CMS Endcap Muon Track Finder Displaced Muon Neural Network

Patrick Kelling[2], Darin Acosta[2], Osvaldo Miguel Colin[2],
Sergo Jindariani[1], Jacobo Konigsberg[3], Jia Fu Low[3], Alexander Madorsky[3], Efe Yiğitbaşı[2],
on behalf of CMS collaboration

October 1st, 2024

TWEPP

1 Fermilab 2 RICE UNIVERSITY 3 UF UNIVERSITY of FLORIDA

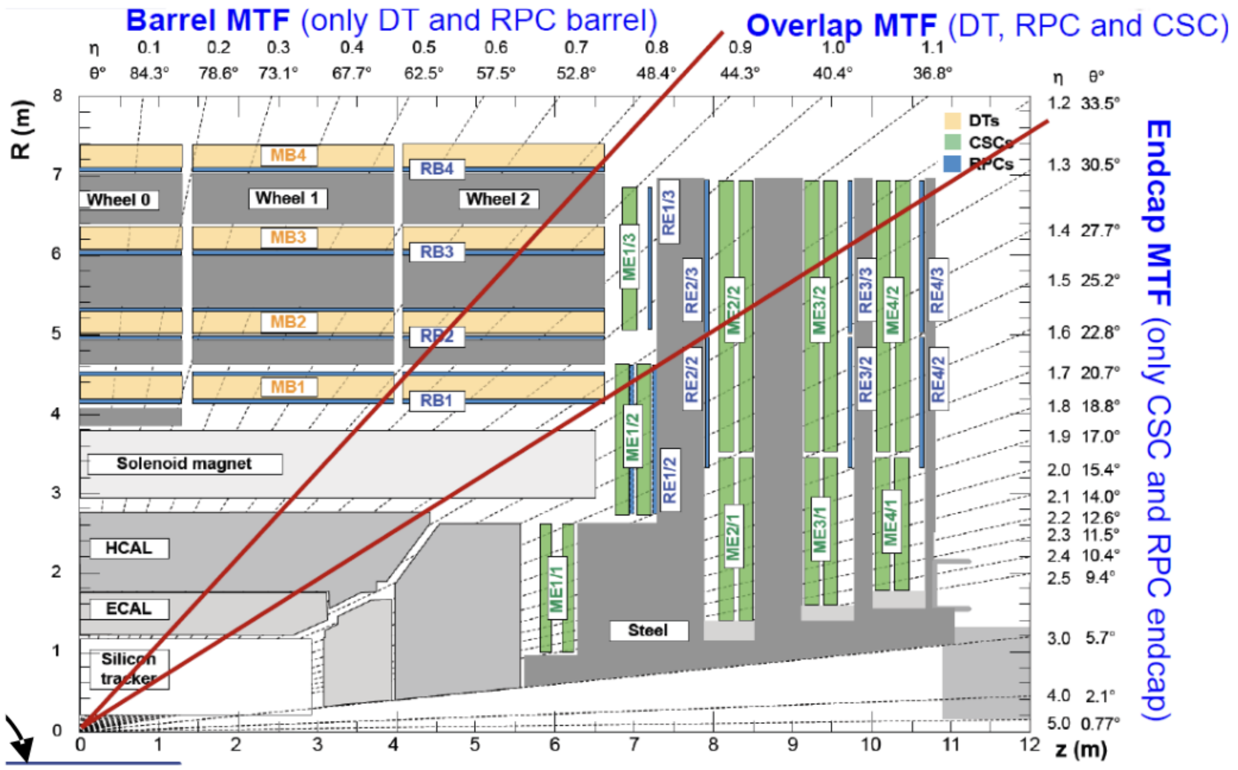
EMTF Introduction

Primary Goal of Endcap Muon Track Finder (EMTF)

- Part of the LHC CMS L1 Trigger, which is responsible for taking ~40M collision events per second and selecting ~100k of them for further processing
- EMTF builds muon tracks by associating muon stubs from different detectors and uses that track information to measure muon transverse momentum

Challenges

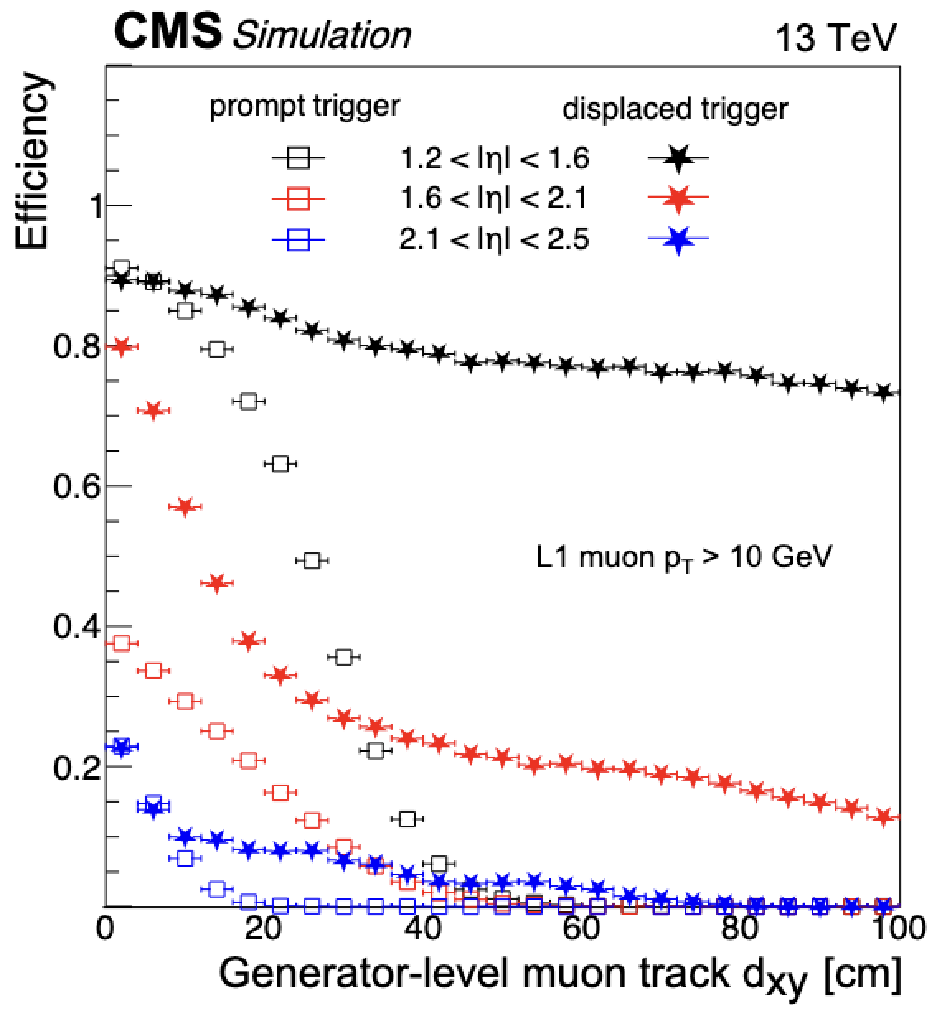
- Run 3 L1T latency
  - < 4 μs for L1 Trigger
  - ~500 ns for EMTF
- Non-uniform magnetic fields in the endcaps
- Large background at high pileup, especially at high η in station 1 (ME1/1)



- Cathode Strip Chambers (CSC)
- Resistive Plate Chambers (RPC)

Purpose of New NN

- New physics models predict long-lived particles that travel away from the collision point before decaying into muons
  - This would lead to muons originating from a vertex with a large displacement from the interaction point ($d_{xy} > 10$ cm)
- Muons with vertices that have a large displacement from the beamline are outside of the scope of the EMTF prompt muon trigger
  - Efficiency for these muons degrades quickly as the vertex is shifted away from the beamline
- Initial simulations for a displaced muon NN in EMTF, showed large efficiency gains for muons from displaced vertices
  - Plot [1] shows efficiency gains from an early NN model, this model did not fit into the FPGA but shows the potential of a new NN to expand our trigger to handle displaced muons



