PROCEEDINGS OF SCIENCE

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

Tommaso Diotalevi,[a,b,1,*] **Marco Lorusso,**[b] **Riccardo Travaglini,**[a] **Carlo Battilana**[a,b] **and Daniele Bonacorsi**[a,b]

[a] *INFN Bologna,*
viale Berti Pichat 6/2, Bologna, Italy
[b] *Deparment of Physics and Astronomy, University of Bologna,*
viale Berti Pichat 6/2, Bologna, Italy
E-mail: tommaso.diotalevi@unibo.it, marco.lorusso8@studio.unibo.it, riccardo.travaglini@bo.infn.it, carlo.battilana2@unibo.it, daniele.bonacorsi@unibo.it

With the advent of the High-Luminosity phase of the LHC (HL-LHC), the instantaneous luminosity of the Large Hadron Collider at CERN is expected to increase up to $\approx 7.5 \cdot 10^{34} cm^{-2} s^{-1}$. Therefore, new strategies for data acquisition and processing will be necessary, in preparation for the higher number of signals produced inside the detectors. In the context of an upgrade of the trigger system of the Compact Muon Solenoid (CMS), new reconstruction algorithms, aiming for an improved performance, are being developed. For what concerns the online tracking of muons, one of the figures that is being improved is the accuracy of the transverse momentum (p_T) measurement. Machine Learning techniques have already been considered as a promising solution for this problem, as they make possible, with the use of more information collected by the detector, to build models able to predict with an improved precision the p_T. This work aims to implement such models onto an FPGA, which promises smaller latency with respect to traditional inference algorithms running on CPU, an important aspect for a trigger system. The analysis carried out in this work will use data obtained through Monte Carlo simulations of muons crossing the barrel region of the CMS muon chambers, and compare the results with the p_T assigned by the current CMS Level 1 Barrel Muon Track Finder (BMTF) trigger system.

International Symposium on Grids & Clouds 2021, ISGC2021 22-26 March 2021
Academia Sinica, Taipei, Taiwan (online)

[1] on behalf of the CMS Collaboration
[*] Speaker

PoS(ISGC2021)005

1. Introduction

Muon detection has a fundamental role in the CMS (C*ompact* M*uon* S*olenoid*) experiment [1]. Such particles are produced, in fact, by an high amount of physical processes of great importance for the High Energy Physics community, in some of the signatures which led to the Higgs boson discovery by the ATLAS and CMS collaborations in 2012 [2]. For this reason, the CMS muon trigger system has to select events containing such particles while keeping the rate of acquisition under control, by operating cuts in transverse momentum (p_T). The current CMS Level-1 muon trigger system [3] performed smoothly during Large Hadron Collider (LHC) Run 1 and, after the "Phase 1" upgrade in 2016, in Run 2 as well.
However with the advent of the High-Luminosity phase of the LHC (HL-LHC), the instantaneous luminosity will increase up to $7.5 \cdot 10^{34}$ cm^{-2}s^{-1}, approximately five times larger than the limit reached during the present LHC run. This corresponds to an average pile-up of up to 200 events per crossing in the interaction region of the detector and an integrated luminosity of up to 4000 fb^{-1} over 10 years of data taking. Such level of machine performance will allow to extend searches for new physics and to perform stringent tests on the Standard Model of particle physics, such as precision measurements of the Higgs Boson couplings. The complexity and the time span of this computing challenge, together with the recent ramp-up in the evolution curve of advanced techniques - like Machine Learning (ML) and Deep Learning (DL) approaches - invites to explore some of them and to implement actual prototypes that test and verify their possible implementation on the experiment hardware itself, using low latency solutions like Field Programmable Gate Arrays (FPGAs).
Focusing on the barrel region of the Level-1 muon trigger, this contribution [4] aims to implement a functioning workflow: from the design of an artificial neural network model capable of predicting the muon p_T, using the track information gathered by the actual Level-1 trigger algorithms, to the conversion of such models in hardware code, with the final goal of importing them in the actual FPGA and perform inference on a microsecond timescale.

2. The Compact Muon Solenoid detector

The Compact Muon Solenoid (CMS) is a general purpose detector 21.6 m long with a diameter of 15 m and a weight of about 14000 tons. It is composed by a cylindrical barrel and two endcaps as shown in Figure 1. A superconducting solenoid generates a magnetic field of 3.8 T inside its volume. In its innermost region, a high quality central silicon inner tracker, capable of achieving a track reconstruction with very high resolution, is installed. A homogeneous electromagnetic calorimeter and a sampling hermetic hadron calorimeter are installed to perform energy measurements on electrons, photons and jets from hadrons. Finally, a redundant and highly performant muon system wraps the detector: it provides detection of muon tracks, and contribute to the measurement of their transverse momentum and trajectory. Its four muon stations are sliced into an iron yoke that returns the flux of the magnetic field and acts as an absorber for traversing particles that are not muons and neutrinos, and hosts and mechanically supports the stations.

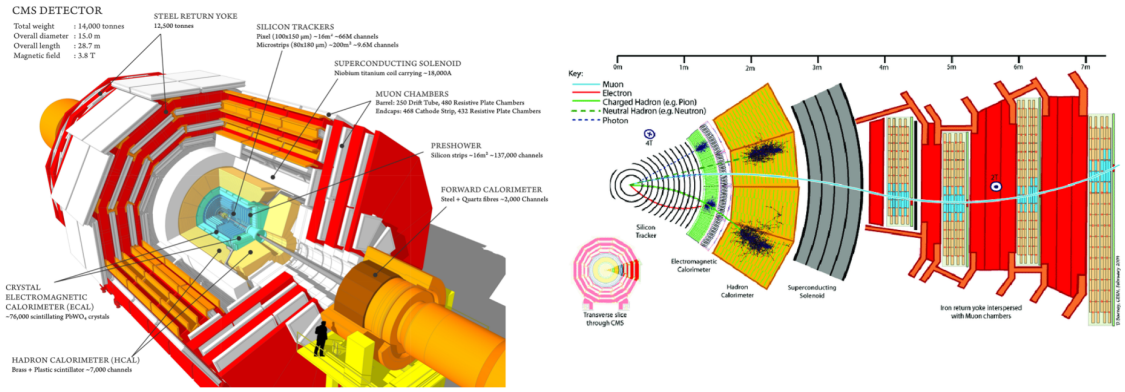


Figure 1: Overall view of the CMS detector [5, 6].

2.1 Tracker

The Tracker is a crucial component in the CMS design. It measures particles' momentum through their path: the greater is their curvature radius across the magnetic field, the larger is their momentum. The Tracker is able to reconstruct muons, electrons and hadrons paths as well as tracks produced by short-lived particles decay, such as b quarks.

