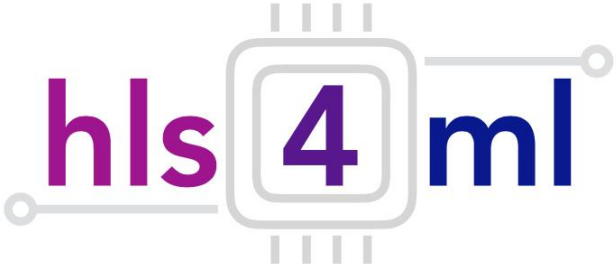


Machine Learning with FPGAs for Trigger and Detector Systems

Vladimir Lončar (CERN)

on behalf of the FastML team
fastmachinelearning.org

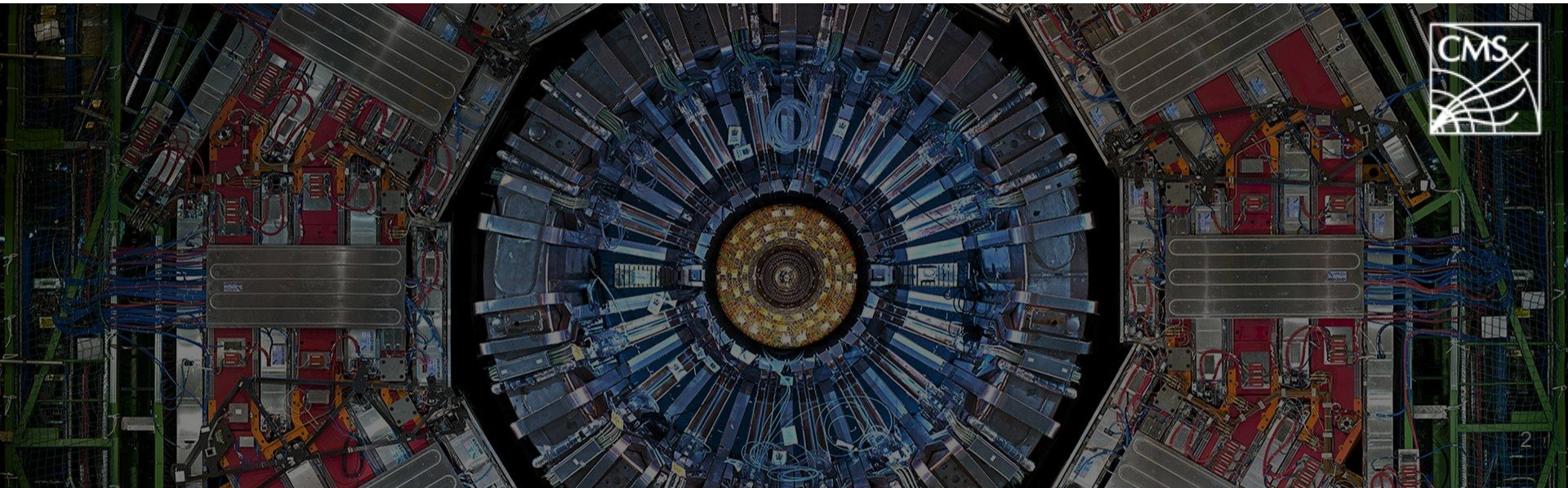
13th Terascale Detector Workshop (6-8 April 2021)

The Big Data of Large Hadron Collider (LHC)

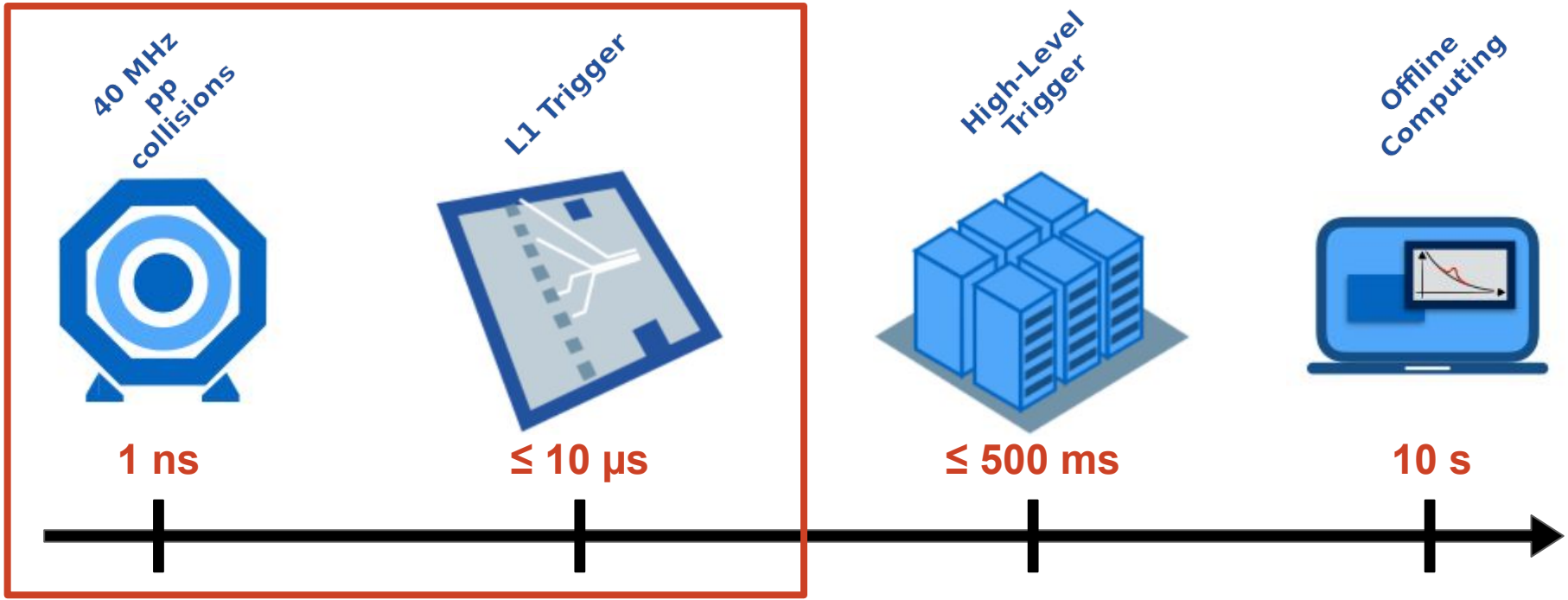
LHC proton beams collide at a frequency of 40 MHz, producing data rates of O(100 TB/s)

