

hls4ml tutorial

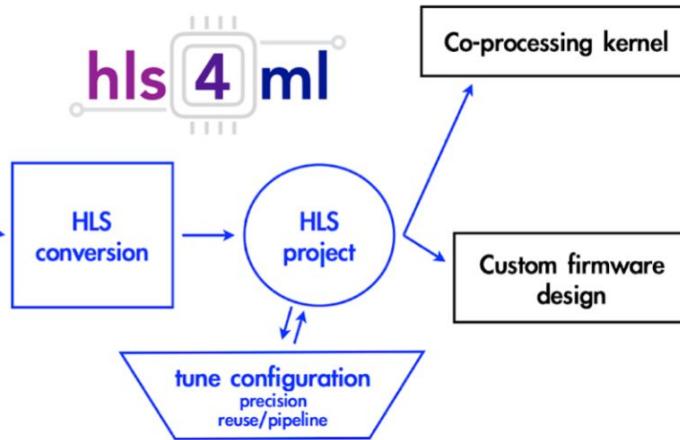
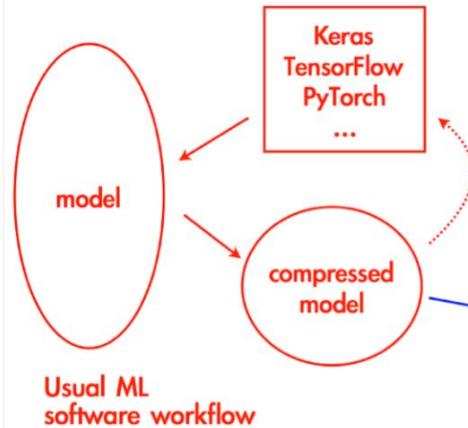
Sioni Summers (CERN)

Introduction

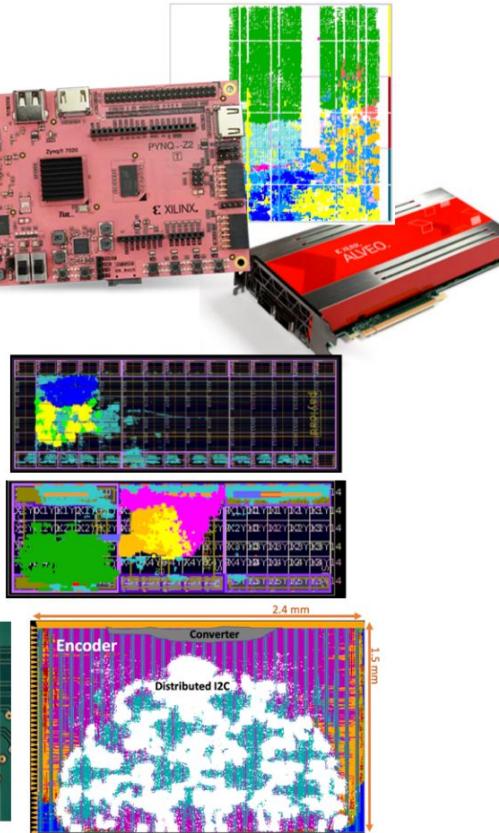
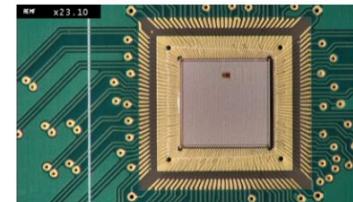
- **hls4ml** is a package for translating neural networks to FPGA firmware for inference with extremely low latency on FPGAs
 - <https://github.com/fastmachinelearning/hls4ml>
 - <https://fastmachinelearning.org/hls4ml/>
 - pip install hls4ml
- In this session you will get hands on experience with the **hls4ml** package
- We'll learn how to:
 - Translate models into synthesizable FPGA code
 - Explore the different handles provided by the tool to optimize the inference
 - Latency, throughput, resource usage
 - Make our inference more computationally efficient with pruning and quantization

high level synthesis for machine learning

fastmachinelearning.org/hls4ml/



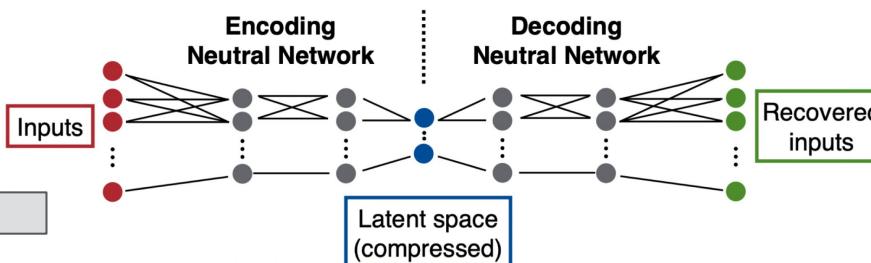
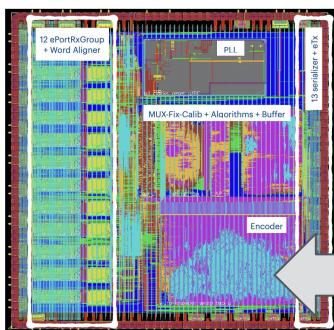
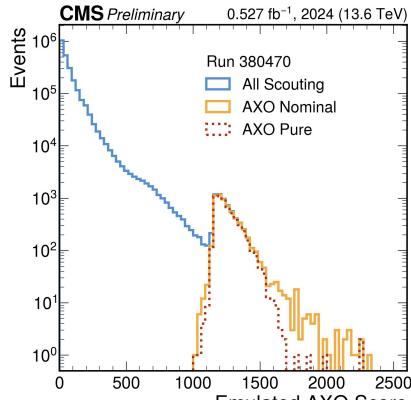
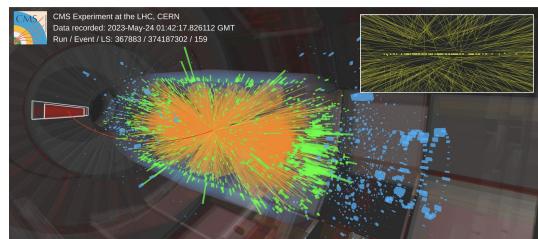
Mentor
Catapult A Siemens Business
Coming soon



Some hls4ml Applications

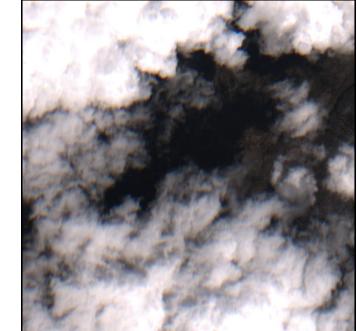
Selecting anomalous LHC collisions at the CMS experiment

- Variational AutoEncoder
- 50 ns



Filtering images onboard Earth Observation satellites

- UNet Image Segmentation
- Optimized for power efficiency and throughput



CMS Detector frontend ASIC for data compression

- Radiation tolerant
- Low power, low latency, high throughput

What are FPGAs?

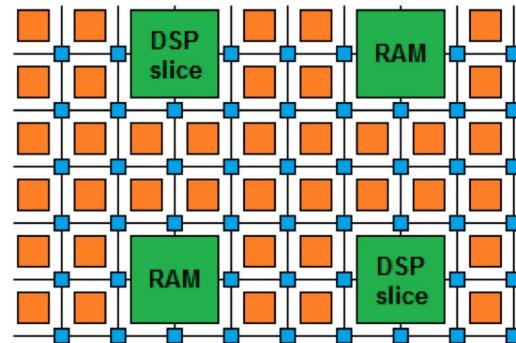
Field Programmable Gate Arrays are reprogrammable integrated circuits

Contain many different building blocks ('resources') which are connected together as you desire

Originally popular for prototyping ASICs, but now also for high performance computing and edge computing



FPGA diagram



What are FPGAs?

