FERMILAB-SLIDES-19-706-SCD

hls4ml: deploying deep learning on FPGAs for L1 trigger and Data Acquisition

Javier Duarte, Sergo Jindariani, Ben Kreis, Ryan Rivera, Nhan Tran (Fermilab)
Jennifer Ngadiuba, Maurizio Pierini, Sioni Summers, **Vladimir Loncar** (CERN)
Edward Kreinar (Hawkeye 360)
Phil Harris, Song Han, Dylan Rankin (MIT)
Zhenbin Wu (University of Illinois at Chicago)
Giuseppe di Guglielmo (Columbia University)

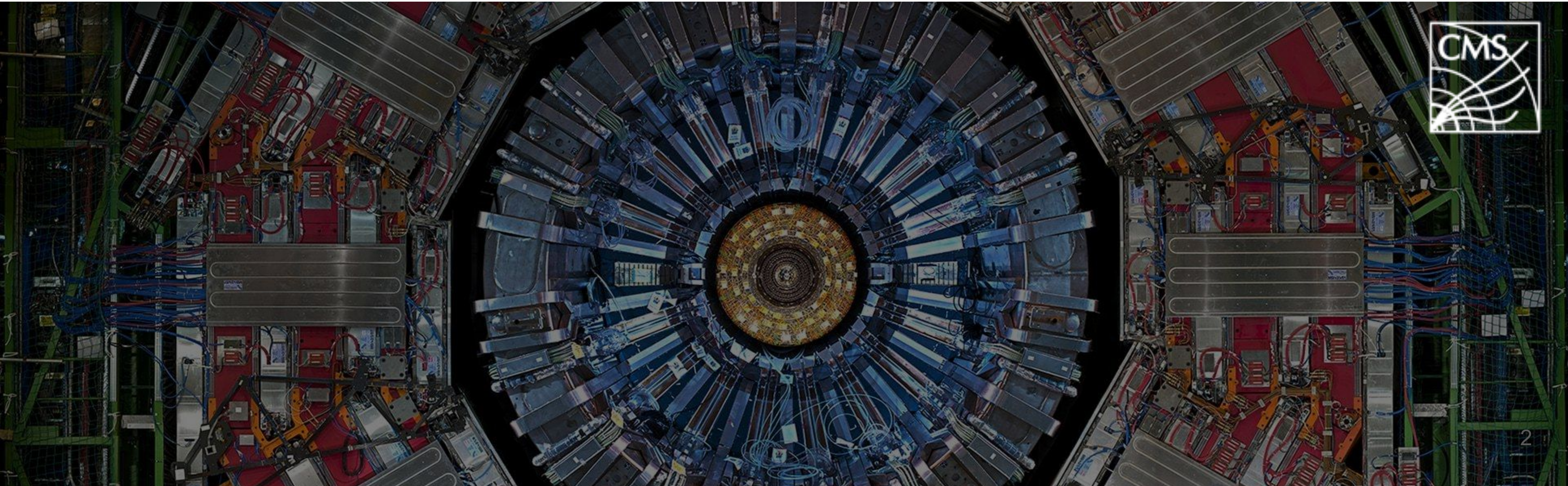
This manuscript has been authored by Fermi Research Alliance, LLC under Contract No. DE-AC02-07CH11359 with the U.S. Department of Energy, Office of Science, Office of High Energy Physics.

Challenges in LHC

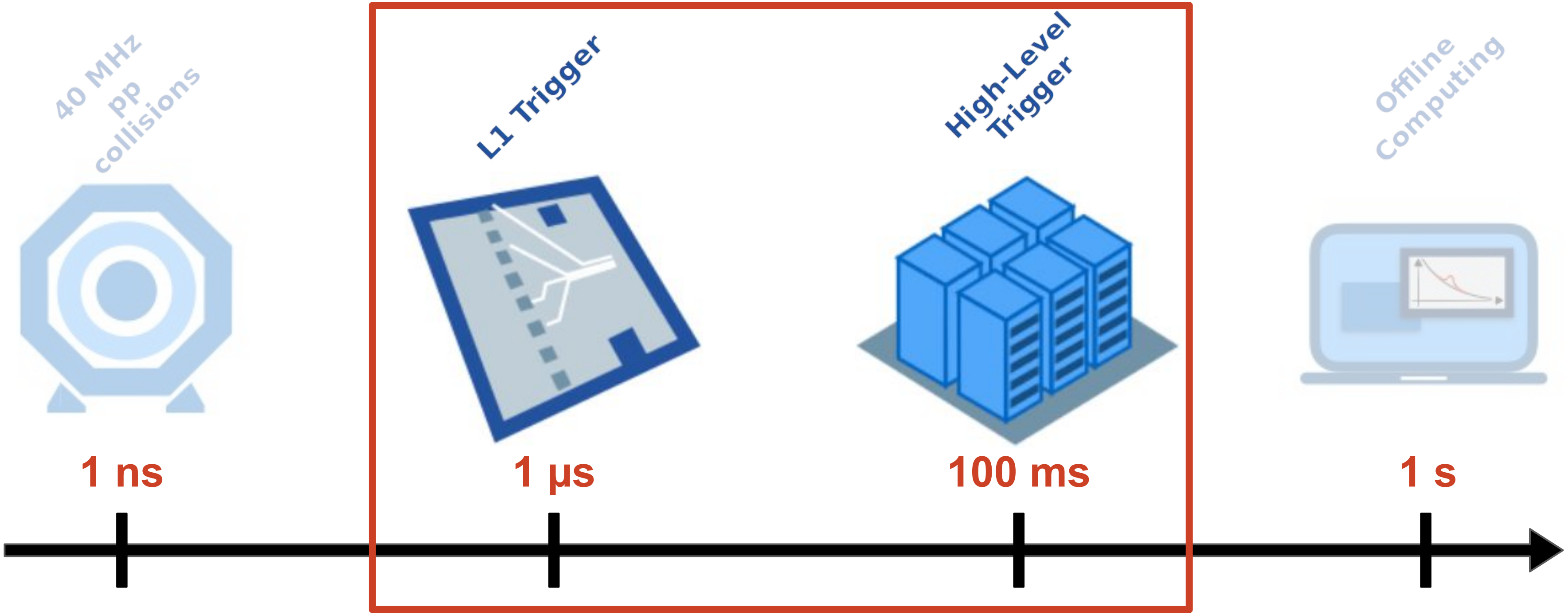
At the LHC proton beams collide at a frequency of 40 MHz