“Triggering” - Filter events to reduce data rates to manageable levels

- Very strict latency constraints! O(1μs)



How do we process data?



Challenge: strict latency constraints!

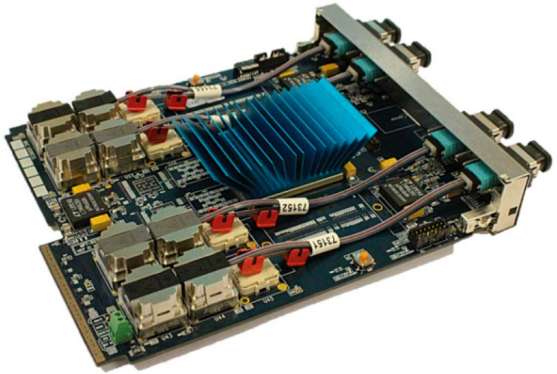
L1 trigger hardware

We need fast processing of raw data *O*(µs)

- Not possible to use common hardware, such as Intel CPUs, nor common operating systems

Must be flexible and modular to support reconfiguration and upgrade/maintenance of modules

➔ Field-programmable gate arrays (FPGAs)



Detector upgrades for HL-LHC

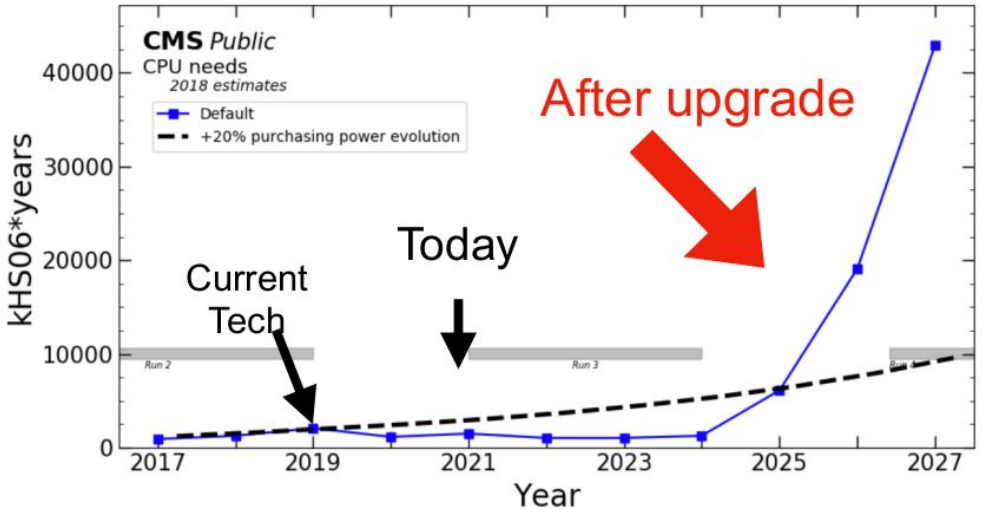
Event size will have to be **10x** larger

We will have to take data at **5x** the current rate

With increased beam intensity physics quality degrades, especially with L1 algo gets worse

Flat budget for computing resources → Current data processing paradigms will not be sustainable!

Can **deep learning** be a way out?



Bring DL to FPGA for L1 trigger with **high-level synthesis for machine learning**

hls4ml - A user-friendly tool that enables fast inference on edge devices

- Dedicated optimization for each network - **O(μs) inference**
- Automatic firmware generation workflow
- Commonly FPGAs, but with expanding hardware support

Input: pre-trained models from popular deep learning tools - Keras, TensorFlow, PyTorch, ONNX

Output: C++/HLS optimized for the target hardware architecture

https://fastmachinelearning.org/hls4ml/

hls4ml pipeline

Supported DL frameworks:

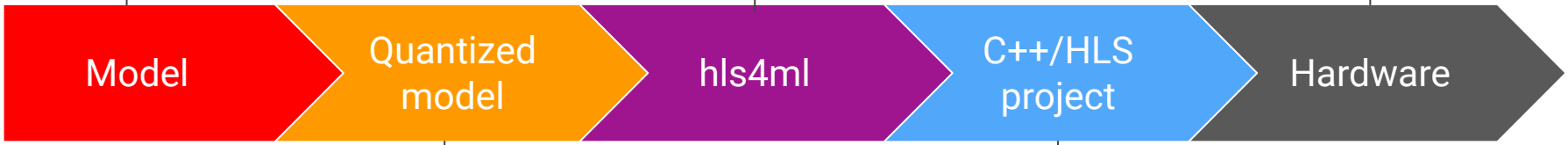
Keras TensorFlow PyTorch ONNX

Model conversion, optimization, profiling & tuning

hls4ml

Xilinx FPGAs, Intel/Altera FPGAs, Intel x86 CPUs

XILINX intel



Quantization and pruning techniques:

