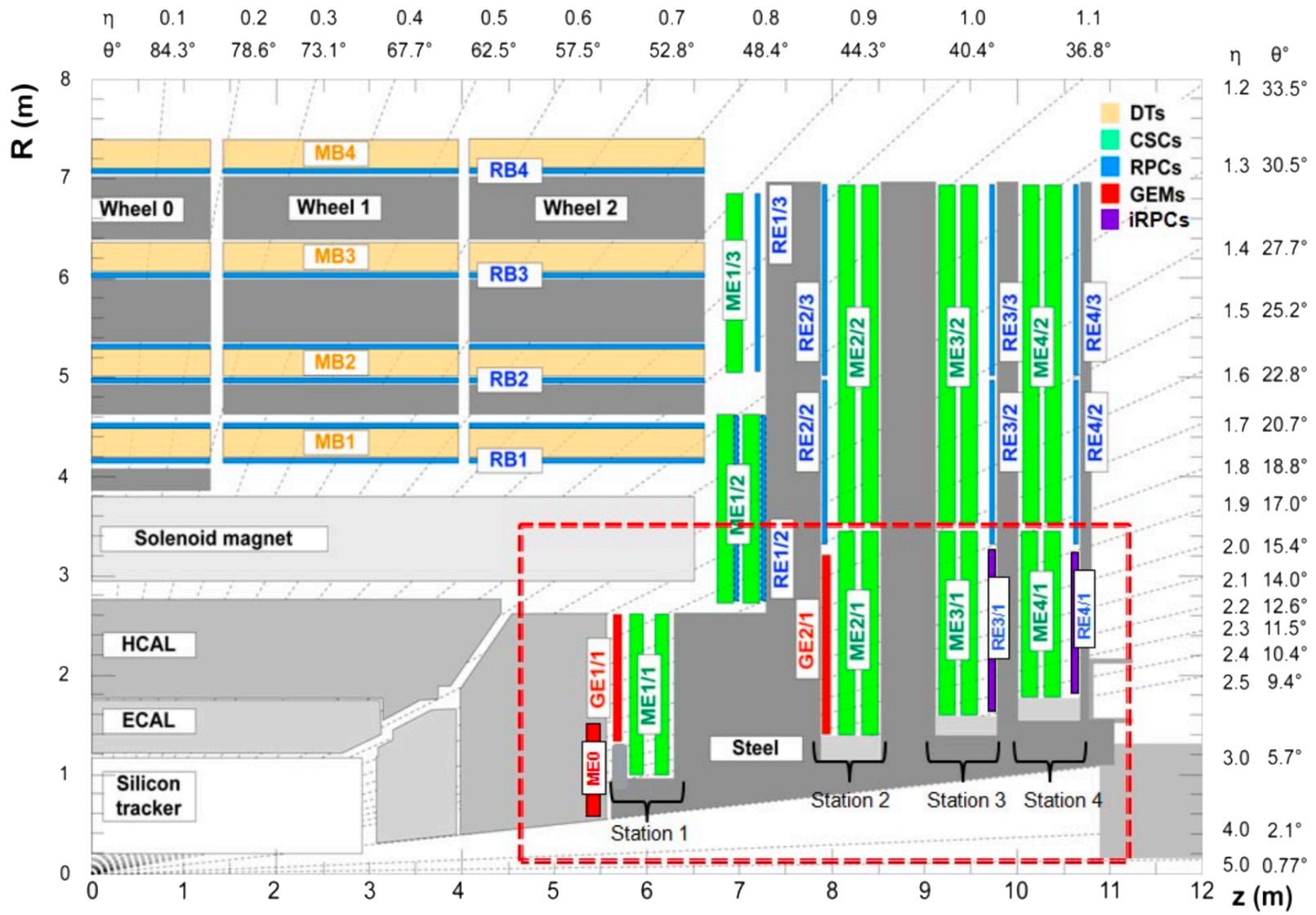
It is composed by a cylindrical barrel and two endcaps.
From inside out, different layers are placed sequentially to detect different types of particle.
A superconducting solenoid, generates a magnetic field of ~3.8T, needed to bend the trajectories of charged particles.

The CMS experiment



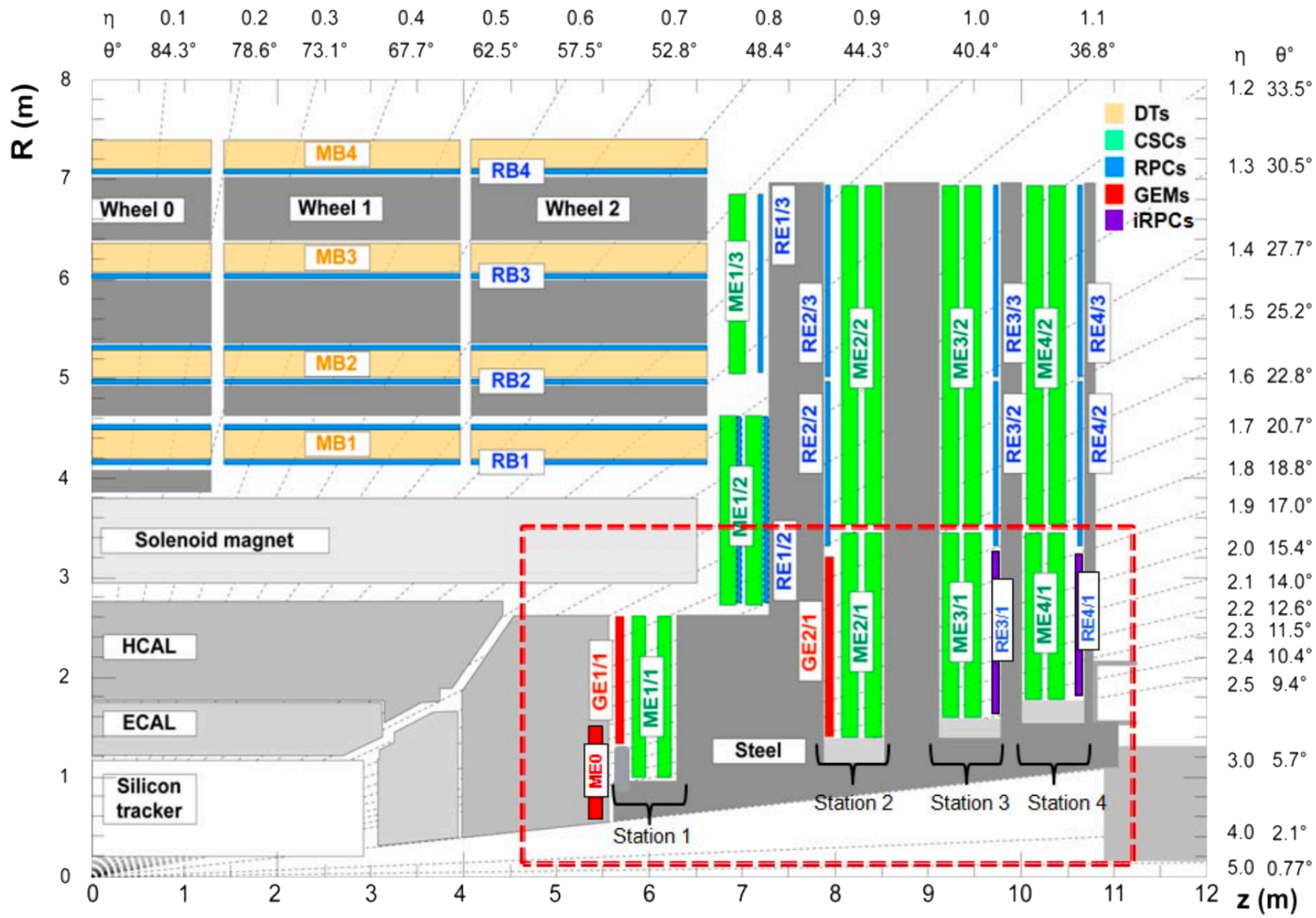
Most particles are absorbed by CMS, a few (mainly muons, neutrinos…) escape.
In particular, **muons** are charged particles (~200 times heavier than electrons) that leave a significant path inside the detector, especially in the inner track and muon chambers.

The CMS muon chambers



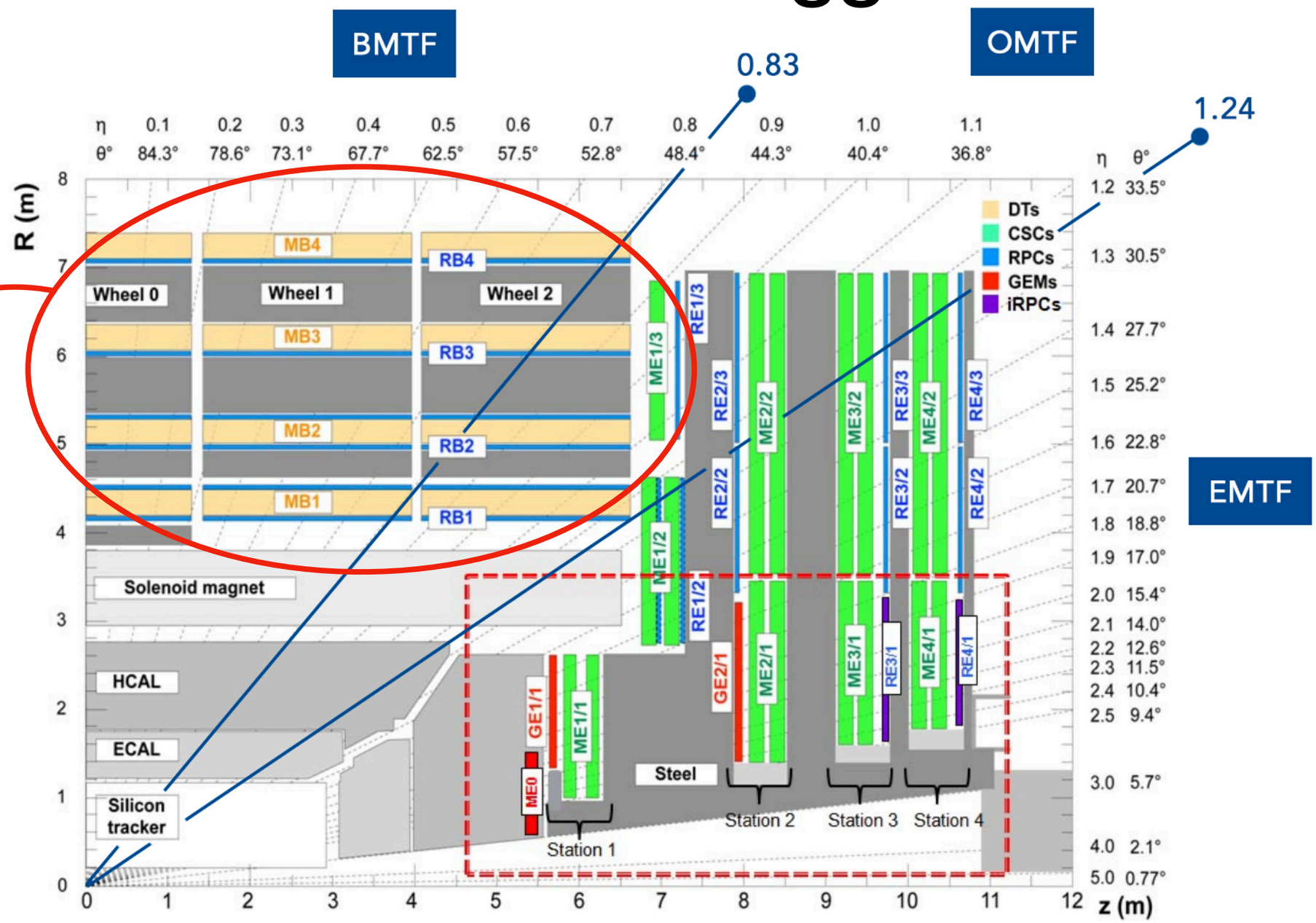
Longitudinal section of the CMS detector, showing the muon chambers.
New detectors from Phase-II contained in the red dashed box.