2.2 Calorimeters

There are two types of calorimeters in the CMS experiment, designed to measure the energy of particles emerging from the collisions: the Electromagnetic calorimeter (ECAL) and the Hadronic calorimeter (HCAL). The ECAL is made of lead tungstate ($PbWO_4$) crystals, that scintillate when electrons and photons pass through them, producing light in proportion to the crossing particle's energy. The energy measurement for hadrons is instead carried out by the HCAL, which is specifically built for measuring strong-interacting particles.

2.3 Muon system

The aim of the Muon System [7] is to provide a robust trigger, capable of performing bunch-crossing (BX) assignment and standalone transverse momentum (p_T) measurement, an efficient identification of muons, and to contribute to the measurement of the p_T of muons with energy as high as few hundreds of GeV or more. As shown in Figure 2, the barrel Drift Tube (DT) chambers [9] cover the psudorapidity region $|\eta| < 1.2$, where the background is small, the muon rate is low and the magnetic field is uniform and mostly contained in the steel return yoke. In the two endcap regions of CMS, instead, the muon system adopts Cathode Strip Chambers (CSC) [10]: with their fast response time, fine segmentation and radiation resistance, the CSCs are used to track muons between $|\eta|$ values of 0.9 and 2.4. Resistive Plate Chambers (RPC) [11] are also deployed throughout the central and forward regions ($|\eta| < 1.8$), offering a fast response and an excellent time resolution. From the Phase-II upgrade of the detector, additional chamber have been installed in the endcap region, designed to improve forward muon triggering capabilities in this challenging region: Gas Electron Multipliers (GEM) [12] and an improved version of the RPC, called iRPC (improved RPC) [13].

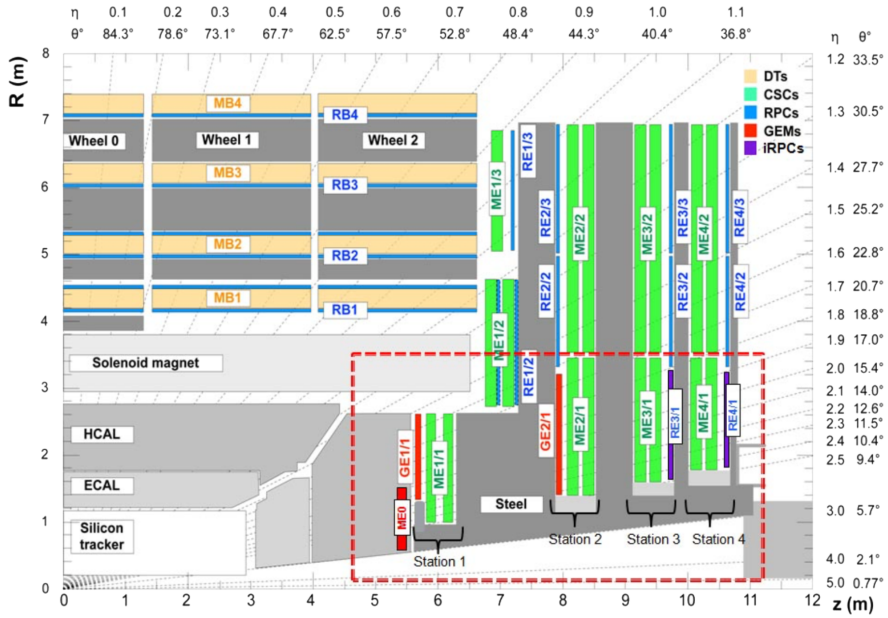


Figure 2: Longitudinal view of the CMS muon region. The dashed red box contains the new detectors added for Phase-II (HL-LHC) [8].

3. The barrel muon Level-1 trigger system

The Level-1 trigger, is designed to take a fast accept/reject decision (every BX), in a pipelined fashion using custom developed programmable hardware. The functional relations between the components are shown in Figure 3. The electronics devoted to muon tracking follows a geographical partitioning: separate systems process local information produced in the barrel, endcap and overlap regions, corresponding to $|\eta| < 0.8$, $0.8 < |\eta| < 1.2$ and $1.2 < |\eta| < 2.4$, respectively, to identify muons and measure their coordinates and momentum.

The information gathered from the barrel gaseous detectors, DTs and RPCs, is collected and synchronised by the TwinMux system [15], in charge of the merging of the two, thus delivering the improved trigger segments, called *primitives*, to the B*arrel* M*uon* T*rack* F*inder* (BMTF) [16]. The BMTF receives the muon primitives from the Barrel area of CMS ($|\eta| < 0.85$) and it reconstructs muon tracks and calculate the physical parameters: transverse momentum (p_T), ϕ and η. The transverse momentum assignment is performed by means of Look-Up Tables (LUTs), which exploit information from the trigger primitives of the two innermost stations used to built a given BMTF track.

The value of p_T assigned after the output produced by the BMTF will be used as a comparison for the p_T 'predicted' by the Machine Learning models.

4. Artificial Neural Network

An Artificial Neural Network (ANN) is a learning method vastly used in Machine and Deep Learning, inspired by the biological neural connections that constitute the human brain, specifically designed to tackle non-linear learning problems [17]. Among the different architectures of ANN

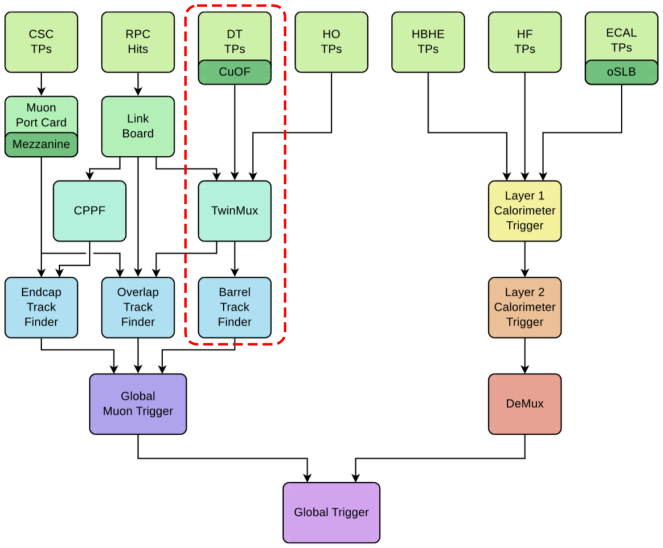
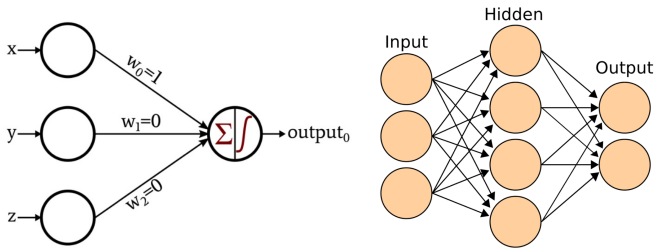


Figure 3: CMS Level-1 muon trigger data flow (the dashed circle highlights the barrel region of the detector) [14].



(a) Graphical representation of a perceptron.