- QKeras + AutoQ (Keras)
- Brevitas (PyTorch)

VIVADO Mentor A Siemens Business

oneAPI Quartus Prime

Features

Supported architectures:

- **Deep Neural Networks (DNNs)**
  - Zero-suppressed weights - arxiv:1804.06913
- **Quantization**
  - Binary/Ternary layers (computation without using DSPs) - arxiv:2003.06308
  - Google QKeras integration - arxiv:2006.10159
- **Convolutional Neural Networks (CNNs)** - arxiv:2101.05108
- **Graph NNs** - GarNet architecture - arxiv:2008.03601

NEW

- **Recurrent Neural Networks (RNNs)**
- **PyTorch quantization with Xilinx Brevitas -** arxiv:2102.11289
- **New hardware platforms**
  - Intel FPGA - Quartus
  - ASICs - Catapult - arxiv:2103.05579
  - Intel x86/Xe - oneAPI

WIP

Efficient implementations of NNs with HLS

Meeting latency constraints

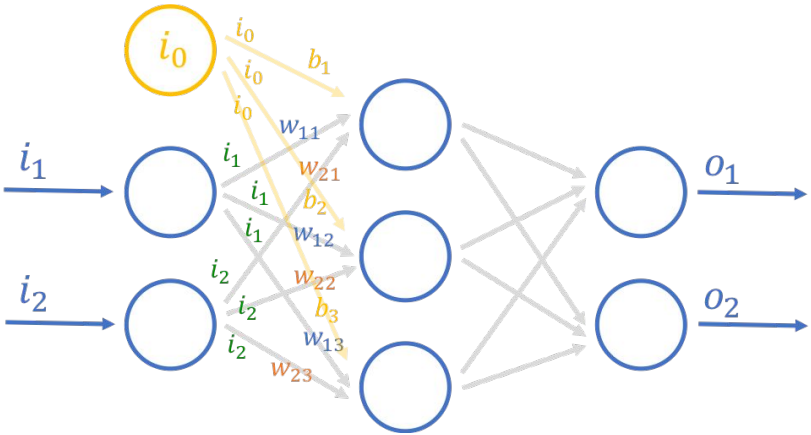
- FPGAs rarely operate at over 500 MHz
- Typically in 200-400 MHz range
- For practical reasons, at L1 trigger we target the frequency of multiple of 40 Mhz
  - Usually we have *O*(10) cycles to complete the entire algorithm!

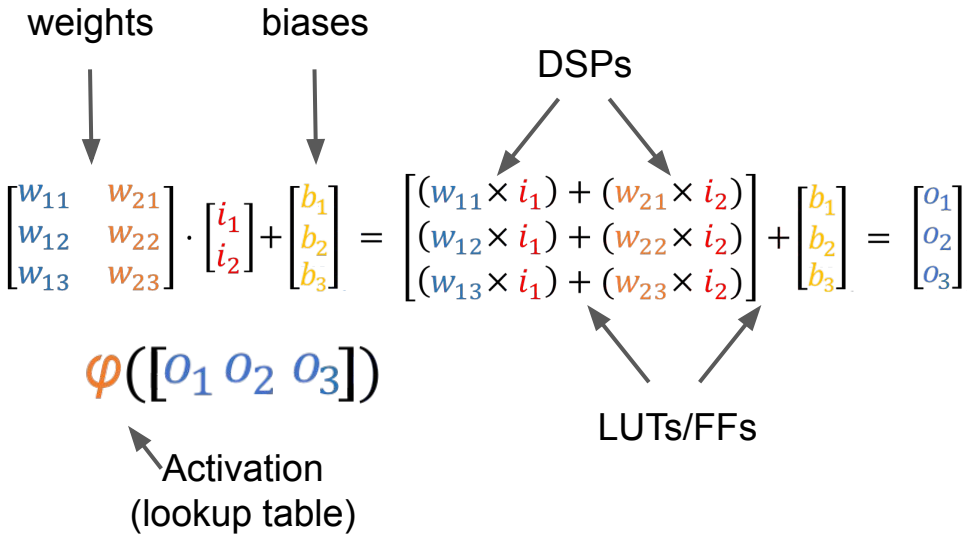
Algorithm design:

- Exploit parallelism as much as possible (unrolling)
- Remove branching as much as possible
- Store all weights in registers of an FPGA ⬅️ **distinguishing feature!**
  - Limited by the amount of resources, has heavy impact on model design

Low-latency matrix multiplication on FPGAs

NN inference → matrix multiplication





Est. latency: **3 cycles** (multiplication + addition + lookup)