Extreme data rates of O(100 TB/s)

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



The LHC big data problem



Deploy ML algorithms very early

Challenge: strict latency constraints!

Field-Programmable Gate Array

Reprogrammable integrated circuits

Configurable logic blocks and embedded components

- Flip-Flops (registers)
- LUTs (logic)
- DSPs (arithmetic)
- Block RAMs (memory)

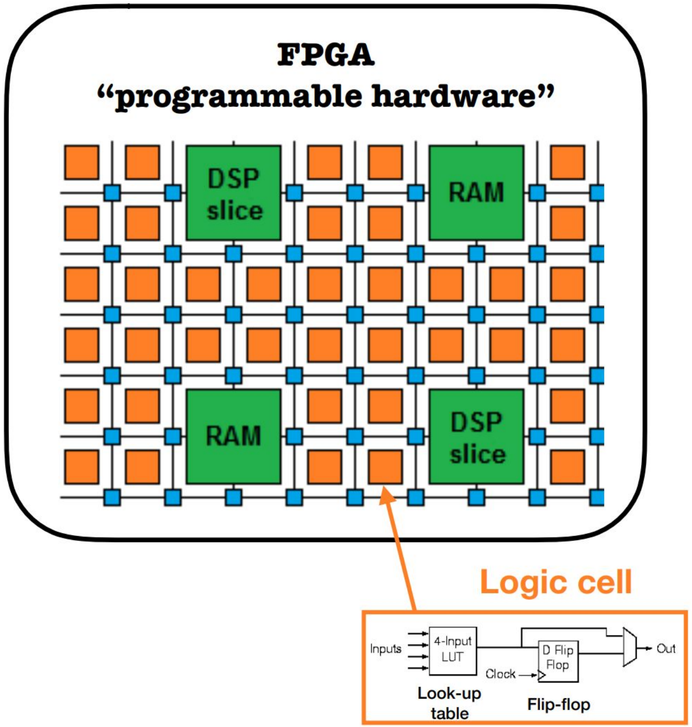
Massively parallel

Low power

Traditionally programmed with VHDL and Verilog

High-Level Synthesis (HLS) tools

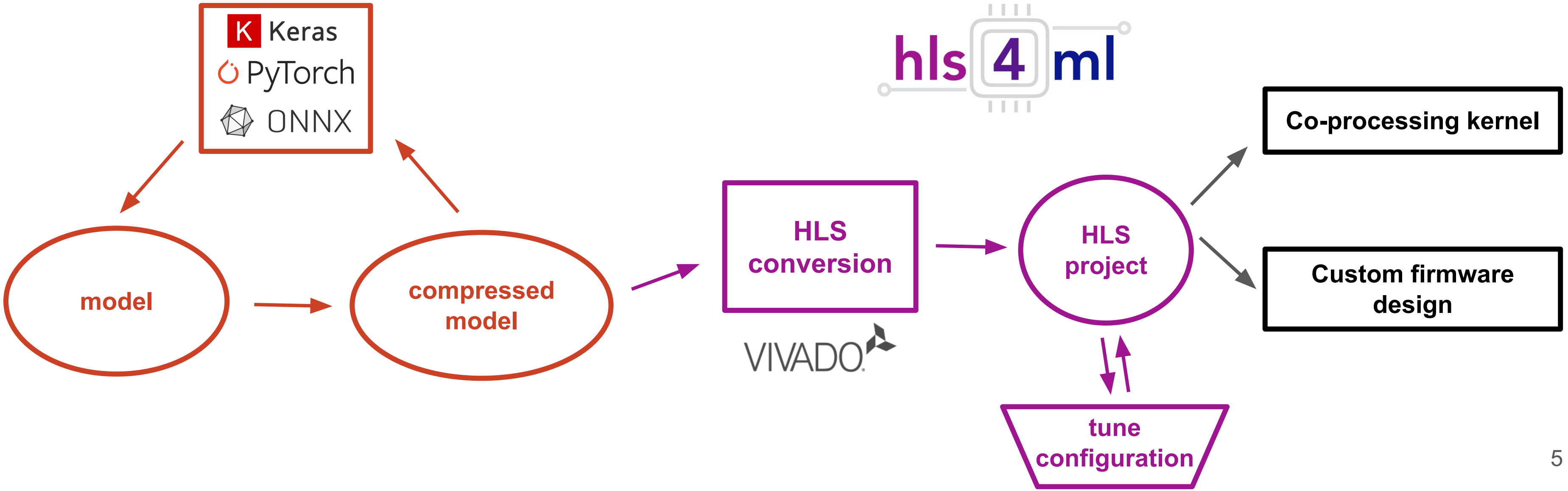
- Use C, C++, System C



high level synthesis for machine learning

User-friendly tool to automatically build and optimize DL models for FPGAs:

- Reads as input models trained with standard DL libraries
- Uses Xilinx HLS software
- Comes with implementation of common ingredients (layers, activation functions, binary NN …)



hls4ml : features

The main idea: Store the full architecture and weights on chip

- Much faster access times
- For longer latency applications, weights storage in on-chip block memory is possible
- No loading weights from external source (e.g. DDR, PCIe)

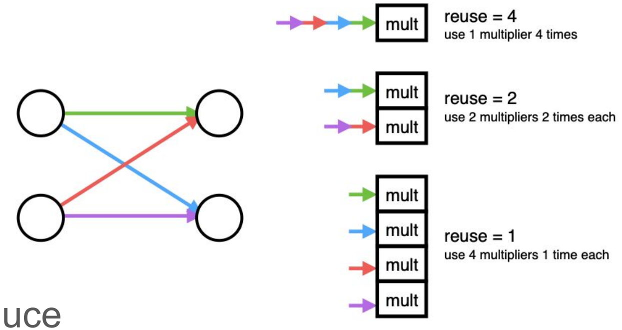
Limitations:

- Constraints on model size
- Not reconfigurable without reprogramming device

Solution: User controllable trade-off between resource usage and latency/throughput

- Tuned via “reuse factor”

hls4ml : exploiting FPGA hardware



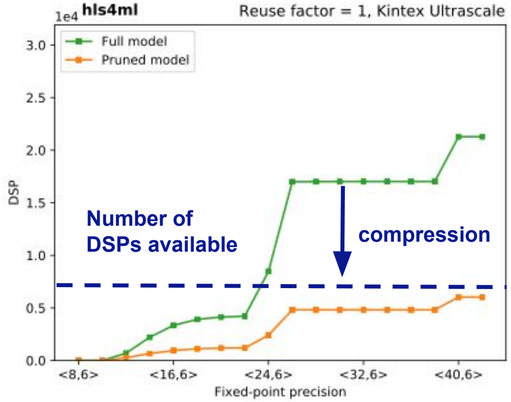
Parallelization: Use reuse factor to tune the inference latency versus utilization of FPGA resources

- Can now be specified per-layer **NEW**

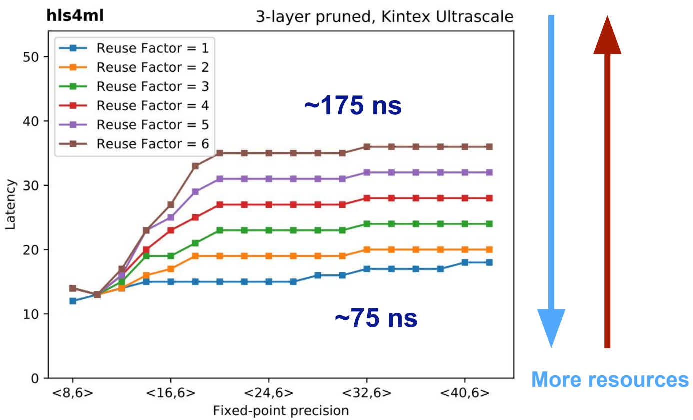
Quantization: Reduce precision of the calculations

Compression: Drop unnecessary weights (zero or close to zero) to reduce the number of DSPs used

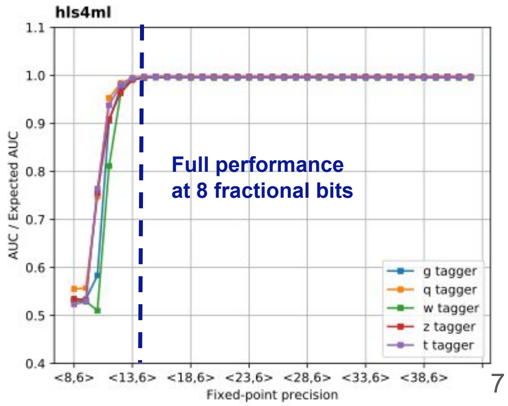
70% compression ~ 70% fewer DSPs



Parallelization



Quantization

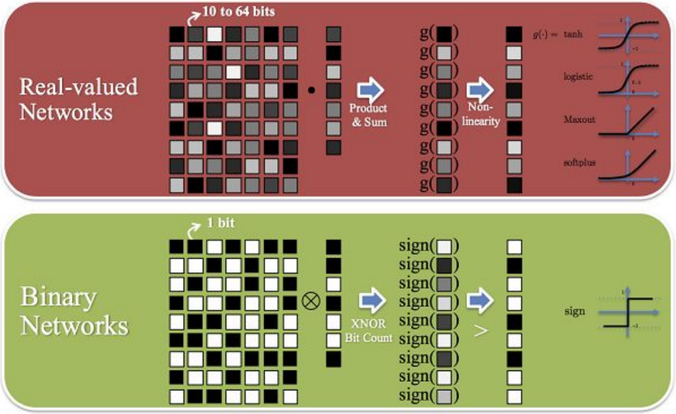


NEW

hls4ml : compression by binarization/ternarization

Replace floating/fixed-point with 1/2-bit arithmetics

- Binary: 1-bit (arXiv:1602.02830)
- Ternary: 2-bits (arXiv:1605.04711)