(b) Neural network basic layout.

Figure 4: Simple diagrams representing the layout and functioning principle of neural networks.

available, the Fully Connected Multilayer Perceptron (MLP) [18] has been chosen for this work, due to its relative simple design. MLPs, as the name suggests, are made up of single units called *perceptrons*. These, as shown in Figure 4(a), are characterised by an input vector of features $\mathbf{x} = (x, y, z)$, each one multiplied with their weight $(w_i x_i)$. All of these are then added together to create the net or weighted sum $\sum_i w_i x_i$, and finally given to the *Activation function* $f(\sum_i w_i x_i)$ which evaluates the perceptron's outputs, adding a non-linearity compared to the simple linear combination alone. Perceptrons can be stacked together to make a layer of neurons, each producing its own outputs. These layers can then be put together to build arbitrarily deep custom networks, by feeding the outputs of a layer to the neurons of the next layer, which will be 'hidden' to the user, and resulting in a structure like the one shown in Figure 4(b).

4.1 Data preparation

The first important step prior to model creation is data preparation. In a standard machine learning problem, this is a delicate phase that deals with aspects such as data cleansing, data scaling (normalisation and binning), feature inspection and eventually feature engineering. The data used is generated out of a Monte Carlo simulated sample of 300k muons in a p_T range from 3 to 200 GeV/c, with different energy and direction (equally distributed in charge) crossing the CMS detector and, as anticipated in the previous section, only trigger primitives are used for a more direct comparison with the actual Level-1 trigger algorithm. Each generated muon passing through the barrel DT chambers results into several trigger primitives. Each muon has an associated number corresponding to the number of chambers crossed by its trajectory.
Therefore, inside a plain *csv* file (Comma Separated Values), extracted from a *TTree* in a ROOT file [19] (format widely used by the high energy physics community), all the necessary information has been stored: both the physical information of each simulated muon and the respective primitive hits in the detector, up to the Level-1 trigger information.

4.2 Neural Network architecture

The neural network architecture has been implemented with the following structure: the first hidden layer has 60 neurons and receives the information directly from the input layer of 27 different features with the ReLU (Rectified Linear Unit) selected as activation function. The second layer is identical to the first one but contains 50 neurons and this is repeated for other 3 additional hidden layers with 30, 40 and 15 neurons, respectively. In the end, the output layer (with only one node) closes the Network. This structure will be referred later as *D model*, standing for 'Deep model'.
The network performs a *regression* task, by predicting the transverse momentum (p_T) in a supervised environment, i.e. providing as a target label the estimated momentum from the Monte Carlo simulation.

4.2.1 Quantisation of the Neural Network

Developing a neural network in a specific hardware with a limited amount of resources, requires a high level of optimisation in terms of compression, to reduce the storage and computation costs for deep models. Quantisation, therefore, means converting the arithmetic used within the network from high-precision floating-points to normalised low-precision integers (fixed-point) [20]. Floating-point representation allows the decimal point to 'float' to different places within the number, depending upon the magnitude. Fixed-point numbers, instead, consist of two parts, integer and fractional, as shown in Figure 5. Compared to floating-point, fixed-point representation maintains the decimal point within a fixed position, allowing for more straightforward arithmetic operations.

The Application Programming Interface (API) environment used for the Machine Learning model - deeply specialised in Neural Networks - is based on Keras [21], an high-level framework capable of running on top of the Google TensorFlow [22] software library, and it is called *QKeras* [23]. Developed by a collaboration between Google and CERN, it is a quantisation extension to Keras that provides a drop-in replacement for layers performing arithmetic operations. This allows efficient training of quantised versions of Keras models. The quantisation is performed during training itself, and not at the end by simply lowering the numbers bitwidth.

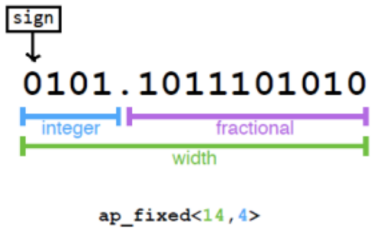


Figure 5: Fixed-point number representation. The green number represents the width, the blue number the integer part and the purple number the fractional part.

For the implementation of the *D model*, the first hidden layers presents a gaussian random number as kernel initialiser (for the initial randomised weight assignment). Then, all the network components are quantised, including the activation functions, with a bitwidth of 16 bits of which 1 is used to represent the integer part of the value (another bit is used for the sign).

4.2.2 Neural Network weight pruning

During the implementation of a neural network model, the final hardware platform where the inference computation will run, has to be considered. In order to minimise the resource utilisation, a further optimisation is needed to reduce the number of parameters and operations involved in the computation, by removing connections, and thus parameters, in between the network layers. This process is commonly called *weight pruning*, i.e. the elimination of unnecessary values in the weight tensor, by setting the network parameters' values to zero, which will be translated into a cut of the connections between the nodes of the neural network.
The pruning is done during the training process to allow the network to adapt to the changes and the TensorFlow Sparsity Pruning (TFSP) API [24] has been selected to perform this optimisation. It uses an algorithm designed to iteratively remove connections, based on their magnitude during training. As training proceeds, the pruning routine is scheduled to execute, eliminating the weights with the lowest magnitude (i.e. those closest to zero), until the current sparsity target is reached. Every time the pruning routine is scheduled to execute, the current sparsity target is recalculated, until it reaches the final target sparsity at the end of the pruning schedule by gradually increasing it according to a smooth ramp-up function. In Table 1, it is clear how this technique reduces the number of weights different from zero, yielding to a lower number of operations performed by the network itself.

5. Neural Network model performance on CPU

Prior to the implementation of the network in the FPGA, the architecture has been tested on a Intel i7 6500U @ 2.50GHz consumer CPU. The QKeras model trained is saved into a HDF5 file [25], containing:

- The architecture of the model, in order to be reproduced when it is needed to be used with other datasets;
- The weights of the model;

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

Table 1: Results of the pruning technique for the *D model*, with a complete list of parameters for every layer before and after pruning. The model is built using QKeras.

- The training configuration (loss, optimiser);
- The state of the optimiser, allowing to resume training exactly where it was left.

Then, using another *csv* file with an independent set of simulated muons (about 15000 muons), created and processed in the same way as the training sample, it is possible to load the models and predict the muon p_T. The results of the prediction are then compared to the p_T estimated by the p_T assignment unit of the BMTF.
Two different types of plots has been produced as output:

- p_T resolution histograms;
- efficiency curves computed as function of the generated muon p_T ("turn-on curves").