Fast convolutional neural networks

New

Direct implementation of convolution is a bad fit for HLS

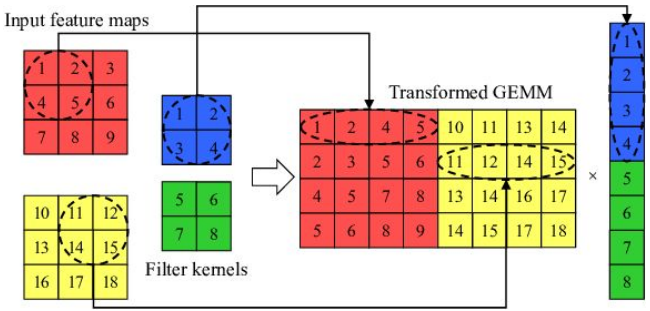
$$\boldsymbol{Y}[v,u,n] = \boldsymbol{\beta}[n] + \sum_{c=1}^{C}\sum_{j=1}^{J}\sum_{k=1}^{K} \boldsymbol{X}[v+j, u+k, c]\, \boldsymbol{W}[j,k,c,n]$$

```
for out_c in range(n_filt):
  for i in range(height):
    for j in range(width):
      for c in range(n_chan):
        for fi in range(k_height):
          for fj in range(k_width):
            elem = input[i+fi, j+fj, c]
            w = weights[fi, fj, c, out_c]
            output[i, j, out_c] += elem * w
```

- Six nested loops, results in long latency

Convolution via matrix multiplication (*im2col*)

- Build an input matrix and multiply it with weight matrix
- Sequential approach:
  - Collect pixels from input image into an internal buffer until we can compute one output
  - Accommodate fast and predictable streaming by eliminating all branches and corner case checks
  - Encode all operations into high-level convolution-specific instructions

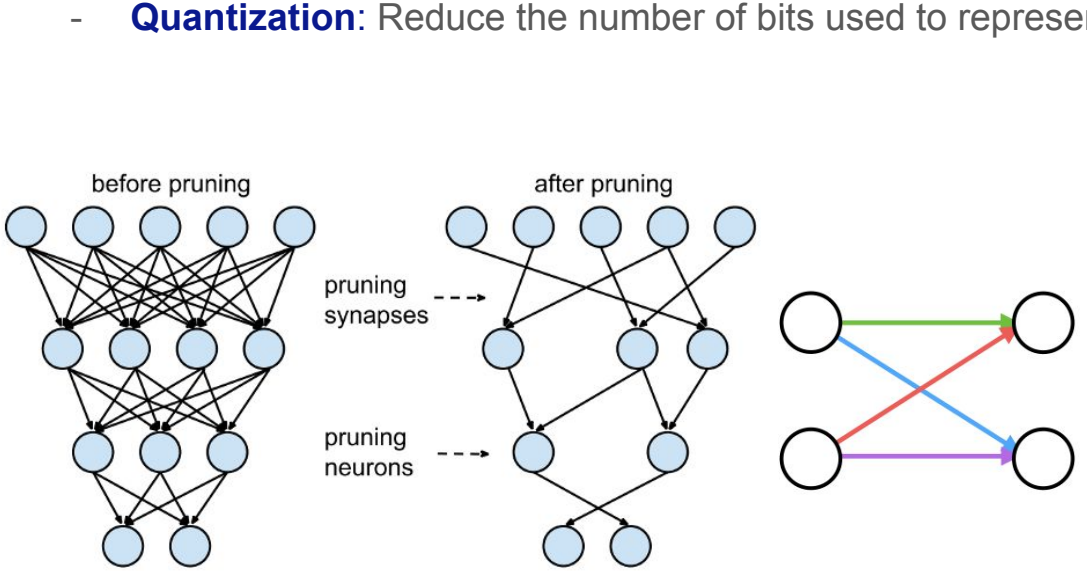


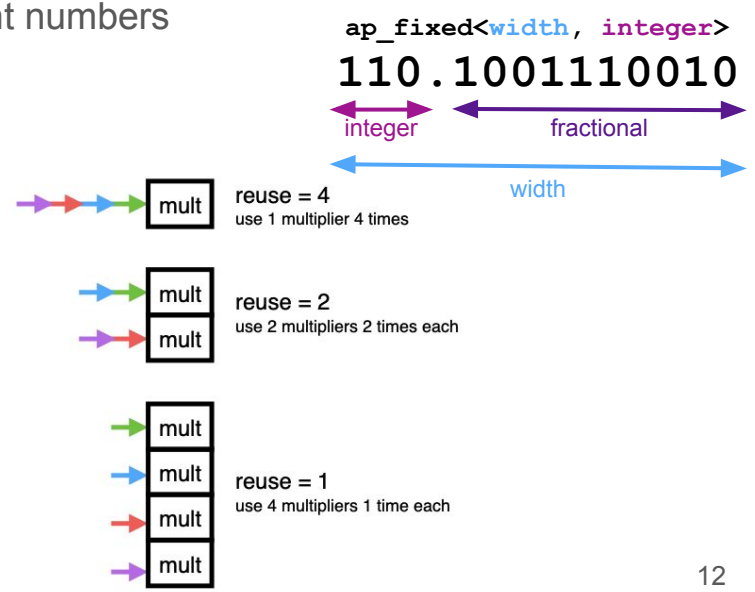
➔ arxiv:2101.05108

Making the model smaller

Exploiting FPGA hardware is key to achieving performance goals

- **Parallelization (reuse)**: Control the inference latency versus utilization of device's resources
- **Pruning**: Remove the connections that play a small role in the final decisions
- **Quantization**: Reduce the number of bits used to represent numbers





Parallelization

Exposed to the user via “reuse factor”

