Boston University Office of the Provost
Undergraduate Research Opportunities Program

Machine Learning Improvements to the CMS Trigger System

Sheila Sagear[1], Jennifer Ngadiuba[2], Indara Suarez[1,2]

[1]Department of Physics, Boston University [2]European Organization for Nuclear Research (CERN)

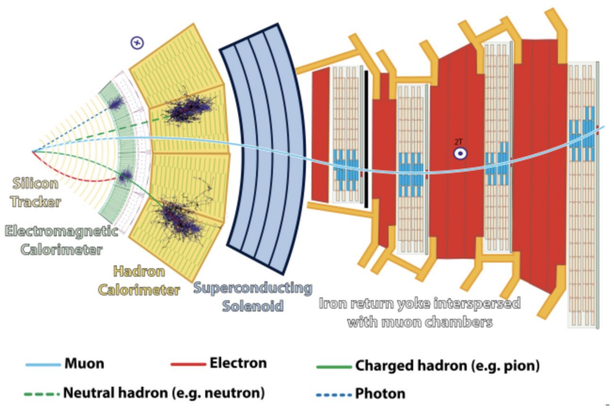
The CMS Trigger System

The Compact Muon Solenoid (CMS) is a particle physics detector on the Large Hadron Collider (LHC) at CERN. The LHC accelerates particles close to the speed of light and collides them together. When the particles collide, they break up, and CMS detects the resulting particles.

There are up to 1 billion collisions per second in CMS, which produces hundreds of terabytes of data per second. It is impossible to store and keep this much data, so we must decide what to keep and discard.

The CMS Trigger System makes extremely fast decisions on which data to keep, based on the possibility of new physics. It uses FPGAs (Field Programmable Gate Arrays), a programmable chip that performs at high speed.

Figure 1: a cross-sectional slice of CMS. Particles collide at the far left, and the particles that result from the collision travel through layers of detectors.



Source: CERN

hls4ml

`hls4ml` (High-Level Synthesis for Machine Learning) is a Python package for machine learning inference in FPGAs. It turns machine learning models into firmware implementations that can be used in FPGAs.

`hls4ml` can be used with any machine learning model, but its goal is to improve the CMS Trigger System using machine learning models to recognize particle jets ("jet tagging").

Jet-Tagging ML Models

We train machine learning models to recognize the jets of different particles.

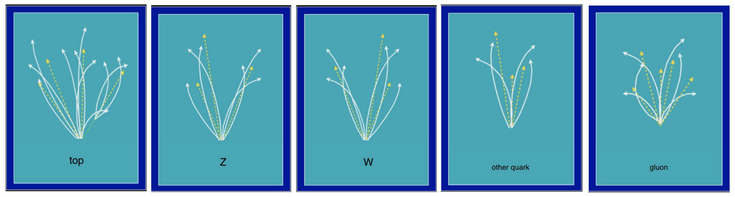
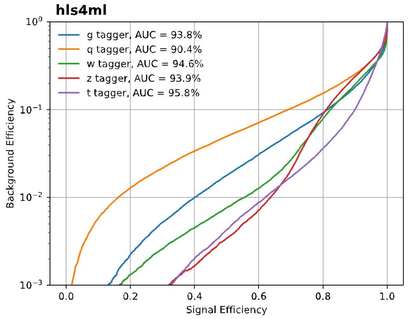


Figure 2: Different particles leave different trails in CMS.

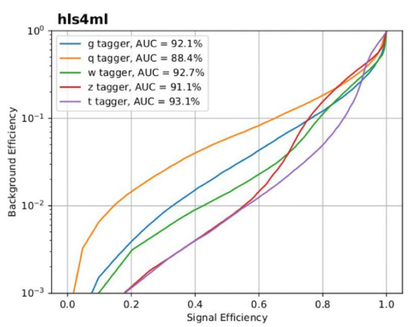
We want to produce the most accurate model possible with the lowest precision: lower precision = lower resource usage when implemented into FPGAs. Our benchmark model uses floating-point precision (32 bits).



16 inputs
64 nodes
32 nodes
32 nodes
5 outputs

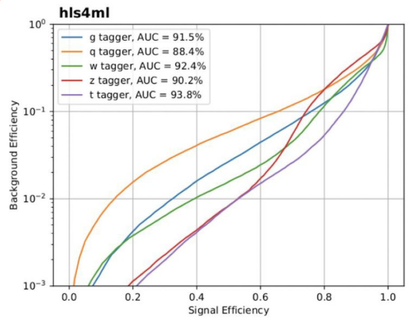
Figure 3: Floating-point precision model and architecture.

We try binary (2 bits) and ternary (3 bits) precision models and measure their accuracy against the floating-point models:



16 inputs
448 nodes
224 nodes
224 nodes
5 outputs

Figure 4: Binary precision model and architecture.



16 inputs
128 nodes
64 nodes
64 nodes
64 nodes
5 outputs

Figure 5: Ternary precision model and architecture.

Binary and ternary models perform sufficiently well – they are nearly as accurate as the floating-point precision model.

Latency and Resource Usage

When we use `hls4ml` to synthesize a model, we can analyze its latency (the time required to complete a calculation) and resource usage.

Architecture	AUCs [%]	Average accuracy	Minimum latency [µs]	DSPs [%]	LUTs [%]	FFs [%]	BRAMs [%]
float model (16x64x32x32x5)	90 - 96	0.75	0.060	60	7	1	0
Optimized BNN (16x448x224x224x5)	88 - 94	0.72	0.210	0	15	7	0
Optimized TNN (16x128x64x64x64x5)	88 - 94	0.72	0.110	0	6	1	0

Table 1: Latency and resource usage for floating-point, binary, and ternary model.

The binary and ternary models use fewer DSPs (Digital Signal Processing blocks) than the floating-point model. DSPs are generally the limiting FPGA resource. In the future, we hope to lower the minimum latency in the binary and ternary models to be smaller than the floating-point model. Latency is controlled by the reuse factor (parallelization of calculations in synthesis) and should be as small as possible.

Funded by Boston University's Undergraduate Research Opportunity Program.