5.1 p_T resolution histogram

The p_T resolution histograms,for each generated muon, are filled using the following relation:

$$\frac{\Delta p_T}{p_T} = \frac{p_{T_{est}} - p_{T_{sim}}}{p_{T_{sim}}}$$

where $p_{T_{est}}$ is the estimation of the transverse momentum, given by the model prediction or the Level-1 trigger p_T assignment unit, and $p_{T_{sim}}$ is the simulated transverse momentum. Even though this metric makes a quick and easy to understand comparison possible, it is important to keep in mind that this resolution is asymmetric, i.e. its range can go from -1 to infinite. This means that, for a constant actual spread, the standard deviation associated to its distribution is affected by the value of its mean: the smaller it is, the smaller the standard deviation will get.
Figure 6 shows the p_T resolution for the entire range analysed (from 3 to 200 GeV/c). The green line indicates the resolution of the Level-1 trigger system while the blue line indicates the resolution of the predictions made by the network *D model*. In particular, it is possible to notice a less broad distribution for the ML resolution, resulting in an overall improvement (yet small) with respect to the Level-1 trigger system. Another noticeable detail is the small peak corresponding to the value -1: this happens when the p_T assigned by the trigger is significantly underestimated with respect to the generated muon p_T. The machine learning based momentum assignment is therefore less prone to large p_T underestimation.
In general, an additional feature visible in the p_T resolution plot is the scale: the peak of the

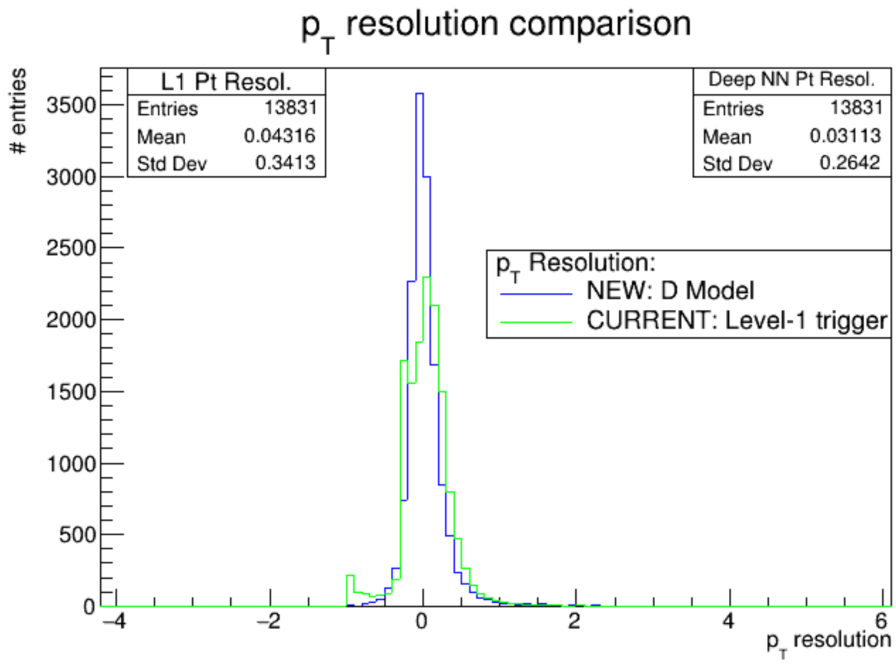


Figure 6: Transverse momentum resolution histograms computed for the machine learning *D model* (blue) and Level-1 trigger (green) based momentum assignment, computed for muons generated in the 3-200 GeV p_T range.

ML-based distribution is closer to zero if compared with the Level-1 trigger one. This is due to a different calibration of the p_T scale in the two cases. The trigger p_T assignment is in fact calibrated defining Level-1 muon p_T as the point where, for a given threshold, the efficiency turn-on curve (described in the next section) reaches the 90% value. On the other hand, the network attempts to estimate the generated muon p_T and is therefore characterised by a smaller scale bias.

5.2 Efficiency turn-on

The efficiency turn-on, as the word states, represents an efficiency ϵ defined as follows:

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

After the definition of a minimal p_T threshold cut, the numerator is filled if the p_T value from the network or the Level-1 p_T assignment is above threshold. Then the ratio with the denominator is performed. Both numerator and denominator are filled only in case a geometrical match between a trigger muon track and a generated muon is found. In Figure 7 a turn-on efficiency curve is shown for the *D model*. A threshold of 22 GeV/c is selected, as it is the typical threshold of the lowest p_T cut used by the lowest momentum unprescaled single muon trigger used over Run-2.
From these plots, a very high efficiency in the plateau region of the ML turn-on curve can be observed: the lower efficiency of the Level-1 curve in the high-p_T region is mainly caused by the underestimation of the muon p_T in some events, which is visible as the small peak at -1 in the p_T resolution plots (Section 5.1). The network model, instead, is not affected by this issue and the efficiency at high values of p_T is nearly 100%. On the other hand, in the low p_T region from 3

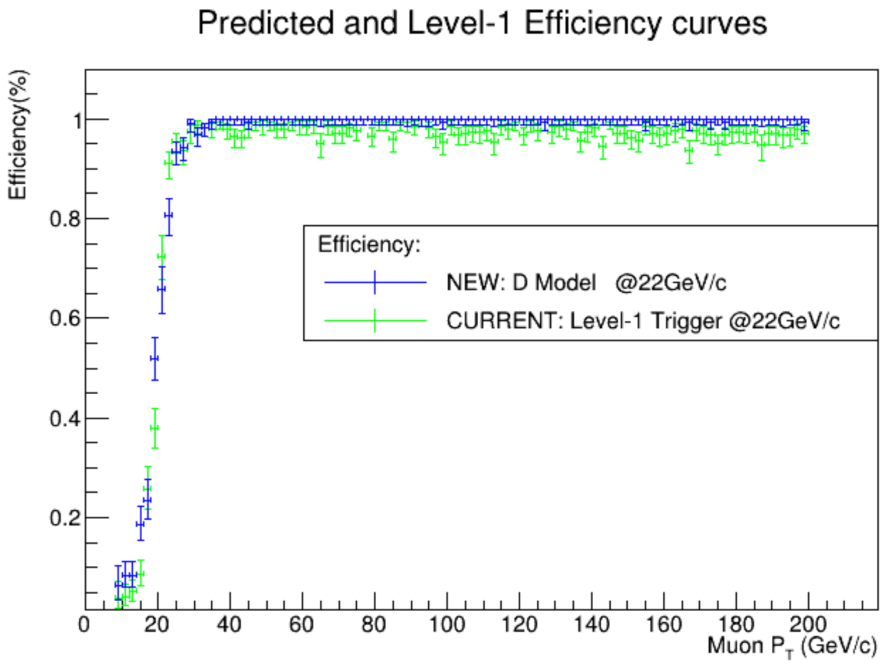


Figure 7: Efficiency turn-on given a p_T threshold cut of 22 GeV. The blue dots show the efficiency for the machine learning based momentum assignment related to the *D model*, while the green dots show the efficiency for the Level-1 trigger p_T assignment.

to 20 GeV, a small increase in efficiency is visible, leading to an higher fraction of muons with low momentum misidentified as high momentum, with a consequently not optimal increase of the expected rate of acquisition.