The CMS muon chambers



Different typologies of gaseous detectors are placed in different regions of the detector: in the barrel region Drift Tubes chambers (DT), in the endcap region Cathode Strip Chambers (CSC). In every region these chambers are interlaced with Resistive Plate Chambers (RPC). From Phase-II added Gas Electron Multiplier (GEM).

The CMS muon trigger



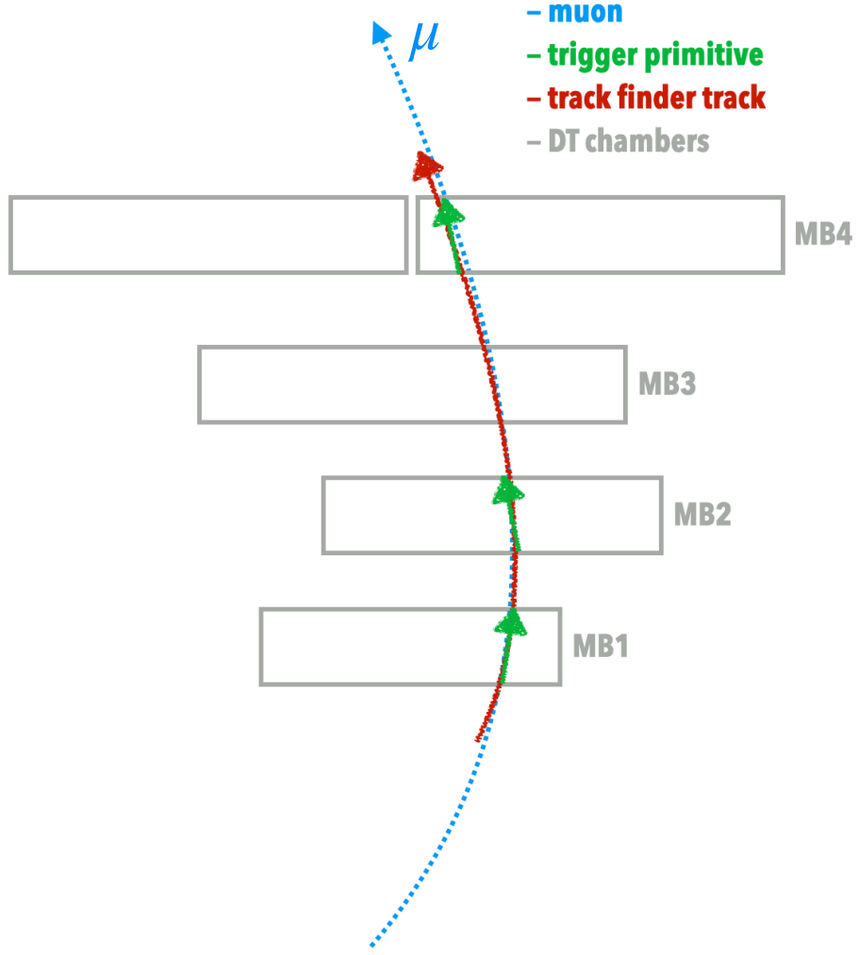
Focused for this work

Level-1 muon trigger divided into three system: **Barrel Muon Track Finder** (BMTF), Overlap Muon Track Finder (OMTF), Endcap Muon Track Finder (EMTF).

Level-1 Trigger in the barrel region

In the barrel region, information coming from the DTs and RPCs are combined to form segments of track (called *trigger primitives*). They include also some information, like: coordinates, bending angles, quality bits.

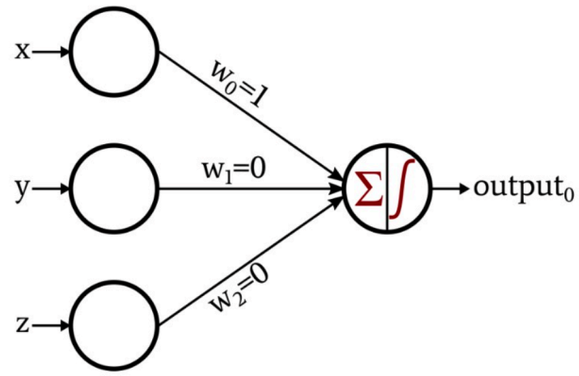
Starting from them, the Barrel Muon Track Finder builds the tracks and assigns the p_T, through precompiled tables (LUTs).



Artificial Neural Networks

An **Artificial Neural Network** (ANN) is a network vaguely inspired by the neurons in the human brain, specifically designed to tackle non-linear learning problems.

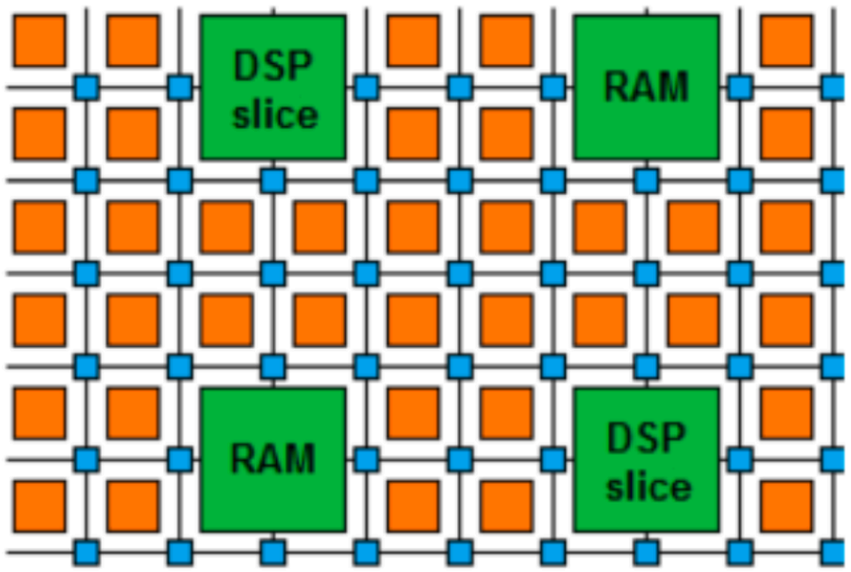
- Fully connected Multilayer perceptron (MLP) is used for this work. MLPs are made up of single units called *perceptrons*;
- All the input values (features) are multiplied by their weights and added together to create a weighted sum, which is then given to the *activation function* $f(\sum_i w_i x_i)$ to form the perceptron's output.
- Perceptrons can be stacked together to create an arbitrary number of middle layers (*hidden*), making the network deeper.



Perceptron representation

FPGAs

FPGA basic structure



FPGAs (Field Programmable Gate Arrays) are integrated circuits designed to be fully configurable by customers after manufacturing.
They are made up of replicated units of digital electronic circuits, *logic blocks*, embedded in a routing structure:

- *Lookup Tables* (LUT) for combinational logic;
- *Flip-flops* for sequential logic;
- *Digital Signal Processor* (DSP) to multiply fixed-point numbers efficiently.

Dataset and features

The dataset used for training:

- 300k MonteCarlo simulated muons (equally distributed in charge) in [3-200 GeV/c] p_T range.

A set of information is associated to the trigger primitives, used to train the NN algorithms and predict the muon p_T (in total = 27 features):

- Primitives' position (wheel, sector) for each station crossed by the particle;
- Bending angles of the primitives in global coordinates (ϕ_B);
- Trigger primitives' quality;
- ϕ variation from one station to another.

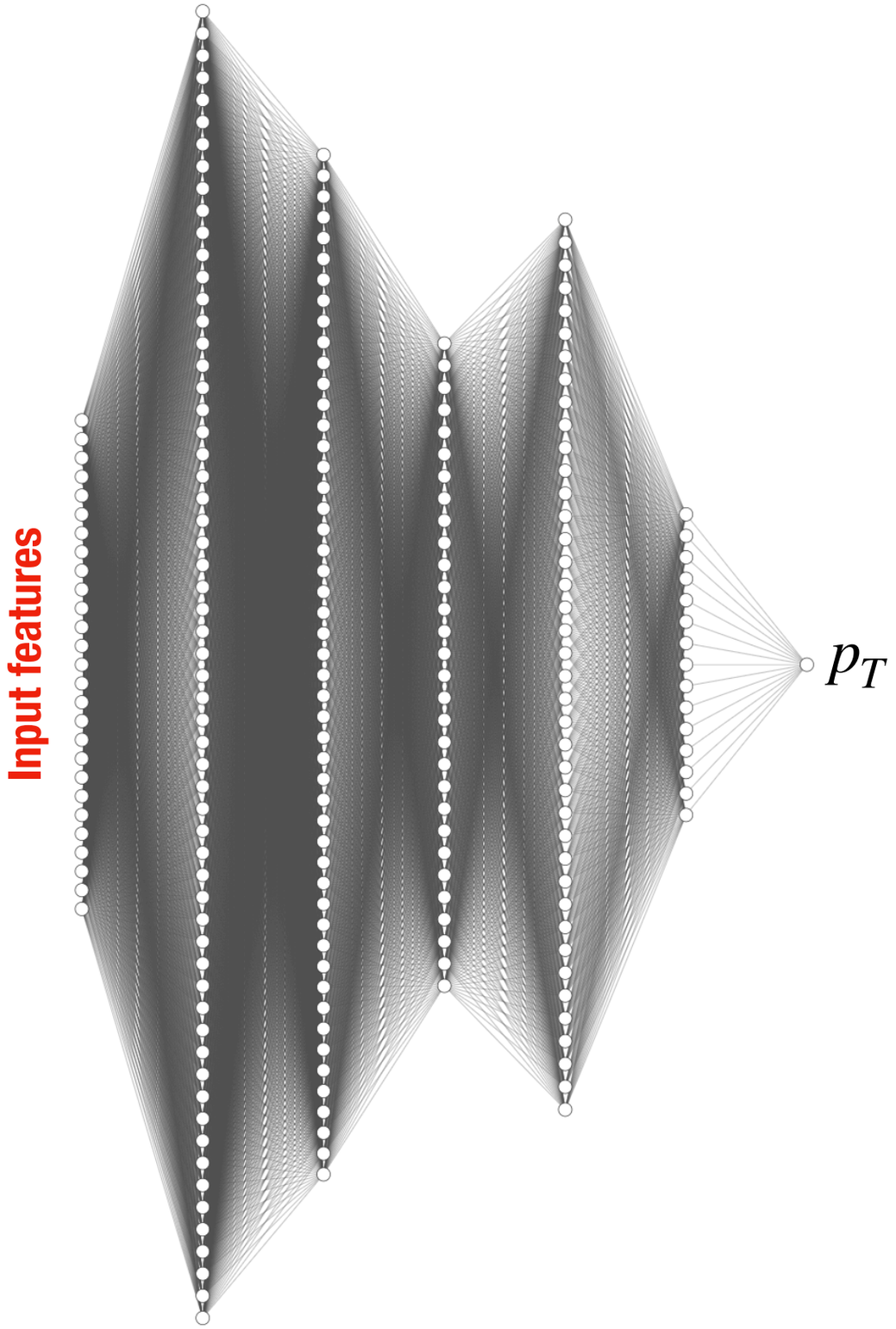
The Machine Learning model used is an Artificial Neural Network, built with the Keras framework (using Google Tensorflow v2 as a back-end).

