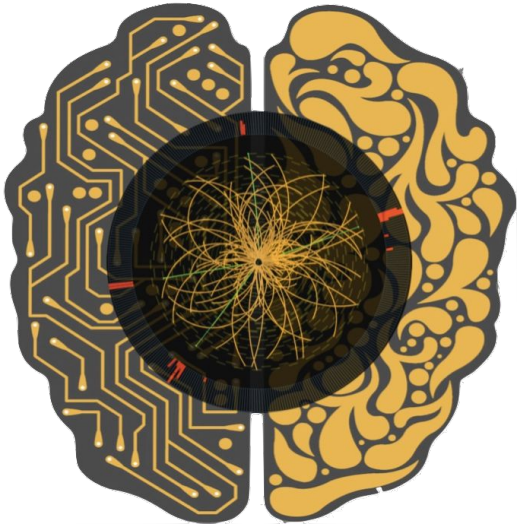


hls4ml Demo Lab

DEFCON 30

Andy Meza, Ben Hawks et al. for the hls4ml team

@not_andy_meza, @quantized_bits, and @hls4ml on Twitter!

Disclaimers, Warnings, etc.

All opinions and statements made here, around the conference, or anywhere else you find us are our own! They do not represent or reflect the US Government, US Department of Energy, Fermilab, Fermi Research Alliance LLC, CERN, University of California San Diego, or any other affiliated group or collaborator.

All work shown today is open source, and all hardware/vendor tools are off the shelf and commercially available!

We're here because we think hls4ml is cool and want to show it to you! Please be nice :)

Work supported by the Fermi National Accelerator Laboratory, managed and operated by Fermi Research Alliance, LLC under Contract No. DE-AC02-07CH11359 with the U.S. Department of Energy. The U.S. Government retains and the publisher, by accepting the article for publication, acknowledges that the U.S. Government retains a non-exclusive, paid-up, irrevocable, world-wide license to publish or reproduce the published form of this manuscript, or allow others to do so, for U.S. Government purposes.

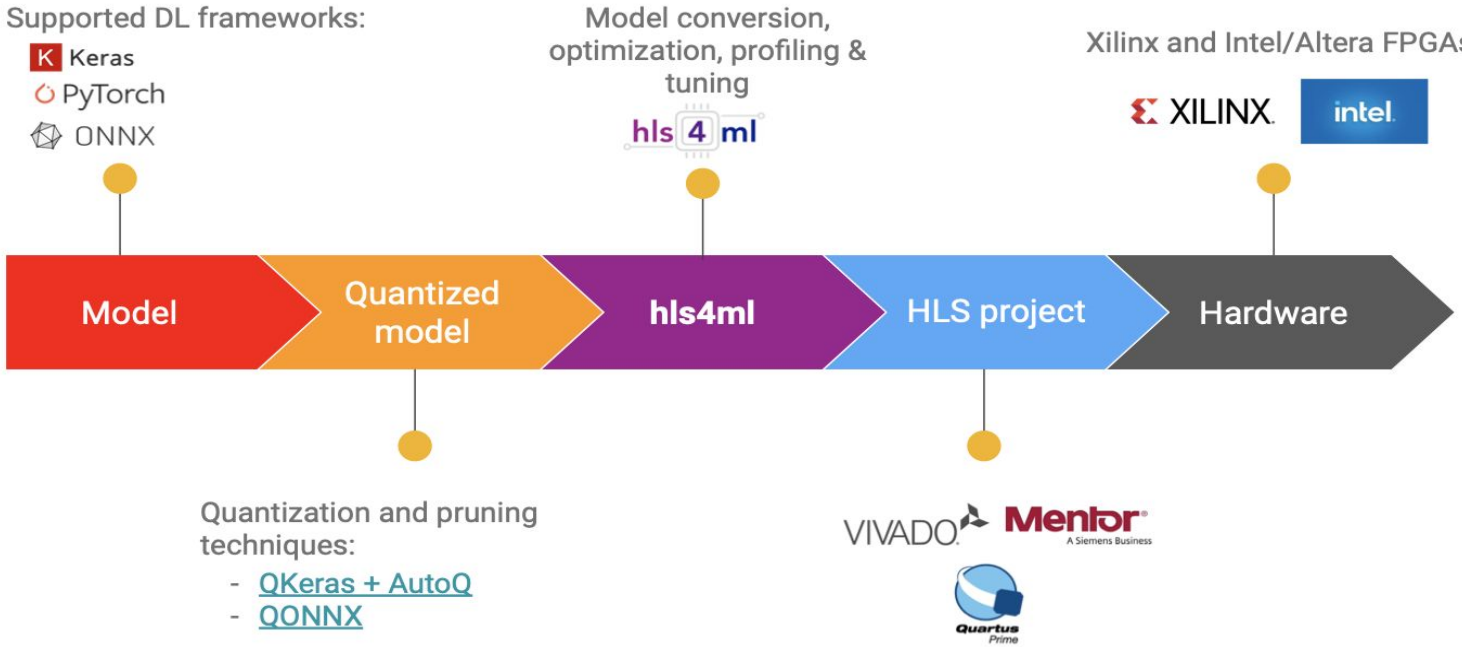
FERMILAB-SLIDES-22-107-SCD

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
    - Demo a real time inference implementation on a low cost, commercially available dev board!