6. Implementing the Neural Network on FPGA

Field Programmable Gate Arrays (FPGAs) are devices that blend the benefits of both hardware and software. They implement circuits just like hardware, providing huge power, area and performance benefits over software, yet they can be reprogrammed cheaply and easily to implement a wide range of tasks. FPGAs implement computations spatially, simultaneously computing millions of operations in resources distributed across a silicon chip. Such systems can be hundreds of times faster than microprocessor based designs. The internal layout of an FPGA, is made up of replicated units of digital electronic circuits, called logic blocks, embedded in a general routing structure, hence the *gate* and *array* in the name of this type of devices. The logic blocks contain processing elements, with different functions:

- L*ook*U*p* T*ables* (LUT) for simple combinational logic;
- F*lip*-F*lops* (FF) for implementing sequential logic;
- D*igital* S*ignal* P*rocessors* (DSP) for efficient multiplication of fixed-point numbers (fundamental for the implementation of the neural network in the FPGA).

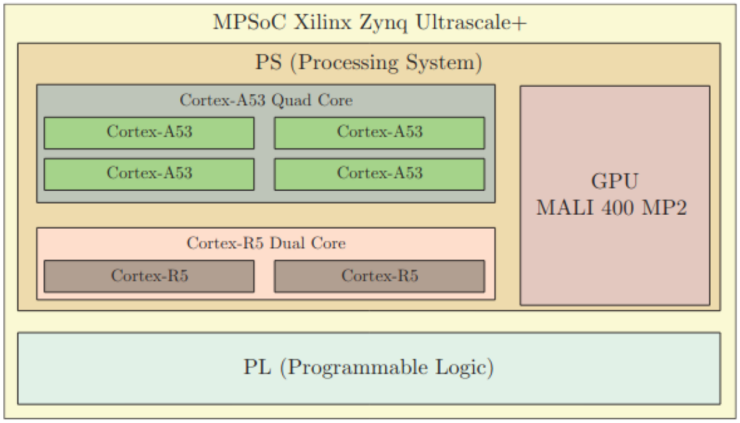


Figure 8: Functional block diagram of the Xilinx Zynq Ultrascale+ MPSoC.

6.1 Hardware characteristics

The target hardware consisted in a Xilinx ZCU102 Evaluation Board [26] featuring the Zynq Ultrascale+ XCZU9EG-2FFVB1156E Multiprocessor System on a Chip (MPSoC) with a quad-core Arm Cortex-A53, dual-core Cortex-R5F real-time processors, and a Mali-400 MP2 graphics processing unit based on Xilinx's 16nm FinFET+ programmable logic fabric. The ZCU102 MPSoC houses 600000 logic cells (flip-flops and LUTs) and 2520 DSP slices, working with an internal memory of 32.1 Mb. This board has been chosen because of the faster development the use of a microprocessor ensures, as only the firmware concerning the network has to be designed, leaving the rest (e.g. I/O interface) manageable via software.
In all MPSoC devices from Zynq Ultrascale family, the system is subdivided in two main parts (shown in Figure 8):

- Processing system (PS): contains all the microprocessors, in particular the quad core ARM Cortex-A53 that implements the ARM v8-A 64-bit instruction set, with an I/O accessible via software using the Xilinx Software Development Kit (SDK);
- Programmable Logic (PL): contains the FPGA logic described above.

Vivado and its SDK can establish a JTAG [27] connection to the MPSoC through the USB-to-JTAG module with a micro-USB connector. This link is used for the programming of the FPGA and for the first steps of debugging. On the other hand, to communicate and so retrieve the output of the computation carried out in the PL and handled by the PS, the Universal Asynchronous Receiver-Transmitter (UART) [28] interface has been used.

6.2 Conversion of the Network architecture to an HLS project

The first step required for the implementation of the neural network on the FPGA is the conversion of the high-level code used for the creation of the network (QKeras) into an High-Level Synthesis (HLS), which is the process of automatic generation of hardware circuit from 'behavioral descriptions' contained in a C or C++ program. To accomplish this task, the *hls4ml* package has been used. The *hls4ml* package [29] has been developed by members of the High Energy Physics

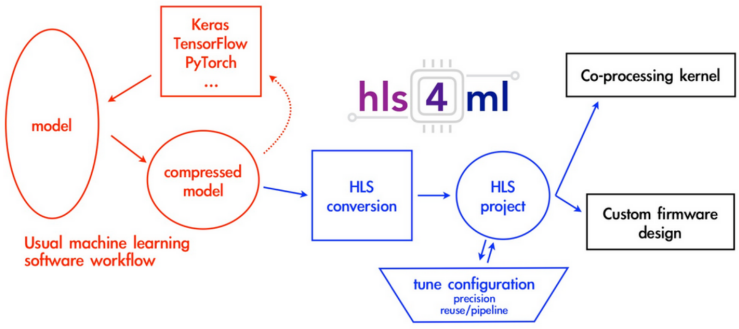


Figure 9: A typical workflow to translate a model into an FPGA implementation using *hls4ml*.

(HEP) community to translate ML algorithm, built using frameworks like TensorFlow, into HLS code. In this work, *hls4ml* is used to perform this transformation on a trained neural network, defined by its architecture, weights, and biases. A schematic workflow of *hls4ml* is illustrated in Figure 9. The parts of the workflow illustrated in red indicates the usual software steps required to design a neural network for a specific task. The blue section of the workflow is the task done by *hls4ml*, which translates the model into an HLS project that can be synthesised and implemented to run on an FPGA. This tool also allows the user to change the level of parallelisation, by adjusting a simple parameter, called *reuse factor*, and it is also fully compatible with the QKeras extension.

6.3 Exporting the HLS project into an IP core

Once the target hardware has been defined, the trained model is converted into an HLS project using the functions provided by the *hls4ml* library. The projects are then opened with Vivado HLS to modify the I/O interface from the one created automatically by *hls4ml* to an AXI4-Lite [30] interface. The Advanced eXtensible Interface (AXI4) is a family of buses defined as part of the fourth generation of the ARM Advanced Microcontroller Bus Architecture (AMBA) standard. AXI4-Lite is a subset of AXI which has a simpler interface than the full AXI4 architecture but with a slower I/O operation compared to the full AXI library. This choice has been performed in order to allow the PS to communicate easily with the PL through the MPSoC Intellectual Property (IP) [31] in the Vivado IP integrator. Then, the network I/O consisted in reading and writing registers in the PS on-board memory, handled by the software. An entire write cycle for a single entry, dealing with all the 27 features making up the test set, takes approximately 500 ns, as shown in Figure 10. The next step is to synthesise the HLS project. After synthesising the project, performance and utilisation estimates can be analysed in the post-synthesis report, which contains information on several aspects, among which:

- the amount of resources required to implement the design, in terms of LUTs, FFs and DSPs;
- the latency, or the number of clock cycles required for the function to compute all output values;
- the initiation interval, or the number of clock cycles before the function can accept new input data,