Neural Network model

A fully connected MLP was built (later called **D model**), with the following architecture:

- Input layer of 27 nodes (features);
- 5 hidden layers with 60, 50, 30, 40, 15 node each;
- An output layer which return the p_T value.
- *ReLU* activation function for all layers;

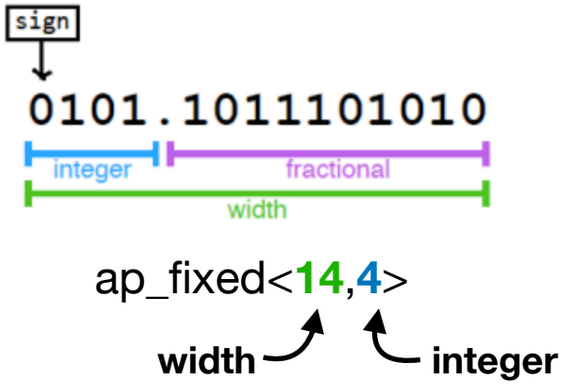
The network will perform a **regression task**, by predicting the transverse momentum p_T in a *supervised* environment, providing as label the generated momentum from the simulation.



Quantisation of the Neural Network

Operations with floating-point numbers in an FPGA require an enormous amount of resources. For this reason, the models need to be *quantised:*

- Conversion of the arithmetic used in the NN from high-precision floating-point to normalised low-precision integers at fixed-point:



The package used to train a quantised NN is called QKeras, developed from a collaboration between Google and HEP researchers. The functionalities are basically the same provided by the classic Keras, with the main difference of using fixed point arithmetic.

The quantisation is performed **during training** itself, and not at the end by lowering the numbers bitwidth.

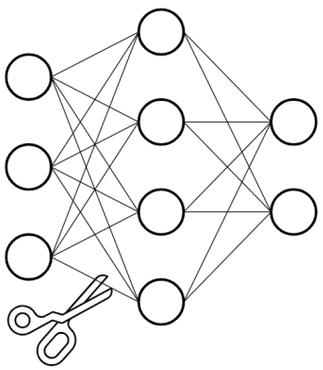
Neural Network weight pruning

Another important aspect to consider when building a Neural Network model on FPGA, is the elimination of unnecessary values from the weight tensor, in order to reduce the number of operations with less resource consumption.

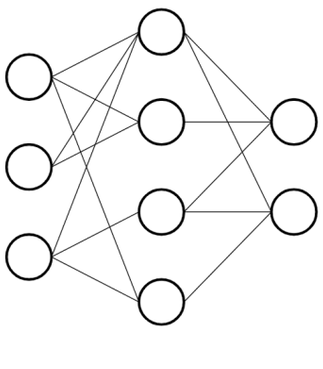
This is performed by *pruning* the weights with the lowest magnitude i.e. closest to zero, until reaching a quota defined by user.

- TensorFlow Sparsity Pruning API was adopted, setting to zero several weights thus resulting in a connection ‘cut’ between different neurons of the Neural Network.

This operation allows the network to be more light and, therefore, more resource-friendly.



Before pruning



After pruning

	Weights		Biases		Total	
	Complete	Pruned	Complete	Pruned	Complete	Pruned
1^{st} layer	1620	409	60	60	1680	469
2^{nd} layer	3000	750	50	50	3050	800
3^{rd} layer	1500	375	30	29	1530	404
4^{th} layer	1200	300	40	40	1240	340
5^{th} layer	600	150	15	15	615	165
Output layer	15	4	1	1	16	5

Results plots

To compare the predictions obtained with the p_T values assigned by the Level-1, two kinds of plot has been produced:

- p_T resolution histogram:

$$\frac{\Delta p_T}{p_T} = \frac{p_{T_{est}}^{ML,L1} - p_{T_{sim}}}{p_{T_{sim}}}$$

where:

$p_{T_{est}}$: transverse momentum estimate, given from the model predictions or the Level-1 trigger;

$p_{T_{gen}}$: simulated value of the transverse momentum.

- efficiency curves (turn-on):

$$\epsilon = \frac{\text{Number of } \mu > \text{given threshold in } p_T}{\text{Total number of muons}}$$

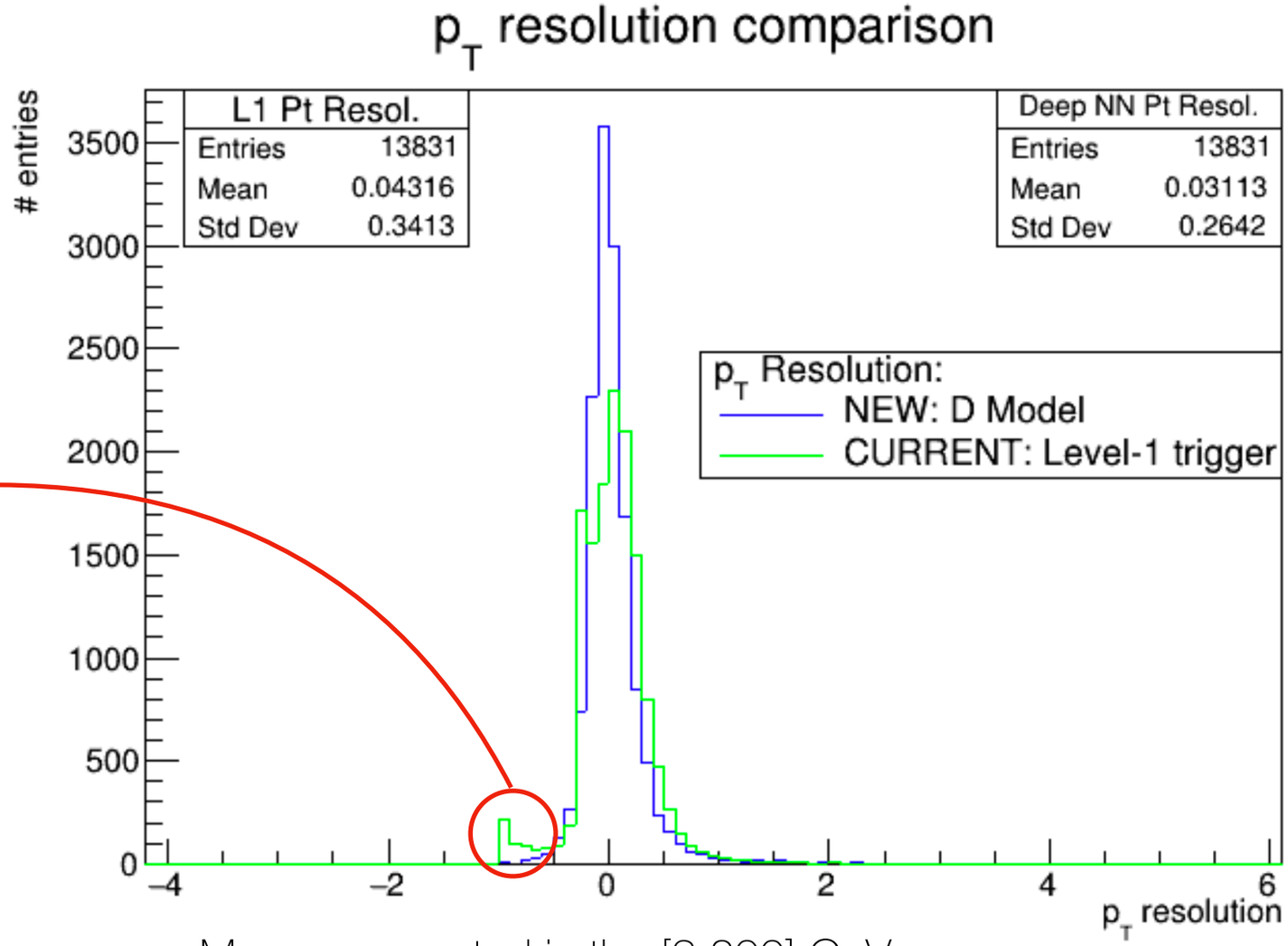
Only muons from which a track is reconstructed from BMTF are considered.
The results are shown as a function of generated p_T.

Preliminary check: p_T resolution on CPU

$$\frac{\Delta p_T}{p_T} = \frac{p_{T_{est}}^{NN,L1} - p_{T_{sim}}}{p_{T_{sim}}}$$

Peak at -1 when:

$$p_{T_{est}}^{L1} = 0 \ , \ \forall p_{T_{sim}}$$



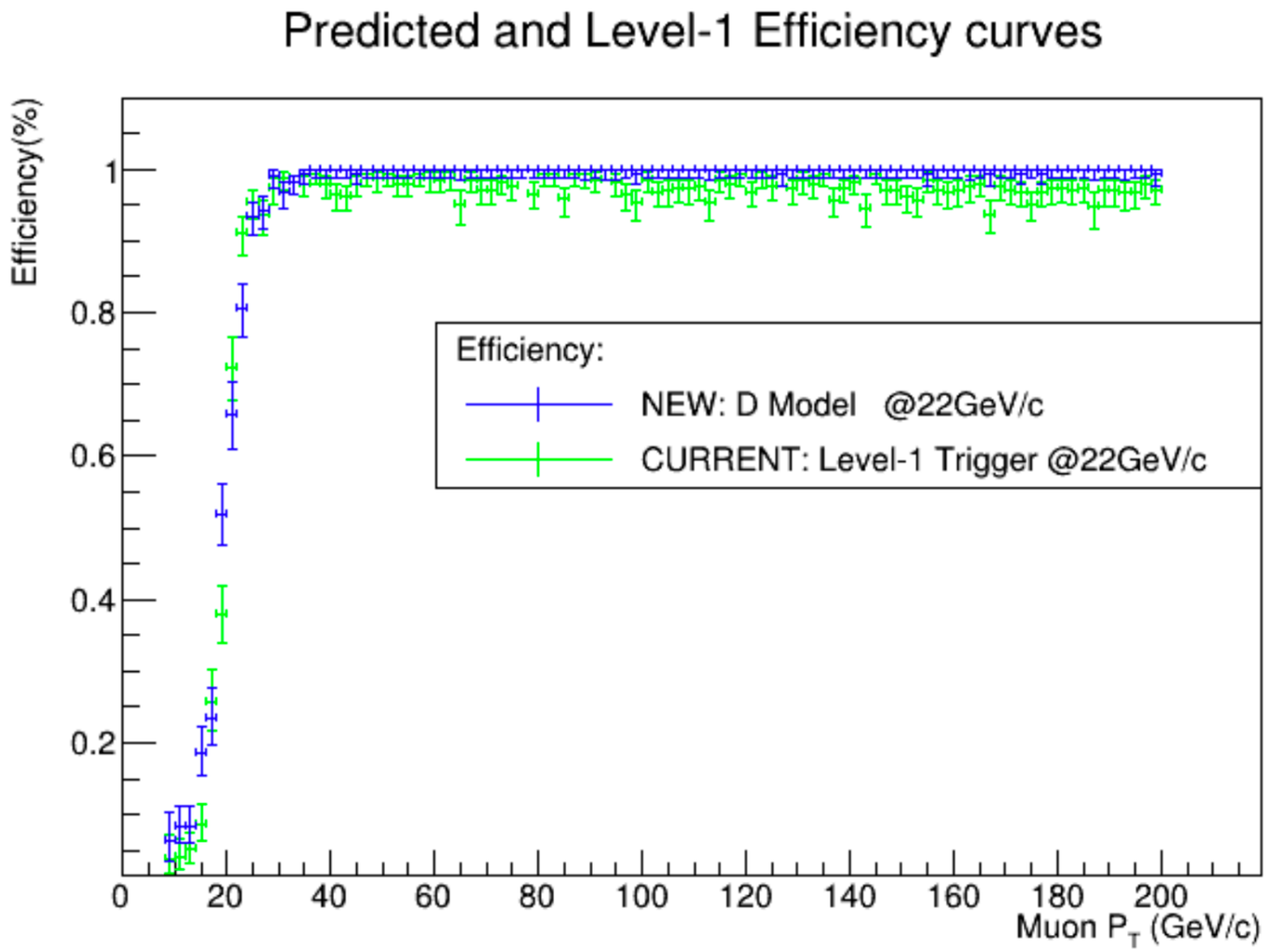
Muons generated in the [3,200] GeV range.