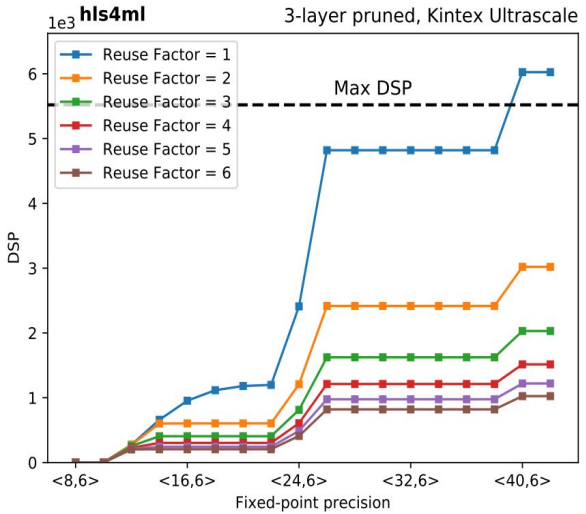
- A handle to control resource usage and latency
- Can be specified per-layer

Reuse = 1: Fully unroll everything

- Fastest, most resource intensive

Reuse > 1: reuse one DSP for several operations

- Increases latency, but uses less resources



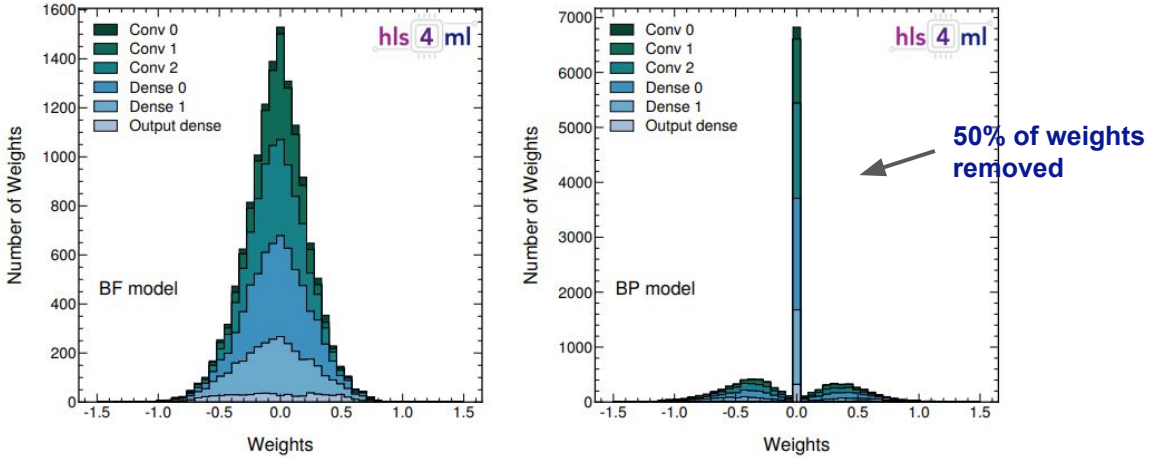
Pruning / Compression

Pruned (zero-valued) weight removes multiplication

- Key optimization, possible only on FPGAs

Applied **during training**, where sparsity is gradually introduced to remove the smallest magnitude weights

- E.g., with *Tensorflow model optimization toolkit* (TFMOT)



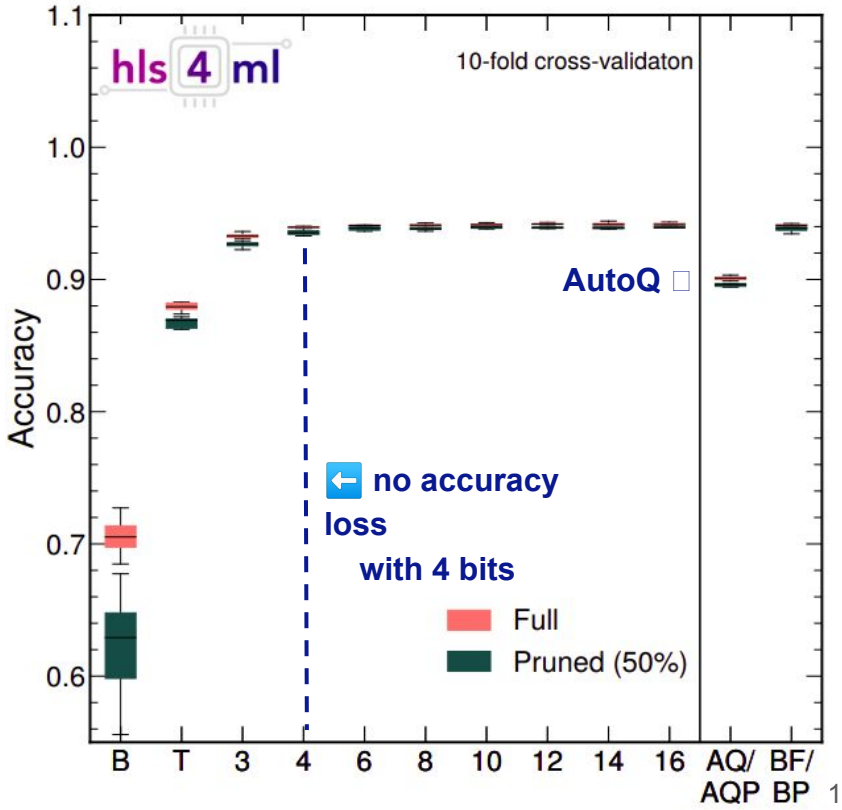
Quantization

New

QKeras: Library for training quantization-aware Keras models - arxiv:2006.10159

- Simple drop-in replacement of Keras layers
- Heterogenous quantization (per layer)
  - Automatic quantization through Bayesian optimization (AutoQ)
- Numerous quantizers available
- Fully supported in **hls4ml**
  - Special case for binary/ternary