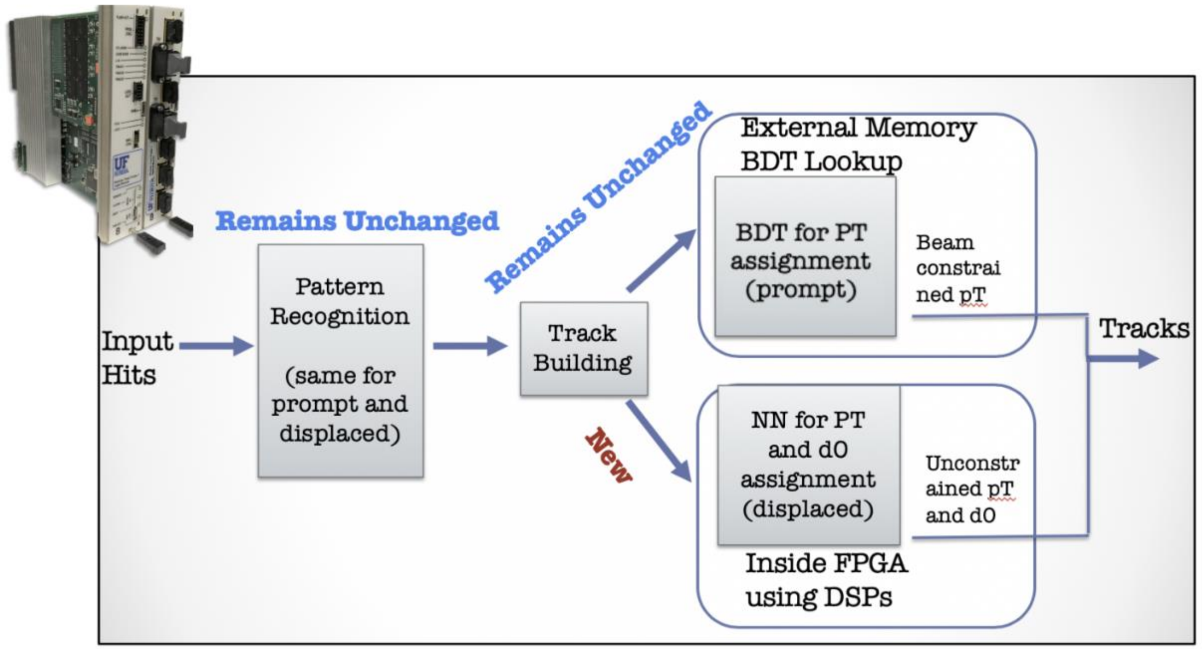
Timing and Resource Constraints

Since 2016, we have performed prompt muon pT assignment using a Boosted Decision Tree

- TWEPP Poster Session 2017 (lots of lessons mentioned here have applied to our new NN)
- This is performed using external RAM, which takes five 40 MHz clock cycles
- The NN parameter assignments happens in parallel with this external pT lookup, leaving ~100 ns for the NN after track serialization

FPGA Context

- FPGA: Virtex 7 chip on MTF7 uTCA board [2]
- Resources used without the NN
  - 74% of FPGA LUTs
  - 2% of DSPs
  - 76% BRAM
  - 25% Flip Flops
- Synthesis and Implementation time: ~3 hours
- Small changes in logic leading to timing failures

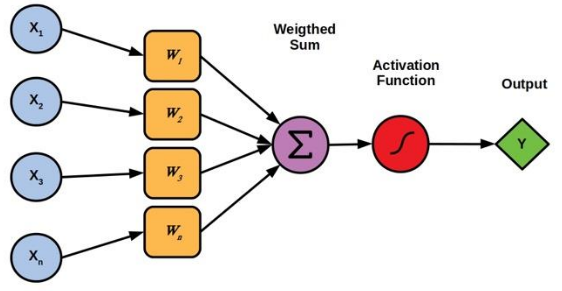

Input Hits
Remains Unchanged
Pattern Recognition (same for prompt and displaced)
Track Building
Remains Unchanged
New
External Memory BDT Lookup
BDT for PT assignment (prompt)
Beam constrained pT
NN for PT and d0 assignment (displaced)
Unconstrained pT and d0
Inside FPGA using DSPs
Tracks

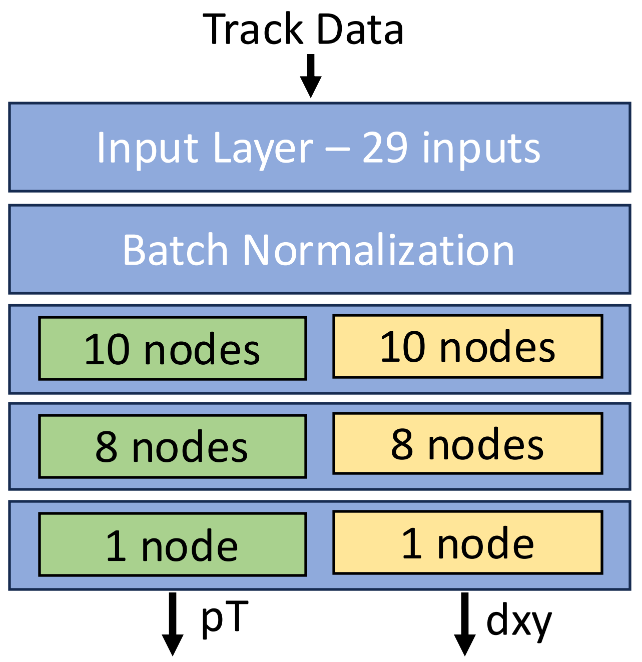
Final NN Constraints

- Pipeline the 3 tracks to a synchronous 120MHz clock
  - Use only 1 NN and input all 3 tracks within 1 BX
- The NN has 11 120MHz clock cycles to finish one track (~92ns)
  - 13 clock cycles to finish 3 tracks, and 1 sync clk cycle on each end of the NN
- Restrict LUT usage as much as possible, basically no restriction on DSP usage
- Basically: Fit into FPGA without breaking anything else

Neural Network Model

Single Node of DNN





Why a Dense NN?

- Given the complexities in the CMS endcap, BDT performed better than former manual method for assigning prompt p_T
- NN allows us to take advantage of ML to handle these complexities while using under-utilized DSP resources

Training Data

- Created in CMSSW emulator by firing a displaced single muon gun into the endcaps and using the tracks built from these displaced muons
- Parameters:
  - Flat in d_{xy} up to 120 cm
  - Flat in $1/p_T$

Keras NN Model

- Model has an initial batch normalization layer and 3 dense layers
  - All activations are Rectified Linear Units (ReLU)
- Train p_T and d_{xy} separately and each gets half of the NN nodes
- Trained using logcosh loss functions (similar to Huber loss used by BDT)
- Normalization is shared between NNs, so we ‘stitch’ the NNs together before converting to HLS using hls4ml [4]

Post Training Quantization

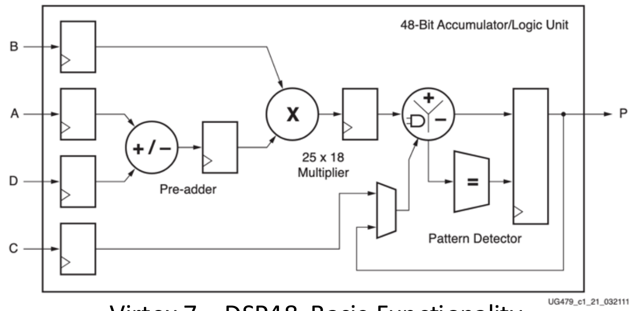
After training we quantize the model to fixed point precision using hls4ml

Choosing Quantization Values

- Layer weights need enough integer bits to hold largest weights
- Needs enough integer bits to hold fully accumulated values in each Dense layer
- Need to check that model still performs well and matches Keras output
- Leaving weights with higher precisions forces multiplication into DSPs,
- Leaving accumulations bit-widths large uses more LUTs and (maybe) latency

Final Values

- Inputs are 13 bits
- Weights and biases: 25 bits with 9 bit integers
- Accumulation type: 25 bits with 9 bit integers
- Activation outputs: 18 bits with 8 bit integers
- Final Output: 8 bit integer