The Neural Network (D model) resolution has a narrower distribution with respect to the L1T. Also, the NN momentum assignment is less prone to large p_T underestimation.

Preliminary check: turn-on efficiency on CPU

$$\epsilon = \frac{\textbf{Number of } \mu > \textbf{given threshold in } p_T}{\textbf{Total number of muons}}$$

- Plateau region: very high efficiency of the NN model, higher w.r.t. Level-1 trigger.
- Low p_T region (10-20 GeV): the NN model has a small increase of efficiency, compared to Level-1, resulting in greater number of low-p_T muons wrongly identified as high-p_T.

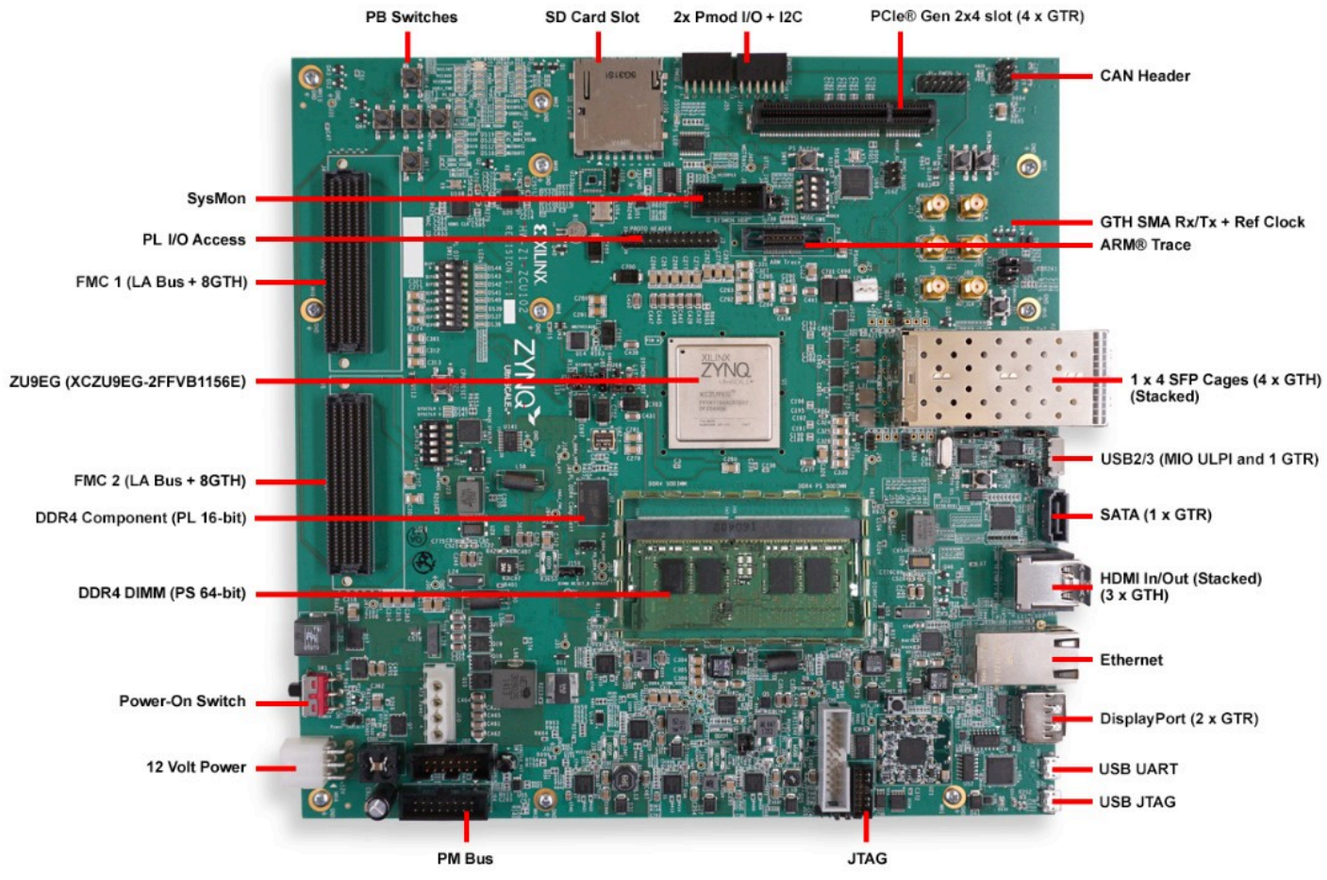


Efficiency turn-on given a p_T threshold of 22 GeV (lowest p_T - cut single muon triggers over Run-2).

Test board characteristics

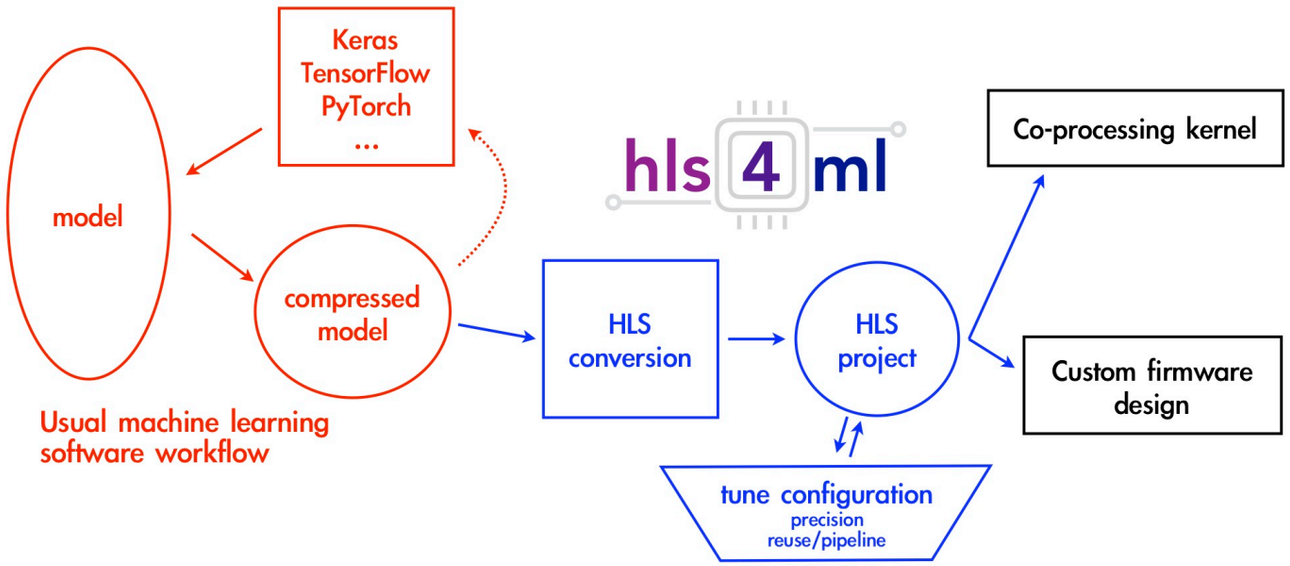
The target hardware consisted in a Xilinx ZCU102 Evaluation Board featuring a Zynq Ultrascale+ MPSoC (Multiprocessor System on a Chip).

- The *Programmable Logic* (PL) houses around 600k logic cells (flip-flops and LUTs) and 2520 DSP slices working with an internal memory of ~32Mb;
- The *Processing System* (PS), consists in a quad-core Arm Cortex-A53, a dual-core Cortex-R5F real-time processor, and a Mali-400 GPU, allowing fast development e.g. I/O interface manageable via software (SDK).



From QKeras to HLS: hls4ml

- **hls4ml** is a package developed by member of the HEP community (link to repo) to translate Machine Learning algorithms into HLS (High-Level Synthesis) code;
- The translation of Python objects into HLS is done by hls4ml as part of an automatic workflow saving the time needed to convert the entire neural network into hardware readable code.

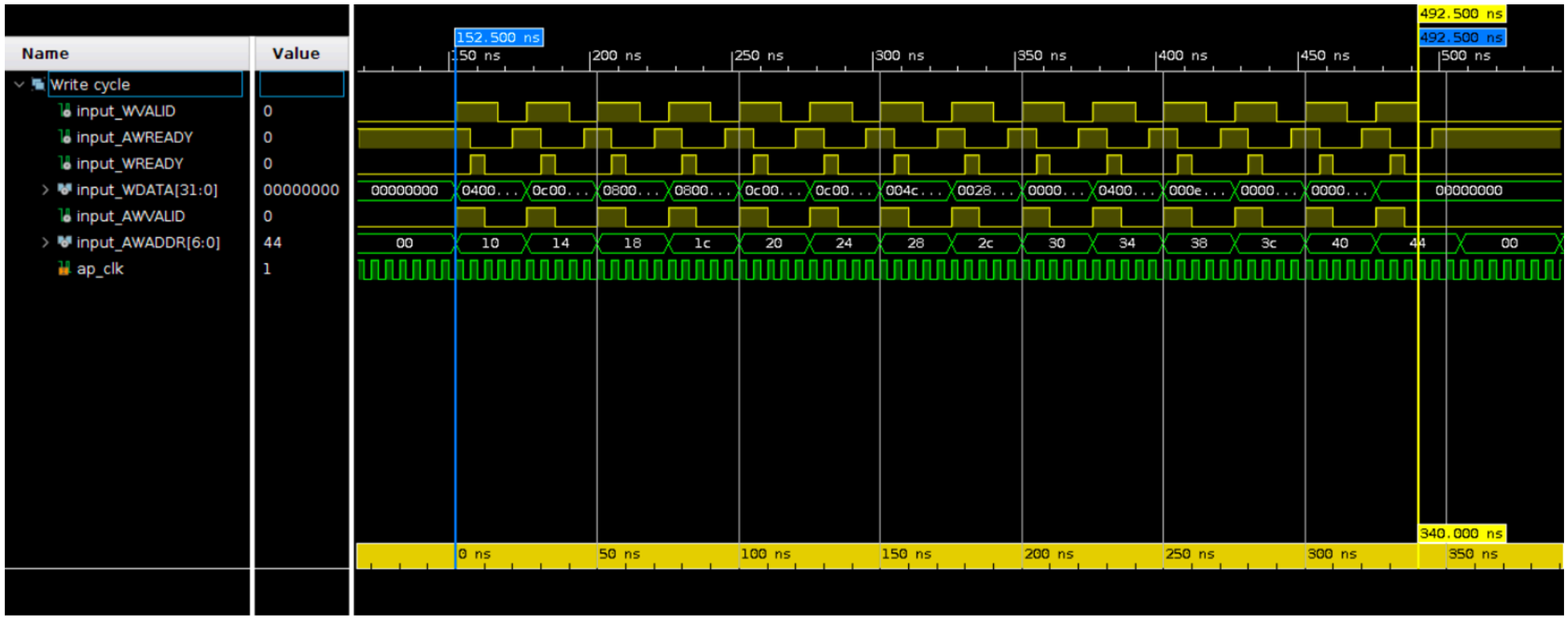


- Ability for the user to control the level of parallelisation, by changing a simple parameter (*reuse factor*).
- This tool is also fully compatible with QKeras, for quantisation aware training.