Field Programmable Gate Arrays are reprogrammable integrated circuits

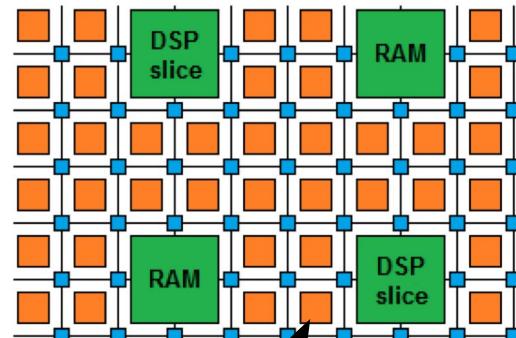
Logic cells / Look Up Tables perform arbitrary functions on small bitwidth inputs (2-6)

These can be used for boolean operations, arithmetic, small memories

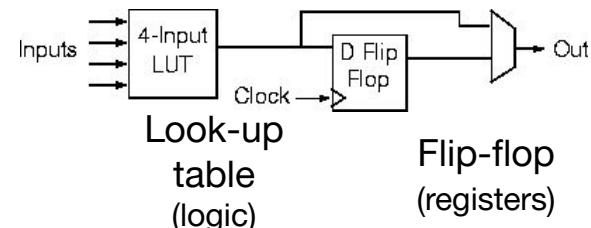
Flip-Flops register data in time with the clock pulse



FPGA diagram



Logic cell



Look-up
table
(logic)

Flip-flop
(registers)

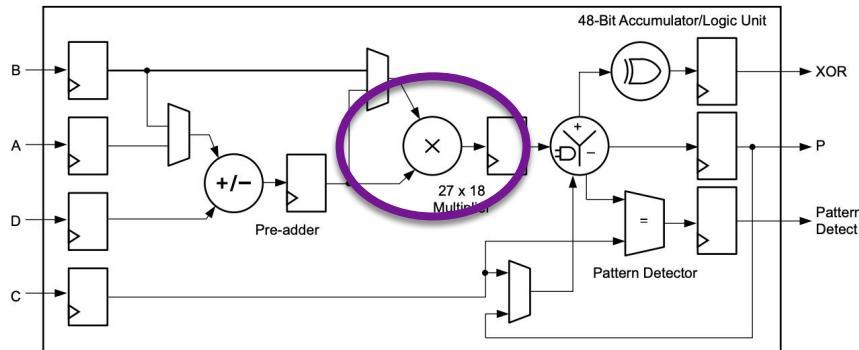
What are FPGAs?

Field Programmable Gate Arrays are reprogrammable integrated circuits

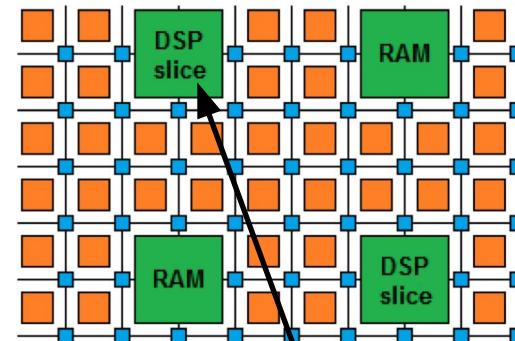
DSPs (Digital Signal Processor) are specialized units for multiplication and arithmetic

Faster and more efficient than using LUTs for these types of operations

Neural Network inference involves lots of linear algebra



FPGA diagram



DSP
(multiplication)

What are FPGAs?

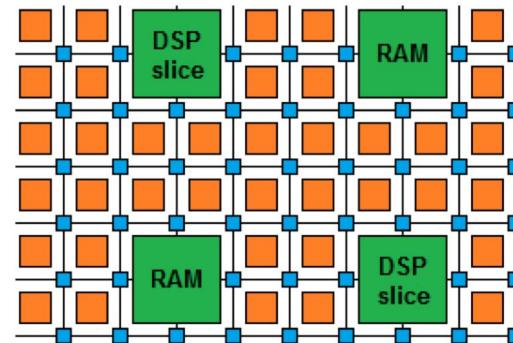
Field Programmable Gate Arrays are reprogrammable integrated circuits

BRAMs are small, fast memories - RAMs, ROMs, FIFOs (18Kb each in Xilinx)

A big FPGA has nearly 100Mb of BRAM, chained together in blocks



FPGA diagram



What are FPGAs?

In addition, there are specialised blocks for I/O, making FPGAs popular in embedded systems and HEP triggers

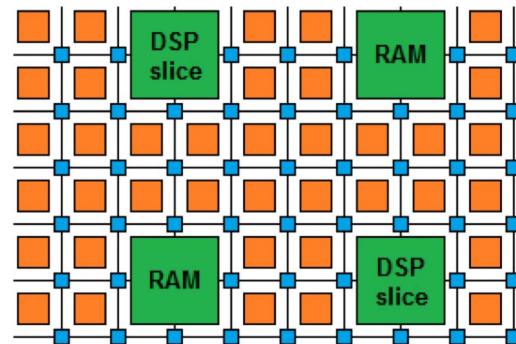
High speed transceivers with Tb/s total bandwidth
PCIe, (Multi) Gigabit Ethernet, Infiniband

AND: Support highly parallel algorithm implementations

Low power per Op (relative to CPU/GPU)



FPGA diagram



Why are FPGAs Fast?

- Fine-grained / resource parallelism
 - Use the many resources to work on different parts of the problem simultaneously
 - Allows us to achieve **low latency**
- Most problems have at least some sequential aspect, limiting how low latency we can go
 - But we can still take advantage of it with...
- Pipeline parallelism
 - Use the register pipeline to work on different data simultaneously
 - Allows us to achieve **high throughput**

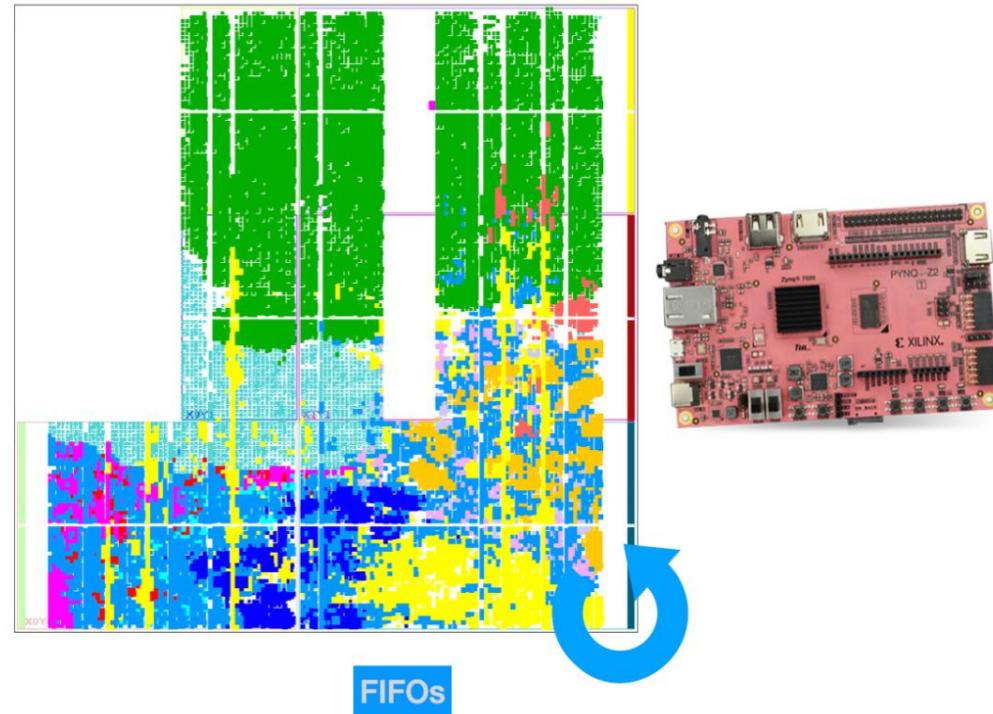


Like a production line for data...

hls4ml dataflow

- **hls4ml** uses a “dataflow architecture” → each compute step is a discrete compute unit

```
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1  
2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2  
3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3  
4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4  
5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5  
6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6  
7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7  
8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8  
9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9
```



How are FPGAs programmed?

Hardware Description Languages

HDLs are programming languages which describe electronic circuits

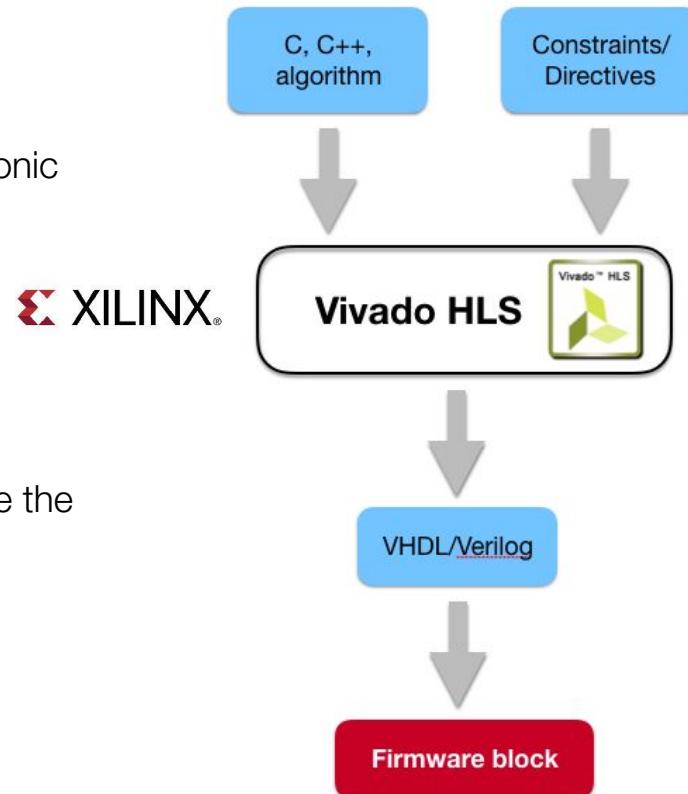
High Level Synthesis

Compile from C/C++ to VHDL

Pre-processor directives and constraints used to optimize the design

Drastic decrease in firmware development time!