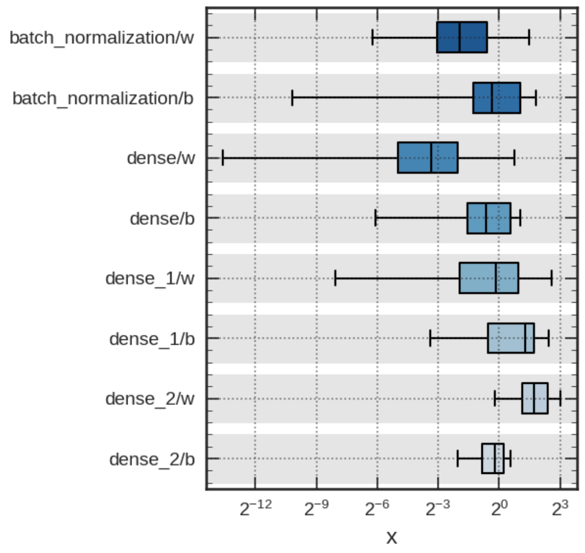


Virtex 7 – DSP48, Basic Functionality

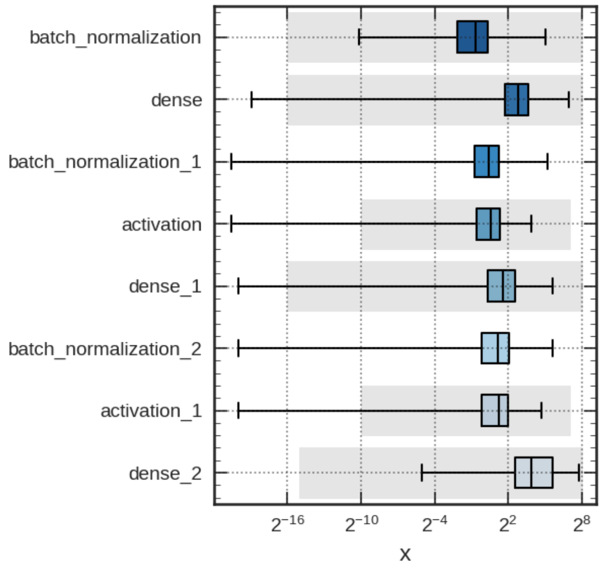
Side Notes

- Weight/Bias bit-widths can be set per layer, but were not in our case
- Accumulation bit-widths are based on default precision, but this behavior is easy to change