Model implementation on FPGA

The Neural Network is then translated into an HLS project and synthesised using Vivado HLS.

The I/O interface used is the AXI4-Lite (Advanced eXtensible Interface), allowing a simpler interface than the full AXI4 interface but making the I/O operation slower due to the handshaking protocol reading/writing Axi registers.

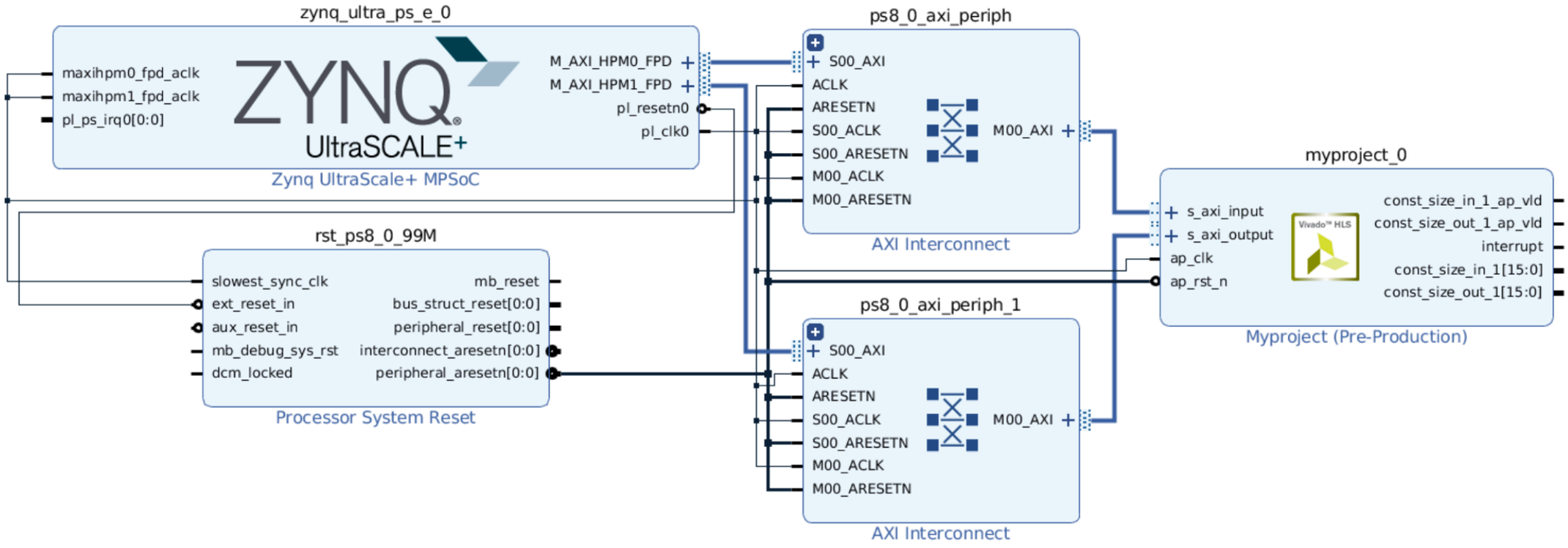


Simulation of write cycle of the 27 features in input for a single muon entry through AXI4-Lite interface between the PS and the PL, taking about 500ns.

Implementing a block design

In order to implement the chosen NN model in a complete design, the project is exported via the Vivado IP (Intellectual Property) Packager. The IP is then added to the Vivado IP Catalog.

Vivado IDE (Integrated Design Environment) provides an user interface with graphic connectivity to select IPs, configure the hardware settings and stick together the IP blocks to create the digital system.



Block design showing the IP core containing the NN (on the right), connected to the PS (on the left), through the I/O interface.

Post synthesis report

After synthesising the project, performance and utilisation estimates can be analysed in the *post-synthesis* report:

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	4	-
FIFO	-	-	-	-	-
Instance	0	432	34523	135580	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	51	-
Register	-	-	6903	-	-
Total	0	432	41426	135635	0
Available	1824	2520	548160	274080	0
Utilization (%)	0	17	7	49	0

The model chosen for this analysis, therefore, have been proven to be suitable for implementation in the target hardware available (Xilinx ZCU102 Evaluation Board).

Inference on the FPGA

After the synthesis of the model, the bitstream could be generated and copied onto the FPGA in order to program it.
An independent **test set**, kept for inference, was then converted in the same format as the NN input: **ap_fixed<16,6>**.

The computation on the FPGA have been performed using Vivado SDK, calling functions in C++, compiled on the host PC that drive the execution into the PL.

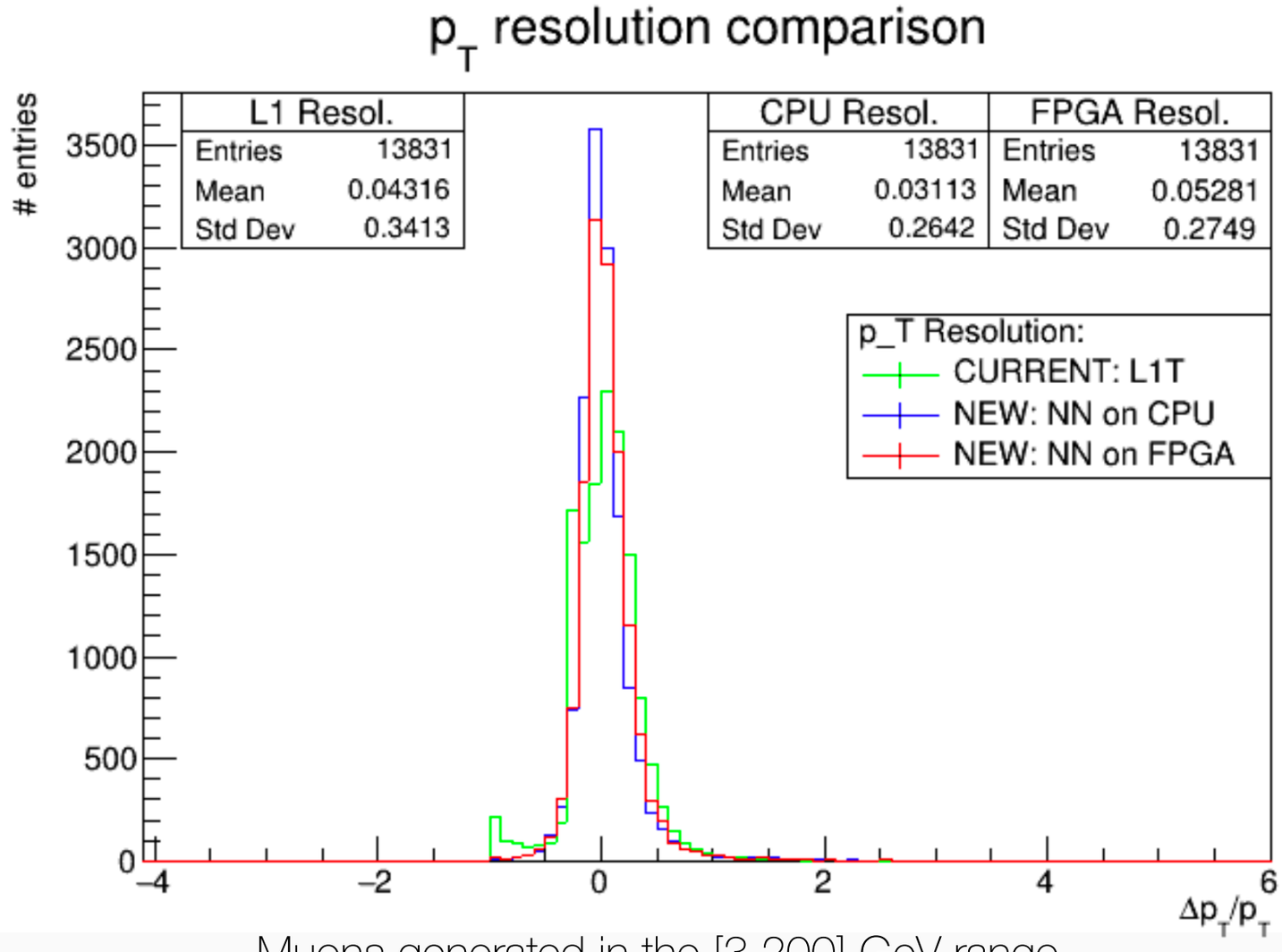
Output: 17 bits fixed-point numbers with 2 bits for sign and integer representation.

The structure is then enclosed in a loop, running on the entire test dataset. The output is then retrieved and re-converted - via python script - back to floating point for the results investigation.

Results: p_T resolution on FPGA

$$\frac{\Delta p_T}{p_T} = \frac{p_{T_{est}}^{NN,L1,FPGA} - p_{T_{gen}}}{p_{T_{gen}}}$$

- Resolution on FPGA is slightly broader than on the CPU with a small bias towards higher values of resolution.
- This small decrease in performance (compared to CPU inference), is likely caused to the effect of the loss in precision undergone by the input features, from floating-point to fixed-point.



Muons generated in the [3,200] GeV range.