high level synthesis for machine learning



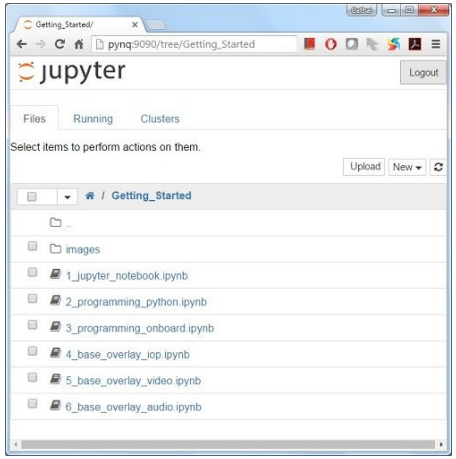
What we *won't* cover today

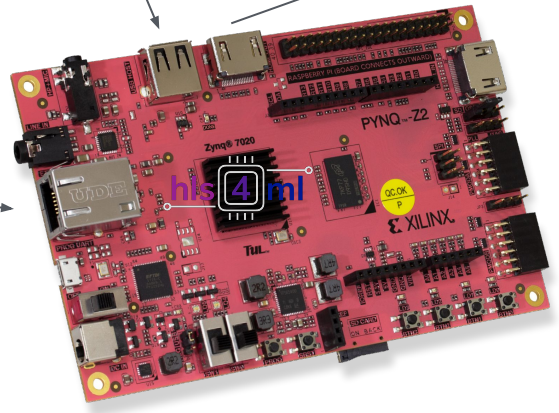
- What comes after hls4ml… you would need to integrate the Neural Network Firmware/'IP Core' into a larger design
  - For more off-the-shelf boards, integration with system-on-chip or host CPU can be more straightforward
    - https://github.com/mlcommons/tiny_results_v0.7/tree/main/open/hls4ml-finn
  - For a custom board, you'd need to do this by hand (e.g. CMS L1 Trigger, National Instruments DAQ framework)

Demo #1 - Live Pokémon Inference

Using the Pynq Software stack (Python api to interact with & program FPGA, hosts Jupyter directly on Pynq-Z2 Board)

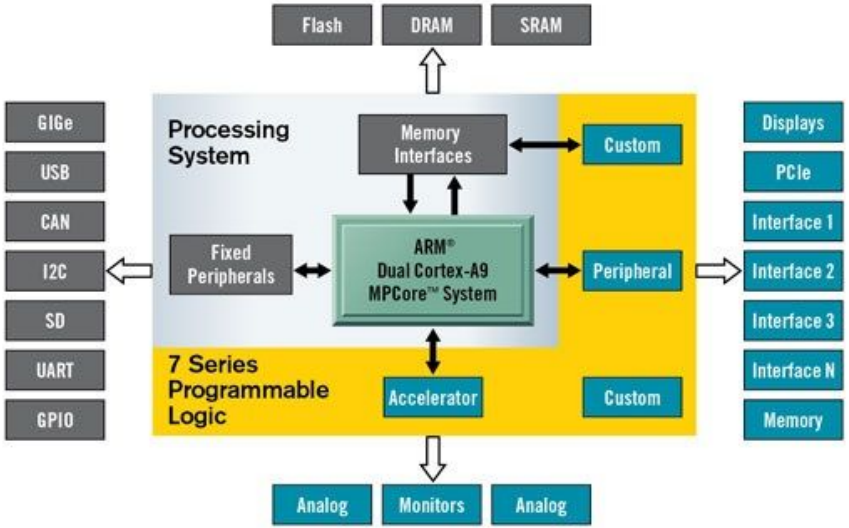
Have a live webcam running inferences via HLS4ML accelerator, outputting to an HDMI display