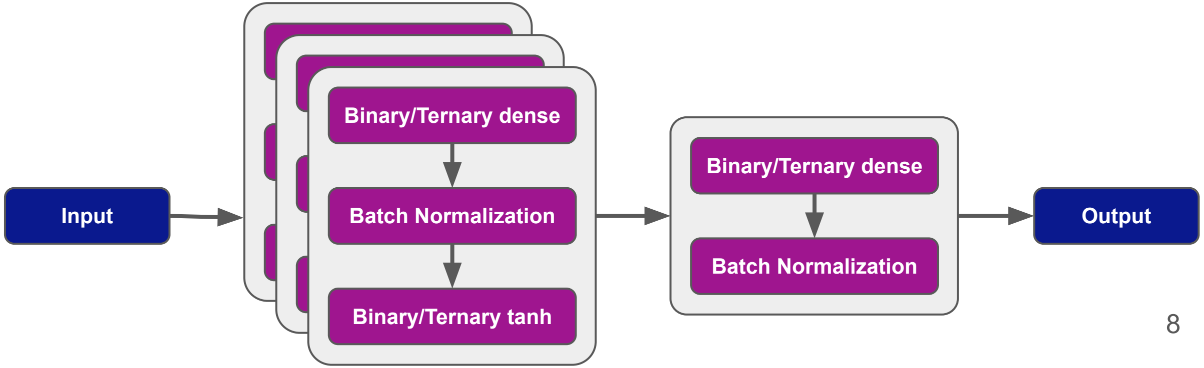


Multiplications (`d * w`) as bit-flip operations:

- Binary: `res = w == 0 ? -d : d;`
- Ternary: `res = w == 0 ? 0 : w == -1 ? -d : d;`

Binary/ternary architecture:

- Binary/Ternary Dense
- Batch Normalization
- Binary/Ternary tanh activation

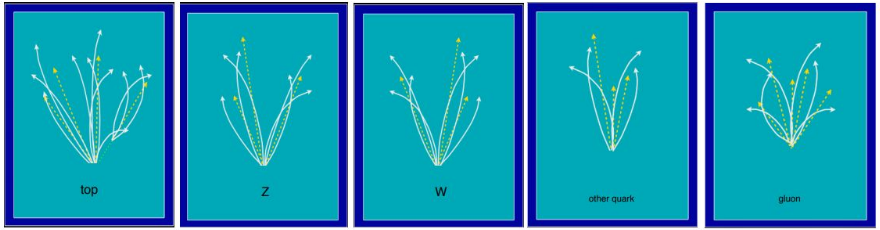


hls4ml : Jet tagging benchmark model

Multi-classification task:

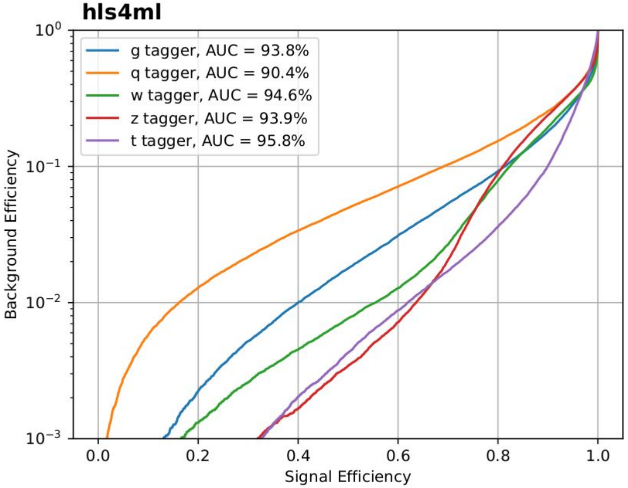
- Discrimination between highly energetic (boosted) **q**, **g**, **W**, **Z**, **t** initiated jets
- 16 inputs, 5 outputs

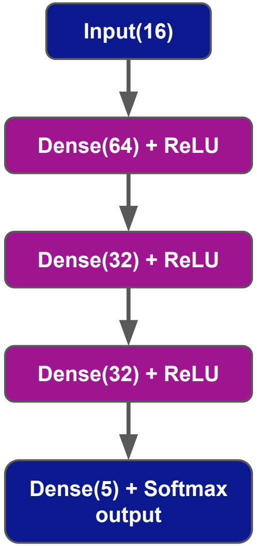
Average accuracy ~ 0.75



t→bW→bqq	**Z→qq**	**W→qq**	**q/g background**
3-prong jet	2-prong jet	2-prong jet	no substructure and/or mass ~ 0

Reconstructed as one massive jet with substructure





hls4ml : Jet tagging benchmark model

Run hyper-parameter bayesian optimization:

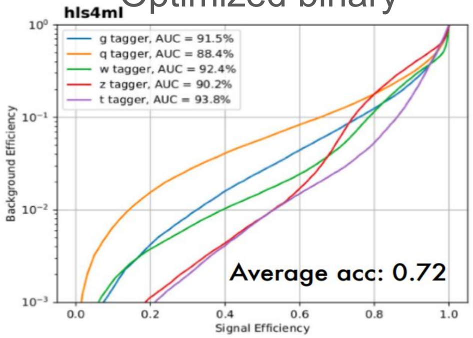
- Number of neurons/layers, batch size, learning rate

Recover performance with larger models

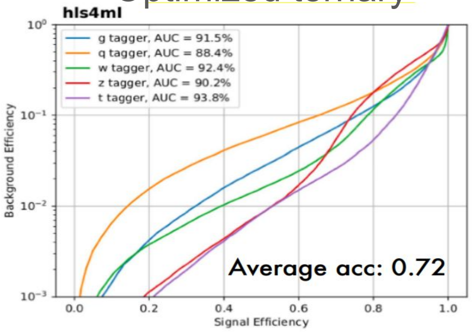
- Binary: **16x448x224x224x5** (7x more neurons)
- Ternary: **16x128x64x64x64x5** (2x more neurons + one more layer)