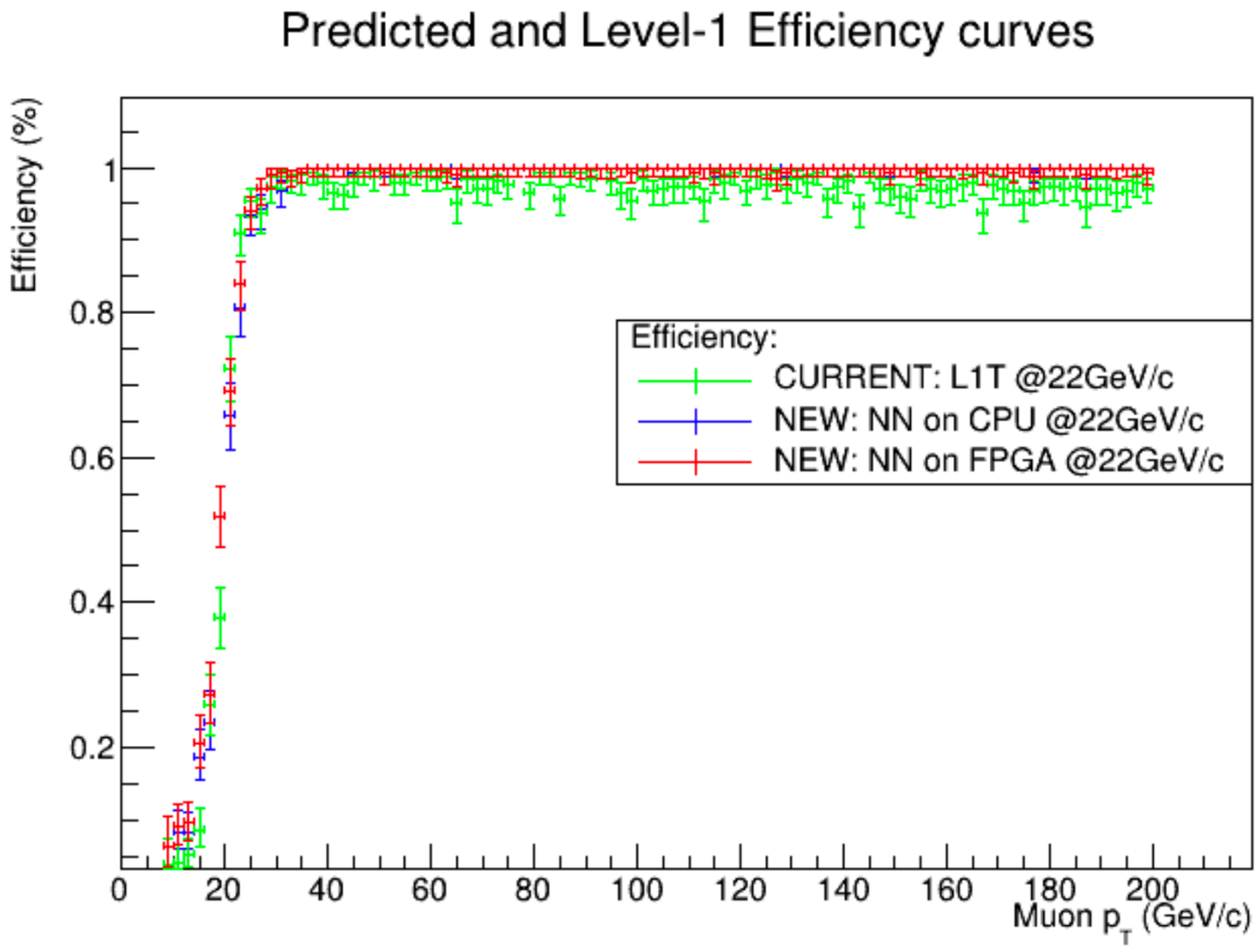
Results: Turn-on efficiency on FPGA

$$\epsilon = \frac{\textbf{Number of } \mu > \textbf{given threshold in } p_T}{\textbf{Total number of muons}}$$

- Plateau region: very high efficiency of the NN model w.r.t. Level-1 trigger also in the FPGA inference (close to 100%).

- Low p_T region (10-20 GeV): similarly to the CPU inference, the small p_T range has an higher efficiency, higher w.r.t. Level-1 trigger.

  This wrong behaviour, that implies an higher selection rate, must be tackled in the next development stages.



Efficiency turn-on given a p_T threshold of 22 GeV (lowest p_T - cut single muon triggers over Run-2).

Latencies

- The main advantage of switching to FPGA, is the reduction of the time needed for a single prediction.

- This information was evaluated by counting the number of clock pulses between the input of a pattern and the production of the related output.

- The model took about 74 clock cycles (**corresponding to** $\approx 0.368\ \mu s$ with a **200MHz** clock frequency) for each candidate on the FPGA. Just as a reference, CPU computation takes around ~ 40 ms for a single prediction.

- The FPGA is, however, comparable with the time needed by the KBMTF (Kalman Barrel Muon Track Finder) at Level-1 to reconstruct a track, with great margins of improvement.

Next steps

- As mentioned, the higher efficiency at low-p_T causes an higher rate of muons that are wrongly identified as high-p_T.
  This issues needs to solved at software level, by building a more fine-tuned NN in the future.
  - e.g. consider not only the p_T but also q/p_T, since the momentum is inversely proportional to the curvature angle of the muon's trajectory. → More weight to low-p_T muons during training.

- Improving the quantisation might improve FPGA performances:
  - e.g. different bitwidth, or new activation functions as well as energy consumption optimisation.

- Implement a new evolved I/O interface to send and retrieve data from the NN, reducing time for computation.

- Try with more performing FPGAs: newly installed Vadatech ATC36 board hosting a Xilinx Virtex-7 FPGA, mounted on an ATCA crate inside the INFN-CNAF Tier-1 Data Center in Bologna.

Conclusions

- This work represents a first stepping stone towards the implementation of a Neural Network in a FPGA hardware, for the muon p_T assignment at Level-1 in the Barrel region of the CMS muon chambers;

- Using a quantised version of Keras called QKeras, a quantised Neural Network was created using fixed-point arithmetics.
  Preliminary results on CPU: better p_T assignment at high momentum regions with an increased selection efficiency at low-p_T;

- *hls4ml* $\rightarrow$ convert the network in a HLS project, then synthesised in the FPGA;

- The inference of the model on the FPGA shows an agreement in terms of p_T resolution and trigger efficiency w.r.t. CPU, with some approximations due to the different arithmetic used;

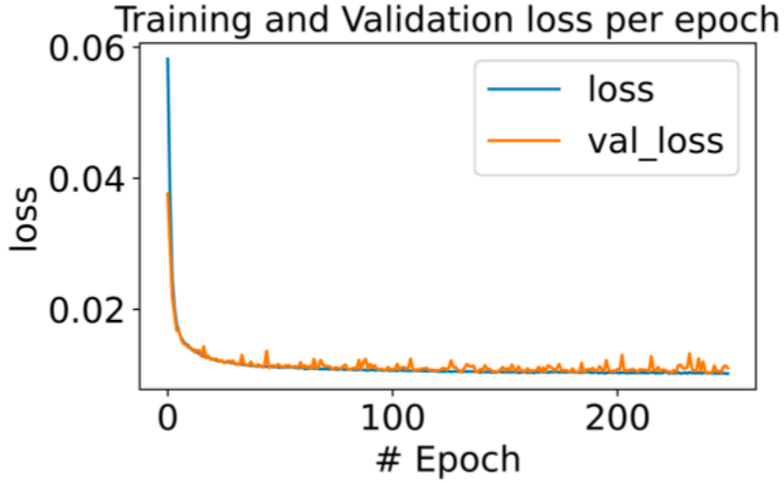
- The inference time is comparable to the actual Level-1 KBMTF trigger.

Backup

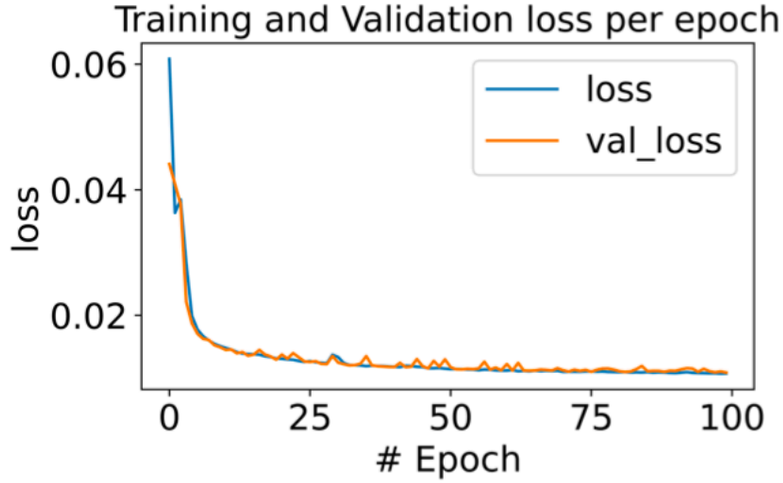
NN training

- 130 epochs with a batch size of 300 events

	Training RMSE	Validation RMSE
S model	0.102409	0.105033
D model	0.109454	0.112008



(a) S model loss values.



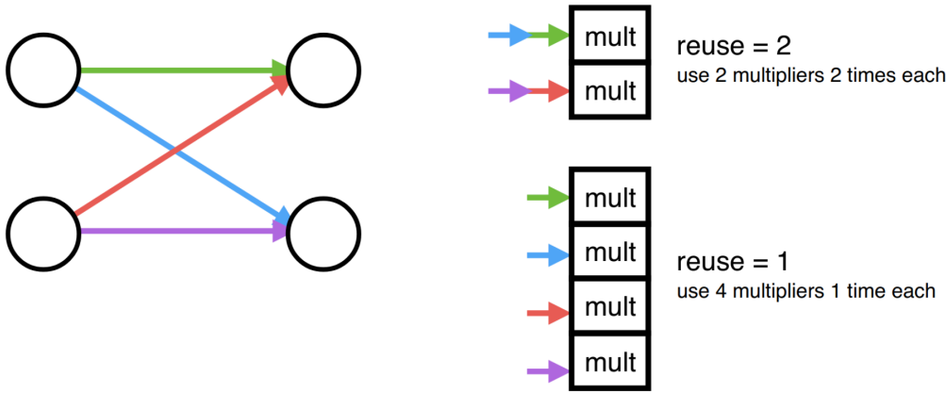
(b) D model loss values.

Reuse factor

hls4ml has a parameter called *reuse factor* that controls the parallelisation or serialisation of the model implementation.