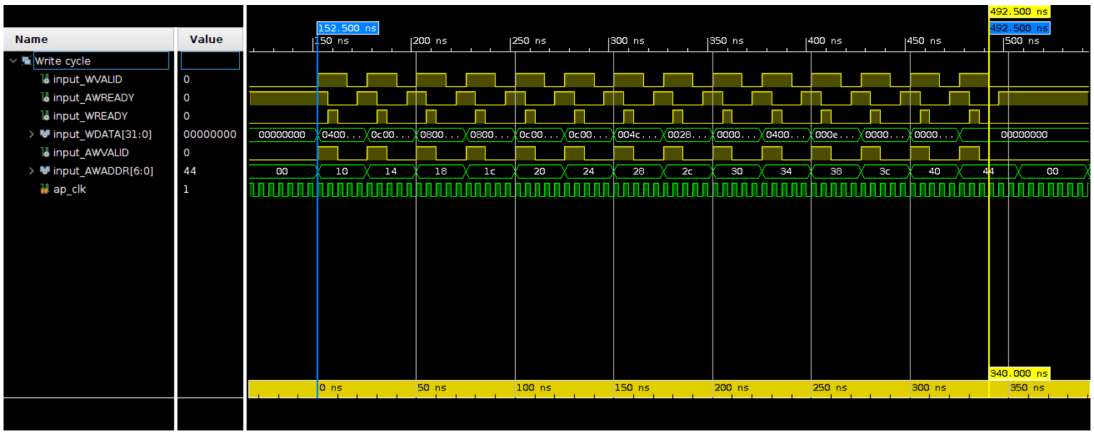


Figure 10: Simulation of a write cycle of the 27 features in input for a single entry through a AXI4-Lite interface between the Processing System to the Programmable Logic.

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

Figure 11: Table containing the resource footprints of the *D model* in terms of BRAM, DSPs, FFs, LUTs and URAM available and used.

The report for the *D model* is shown in Figure 11. In general, when implementing neural networks into FPGAs, the main problem is the lack of DSP slices. The model chosen for the analysis has been proven suitable for the implementation in the target hardware available for this work. Before going further, a simulation of the model behaviour on the FPGA is carried out, yielding the results depicted in Figure 12, which shows a comparable behaviour with the network inference on software.

6.4 Implementing a block design

To implement the chosen model in a complete design, the project is exported via the Vivado IP Packager: it enables the creation of a reusable Intellectual Property (IP) module that can be accessed from the Vivado IP Integrator. Then, a graphic connectivity canvas has been used to select peripheral IPs, configure the hardware settings, and stitch together the IP blocks to create the digital system. This is done in workspaces called Block Designs (BDs). In the case under study, the model is represented inside a Block Design by the IP core produced by Vivado HLS. On the other hand, the PS is represented by the Zynq Ultrascale+ MPSoC IP. In this way, through the addition of IPs handling the AXI4-Lite interfaces, it is possible to connect the application with the ARM processor, which will be used to communicate with the PL. The final diagram representing the entire design is

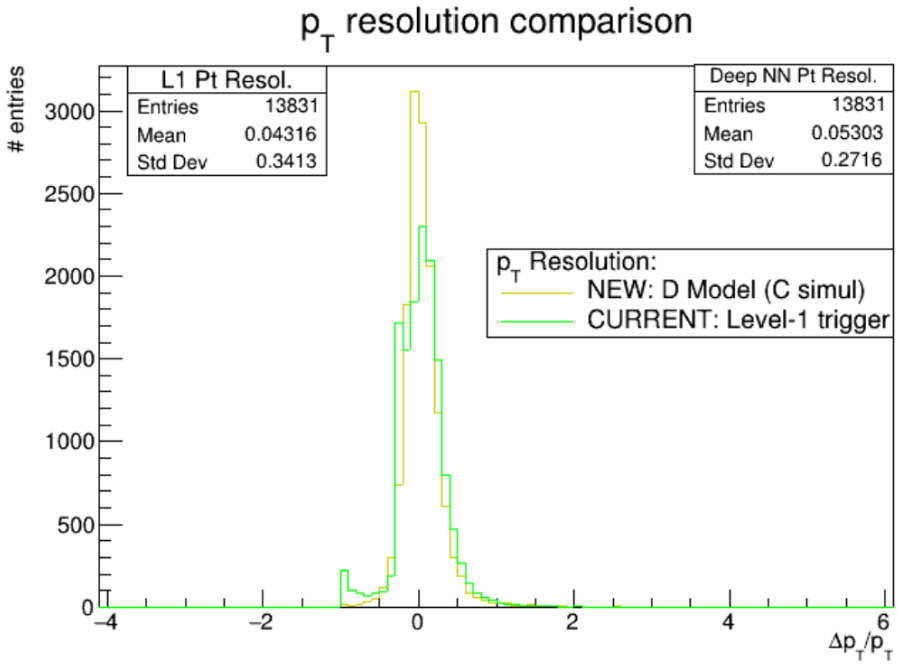


Figure 12: Transverse momentum resolution histograms computed for the deeper ML-based simulated on the FPGA (dark yellow) and Level-1 trigger based momentum assignment, computed for muons generated in the 3-200 GeV p_T range.

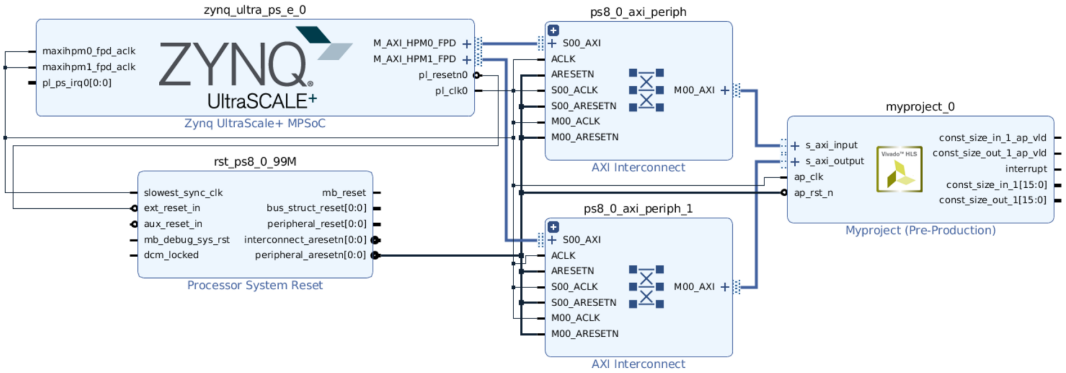


Figure 13: Block Design showing the IP core containing the neural network connected to the PS, through IPs handling the I/O interface of the implemented network.

shown in Figure 13.
Once the design is ready, an HDL wrapper is created in order to synthesise and implement the project. Finally, the bitstream can be generated, which is copied onto the FPGA in order to program it.