Distribution of Weights

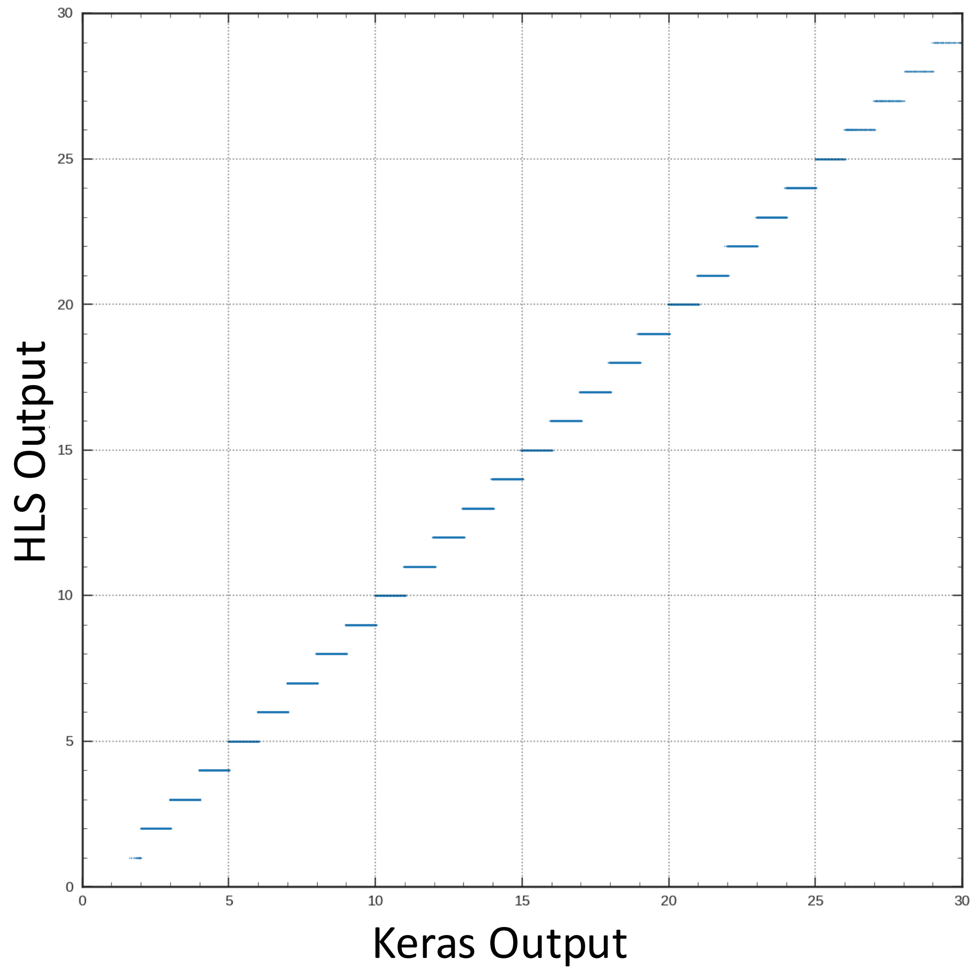


Distribution of layer Outputs

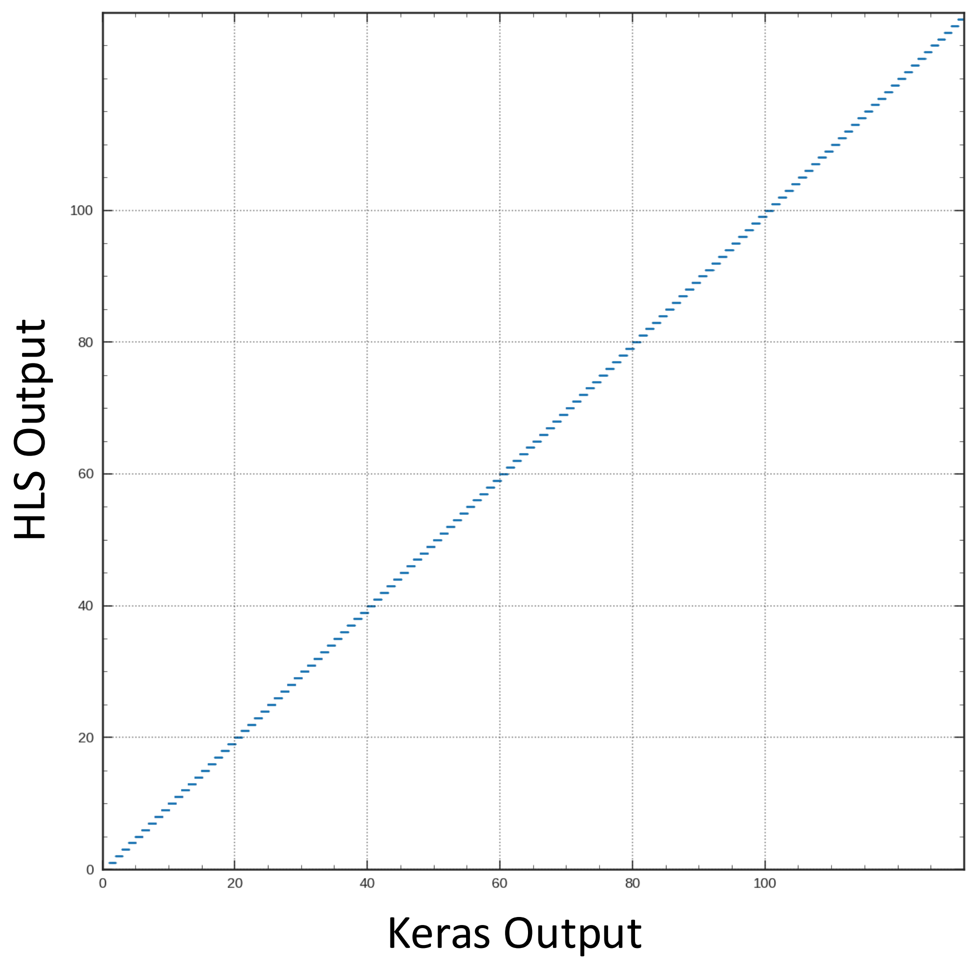


Keras Floating Point vs HLS Fixed Point Outputs

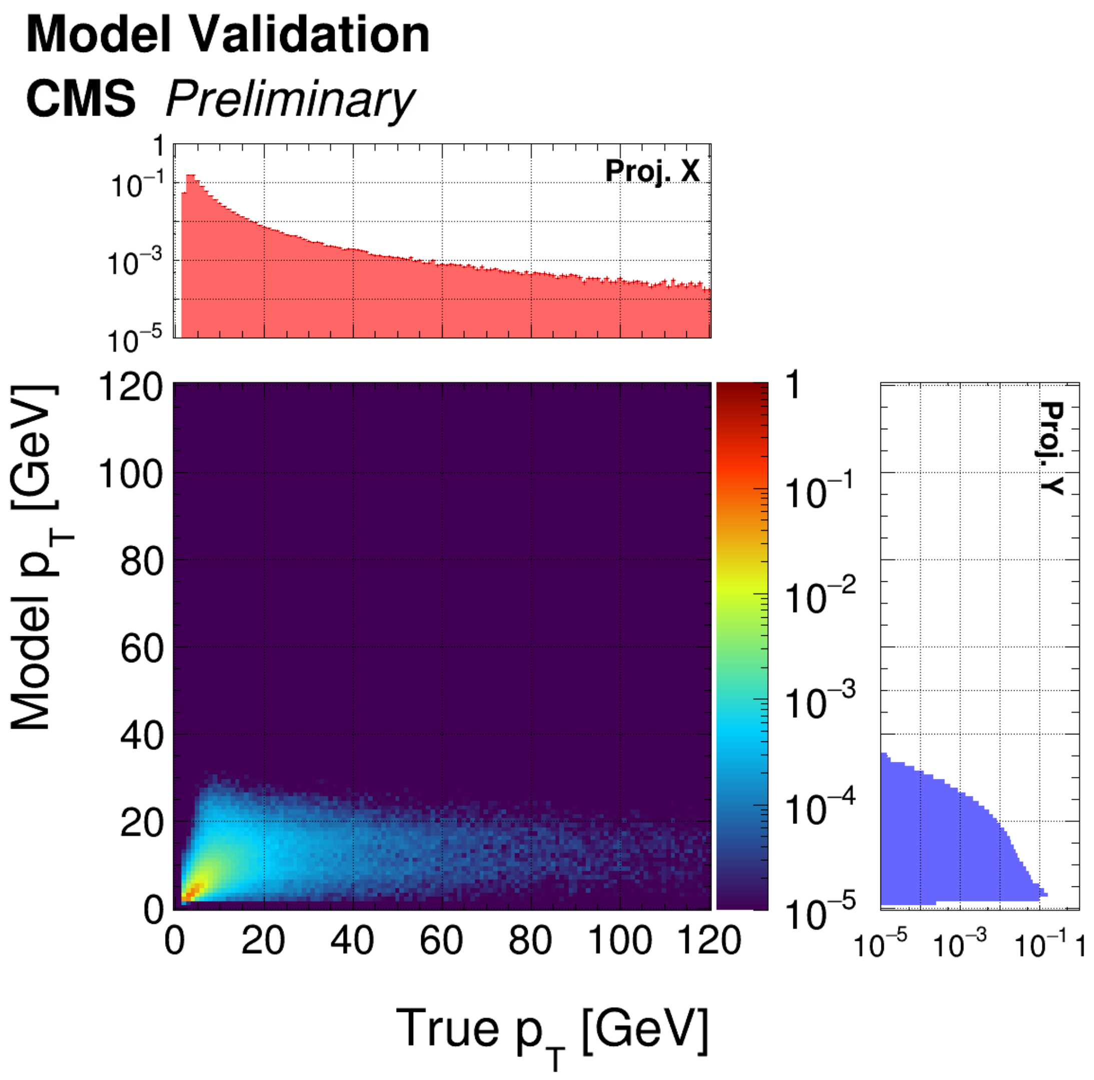
HLS vs Keras Output – pT
CMS Simulation Preliminary

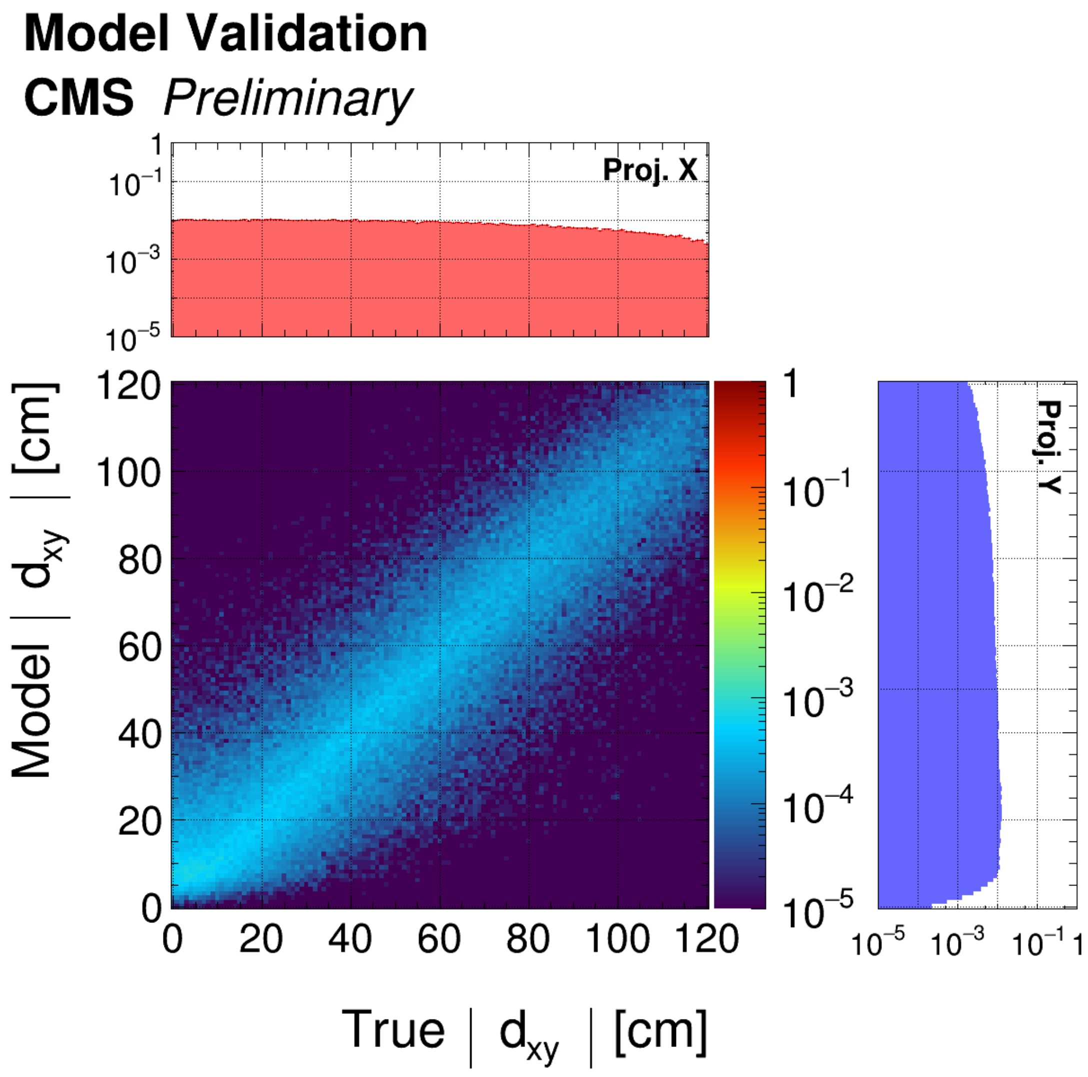


HLS vs Keras Output – Dxy
CMS Simulation Preliminary



Model Validation





Final Model

- Final Latency: 10 clocks (83 ns)