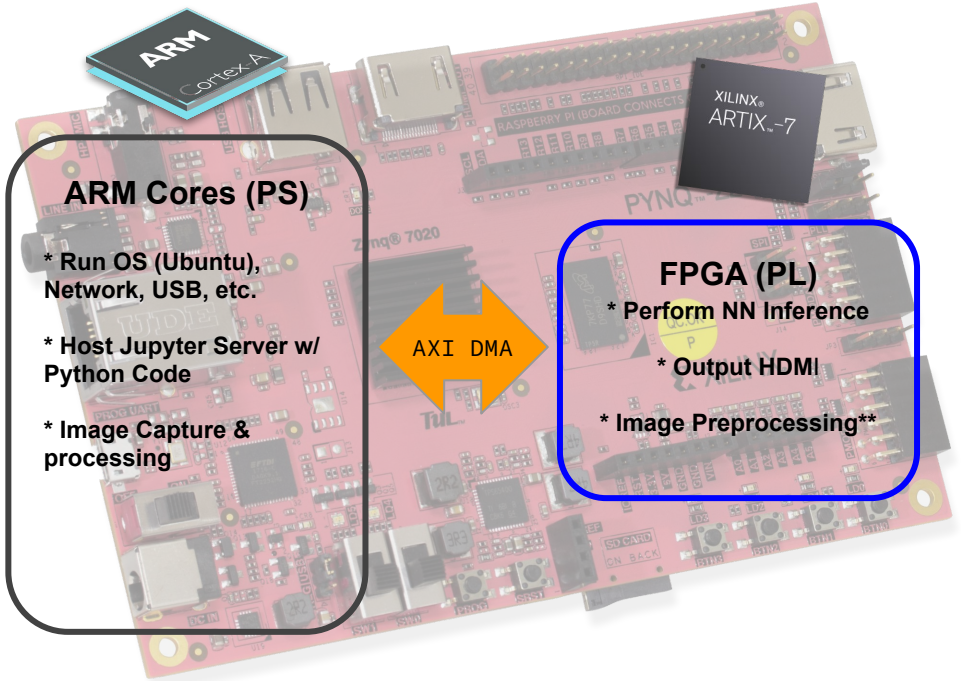


Demo Hardware - Pynq Z2

Zynq XC7Z020 Block Diagram



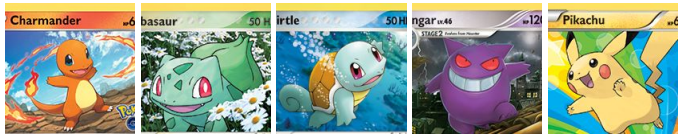
TUL Pynq-Z2 w/ Xilinx Zynq XC7Z020

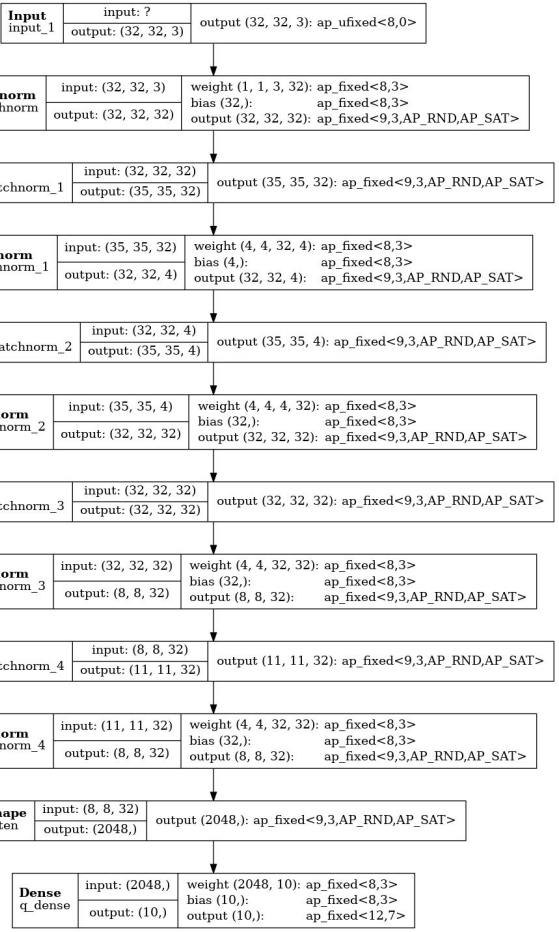


** Capable of accelerating some OpenCV operations, but we ran out of time :)

Example Model - Image Classification

- This is a 2D Convolutional Neural Network
  - Originally based on Resnet-8…
  - …but we removed the residual connections and changed the architecture a bit
  - Quantized weights, biases, and inputs to 8b (via QKeras)
- Trained to distinguish between 10 Classes, originally from CIFAR-10 (32x32 px, 24b RGB images)
  - 

    etc.
- …but we also retrained it on Pokémon for this live demo
  - 



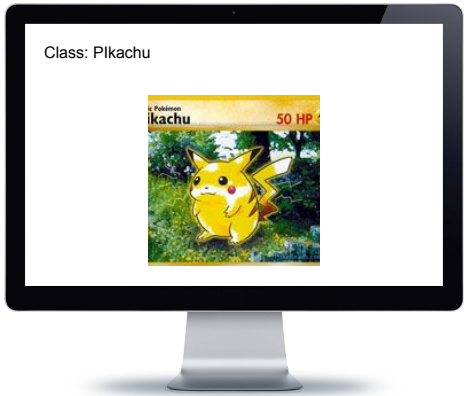
"RN07" (v0.7):
58,115 parameters
83.5% acc. on CIFAR-10*
(note: removed activations)