7. Inference on the FPGA

At this stage the *D model* has been successfully translated in a bitstream. The computation on the FPGA is performed through Vivado SDK, calling functions in a C++ code, compiled on a host

PC that drive the execution into the PL. By defining the input as objects of the *ap_fixed*$< 16, 6 >$ class, the input from the network test set is cast to fixed point numbers of 16 bits. The entire structure is then enclosed in a loop, allowing the inference for the entire test dataset. The output is then retrieved via the UART interface, connected to a serial port on the host PC, using a Python script which also converts the integer representation back into floating point numbers for further analysis.
The results are evaluated by counting the number of clock pulses between the input of a pattern and the production of the related output. The model takes approximately 74 clock cycles (corresponding to ≈ 0.368 μs) for each candidate on the FPGA, in contrast with the times obtained with a consumer CPU of 37.29 ms for a single prediction. It is, however, comparable with the 9 BXs (≈ 0.225 μs) expected to be needed by the Phase-II upgrade of the track finder, the Kalman BMTF.

8. Neural Network model performance on FPGA

After the implementation, synthesis and inference of the muon p_T from the neural network model, the same plots shown in Section 5 has been produced for the FPGA results.

8.1 p_T resolution histogram

In Figure 14 the resolution histogram for the *D model* is shown. In this overall picture, it is clear that the model infer momenta with a resolution which is narrower when the computation is carried out on a CPU. When the assignment is performed on an FPGA, slightly worse results are produced with respect to a traditional CPU, with a small bias towards higher values of $\Delta p_T / p_T$. The passage from software inference to FPGA has caused a small decrease in the advantages achieved by using a machine learning based algorithm to assign p_T to muons, as demonstrated by the slightly wider resolutions' distributions. This could be the effect of the loss in precision the input features undergo, due to the conversion to fixed-point representation needed to perform computations efficiently in an FPGA. Nevertheless, the hardware approach still appears compatible or, in case of higher momenta, even better than the Level-1 trigger based momentum assignment.

8.2 Efficiency turn-on

As shown in Section 5.2, the accuracy of the model under study in the range 3-20 GeV/c results worse than the Level-1 trigger. This can cause problems when cuts in momentum are applied in the trigger. Figure 15 depicts the turn-on curves for the *D model* when a threshold cut at 22 GeV/c is applied. Only small differences between software and hardware based computation are observed. From both FPGA and CPU (Figure 7) turn-on curves, almost 100% efficiency for p_T values well above the threshold cut is also shown.

9. Conclusions

The precise evaluation of p_T is crucial to allow the trigger rate to be ultimately reduced and it represents a possible improvement of the CMS Level-1 muon trigger system during the High Luminosity phase of the LHC. By implementing neural network models onto Field Programmable Gate Arrays (FPGAs), an alternative to the Level-1 trigger track transverse momentum assignment

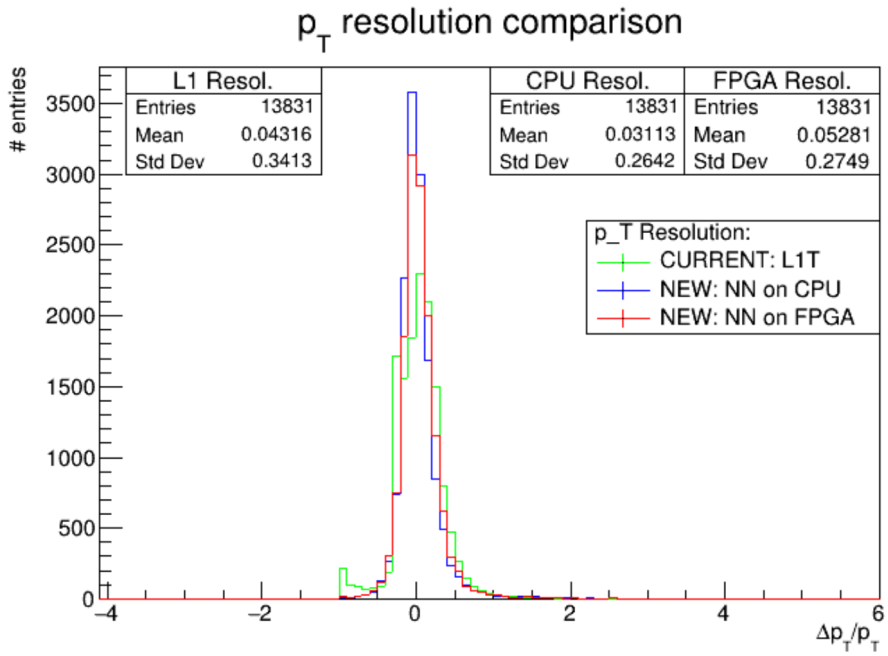


Figure 14: Transverse momentum resolution histograms computed for the *D model* running on a consumer CPU (blue), running on an FPGA (red) and Level-1 trigger based momentum assignment (green), computed for muons generated in the 3-200 GeV p_T range.

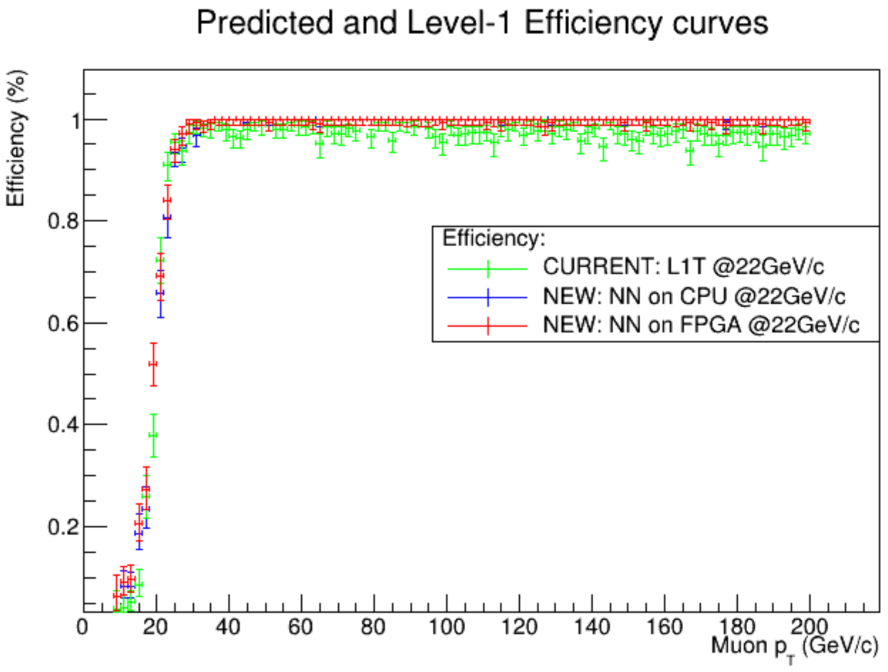


Figure 15: Efficiency turn-on given a p_T threshold cut of 22 GeV/c. The blue dots show the efficiency for the network based momentum assignment related to the *D model* running on a consumer CPU, the red dots show the results for the same model implemented on the FPGA, while the green dots show the efficiency for the Level-1 trigger p_T assignment.