  arxiv:2003.06308

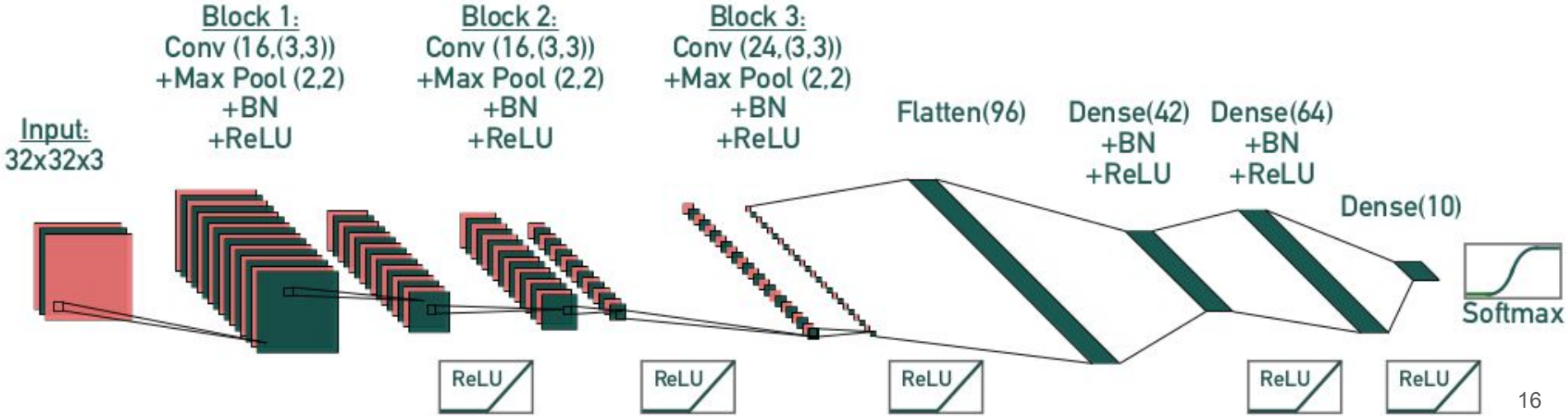


Evaluation of model optimization techniques

Street-view house numbers dataset (SVHN) - “A tougher MNIST”

- 32x32x3 images

Model architecture (obtained through Bayesian optimization with *Keras Tuner* and *AutoQ*):



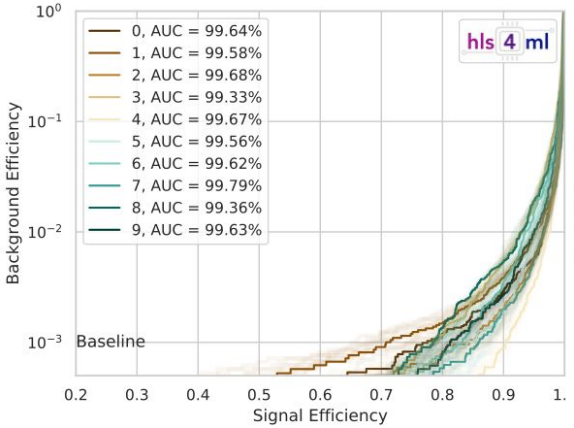
Model performance

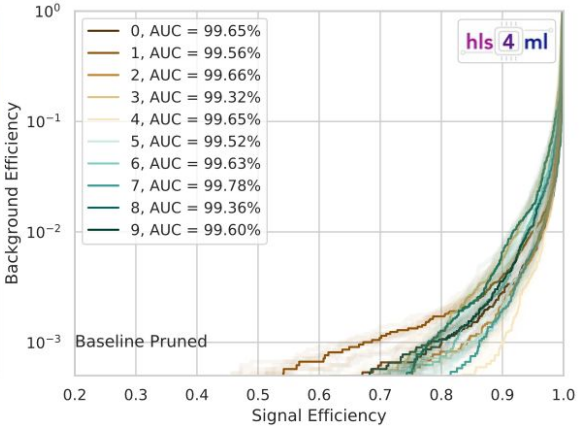
Baseline models

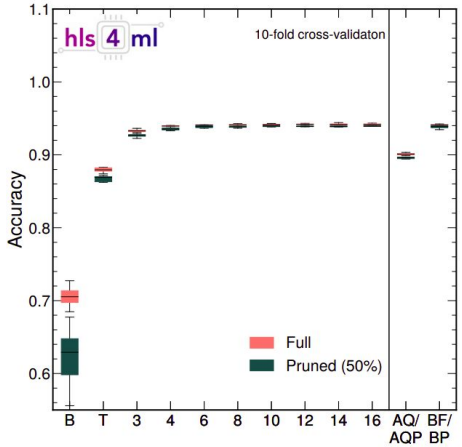
- Full 32-bit precision (BF)
- Full 32-bit precision, pruned (BP)
  - 50% sparsity
  - Polynomial decay

QKeras models

- Quantized (Q)
  - Binary (1-bit)
  - Ternary (2-bit)
  - Quantized to 3-16 bits
- Pruned (QP)
  - 50% sparsity







Model performance on an FPGA

Using *Xilinx Virtex UltraScale+ VU9P* series FPGA

- Target device for CMS L1 trigger upgrade

200MHz clock

Table 3: Accuracy, resource consumption and latency for the Baseline Full (BF) and Baseline Pruned (BP) models quantized to a bit width of 14, the QKERAS (Q) and QKERAS Pruned (QP) models quantized to a bit width of 7 and the heterogeneously quantized AutoQ (AQ) and AutoQ Pruned (AQP) models. The numbers in parentheses correspond to the total amount of resources used.