Demo #1 - Image Processing Flow

CPU

Crop

Image is natively 640x480,
24 bit (3x8b) RGB



FPGA

Image to Display

Resize to
32x32px
(Bilinear)

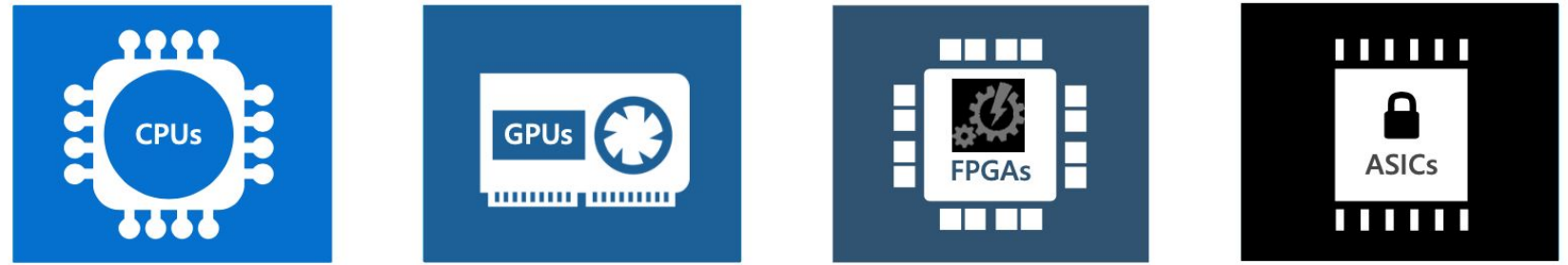
Neural
Network

Pred. Class

Add Text
(Prediction)

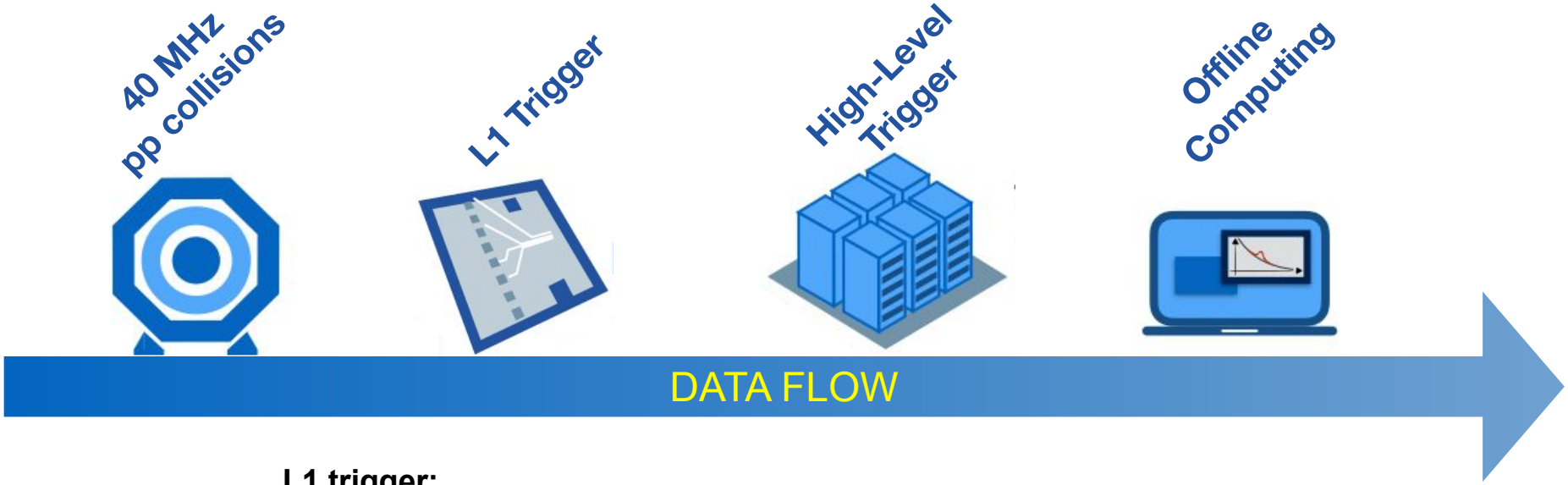
HDMI Out

Why FPGAs?