Model	Accuracy	Latency	DSP	BRAM	FF	LUT
Base model	0.75	0.06 μs	60%	0%	1%	7%
Optimized Binary	0.72	0.21 μs	0%	0%	7%	15%
Optimized Ternary	0.72	0.11 μs	0%	0%	1%	6%

Optimized binary



Optimized ternary



hls4ml : MNIST benchmark

Dense networks trained with the MNIST dataset

- 784 inputs (28x28 grayscale image), 10 outputs (digits)

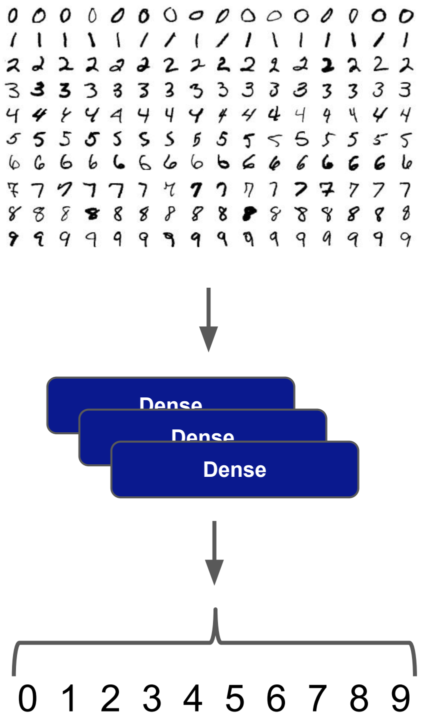
Base model:

- 3 hidden layers with 128 neurons and ReLU activation

Binary/Ternary model:

- 3 hidden layers with batch normalization and binary/ternary tanh

Xilinx VU9P FPGA at 200 MHz, reuse factor 128



Model	Accuracy	Latency	DSP	BRAM	FF	LUT
Dense model	0.97	2.6 μs	21%	45%	12%	33%
Binary dense model	0.93	2.6 μs	0%	33%	7%	39%
Ternary dense model	0.95	2.6 μs	0%	33%	7%	40%

hls4ml : current status

Supported architectures:

- **DNN**
  - Support for very large layers **NEW**
  - Zero-suppressed weights
- **Binary and Ternary DNN** **NEW**
  - 1- or 2-bit precision with limited loss of performance
  - Computation without using DSPs, only LUTs
- **Convolutional NNs**
  - 1D and 2D with pooling
  - Currently limited to very small layers, working on support for larger layers **WIP**

Other:

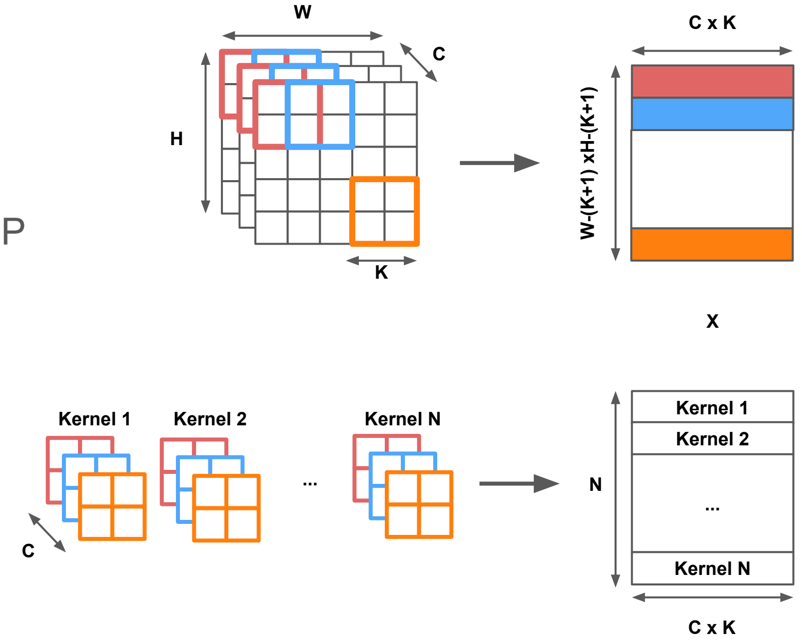
- Batch normalization
- Merge layers (concatenation, addition, subtraction etc)
- Numerous activation functions

hls4ml : ongoing work

Convolutional layers

Support for "large" convolutional layers SOON

- Express convolution as matrix multiplication
- im2col algorithm
- Reuse "large" matrix multiplication algorithm from MLP
- Quantized (binary and ternary) weights



Credit: Jennifer Ngadiuba, Sioni Paris Summers

hls4ml : ongoing work

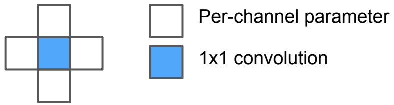
Convolutional layers

Depthwise separable convolution (arXiv:1610.02357)

- First step: depthwise convolution
- Second step: pointwise convolution
- For 3x3 kernels this can yield 8-9 times less multiplications

LeanConvNet (arXiv:1904.06952)

- Depth-wise (block diagonal) operator operating on each channel separately and 1×1 convolution
- 5-point convolution kernel