Today we'll use Xilinx Vivado HLS [*]



[*] https://www.xilinx.com/support/documentation/sw_manuals/xilinx2020_1/ug902-vivado-high-level-synthesis.pdf

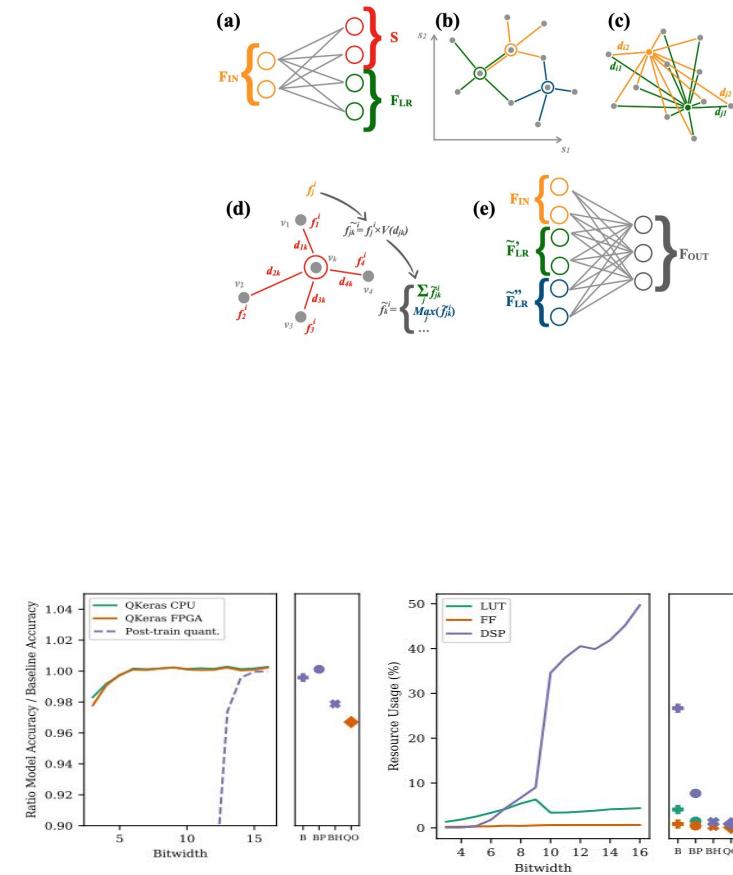
Jargon

- **LUT - Look Up Table aka 'logic'** - generic functions on small bitwidth inputs. Combine many to build the algorithm
- **FF - Flip Flops** - control the flow of data with the clock pulse. Used to build the pipeline and achieve high throughput
- **DSP - Digital Signal Processor** - performs multiplication and other arithmetic in the FPGA
- **BRAM - Block RAM** - hardened RAM resource. More efficient memories than using LUTs for more than a few elements
- **HLS** - High Level Synthesis - compiler for C, C++, SystemC into FPGA IP cores
- **HDL** - Hardware Description Language - low level language for describing circuits
- **RTL** - Register Transfer Level - the very low level description of the function and connection of logic gates
- **Latency** - time between starting processing and receiving the result
 - Measured in clock cycles or seconds
- **II - Initiation Interval** - time from accepting first input to accepting next input

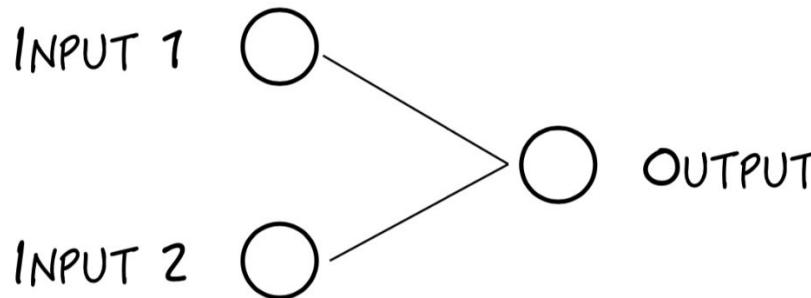
Recent Developments

hls4ml community is very active!

- Quantization aware training QKeras + support in hls4ml: [\[arXiv: 2006.10159\]](#)
- Convolutional neural networks
[Mach. Learn.: Sci. Technol. 2 045015 \(2021\)](#)
- Building accelerators (e.g. pynq, Alveo)
- Binary & Ternary neural networks:
[\[2020 Mach. Learn.: Sci. Technol.\]](#)
 - Compressed weights for low resource inference
- GarNet / GravNet: [\[arXiv: 2008.03601\]](#)
 - Distance weighted graph neural networks suitable for sparse/irregular point-cloud data



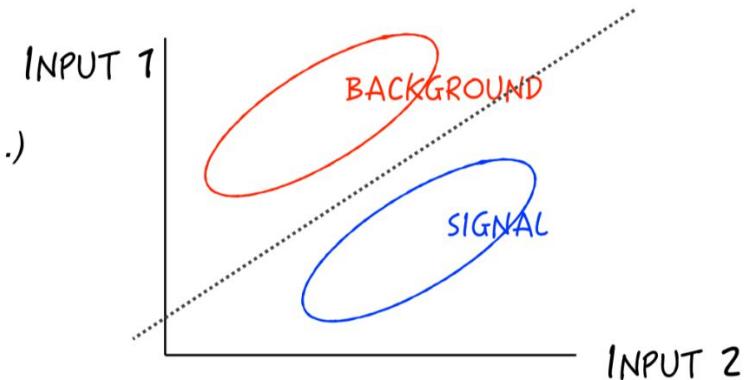
Neural network inference



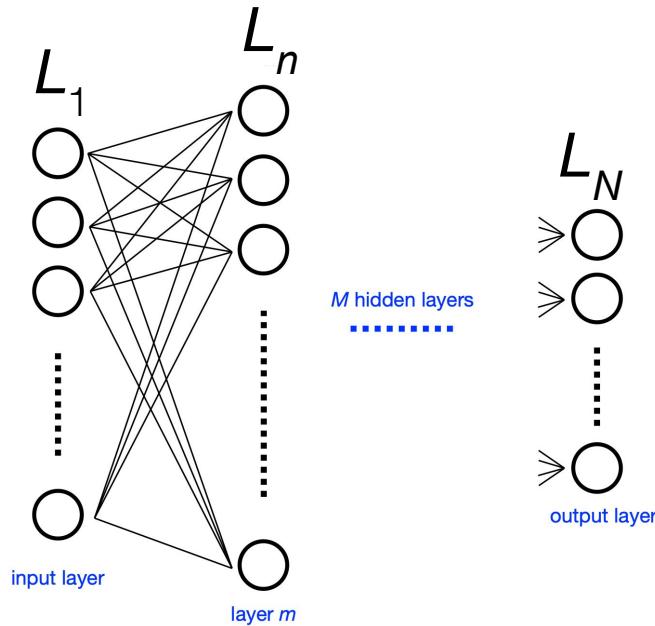
$$\vec{O}_j = \vec{l}_i \times \vec{W}_{ij} + \vec{b}_j$$

Simple 2 input example
(Fisher linear discriminant, linear support vector machine,...)

$$O_1 = l_1 \times W_{11} + l_2 \times W_{21} + b_1$$



Neural network inference



$$N_{\text{multiplications}} = \sum_{n=2}^N L_{n-1} \times L_n$$

$\mathbf{x}_n = g_n(\mathbf{W}_{n,n-1}\mathbf{x}_{n-1} + \mathbf{b}_n)$

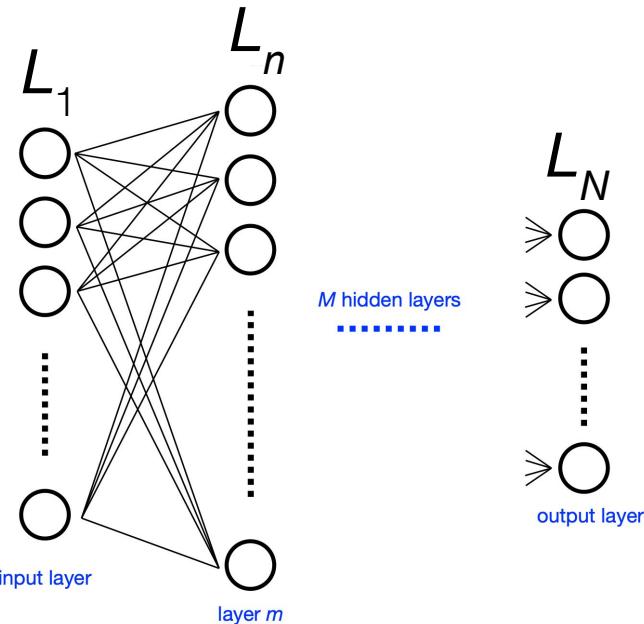
precomputed and stored in BRAMs

DSPs

logic cells

16 inputs	8
64 nodes activation: ReLU	8
32 nodes activation: ReLU	8
32 nodes activation: ReLU	8
5 outputs activation: SoftMax	