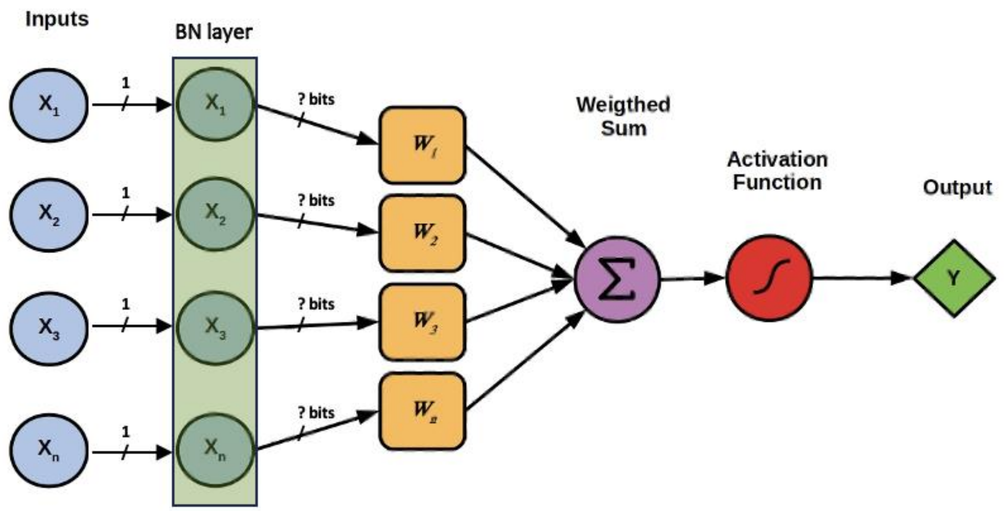
- Synthesis Report in HLS
  - LUTs: 14k (3.2%)
  - DSP: 767 (21.3%)

- Final Vivado Implementation resources with NN
  - LUTs: 76.3% (from 74%)
  - DSPs: 14.4% (from 2%)

- Model has been running since June 2023 and has been used for triggering from the start of runs in 2024
  - This was the first NN running in CMS L1T FPGAs for data taking
  - Model performance plots in data will be available soon!

Lessons Learned – Wrappers and Batch Normalization

- About half of our inputs are 1-bit, by using a standard hls4ml model, the HLS is synthesized without this information
  - Can solve this by wrapping hls4ml model in a wrapper with real bit-widths as inputs and running that in HLS
  - This will be known by the tool during Vivado synthesis regardless, but latency is already set by this point
- Normalizing inputs can really help with training, but if your NN inputs are low precision numbers this comes at a very high cost
- With a wrapper and without an initial BN layer, about half of our first dense layer multiplications are: (weight * 1-bit number)
  - HLS can easily handle these and start pre-accumulating these numbers while other numbers are multiplying. Leads to lots of resource savings and lowers latency



Lessons Learned – Quantization Aware Training

One problem we have run into with our models for this ML problem is that we cannot quantize to very low precisions without degrading performance

- Dense layer weights can go to ~10 bits
- Relu Activations are stuck at ~12 bits

This puts us in an in-between area where we do not see LUT or latency savings from quantization directly

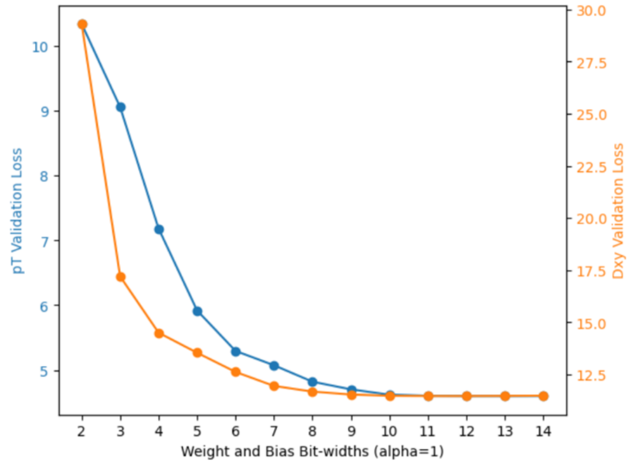
- DSP usage goes down, but this is not a number we are worried about
- LUTs at this range go up because they start replacing DSPs but are not small enough to make up for that

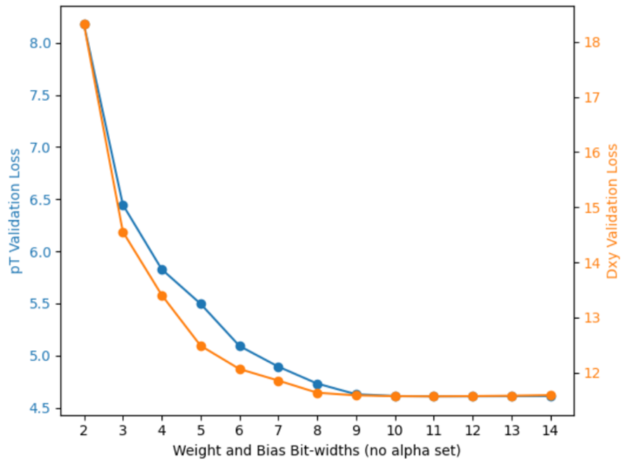
The largest benefit we see from this is by allowing us to tune our accumulation bit-widths to smaller values while still retaining performance

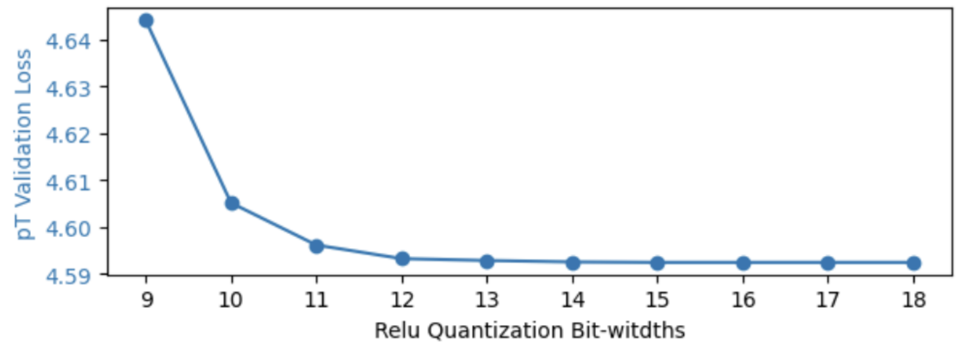
- Leads to large LUT savings
- **This may not pair well with quantizing relu activations and can lead to large discrepancies between Keras and HLS
    - Qkeras accumulations happen at floating point precision and the values are rounded to the quantization value at the end, so if we don't accumulate our numbers with enough bits we end up rounding the wrong direction and the output is further off than it would be without quantization at the activation function