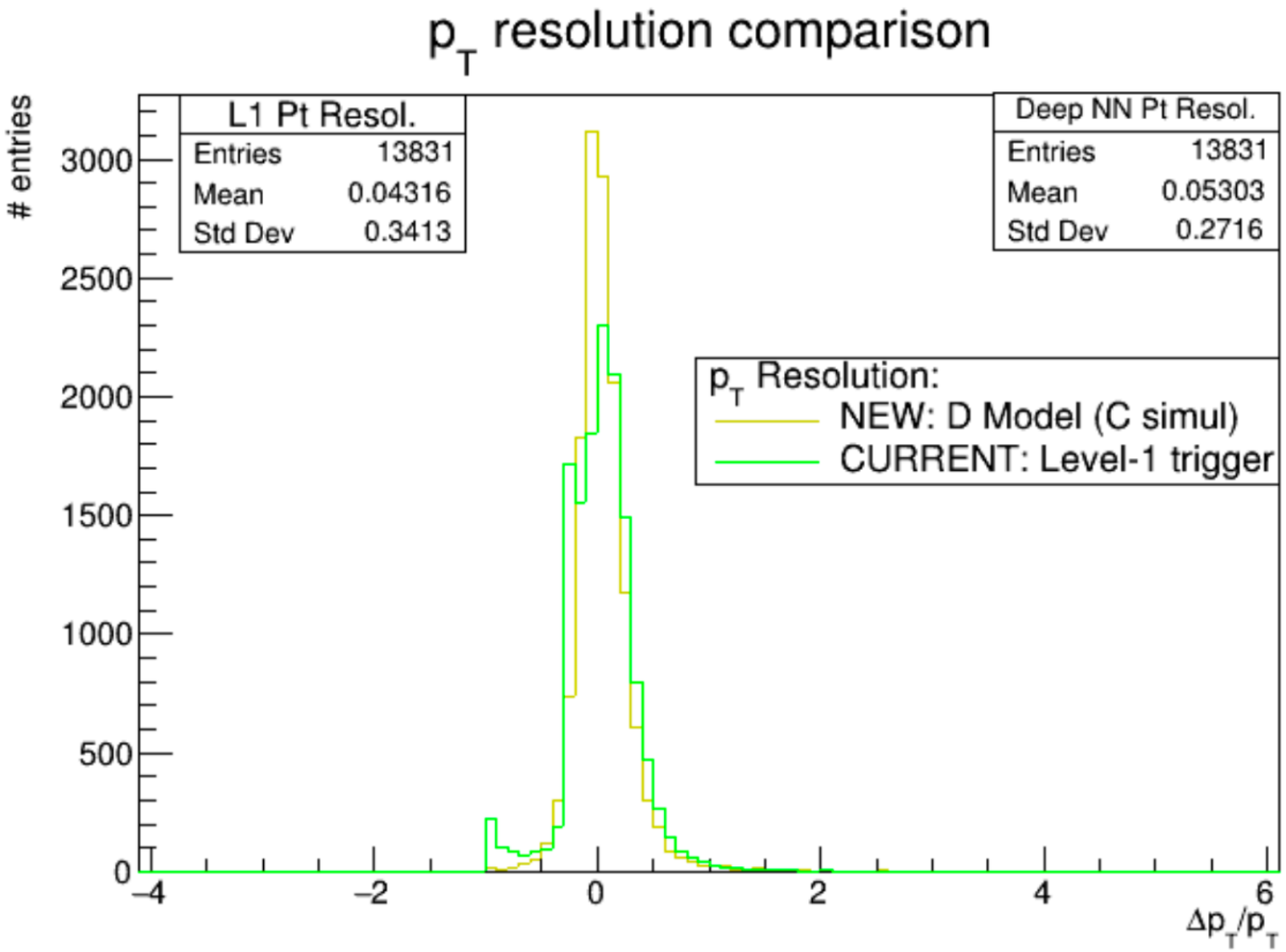
- When this factor is 1, the model is fully parallel exploiting more resources with minor latencies;
- When this factor is >1, the model is more serial with the reuse of the same resource (increasing latency).

For our study, maximum reuse was selected. However, by setting no reuse the following resource table is obtained:



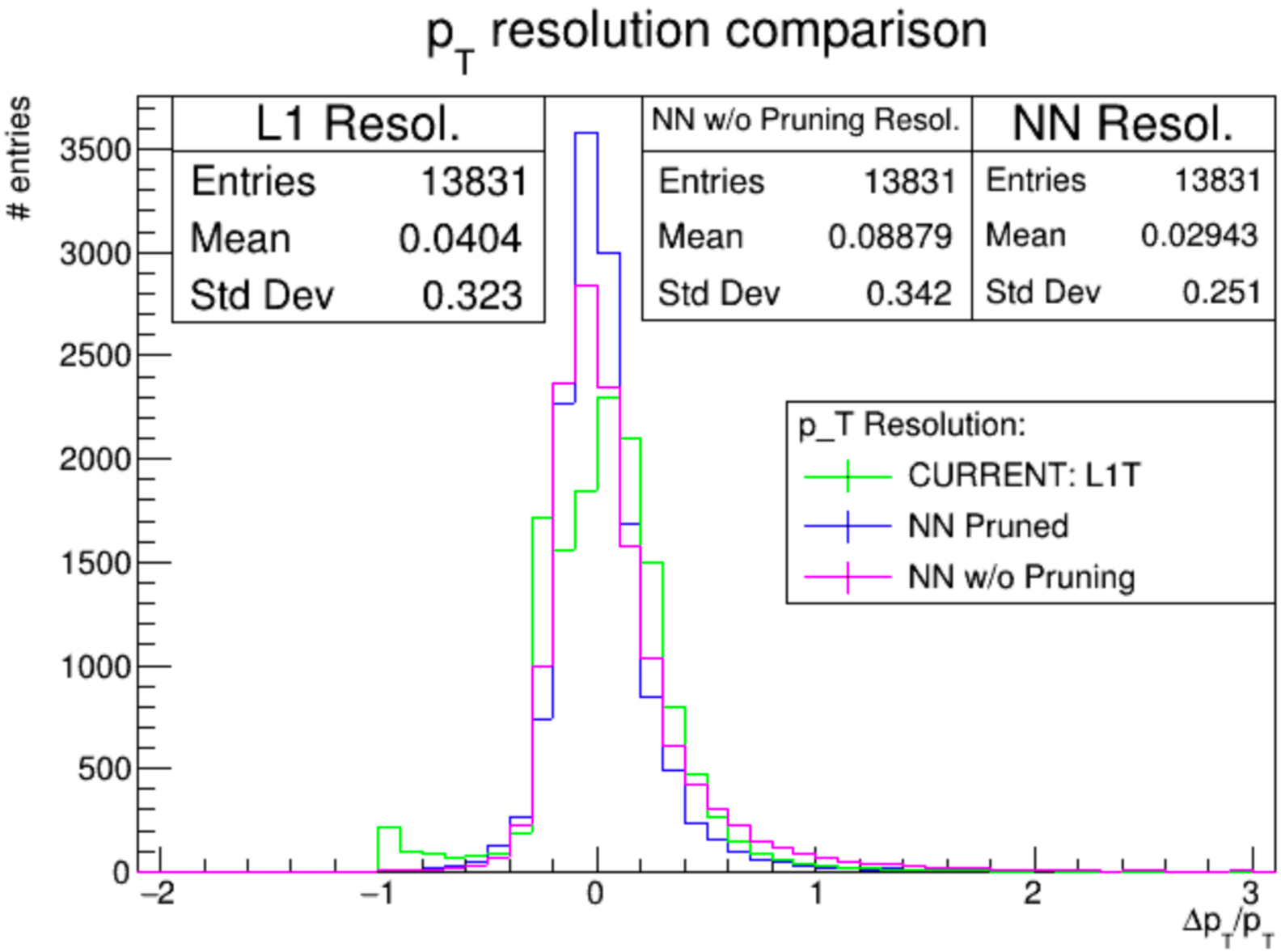
Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	4	-
FIFO	-	-	-	-	-
Instance	0	2149	34909	109557	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	-	-
Register	-	-	5679	-	-
Total	0	2149	40588	109561	0
Available	1824	2520	548160	274080	0
Utilization (%)	0	85	7	39	0

p_T resolution FPGA simul.