Neural network inference



$$N_{\text{multiplications}} = \sum_{n=2}^N L_{n-1} \times L_n$$

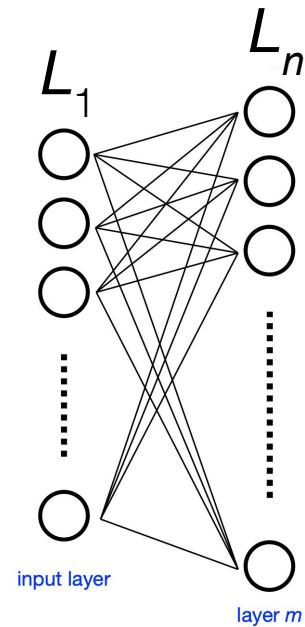
$$\mathbf{x}_n = g_n(\mathbf{W}_{n,n-1}\mathbf{x}_{n-1} + \mathbf{b}_n)$$

precomputed and stored in BRAMs DSPs logic cells

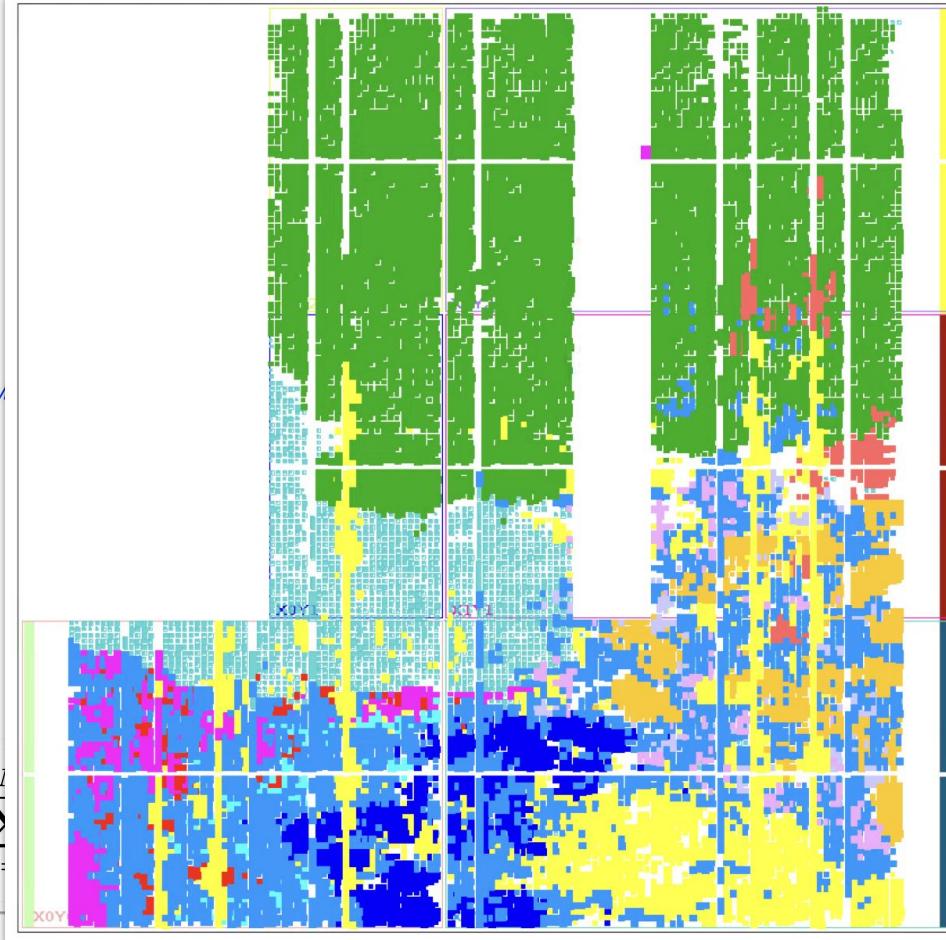
How many resources? DSPs,
LUTs, FFs?
Does the model fit in the latency
requirements?

5 outputs
activation: SoftMax

Neural network inference



$$N_{\text{multiplications}} = \sum_{n=1}^l M_n \cdot M_{n+1}$$



$-1x_{n-1} + b_n)$
SPs
logic cells
uts
ources? DSPs,
Fs?
t in the latency
ents?
uts
SoftMax

Efficient NN design for FPGAs

FPGAs provide huge flexibility

*Performance depends on how well you take
advantage of this*

Constraints:

Input bandwidth

FPGA resources

Latency

Today you will learn how to optimize your project through:

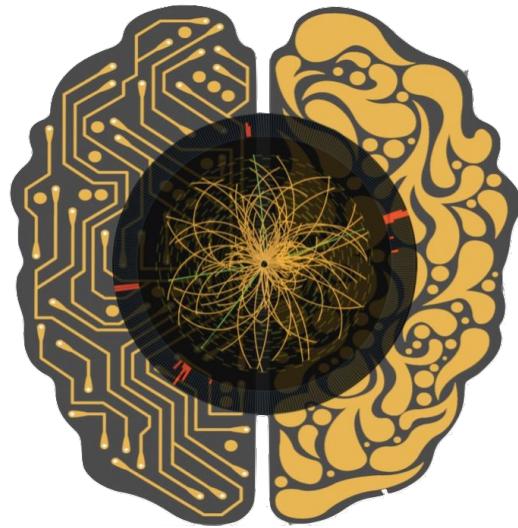
- compression:** reduce number of synapses or neurons
- quantization:** reduces the precision of the calculations (inputs, weights, biases)
- parallelization:** tune how much to parallelize to make the inference faster/slower versus FPGA resources

NN training

FPGA project
designing

Today's **hls4ml** hands on

- Part 1:
 - Get started with hls4ml: train a basic model and run the conversion, simulation & synthesis steps
- Part 4:
 - Train using Quantization Aware Training using QKeras and study impact on FPGA metrics

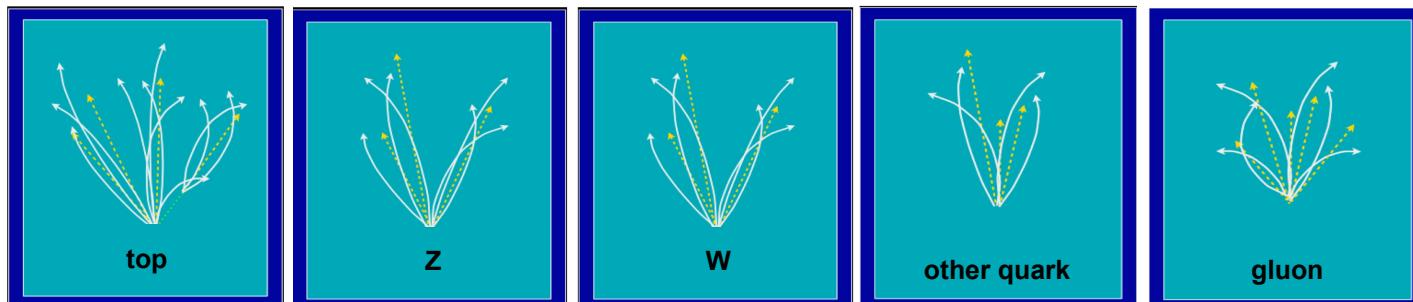


hls4ml tutorial *Part 1: Model Conversion*

Dataset: jet tagging

Study a **multi-classification task to be implemented on FPGA**: discrimination between highly energetic (boosted) **q , g , W , Z , t** initiated jets