Guidelines:
> 100 Gbps throughput
< 1ms computational latency
< 10W power budget

LHC Experiment Data Flow

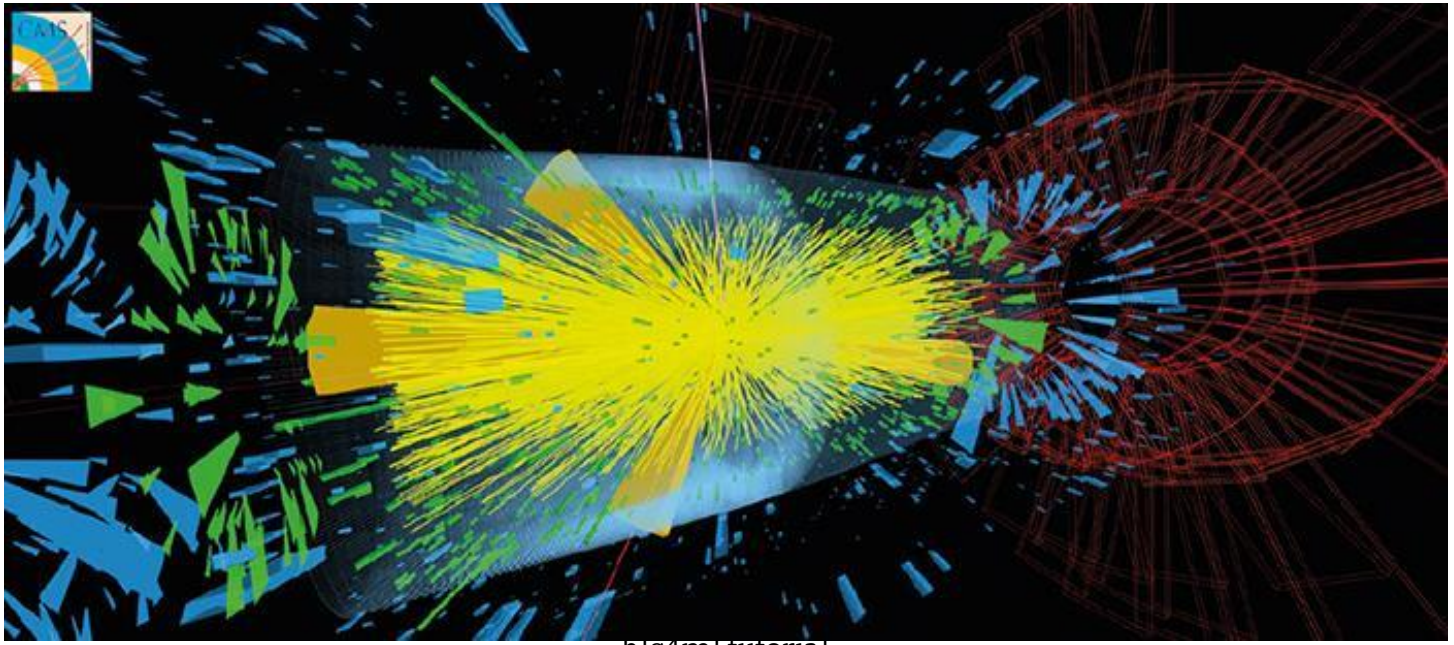


L1 trigger:

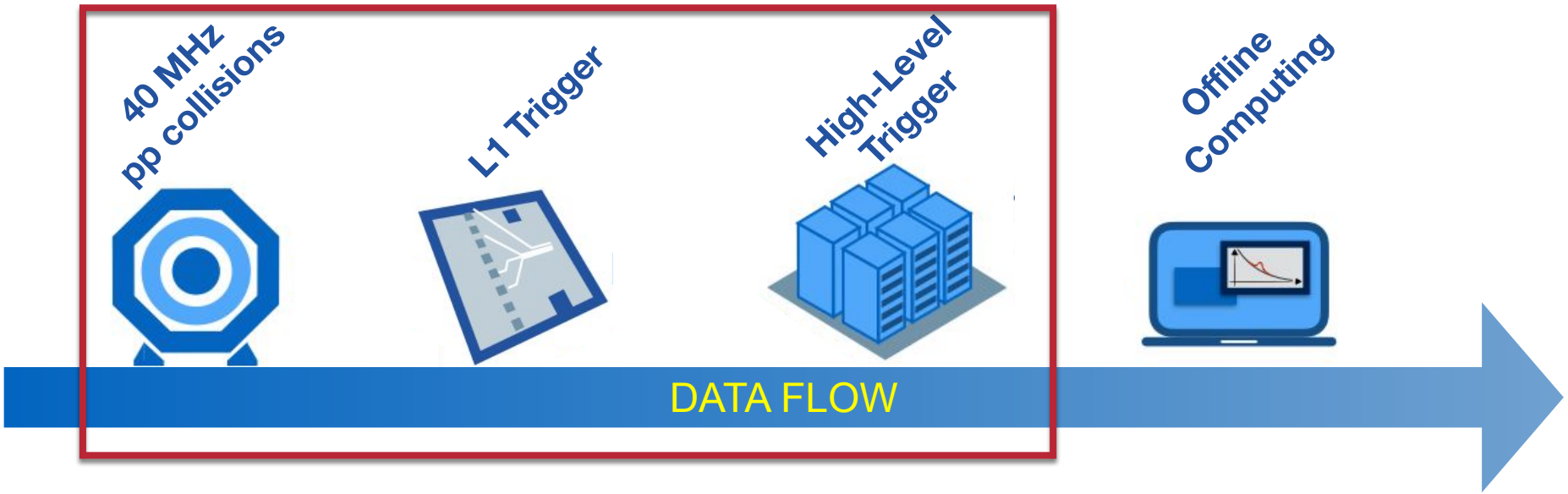
- 40 MHz in / 100 KHz out (GB Data/ per event)
- Process 100s TB/s
- Trigger decision to be made in ≈ **10 μs**