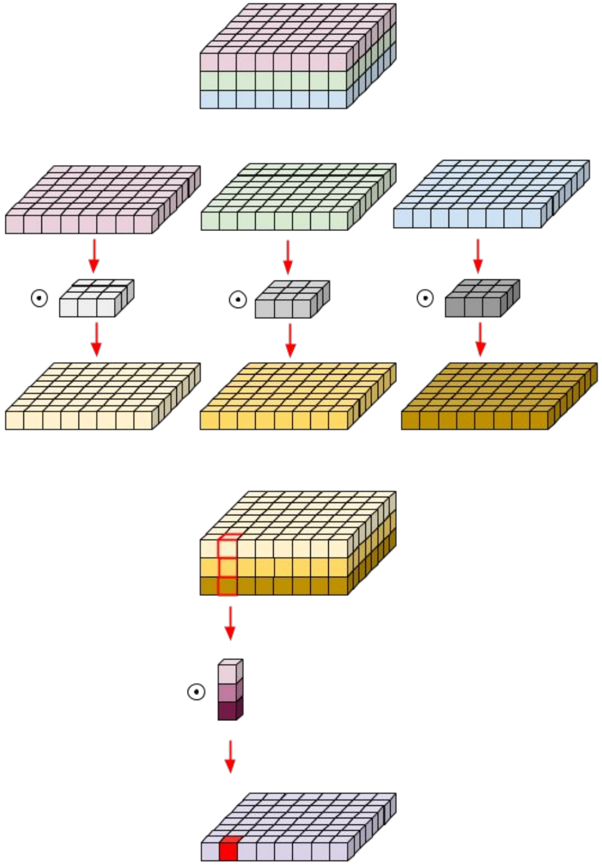
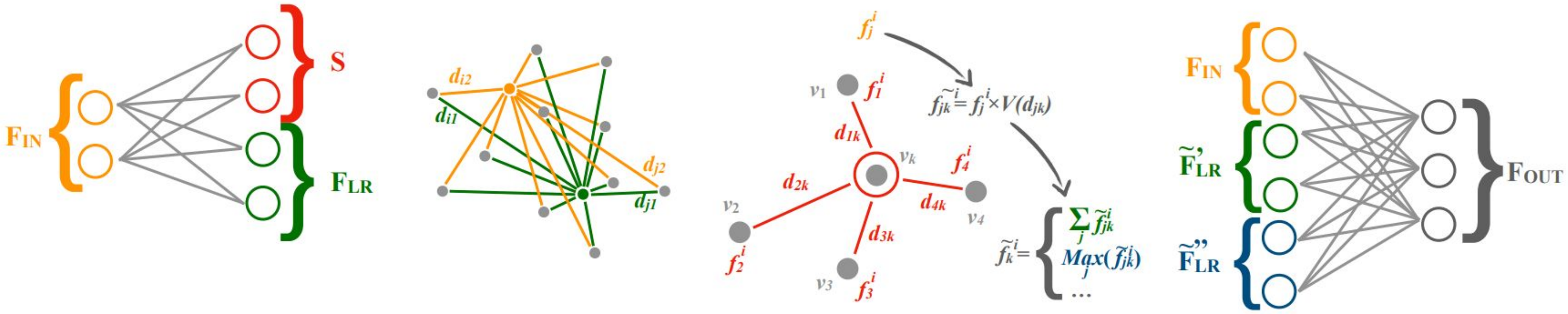


Image source: Atul Pandey

hls4ml : ongoing work

Graph networks (GarNet) H1 2020

- Distance-weighted GNN capable of learning irregular patterns of sparse data (arXiv:1902.07987)
- Suitable for irregular particle-detector geometries
- Early stage of HLS implementation



Credit: Abhijay Gupta, Yutaro Iiyama, Jan Kieseler and Maurizio Pierini

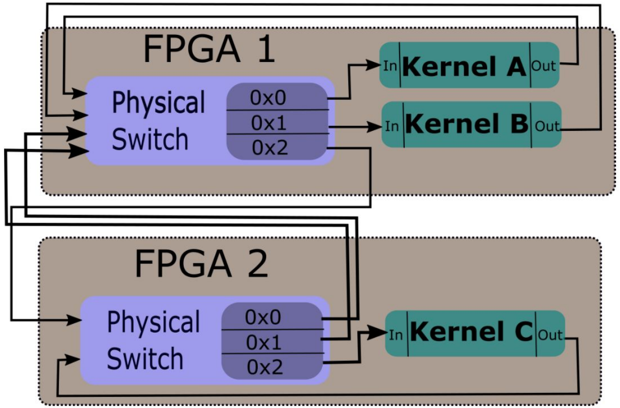
hls4ml : future directions

Multi-FPGA inference H1 2020

- Main idea: place layers onto multiple FPGAs and pipeline the execution

Leverage **Galapagos** framework (https://github.com/tarafdar/galapagos)

- *"...a framework for creating network FPGA clusters in a heterogeneous cloud data center."*
- Given a description of how a group of FPGA kernels are to be connected, creates a ready-to-use network device
- Possible to use MPI programming model



Credit: Naif Tarafdar, Phil Harris

hls4ml : other future developments

Recurrent Neural Networks (RNNs) **Q4 2019**

Boosted decision trees **Q4 2019**

Autoencoders **H2 2020**

HLS implementations beyond Xilinx/Vivado **H1 2020**

- Quartus HLS Compiler for Intel/Altera FPGAs
- Mentor Catapult HLS

Inference engine for CPUs based on hls4ml **H1 2020**

- Targeting integration with CMSSW

Many more...

hls4ml in production in HEP

CMS designing DL-based triggers for Run III, using **hls4ml** for deployment

- Reduce muon rate by factor 4 (link)
- Run inference in 160ns on currently used boards (Virtex 7)

NN firmware resource usage

- Converted an earlier version of NN using HLS4ML
  - 80 input nodes, 3 hidden layers with 64/32/16 nodes, and 1 output node.
  - Pruning is applied to remove 50% of the unimportant synapses
  - Use <18,8>-precision for inputs & output. 8 integer bits, 18 total bits
- Xilinx Virtex-7 target FPGA (as in the MTF7 board)
  - Clock of 250 MHz
- Currently re-optimizing the NN structure for less resource usage.
  - A smaller version with 30/25/20 nodes uses only 50% DSP resource.