Jet = collimated ‘spray’ of particles from hadronization of quarks and gluons



$t \rightarrow bW \rightarrow bq\bar{q}$

3-prong jet

$Z \rightarrow q\bar{q}$

2-prong jet

$W \rightarrow q\bar{q}$

2-prong jet

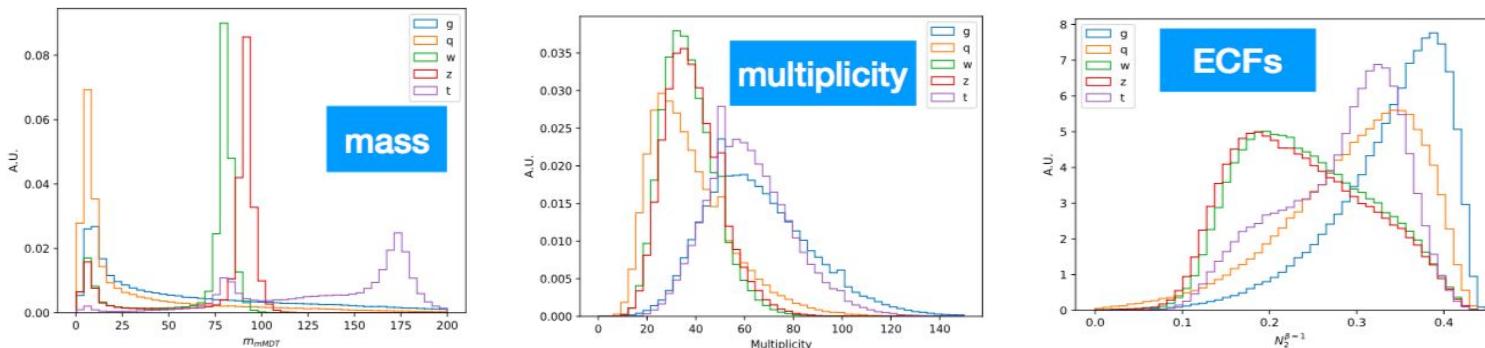
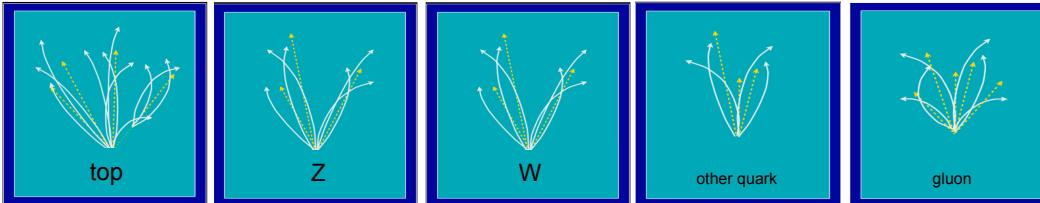
q/g background

no substructure
and/or mass ~ 0

Reconstructed as one massive jet with substructure

Dataset: jet tagging

Classes:



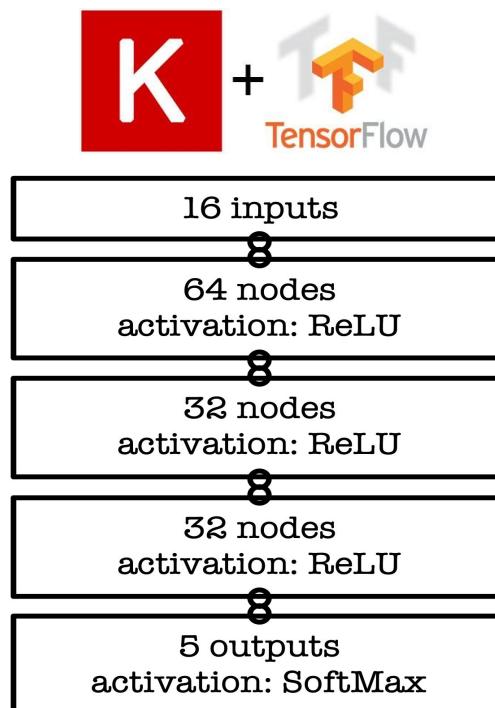
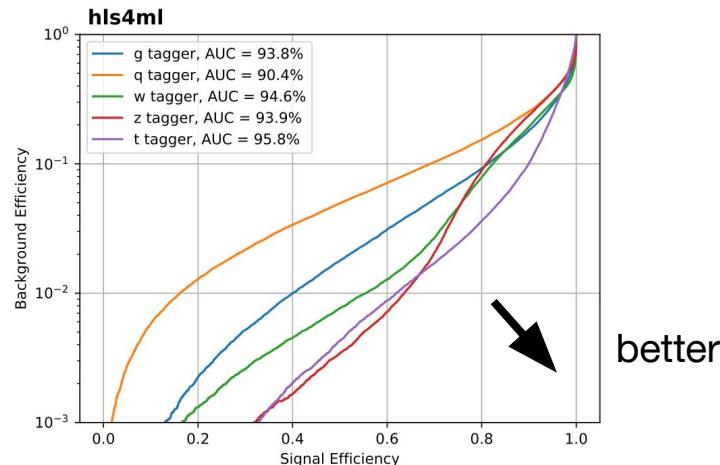
m_{mMDT}
 $N_2^{\beta=1,2}$
 $M_2^{\beta=1,2}$
 $C_1^{\beta=0,1,2}$
 $C_2^{\beta=1,2}$
 $D_2^{\beta=1,2}$
 $D_2^{(\alpha,\beta)=(1,1),(1,2)}$
 $\sum z \log z$
Multiplicity

Input variables: several observables known to have high discrimination power from offline data analyses and published studies [*]

[*] E. Coleman *et al.* [JINST.2018.T01003](#), D. Guest *et al.* [PhysRevD.94.112002](#), G. Kasieczka *et al.* [JHEP05\(2017\)006](#), J. M. Butterworth *et al.* [PhysRevLett.100.242001](#)

Dataset: jet tagging

- We'll train the **five class multi-classifier** on a sample of ~ 1M events with two boosted WW/ZZ/tt/qq/gg anti- k_T jets
 - Dataset DOI: 10.5281/zenodo.3602254
 - OpenML: <https://www.openml.org/d/42468>
- Fully connected neural network with **16 expert-level inputs**:
 - Relu activation function for intermediate layers
 - Softmax activation function for output layer



Hands On - Setup

- Open a web browser, go to <https://ssummers.web.cern.ch/hls4ml-tutorial.php>
- Use your GitHub account to authenticate

The screenshot shows a Jupyter Notebook environment. On the left, there's a file browser with a sidebar for navigating through a directory structure. The main area contains two code cells and their outputs.

File Browser:

Name	Last Modified
docker	41 minutes ago
images	41 minutes ago
model_1	16 minutes ago
model_3	3 minutes ago
pruned_cnn	41 minutes ago
quantized_pruned_cnn	41 minutes ago
sr	41 minutes ago
Y: _config.yml	41 minutes ago
Y: _toc.yml	41 minutes ago
callbacks.py	41 minutes ago
classes.npy	17 minutes ago
Y: environment.yml	41 minutes ago
nn_utils.py	41 minutes ago
part1_getting_started.ipynb	4 minutes ago
part2_advanced_config.ipynb	41 minutes ago
part3_compression.ipynb	41 minutes ago
part4_quantization.ipynb	1 minute ago
part4_1_HQ_quantization.ipynb	41 minutes ago
part5_bdt.ipynb	41 minutes ago
part6_cnns.ipynb	41 minutes ago
part7a_bitstream.ipynb	41 minutes ago
part7b_deployment.ipynb	41 minutes ago
part7c_validation.ipynb	41 minutes ago
part8_symbolic_regression.ipynb	41 minutes ago
plotting.py	41 minutes ago
README.md	41 minutes ago
test.py	17 minutes ago

Terminal 1 Output:

```
[1]: from tensorflow.keras.utils import to_categorical
from sklearn.datasets import fetch_openml
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
import numpy as np

%matplotlib inline
seed = 0
np.random.seed(seed)
import tensorflow as tf

tf.random.set_seed(seed)
import os

os.environ['PATH'] = os.environ['XILINX_VIVADO'] + '/bin:' + os.environ['PATH']

2024-12-03 09:16:53.893727: I tensorflow/core/platform/cpu_feature_guard.cc:193] This TensorFlow binary is optimized with oneAPI Deep Neural Network Library (oneDNN) to use the following CPU instructions in performance-critical operations: SSE4.1 SSE4.2 AVX AVX2 AVX512F AVX512-VNNI FMA. To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.
```