Quantization Aware Training – Bit-width Effect

CMS Simulation Preliminary

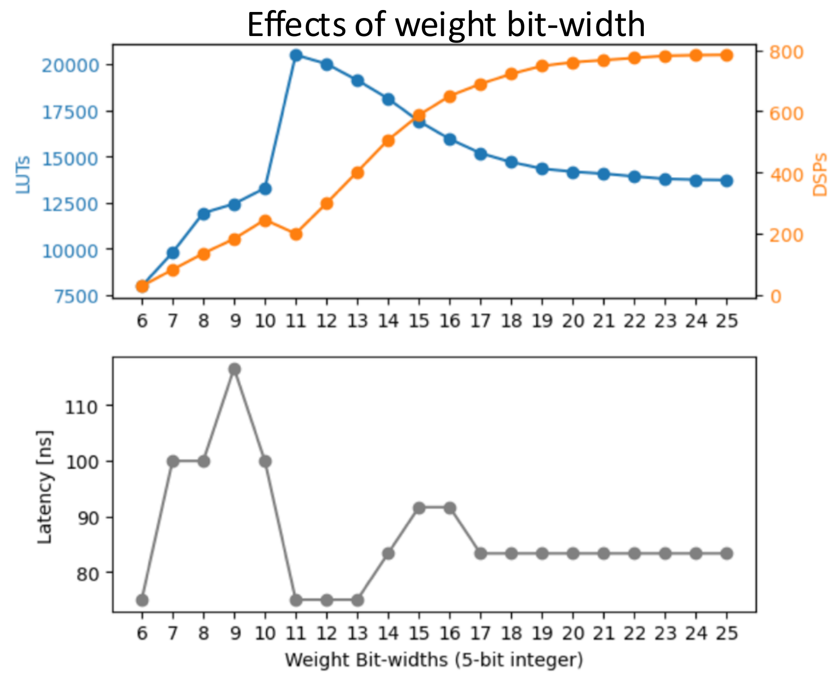


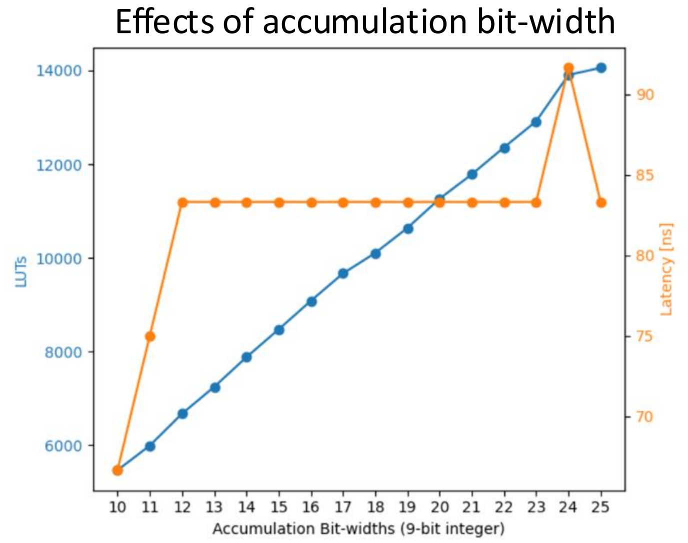




Lessons Learned for Future Models - Quantization

- Initial model used a uniform post-training quantization
  - This leaves a lot of room for improvement, to reduce latency and resources to fit in larger models with better performance
- Weight bit-widths
  - DSP usage is largely effected by weight bit widths, with our constraints this isn't a problem
  - LUT usage at in-between bit-widths is actually worse as multiplications shift into LUTs, but at small ranges there are savings in LUTs and DSPs
  - Latency effects are often hard to predict until you get to small bit-widths. Even in DSPs, all of our multiplications are done in parallel
- Accumulation bit-widths
  - Accumulations of multiplications are all handled in LUTs, so the bit-width that you accumulate is directly correlated with LUT usage
  - Accumulations can also have a large effect on latency, this effect can be better taken advantage of by inlining layers in HLS
  - These plots were made using regular fixed point numbers (truncation and wrapping). In my experience, adding rounding or saturation fixed point numbers in HLS is worse than just expanding the accumulation bit-width





Conclusions

With tight latency constraints and a nearly full chip, its still possible to squeeze in a neural network. With hls4ml converting models to HLS for use in FPGAs is simple and with some additional tweaks, you can fit models in tight constraints.

My main tips for anyone trying to do the same using hls4ml:

- Standard NN optimization strategies
  - ReLU activations
  - Model pruning
- Remove input batch-normalization and wrap NN model in wrapper with real bit-widths
- Dense Layer Accumulations take a lot of time and LUTs
  - It really just takes experimentation to see how changing these effects performance, resources, and timing, but there are savings to be had
- Use Quantization aware training
  - There are still benefits even when you cannot quantize model to very small values an significant benefits if you can
- If you really need to squeeze the latency - try inlining everything using HLS inline pragma
  - Inlining may also help take advantage of the other changes
  - Don’t be surprised if something weird happens with timing or resources (not uncommon in HLS)
- Meeting timing in HLS doesn’t mean meeting timing in a full FPGA
  - Experiment with the HLS synthesis frequency and leave room for longer path delays with fuller chips
  - To meet timing at 120 MHz (8.33 ns), we synthesized the HLS using a clock of 6.5 ns

Using these optimizations, we can get this model down from 83 ns to ~58 ns (7 clks). This has opened up extra latency for us to make our model much larger and hopefully we can get a new model in this winter with improved performance.

References

[1] CMS Collaboration, “Development of the CMS detector for the CERN LHC Run 3”, arXiv:2309.05466.

[2] D. Acosta et al., “The CMS Modular Track Finder boards, MTF6 and MTF7”, Journal of Instrumentation 8 (2013) C12034.

[3] D. Acosta, A. Brinkerhoff, A. Carnes, I. Furic, S. Gleyzer, K. Kotov, J.F. Low, A. Madorsky and B. Scurlock, “Boosted Decision Trees in the CMS Level-1 Endcap Muon Trigger”, Proceedings of Science, TWEPP- 17 (2017) 143

[4] Official hls4ml Documentation

Backup

EMTF Algorithm

- EMTF is split into 6 sectors per endcap, each covering 60°, with overlap for full coverage
- Our algorithm runs independently for each sector

Primitive Conversion: Muon stubs from different chambers are converted into uniform EMTF Stubs, which include local sector ϕ and θ coordinates and other info depending on chamber type

↓

Pattern Finding: Use muon patterns to find stubs in different stations that are consistent with a muon track

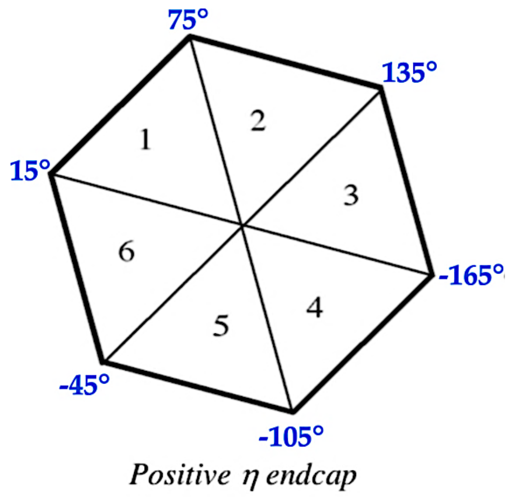
↓

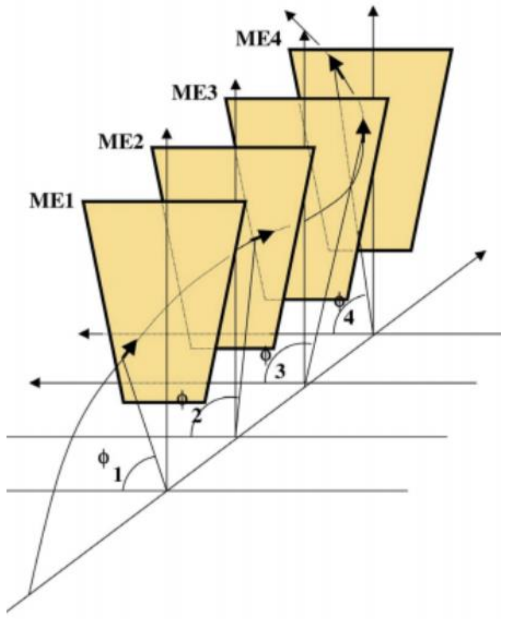
Track Building: Tracks are built by attaching muon stubs that best fit the original matched pattern for each station.

↓

Parameter Assignment: First, calculate station to station track parameters (Δϕ, Δθ)
- For prompt muons, a pre-computed boosted decision tree lookup table is used to measure p_T
- For displaced muons, a NN is used to measure p_T and d_{xy}