presently used by the Compact Muon Solenoid (CMS) was developed and tested.
The inference for the chosen network model on FPGA produced overall comparable resolutions with respect to the current Level-1 trigger system, with slightly better results at higher p_T values, but worse assignments in the 3-20 GeV/c range. In the plateau region of the ML turn-on curve an efficiency very close to 100% is observed, better than in the present trigger. On the other hand, compared to the present Level-1 trigger, the proposed algorithm shows a higher efficiency in the low p_T region (e.g. 10-15 GeV/c, for a 22 GeV/c cut) of the turn-on efficiency curve. This feature, which would definitely imply a high rate for the newly proposed solution, must be tackled in the next development stages. Regarding the time needed for the p_T assignment, the FPGA showed an inference time of the order of ≈ 70 clock cycles.

This work represents a first stepping stone towards more refined and fine-tuned neural networks, implemented in a more optimised way onto FPGAs. There is space for improvements in every phase of the workflow, from finding more suitable bitwidths for the weights, to a more evolved I/O interface used to send and retrieve data from the network, in order to process data in a pipelined way and to reduce even more the time needed for computation.

References

[1] CMS Collaboration, *The CMS Experiment at the CERN LHC*. JINST 3 S08004 (2008)

[2] S. Chatrchyan et al., *Observation of a New Boson at a Mass of 125 GeV with the CMS Experiment at the LHC* Phys. Lett. B, vol. 716, pp. 30–61, 2012.

[3] CMS Collaboration, *CMS Technical Design Report for the Level-1 Trigger Upgrade*, Tech. Rep. CERN-LHCC-2013-011, CMS-TDR-12, CMS-TDR-012, CERN, Geneva, June 2013.

[4] M. Lorusso, *FPGA implementation of Muon Momentum assignment with Machine Learning at the CMS Level-1 Trigger*. University of Bologna master thesis (unpublished).

[5] The CMS Experiment Webpage. https://cms.cern/detector

[6] D.Barney, *Presentation for public - Introduction to CMS for CERN guides*. https://cds.cern.ch/record/2629323

[7] J. G. Layter. The CMS muon project: Technical Design Report. Technical design report. CMS. Geneva: CERN, 1997. url: https://cds.cern.ch/record/343814.

[8] CMS Collaboration. *The Phase-2 Upgrade of the CMS Level-1 Trigger* . CMS-TDR-021

[9] M. Aguilar-Benitez et al. *Construction and test of the final CMS Barrel Drift Tube Muon Chamber prototype*. In: *Nucl. Instrum. Meth.* A480 (2002), pp. 658–669. doi: 10.1016/S0168-9002(01)01227-X.

[10] J. Hauser. *Cathode strip chambers for the CMS endcap muon system*. In: *Nucl. Instrum. Meth.* A384 (1996), pp. 207–210. doi: 10.1016/S0168- 9002(96)00905-9

[11] G. Wrochna. *The RPC system for the CMS experiment at LHC*. In: Resistive plate chambers and related detectors. Proceedings of *3rd International Workshop*, Pavia, Italy, October 11-12, 1995. 1995, pp. 63–77.

[12] CMS Collaboration, *CMS Technical Design Report for the Muon Endcap GEM Upgrade*, Technical Report CERN-LHCC-2015-012. CMS-TDR-013, CERN, 2015. Link: https://cds.cern.ch/record/2021453.

[13] CMS Collaboration, *Improved-RPC for the CMS muon system upgrade for the HL-LHC*, in proceedings of *15th Workshop on Resistive Plate Chambers and Related Detectors (RPC2020)*. Published in *JINST* 15 (2020) 11, C11012. doi: 10.1088/1748-0221/15/11/C11012

[14] CMS Collaboration. *Performance of the CMS Level-1 trigger in proton-proton collisions at* $sqrts = 13$ *TeV*. JINST 15 (2020) P10017

[15] CMS Collaboration, *Performance of the CMS TwinMux Algorithm in late 2016 pp collision runs*. In proceedings of *15th Workshop on Resistive Plate Chambers and Related Detectors (RPC2020)*. url: http://cds.cern.ch/record/2239285.

[16] J. Ero et al., *The CMS Level-1 Trigger Barrel Track Finder*, in 2016 *JINST* 11 C03038

[17] Samarasinghe, Sandhya. *Neural networks for applied sciences and engineering. From fundamentals to complex pattern recognition* (2007).

[18] Marius-Constantin Popescu, Valentina E. Balas, Liliana Perescu-Popescu, and Nikos Mastorakis. 2009. *Multilayer perceptron and neural networks*. WSEAS Trans. Cir. and Sys. 8, 7 (July 2009), 579–588.

[19] R Brun, F Rademakers, and S Panacek. *ROOT, an object oriented data analysis framework* (2000). url: http://cds.cern.ch/record/491486.

[20] Adam Taylor. *The basics of FPGA mathematics*. In: *Xilinx Xcell Journal* 80 (2012).

[21] *Keras Documentation*, url: https://keras.io

[22] *TensorFlow Documentation*, url: https://www.tensorflow.org

[23] *QKeras Github repository*. url: https://github.com/google/qkeras

[24] *TensorFlow Model Optimization Toolkit - Pruning API*. url: https://blog.tensorflow.org/2019/05/tf-model-optimization-toolkit-pruning-API.html

[25] *The HDF Group*. url: https://portal.hdfgroup.org/display/support.

[26] *Zynq UltraScale+ MPSoC ZCU102 Evaluation Kit*. url: https://www.xilinx.com/products/boards-and-kits/ek-u1-zcu102-g.html

[27] *The JTAG Connection*. url: https://semiengineering.com/the-jtag-connection

[28] Eric Peña and Mary Grace Legaspi. *UART: A Hardware Communication Protocol Understanding Universal Asynchronous Receiver/Transmitter*. In: *AnalogDialogue* 54 (Dec. 2020). url: https://www.analog.com/en/analog-dialogue/articles/uart-a-hardware-communication-protocol

[29] J. Duarte et al. *Fast inference of deep neural networks in FPGAs for particle physics*. In: *Journal of Instrumentation* 13.07 (July 2018), P07027. doi: 10.1088/1748-0221/13/07/p07027.

[30] *Introduction to AXI4-Lite*. url: https://www.realdigital.org/doc/a9fee931f7a172423e1ba73f66ca4081

[31] Sanjay Churiwala. *Designing with Xilinx® FPGAs: Using Vivado*. 1st ed. *Springer International Publishing*, 2017. isbn: 9783319424385.