Code Cell 1:

```
[1]: from tensorflow.keras.utils import to_categorical
from sklearn.datasets import fetch_openml
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
import numpy as np

%matplotlib inline
seed = 0
np.random.seed(seed)
import tensorflow as tf

tf.random.set_seed(seed)
import os

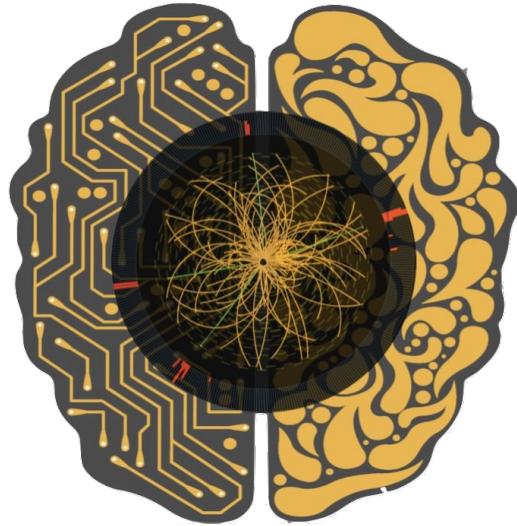
os.environ['PATH'] = os.environ['XILINX_VIVADO'] + '/bin:' + os.environ['PATH']

2024-12-03 09:16:53.893727: I tensorflow/core/platform/cpu_feature_guard.cc:193] This TensorFlow binary is optimized with oneAPI Deep Neural Network Library (oneDNN) to use the following CPU instructions in performance-critical operations: SSE4.1 SSE4.2 AVX AVX2 AVX512F AVX512-VNNI FMA. To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.
```

Code Cell 2:

```
[2]: data = fetch_openml('hls4ml_lhc_jets_hlf')
X, y = data['data'], data['target']

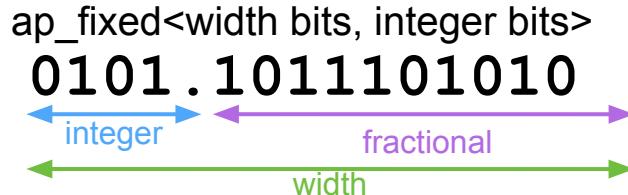
/opt/conda/lib/python3.10/site-packages/sklearn/datasets/_openml.py:968: FutureWarning: The default value of 'parser' will change from 'libarff' to 'auto' in 1.4. You can set 'parser='auto'' to silence this warning. Therefore, an ImportWarning will be raised from 1.4 if the dataset is dense and pandas is not installed. Note that the pandas parser may return different data types. See the Notes Section in fetch_openml's API doc for details.
warn(
```



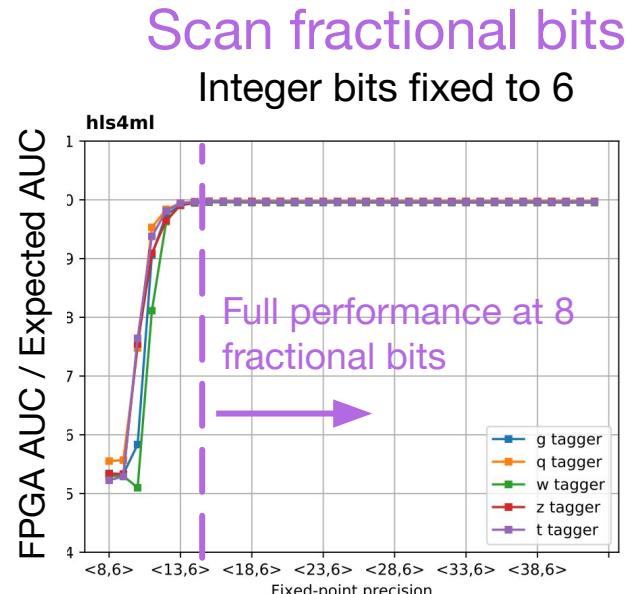
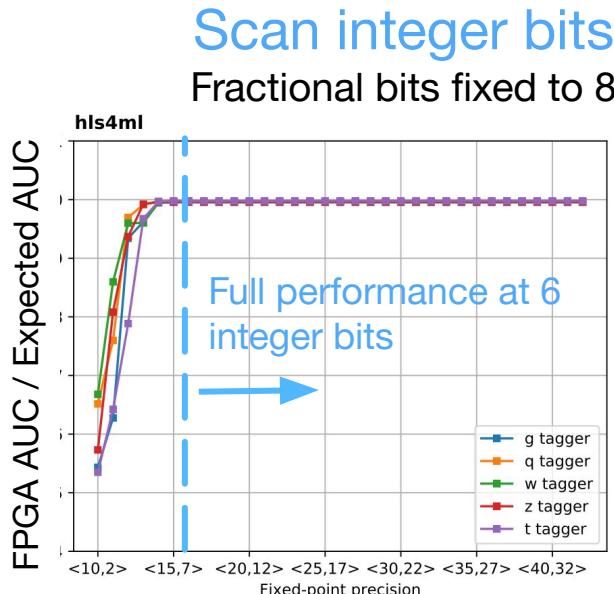
hls4ml Tutorial

Part 2: Advanced Configuration

Efficient NN design: quantization

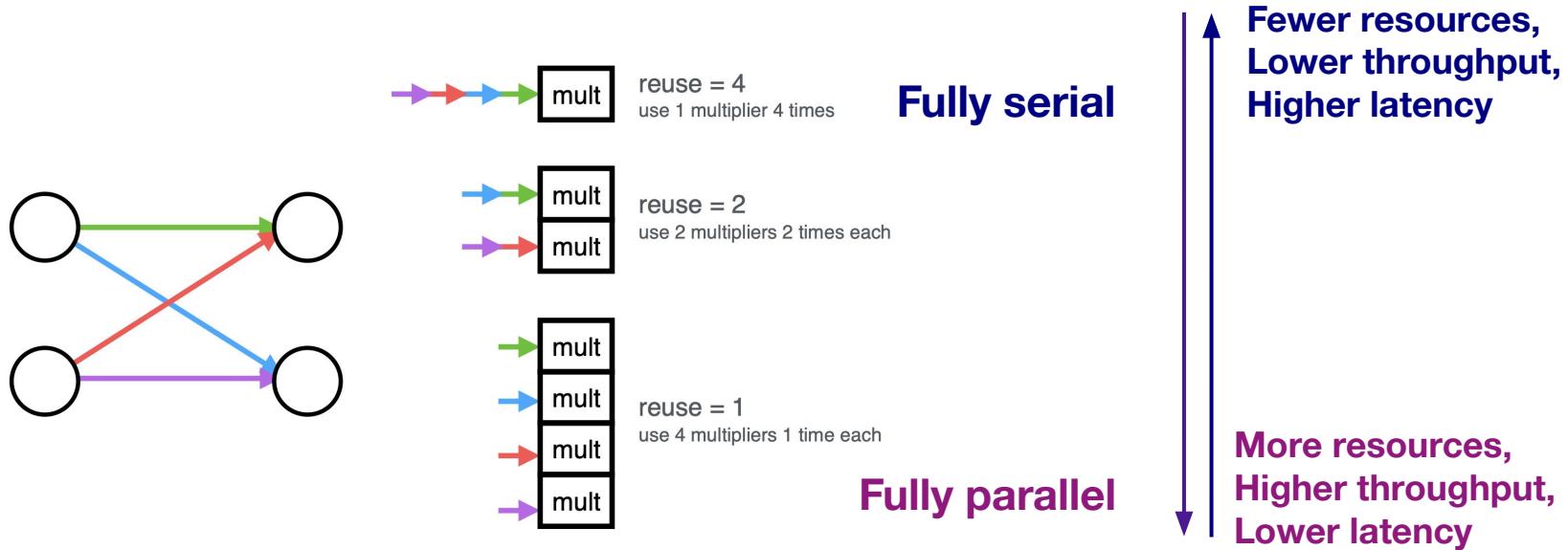


- In the FPGA we use fixed point representation
 - Operations are integer ops, but we can represent fractional values
- But we have to make sure we've used the correct data types!