hls4ml origins: triggering at (HL-)LHC

Extreme collision frequency of 40 MHz → extreme data rates O(100 TB/s)
Most collision "events" don't produce interesting physics
"Triggering" = filter events to reduce data rates to manageable levels



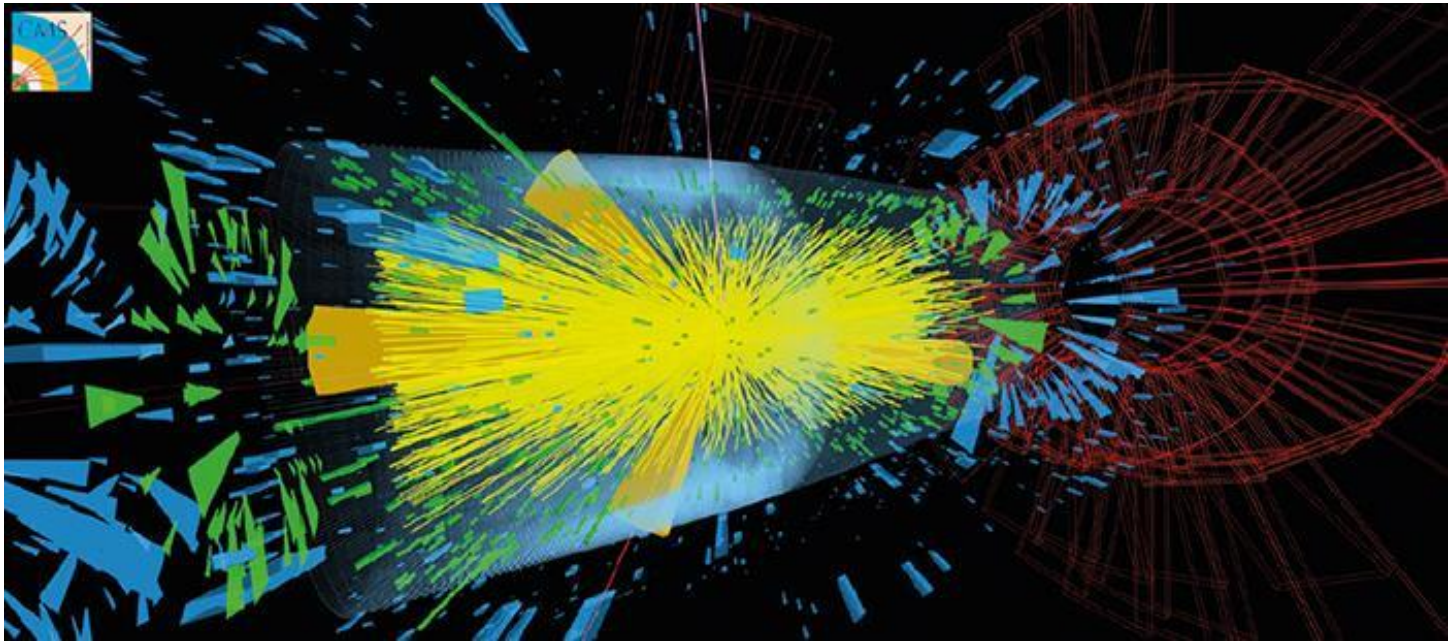
LHC Experiment Data Flow



Deploy ML algorithms very early in the game
Challenge: strict latency constraints!