FPGA software simulation of the model

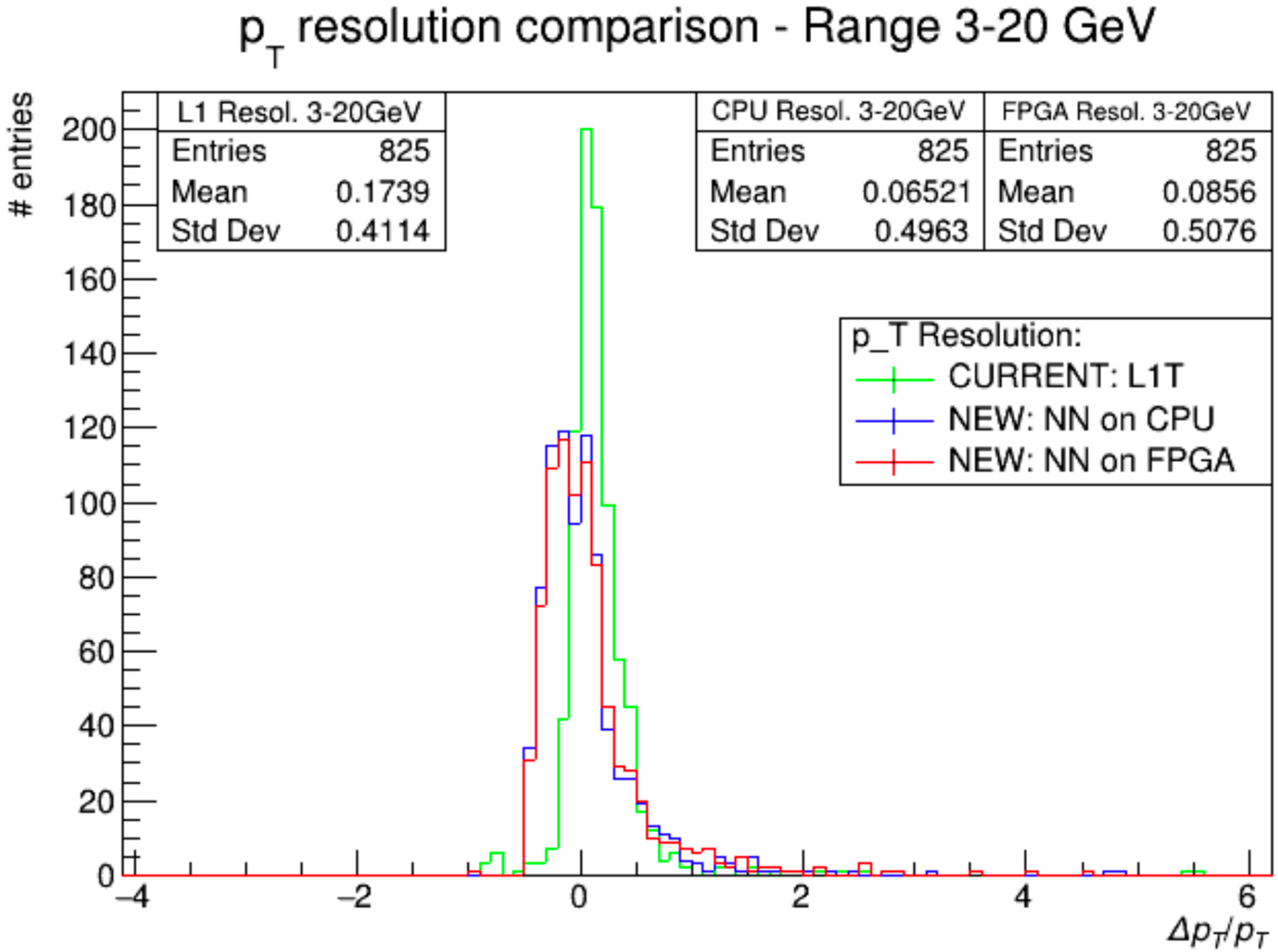
p_T resolution on CPU



Comparison between Pruned NN and non-Pruned NN

p_T resolution on FPGA

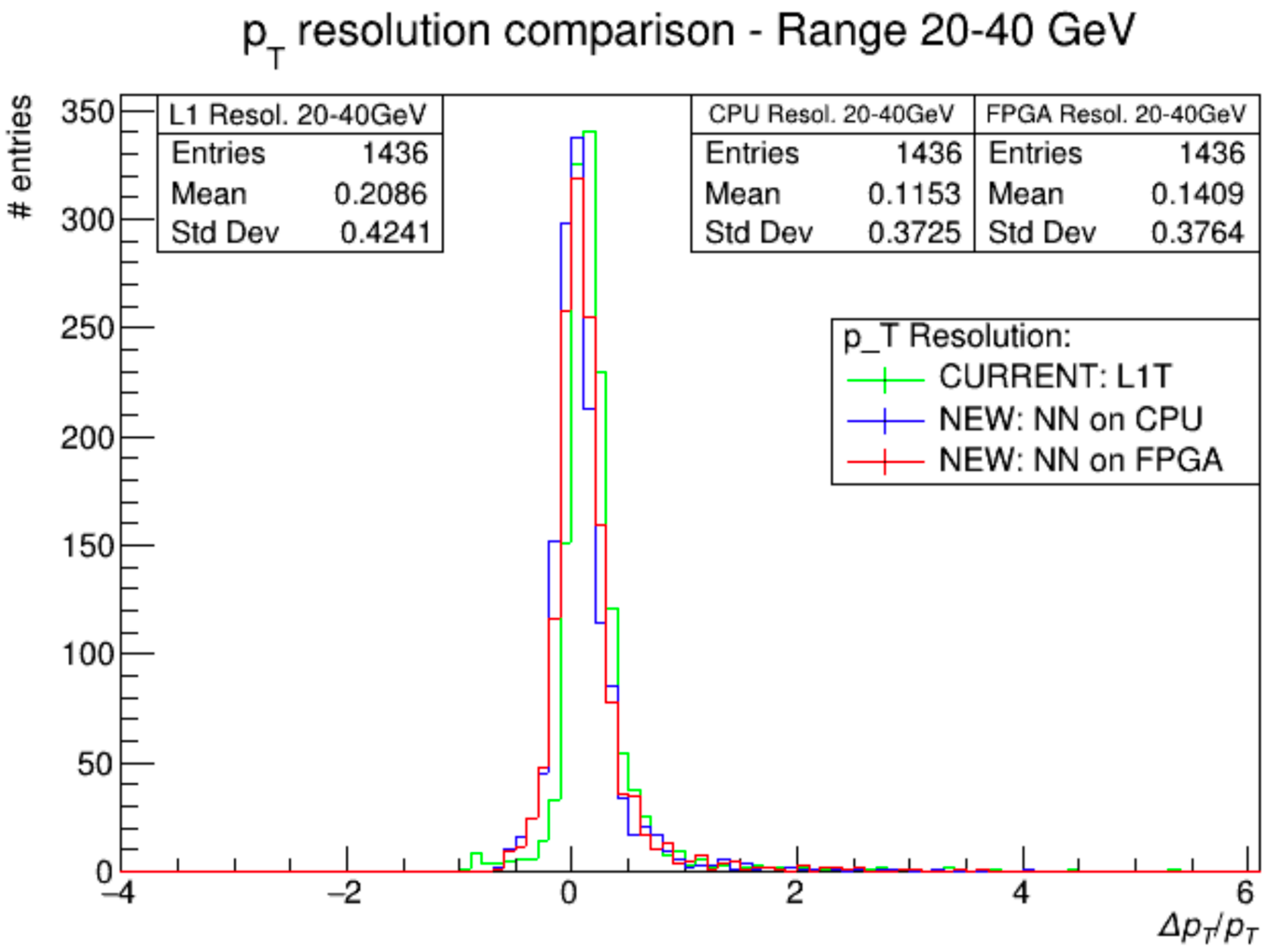
$$\frac{\Delta p_T}{p_T} = \frac{p_{T_{est}}^{NN,L1,FPGA} - p_{T_{sim}}}{p_{T_{sim}}}$$



Muons generated in the [3,20] GeV range.

p_T resolution on FPGA

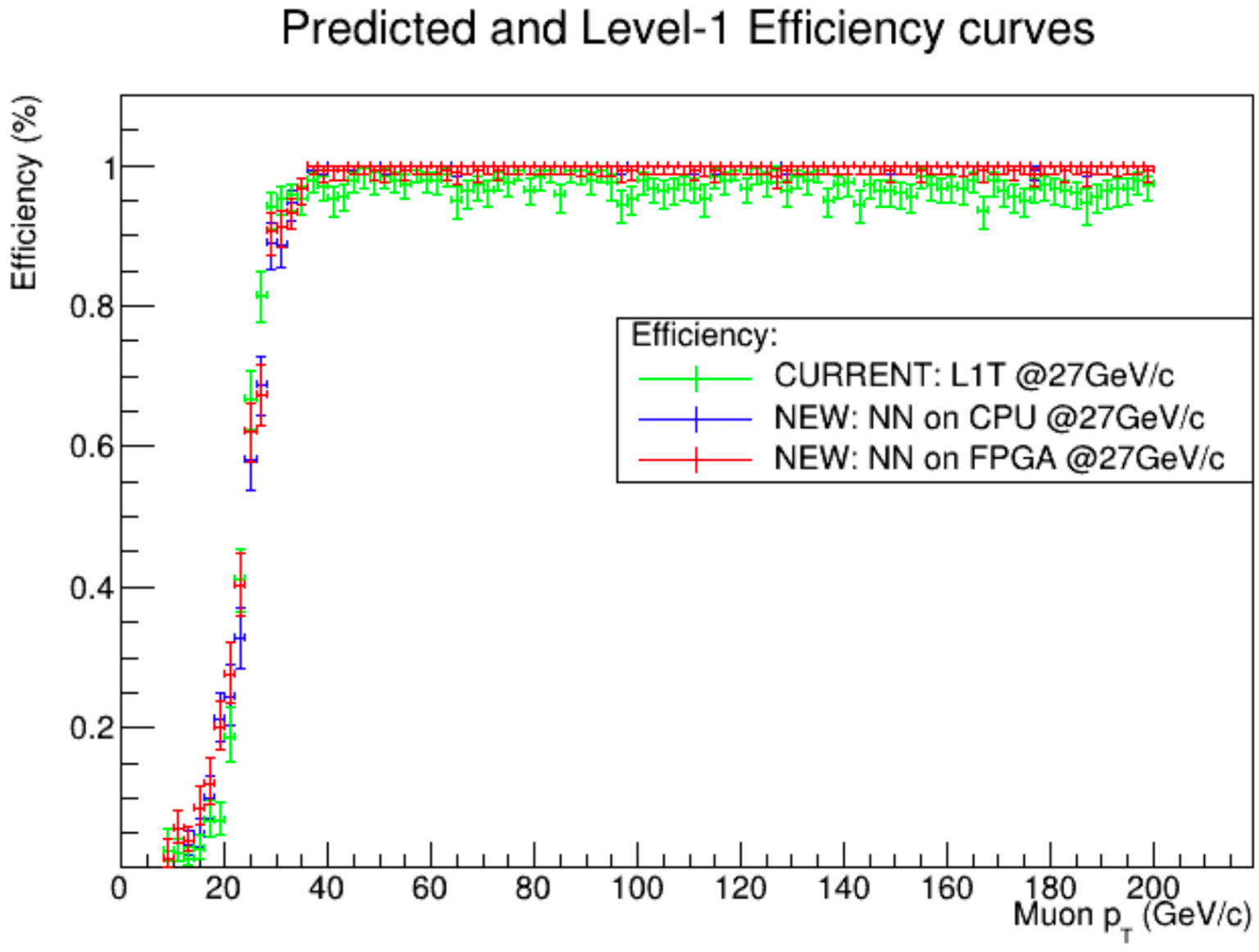
$$\frac{\Delta p_T}{p_T} = \frac{p_{T_{est}}^{NN,L1,FPGA} - p_{T_{sim}}}{p_{T_{sim}}}$$



Muons generated in the [20,40] GeV range.

Turn-on efficiency on FPGA

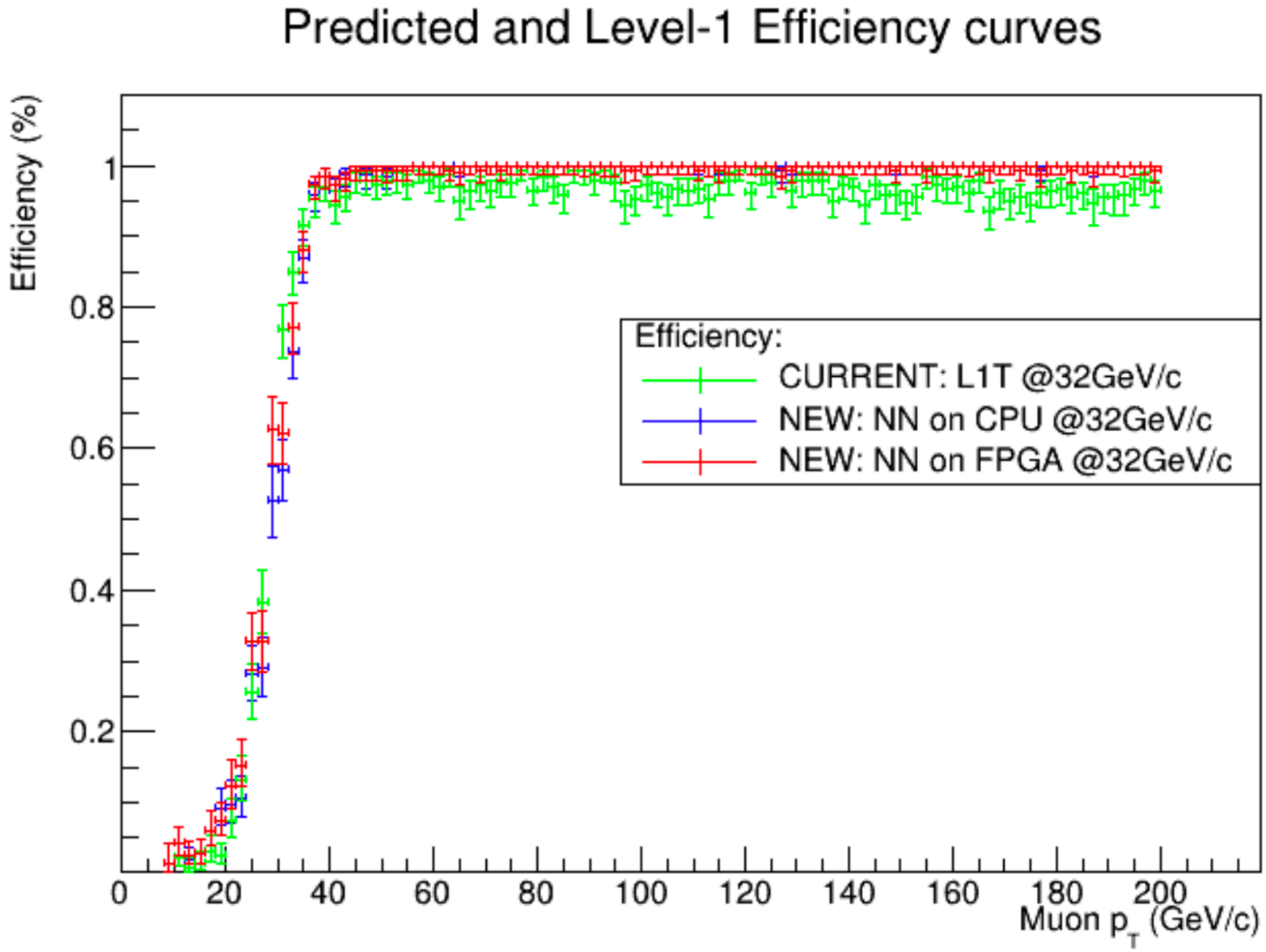
$$\epsilon = \frac{\textbf{Number of } \mu > \textbf{given threshold in } p_T}{\textbf{Total number of muons}}$$



Efficiency turn-on given a p_T threshold of 27 GeV

Turn-on efficiency on FPGA

$$\epsilon = \frac{\textbf{Number of } \mu > \textbf{given threshold in } p_T}{\textbf{Total number of muons}}$$



Efficiency turn-on given a p_T threshold of 32 GeV