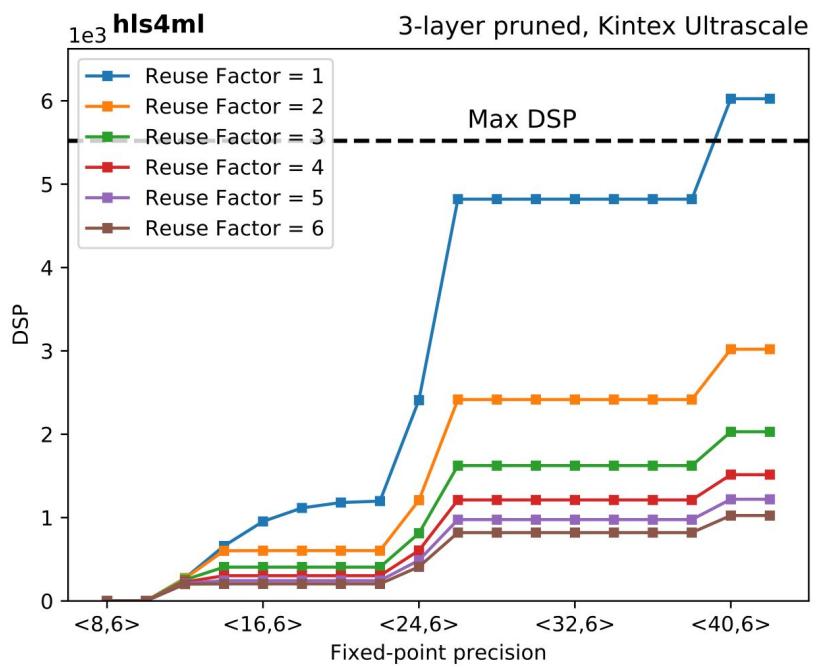
Efficient NN design: parallelization

- Trade-off between latency and FPGA resource usage determined by the parallelization of the calculations in each layer
- Configure the “reuse factor” = number of times a multiplier is used to do a computation



Reuse factor: how much to parallelize operations in a hidden layer

Parallelization: resource usage



More resources

Fully parallel
Each mult. used 1x

Each mult. used 2x

Each mult. used 3x

⋮

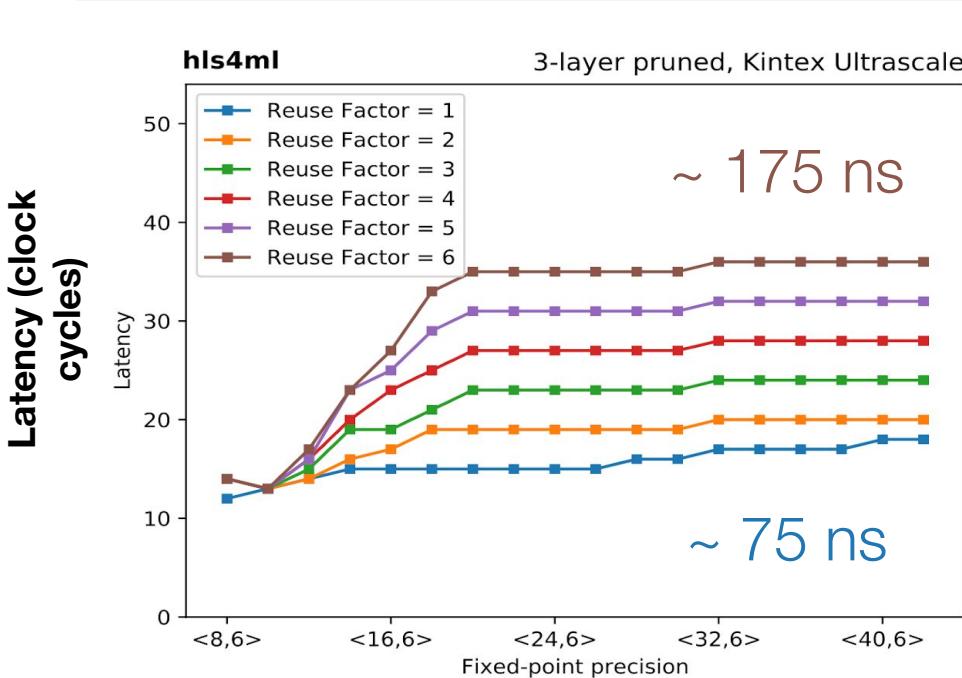
Longer latency

A vertical axis on the right side of the graph shows the relationship between resource usage and latency. An upward arrow is labeled "More resources" and points from the bottom to the top of the chart. A downward arrow is labeled "Longer latency" and points from the top back down to the bottom of the chart. Between the arrows, three text labels describe the mapping: "Fully parallel Each mult. used 1x" (top), "Each mult. used 2x" (middle), and "Each mult. used 3x" (bottom).

Parallelization → speed

Latency of layer m

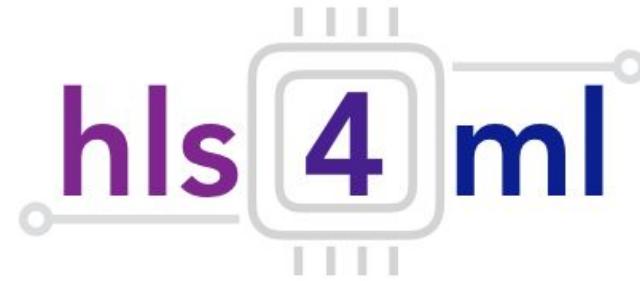
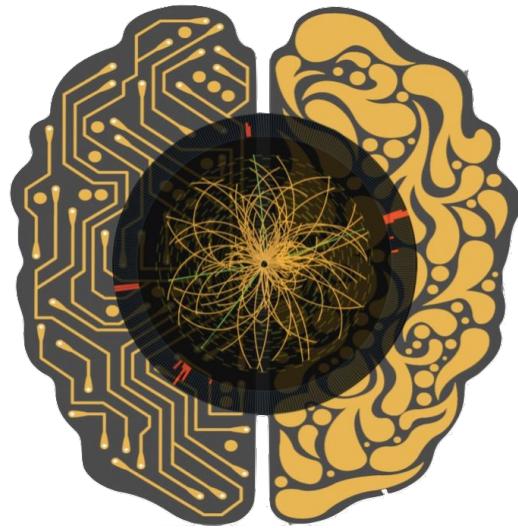
$$L_m = L_{\text{mult}} + (R - 1) \times II_{\text{mult}} + L_{\text{activ}}$$



Longer latency
More resources

Each multiplier used 6x
Each multiplier used 3x
Each multiplier used 1x

A vertical blue arrow points downwards from the top text to the bottom text, indicating a trade-off between latency and resource usage.

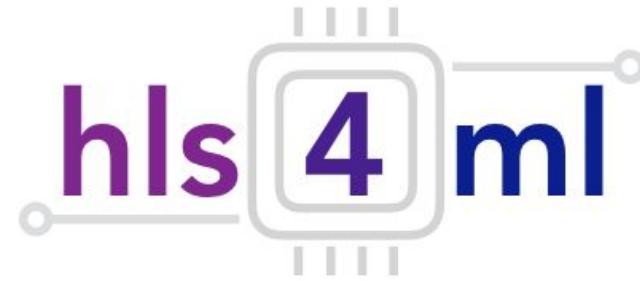
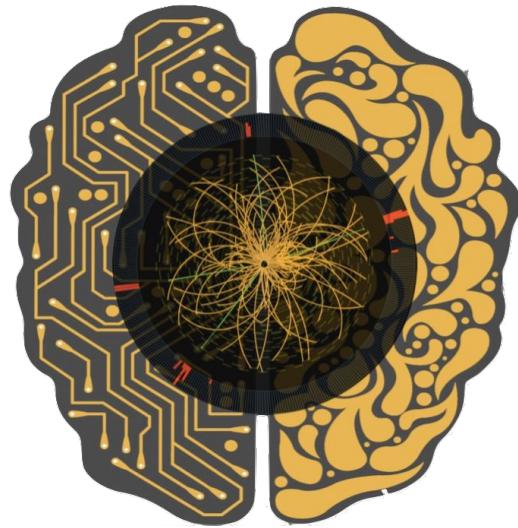