The challenge: triggering at (HL-)LHC

The trigger discards events ***forever***, so selection must be very precise
ML can improve sensitivity to rare physics
Needs to be ***fast!***
Enter: **hls4ml** (high level synthesis for machine learning)

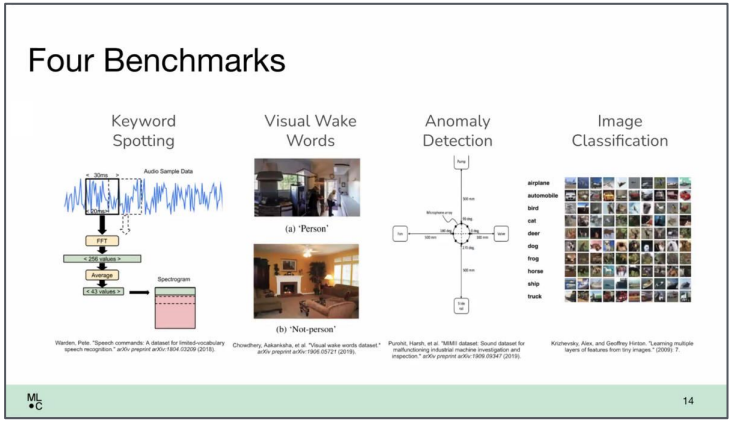


hls4ml: progression

- Previous slides showed the original motivation for hls4ml
  - Extreme low latency, high throughput domain
- Since then, we have been expanding!
  - Longer latency domains, larger models, resource constrained
  - Different FPGA vendors
  - New applications, new architectures
- While maintaining core characteristics:
  - “Layer-unrolled” HLS library → not another DPU
  - Extremely configurable: precision, resource vs latency/throughput tradeoff
  - Research project, application- and user-driven
  - Accessible, easy to use

Coming Soon

- A few exciting new things are being developed and should become available soon:
  - Intel Quartus HLS, Mentor Catapult HLS, Intel One API ‘Backends’
  - Recurrent Neural Networks
  - **More integrated ‘end-to-end’ flow with bitfile generation and host bindings for platforms like Alveo, PYNQ**
    - What we’re showing today 😉



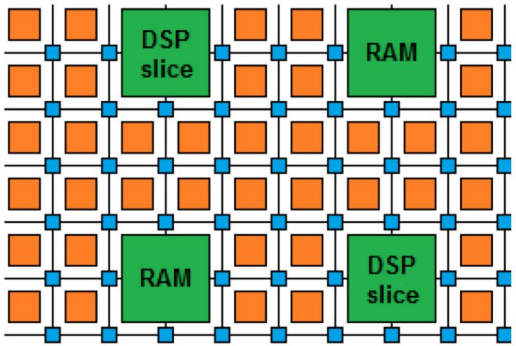
What are FPGAs?

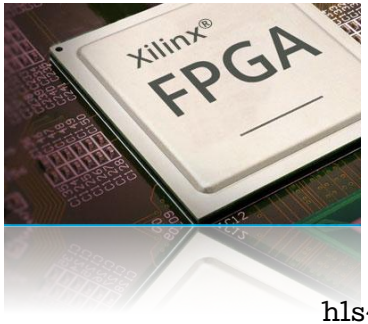
Field Programmable Gate Arrays are reprogrammable integrated circuits

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

FPGA diagram





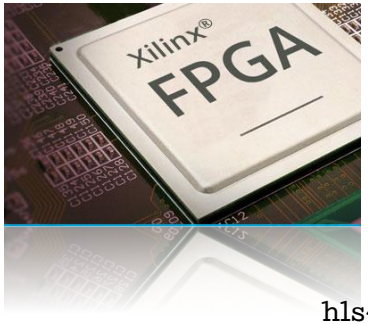
What are FPGAs?

Field Programmable Gate Arrays are reprogrammable integrated circuits

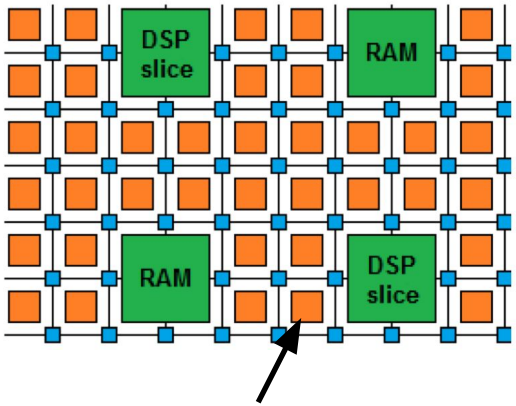
Logic cells / Look Up Tables perform arbitrary functions on small bitwidth inputs (2-6)

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

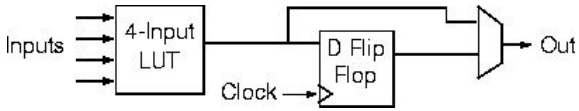
Flip-Flops register data in time with the clock pulse



FPGA diagram



Logic cell



Look-up table (logic)

Flip-flop (registers)