EMTF Sectors





Final Track Information – Run 3 ML Inputs

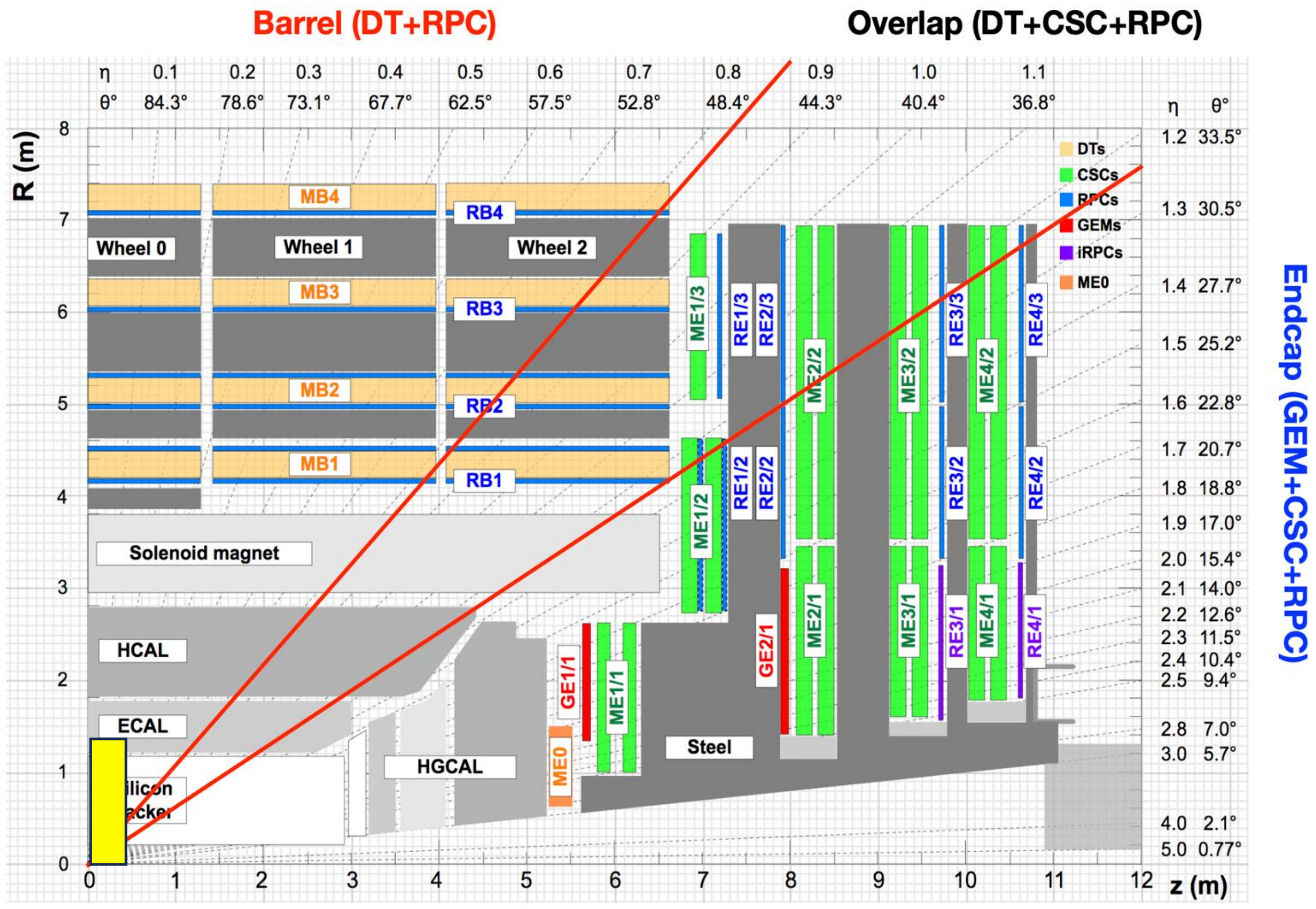
- Algorithm builds up to 3 tracks per FPGA
- The final track for each muon includes up to 4 trigger primitive hits, one per station
  - Hits may be a combination of CSC and RPC hits
  - These hits have info from the original TP as well as the converted EMTF local sector ϕ and θ coordinates
  - Additionally, station to station parameters are calculated (Δϕ, Δθ) between hits

NN inputs available

Input Name	Param count	Bit count	Info
Delta Phi Values	6	13	Stations: 12, 13, 14, 23, 24, 34
Sign of Delta Phis	6	1	
Delta Theta Values	6	7	
Sign of Delta Thetas	6	1	
CSC pattern	4	4	Converted to Run 2 style
Theta	1	7	

Training Data – Displaced Muon Gun

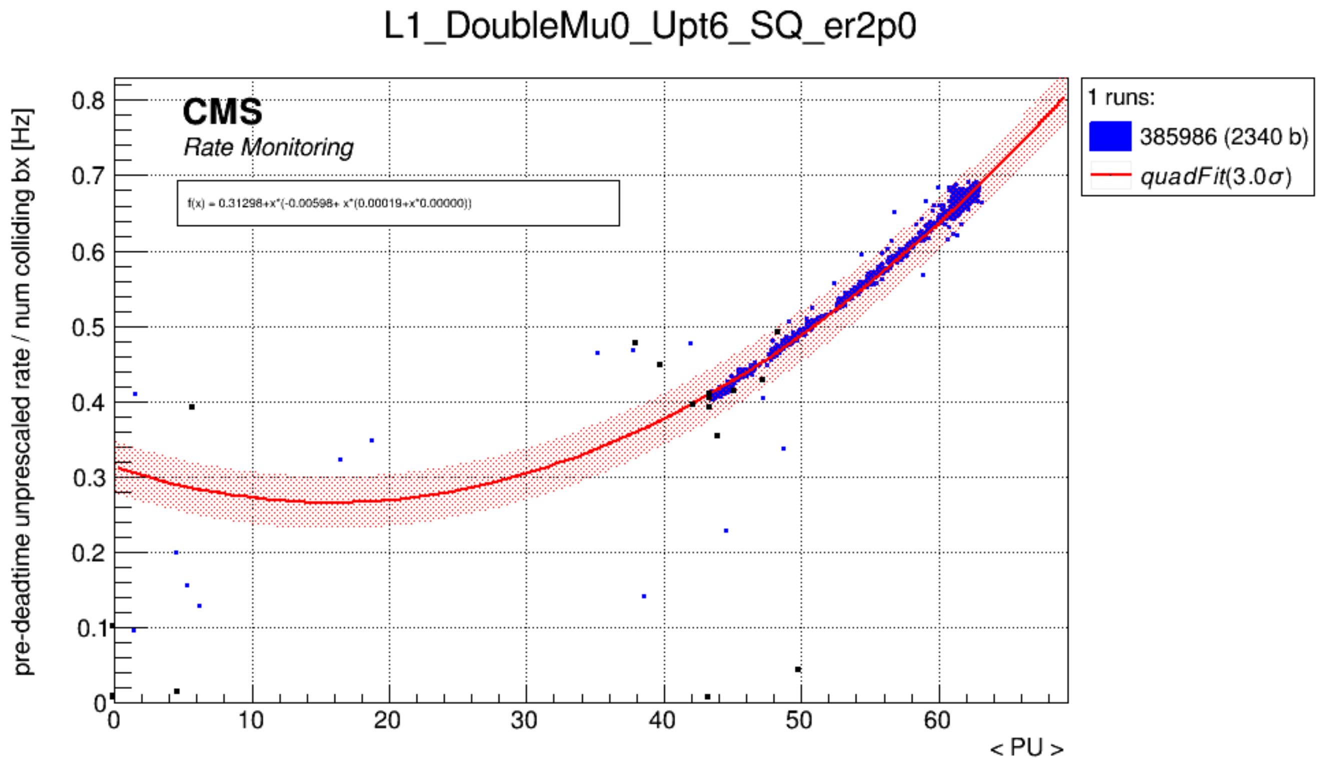
- We create training data in the emulator by firing a single muon gun into the endcap and using the tracks built from these displaced muons
- Muon gun
  - Flat in dxy up to 120cm
  - Vertex smearing in z
  - Flat in 1/pT
  - Vertex range shown in yellow