hls4ml Tutorial

Part 3: Compression

NN compression methods

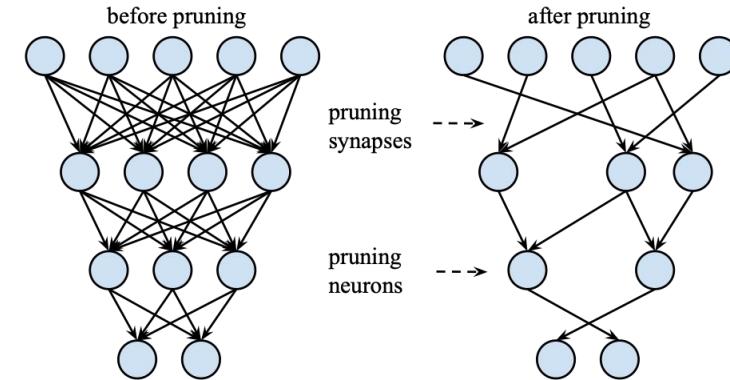
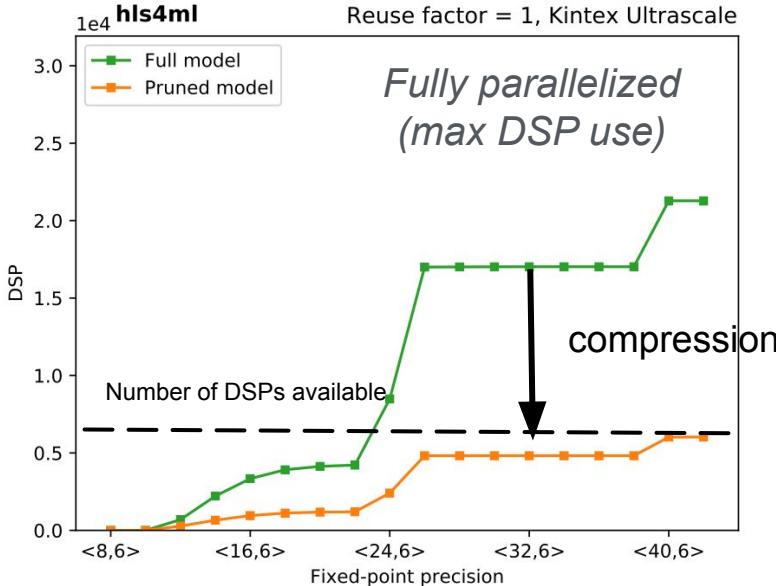
- Network compression is a widespread technique to reduce the size, energy consumption, and overtraining of deep neural networks
- Several approaches have been studied:
 - **parameter pruning:** selective removal of weights based on a particular ranking
[\[arxiv.1510.00149\]](https://arxiv.org/abs/1510.00149), [\[arxiv.1712.01312\]](https://arxiv.org/abs/1712.01312)
 - **low-rank factorization:** using matrix/tensor decomposition to estimate informative parameters
[\[arxiv.1405.3866\]](https://arxiv.org/abs/1405.3866)
 - **transferred/compact convolutional filters:** special structural convolutional filters to save parameters
[\[arxiv.1602.07576\]](https://arxiv.org/abs/1602.07576)
 - **knowledge distillation:** training a compact network with distilled knowledge of a large network
[\[doi:10.1145/1150402.1150464\]](https://doi.org/10.1145/1150402.1150464)
- Today we'll use the tensorflow model sparsity toolkit
 - <https://blog.tensorflow.org/2019/05/tf-model-optimization-toolkit-pruning-API.html>
- But you can use other methods!



hls4ml Tutorial

Part 4: Quantization and Compression

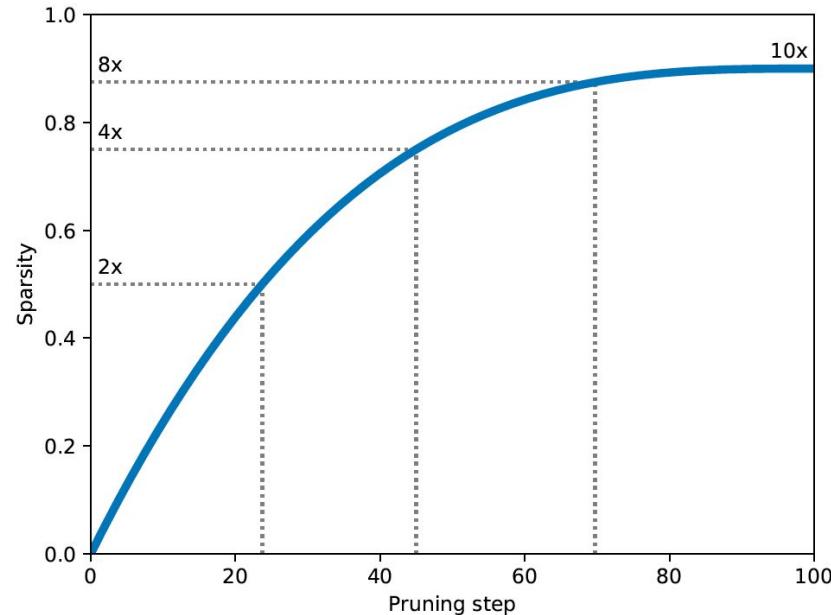
Efficient NN design: compression



- DSPs (used for multiplication) are often limiting resource
 - maximum use when fully parallelized
 - DSPs have a max size for input (e.g. 27x18 bits), so number of DSPs per multiplication changes with precision

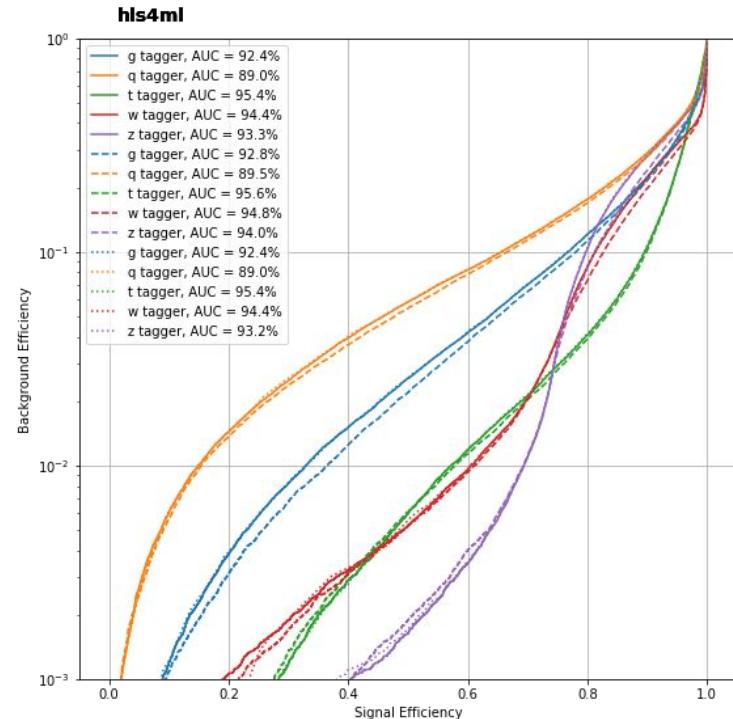
TF Sparsity

- Iteratively remove low magnitude weights, starting with 0 sparsity, smoothly increasing up to the set target as training proceeds



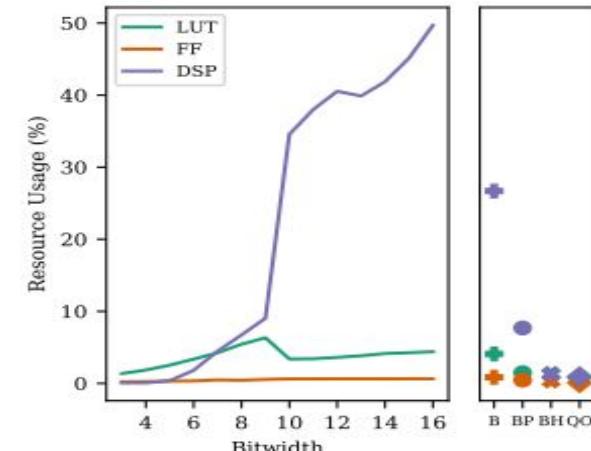
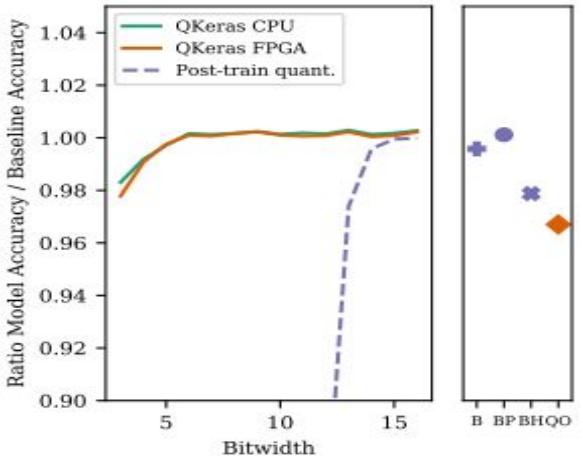
Efficient NN design: quantization

- hls4ml allows you to use different data types everywhere, we saw how to tune that in part 2
- We will also try quantization-aware training with QKeras (part 4)
- With quantization-aware we can even go down to just 1 or 2 bits
 - See our recent work:
<https://arxiv.org/abs/2003.06308>



QKeras

- QKeras is a library to train models with quantization in the training
 - Developed & maintained by Google
- Easy to use, drop-in replacements for Keras layers
 - e.g. Dense → QDense
 - e.g. Conv2D → QConv2D
 - Use ‘quantizers’ to specify how many bits to use where
 - Same kind of granularity as hls4ml
- Can achieve good performance with very few bits
 - And reduce resource usage easily!
- hls4ml supports QKeras and QONNX formats
 - The number of bits used in training is also used in inference



Summary

- After this session you've gained some hands on experience with **hls4ml**
 - Fast Inference of Neural Networks in FPGAs
 - Translated neural networks to FPGA firmware, run simulation and synthesis
 - Learned how to prune and quantize Neural Networks with QKeras
- You can find these tutorial notebooks to run locally at:
<https://github.com/fastmachinelearning/hls4ml-tutorial>
- Use **hls4ml** in your own environment: `pip install hls4ml`