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	-	-	-
Expression	-	-	0	6
FIFO	-	-	-	-
Instance	56	2822	315515	112745
Memory	-	-	-	-
Multiplexer	-	-	-	36
Register	-	-	4689	-
Total	56	2822	320204	112787
Available	2940	3600	866400	433200
Utilization (%)	1	78	36	26

Performance Estimates

Timing (ns)

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	4.00	3.49	0.50

Latency (clock cycles)

Summary

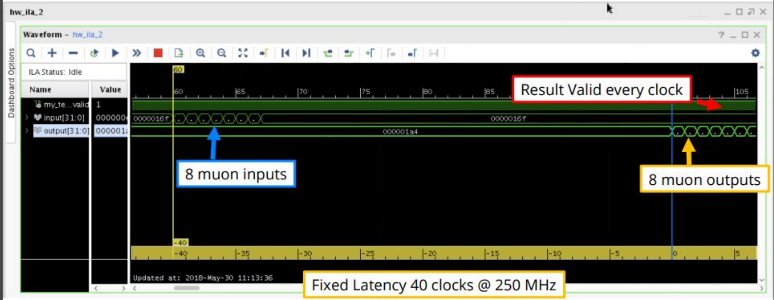
Latency		Interval		
min	max	min	max	Type
40	40	1	1	function

~160ns fixed latency

CPAD Workshop 2018 17 Dec 9, 2018

NN in MTF7 hardware

- Reading inputs corresponding to 8 simulated muons and calculating their p_T.
- HLS IP core target frequency 250 MHz (which agrees with design frequency)



Expect latency of ~40 + N_{muons} clocks

99% percentile of tracks to fit in both endcaps is 5. Per processor 5/12 = 0.4

CPAD Workshop 2018 19 Dec 9, 2018

Conclusions

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

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

More information:

- Website: https://hls-fpga-machine-learning.github.io/hls4ml/
- Paper: https://arxiv.org/abs/1804.06913
- Code: https://github.com/hls-fpga-machine-learning/hls4ml

Bonus

hls4ml : mini tutorial

Install:

```
pip install hls4ml
```

SOON

(for now: `git clone … && cd hls4ml && pip install .`)

Translate to HLS:

```
hls4ml convert -c my_model.yml
```

Run synthesys etc.:

```
hls4ml build -p my_project_dir -a
```

Get help:

```
hls4ml <command> -h
```

...or visit: https://fastmachinelearning.org/hls4ml/

...or contact us at hls4ml.help@gmail.com

Keras

ONNX

```yaml
OnnxModel: models/my_model.onnx
InputData: data/my_input_features.dat
OutputPredictions: data/my_predictions.dat
OutputDir: my_project_dir
ProjectName: myproject
XilinxPart: xcku115-flvb2104-2-i
ClockPeriod: 5

IOType: io_parallel
HLSConfig:
  Model:
    Precision: ap_fixed<16,6>
    ReuseFactor: 2
    Strategy: Resource
```

Degree of parallelism

Support for large models

Default precision (weights, biases...)

hls4ml : Advanced configuration example

```yaml
KerasJson: models/my_model.json
KerasH5: models/my_model_weights.h5
OutputDir: my_project_dir
ProjectName: myproject
XilinxPart: xcku115-flvb2104-2-i
ClockPeriod: 5

IOType: io_parallel
HLSConfig:
  Model:
    Precision: ap_fixed<16,6>
    ReuseFactor: 8
    Strategy: Resource
  LayerName:
    fc1_relu:
      Precision:
        weight: ap_fixed<18,6>
        bias: ap_fixed<16,8>
        result: ap_fixed<18,8>
      ReuseFactor: 4
```

Applies to the whole model

Specific to this layer by name

```yaml
LayerType:
  Dense:
    Precision:
      default: ap_fixed<18,8>
      weight: ap_fixed<14,6>
    ReuseFactor: 2
  Activation:
    Precision: ap_fixed<12,8>
```

Applies to all other Dense layers

Applies to all Activation layers

hls4ml : ongoing work

Boosted decision trees Q4 2019

- BDTs have been popular for a long time in HEP reconstruction and analysis
- Suitable for highly parallel implementation in FPGAs
- Implementation in hls4ml optimised for low latency
- No ‘if/else’ statement in FPGAs → evaluate all options and select the right outcome
    - Compare all features against thresholds, chain together outcomes to make the ‘tree’

Test for model with 16 inputs, 5 classes, 100 trees, depth 3 on VU9P FPGA:

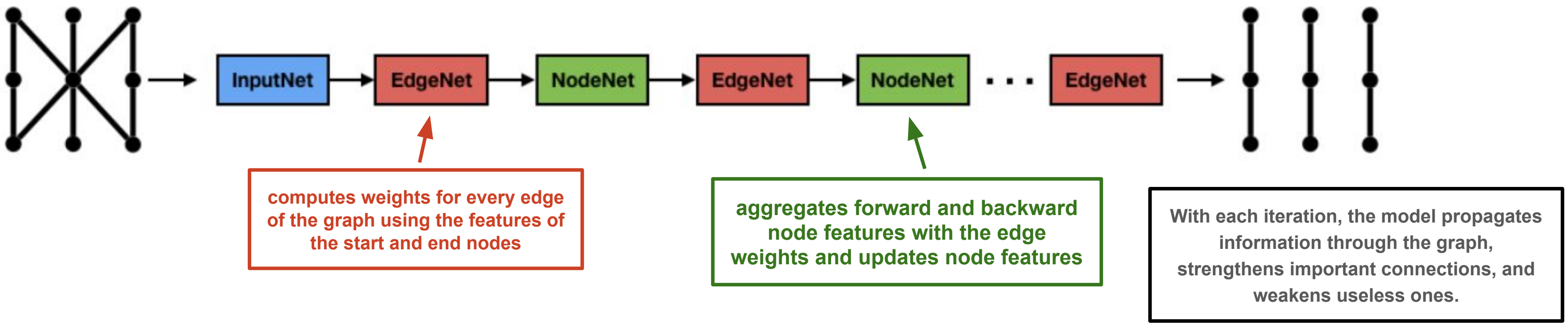
- 4% LUTs, 1% FFs (0 DSPs, 0 BRAMs)
- 25 ns latency with II=1

Credit: Sioni Paris Summers

hls4ml : ongoing work

Graph networks H1 2020

- Natural solution for reconstructing the trajectories of charged particles



Preliminary implementation:

- Implemented as an HLS project, not supported in conversion tools
- Successfully tested a small example with 4 tracks, 4 layers
- Major effort required to scale up to larger graphs

Credit: Javier Duarte and Kazi Asif Ahmed Fuad

hls4ml : ongoing work

Recurrent neural networks Q4 2019

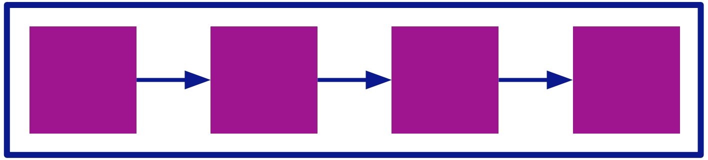
- Simple RNN, LSTM, GRU

Two implementations:

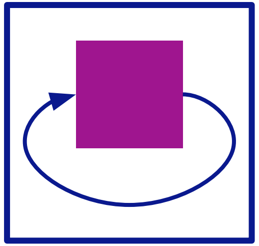
- **Fully unrolled**:
  - Latency optimized with II=1
  - Large resource usage
- **Static:** same resources used for weights and multiplications
  - N (N=latency of layer) copies can go through at the same time
  - Latency is larger and II limited to clock time for each layer

Supports small networks → scale it up using “large” matrix multiplication algorithm

Fully unrolled



Static



Credit: Phil Harris, Nhan Tran, Richa Rao

hls4ml : future directions

Training on FPGAs
H2 2020

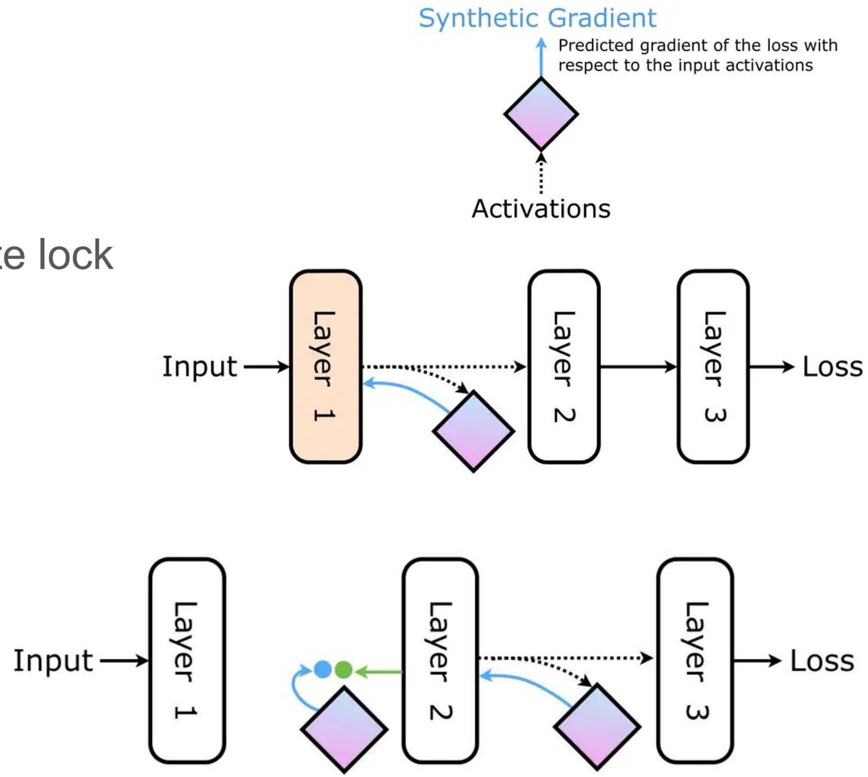
- Build on top of multi-FPGA idea

Use synthetic gradients (SG) to remove the update lock

- Individual layers to learn in isolation

Train SGs by another NN

- Each SG generator is only trained using the SGs generated from the next layer
- Only the last layer trains on the data



Images source: https://deepmind.com/blog/article/decoupled-neural-networks-using-synthetic-gradients