Model	Accuracy	DSP [%]	LUT [%]	FF [%]	BRAM [%]	Latency [cc]	II [cc]
BF 14-bit	0.87	93.23 (6377)	19.36 (228823)	3.40 (80278)	3.08 (66.5)	1035	1030
BP 14-bit	0.92	48.85 (3341)	12.27 (145089)	2.77 (65482)	3.08 (66.5)	1035	1030
Q 7-bit	0.93	2.56 (175)	12.77 (150981)	1.51 (35628)	3.10 (67.0)	1034	1029
QP 7-bit	0.93	2.54 (174)	9.40 (111152)	1.38 (32554)	3.10 (67.0)	1035	1030
AQ	0.85	1.05 (72)	4.06 (48027)	0.64 (15242)	1.5 (32.5)	1059	1029
AQP	0.88	1.02 (70)	3.28 (38795)	0.63 (14802)	1.4 (30.5)	1059	1029

More details in our paper!

hls4ml for triggering @ 40 MHz

New trigger algorithms with **hls4ml**

- Replace standard cut-based algorithms
  - Better signal efficiency with NN!

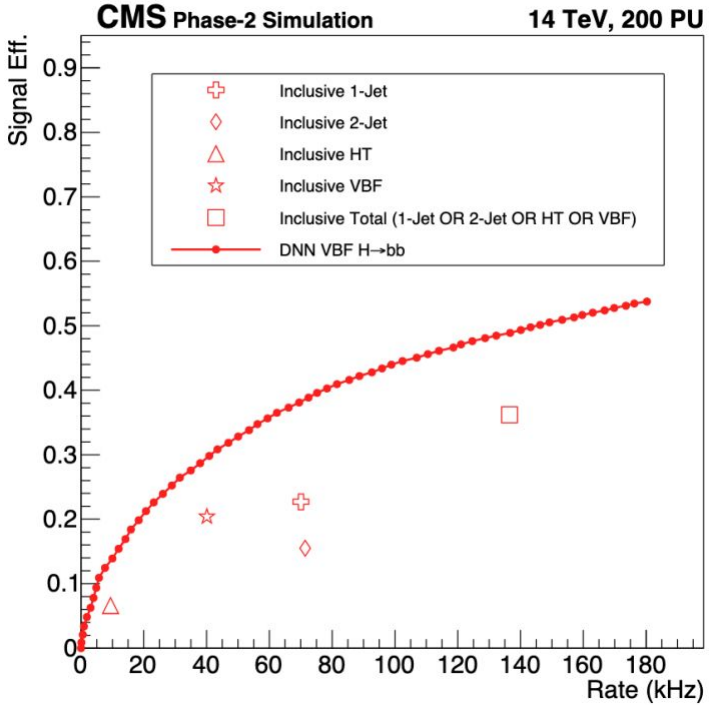
24 INPUT FEATURES:
pT, η, and Φ of the 3 leading jets; the total event H_T; the di-jet pT, invariant mass, ΔR. E_T^{miss}

⬇️

3 HIDDEN LAYERS (72x72x72)

⬇️

1 OUTPUT: final discriminant score



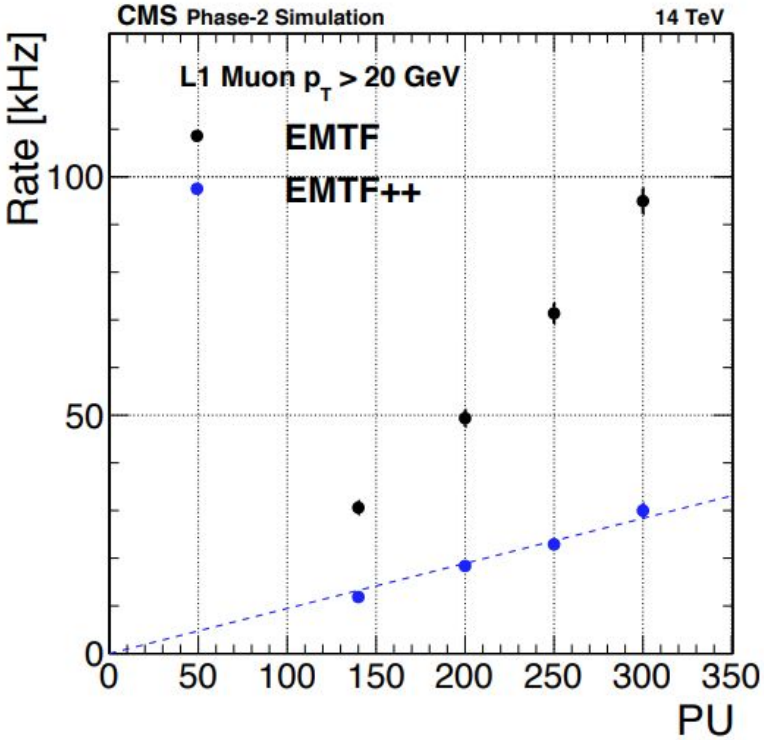
➔ [CMS Phase-2 L1 trigger upgrade TDR]

hls4ml for triggering @ 40 MHz

New trigger algorithms with **hls4ml**

- Replace standard cut-based algorithms
  - Better signal efficiency with NN!
- Improve physics objects reconstruction (muons, taus, jets)
  - 2.5x rate reduction!