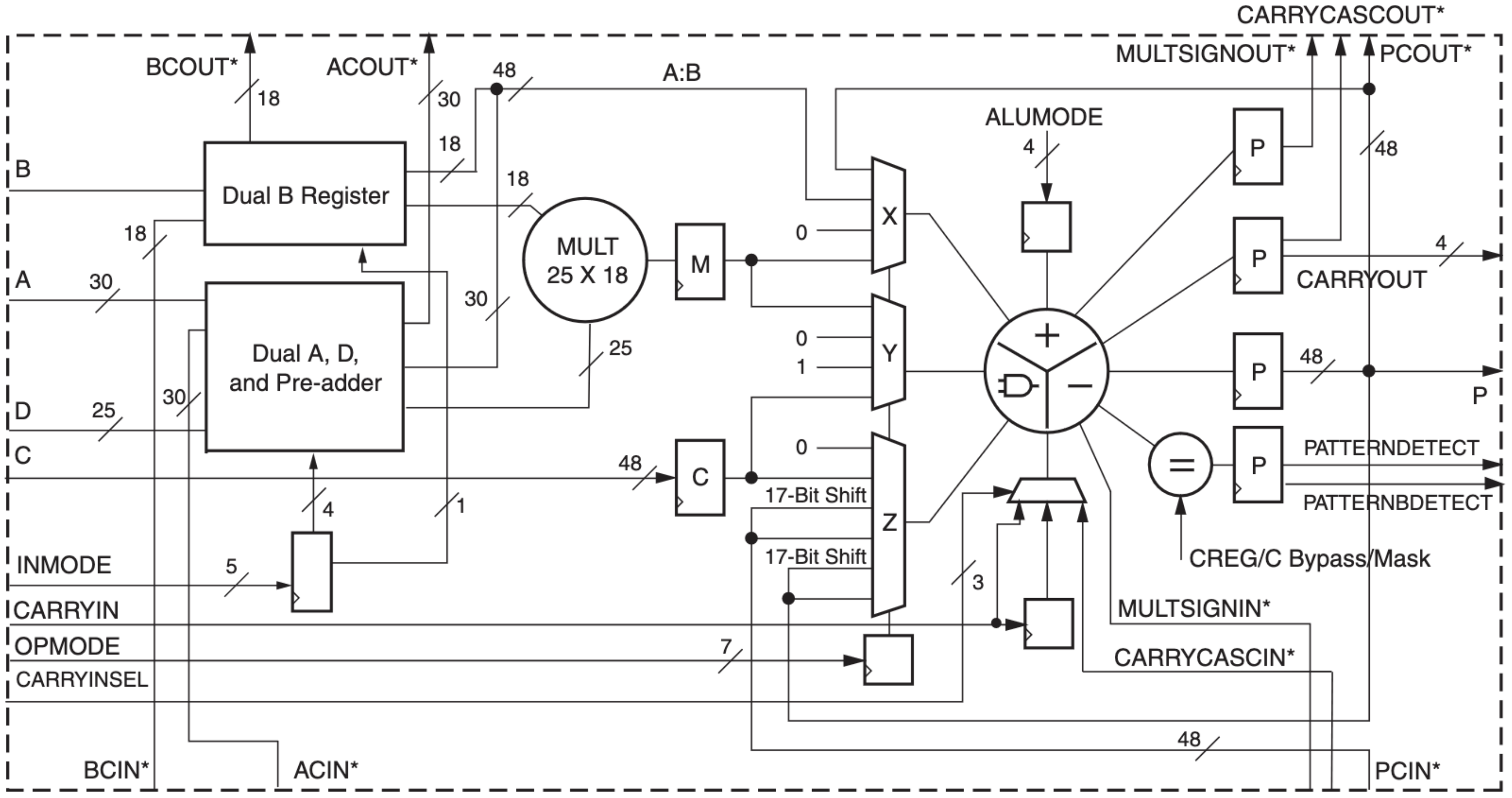
Trigger Rates coming from NN in Collisions

- New seed was implemented for 2024 data taking that makes use of the EMTF NN
  - Extends the eta coverage for to the endcaps
  - Triggers on 2 high quality muons
    - Vertex unconstrained pT > 6GeV
    - Abs(eta) < 2

- Plot shows rate vs pile-up for this new displaced dimuon seed for LHC run number 385986



Virtex 7 – DSP48



*These signals are dedicated routing paths internal to the DSP48E1 column. They are not accessible via fabric routing resources.

UG369_c1_01_052109

Floating Point in HLS

- It is possible to use HLS for floating point dense layers
    - 16-bit, 32-bit, and 64-bit floats are supported by default

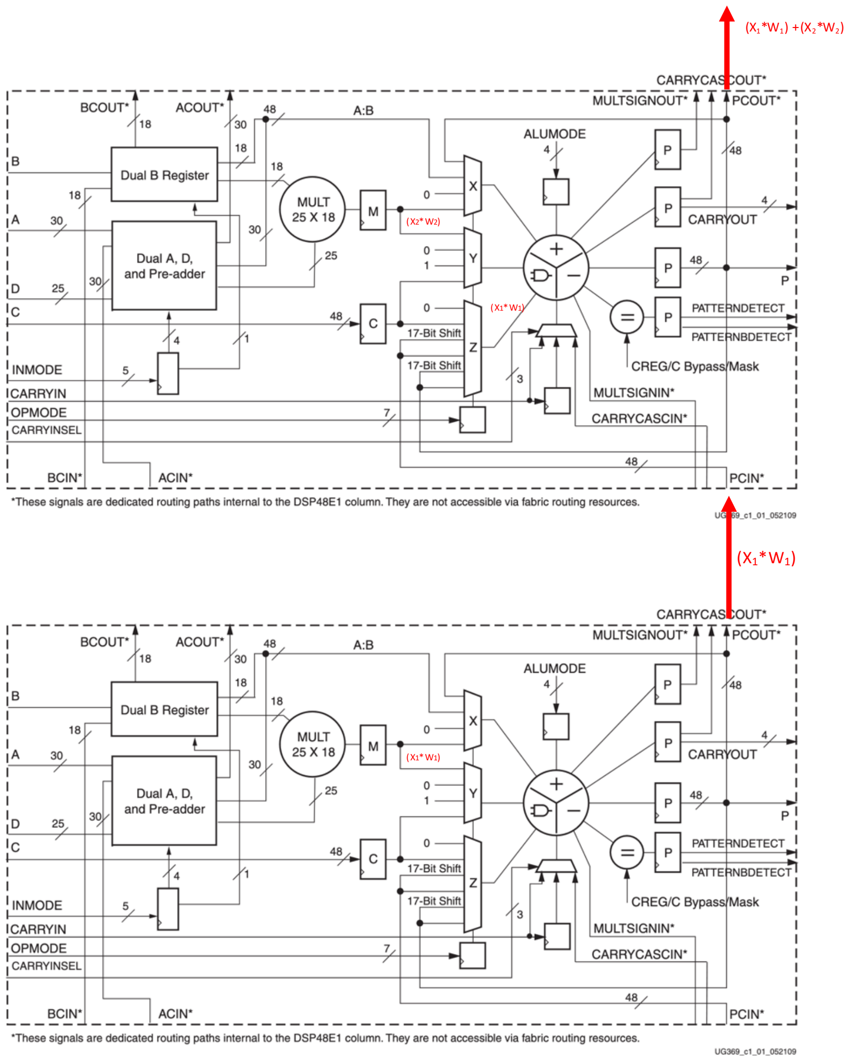
- Just for some numbers, with our first dense layer (29 inputs -> 20 outputs)
    - Multiplications take ~3clks (only tested at 120MHz)
    - Accumulations take ~30clks and must be programmed to run in parallel
        - Floating point accumulations are dependent on the accumulation order, so to get them to run in parallel, they need be programmed by hand to add in pairs.
    - For this dense layer, 580 DSPs were used (29*20) and ~120k LUTs were used, so things really add up fast, but almost all of the time and resources are from the accumulations not the multiplications.

Using DSPs for Accumulation

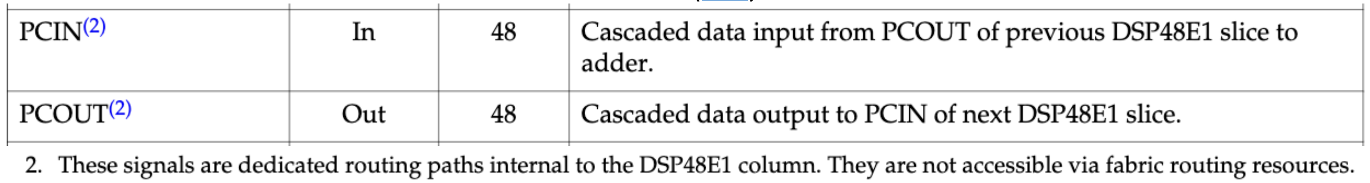
- Xilinx FPGA DSPs have dedicated hardware for accumulating the results of multiplications

- Either need second multiplication to come from PCIN or C (in diagram)
  - In both cases, the multiply output needs to be the full bit-width possible
    - There is no logic to cut multiplication bw before accumulator
    - For inputs of 25 bits and 18 bits, full bw possible is 48 bits
  - For PCIN, requires PCOUT to be directly connected to PCIN of another DSP
    - Cutting the multiplication output before the accumulation would require logic between PCOUT and PCIN that doesn't exist

- We have tested using this hardware for the accumulations by coupling multiplications and doing a first layer of accumulation in parallel (in diagram)



From DSP48E1 User Guide (link)



PCIN(2)	In	48	Cascaded data input from PCOUT of previous DSP48E1 slice to adder.
PCOUT(2)	Out	48	Cascaded data output to PCIN of next DSP48E1 slice.

2. These signals are dedicated routing paths internal to the DSP48E1 column. They are not accessible via fabric routing resources.