36 INPUT FEATURES:
ϕ,θ of track segments in muon stations
track segment quality
track segment curvature
⬇️
3 HIDDEN LAYERS (30x25x20)
⬇️
1 OUTPUT: muon pT

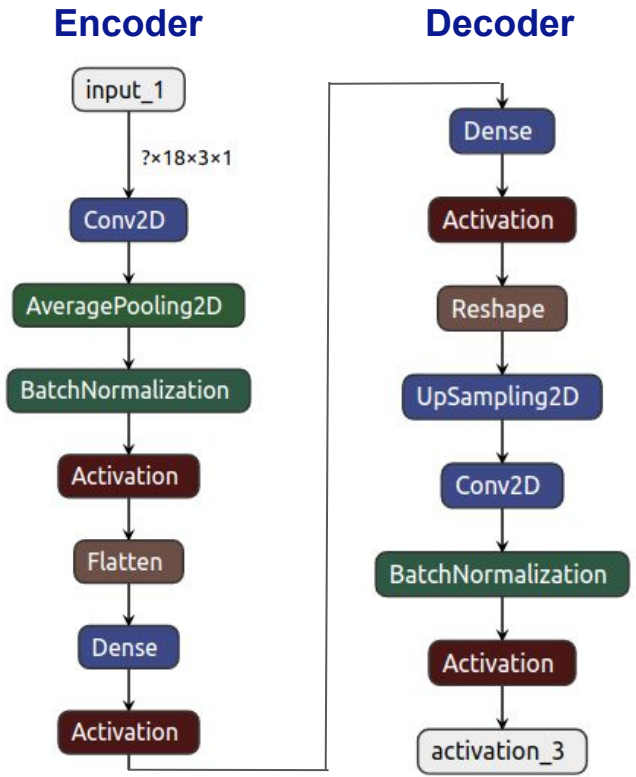


CMS Phase-2 L1 trigger upgrade TDR

hls4ml for triggering @ 40 MHz

New trigger algorithms with **hls4ml**

- Replace standard cut-based algorithms
  - Better signal efficiency with NN!
- Improve physics objects reconstruction (muons, taus, jets)
  - 2.5x rate reduction!
- Develop new strategies like anomaly detection with autoencoders for signal-agnostic triggering



CMS Phase-2 L1 trigger upgrade TDR

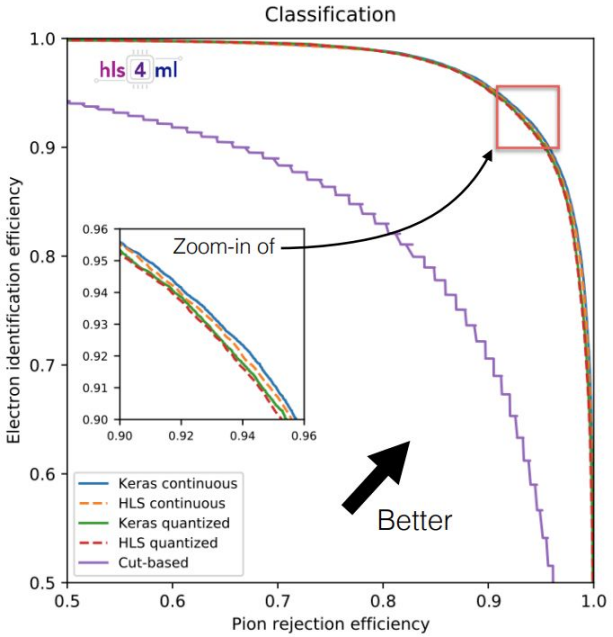
K.Govorkova @ FastML workshop '20

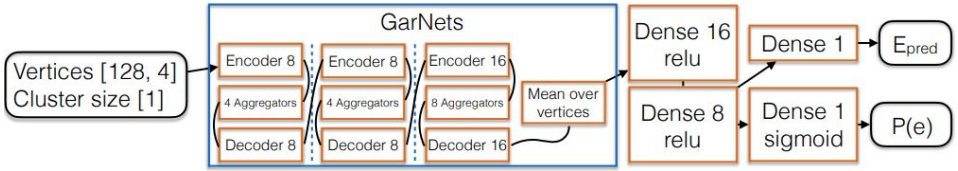
hls4ml for triggering @ 40 MHz

New

New trigger algorithms with **hls4ml**

- Replace standard cut-based algorithms
  - Better signal efficiency with NN!
- Improve physics objects reconstruction (muons, taus, jets)
  - 2.5x rate reduction!
- Develop new strategies like anomaly detection with autoencoders for signal-agnostic triggering
- Custom NNs @ L1
  - Calorimeter clusters classification - arxiv:2008:03601
  - Charged particles track reconstruction - arxiv:2012:01563





CMS Phase-2 L1 trigger upgrade TDR

Summary

hls4ml - software package for translation of trained neural networks into synthesizable FPGA firmware

- Tunable resource usage latency/throughput
- Fast inference times, O(1μs) latency

Currently being extended to multiple hardware architectures and new neural networks

Many applications in science

More information:

- Website: https://hls-fpga-machine-learning.github.io/hls4ml/
- Code: https://github.com/hls-fpga-machine-learning/hls4ml
- Tutorial: http://cern.ch/ssummers/hls4ml-tutorial