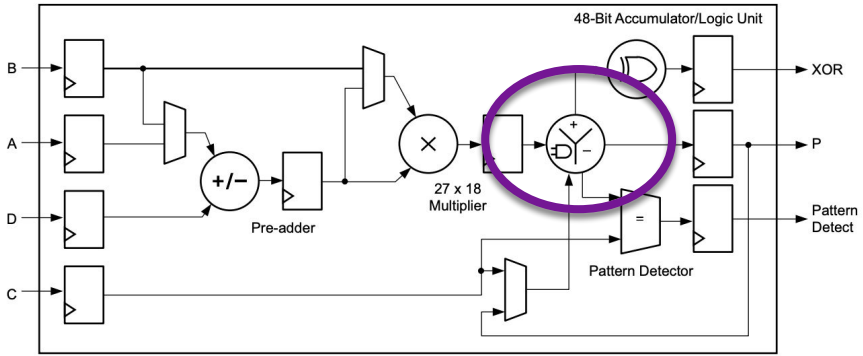
What are FPGAs?

Field Programmable Gate Arrays are reprogrammable integrated circuits

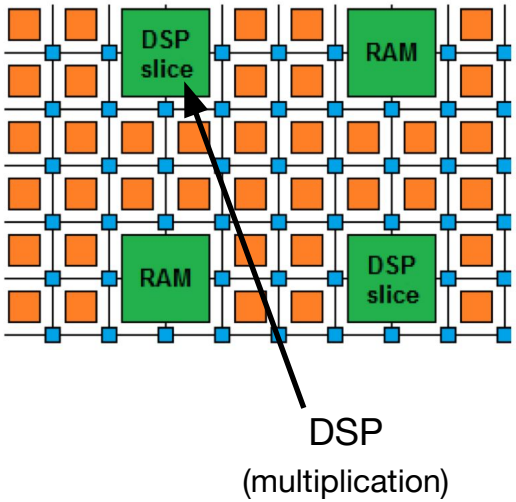
DSPs (Digital Signal Processor) are specialized units for multiplication and arithmetic

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

And for Neural Nets, DSPs are often the most scarce



FPGA diagram



What are FPGAs?

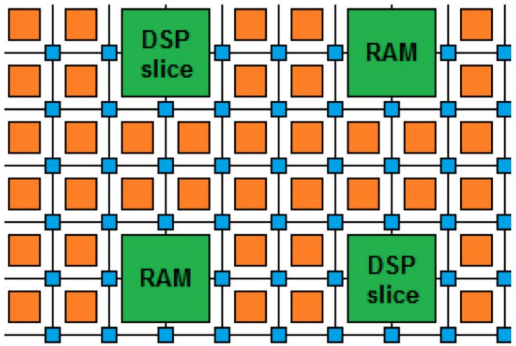
Field Programmable Gate Arrays are reprogrammable integrated circuits

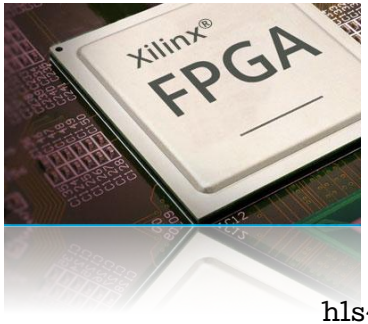
BRAMs are small, fast memories - RAMs, ROMs, FIFOs (18Kb each in Xilinx)

Memories using BRAMs more efficient than using LUTs

A big FPGA has nearly 100Mb of BRAM, chained together as needed

FPGA diagram





What are FPGAs?

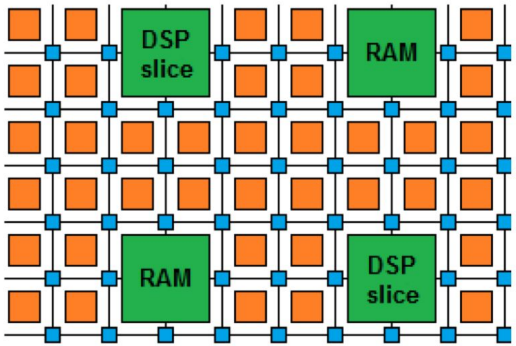
In addition, there are specialised blocks for I/O, making FPGAs popular in embedded systems and HEP triggers

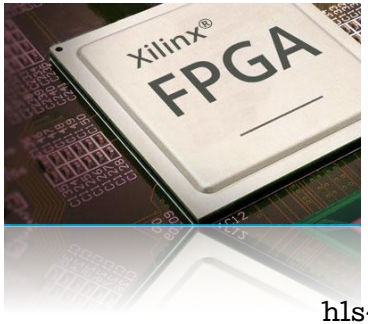
High speed transceivers with Tb/s total bandwidth
PCIe, (Multi) Gigabit Ethernet, Infiniband

AND: Support highly parallel algorithm implementations

Low power per Op (relative to CPU/GPU)

FPGA diagram





How are FPGAs programmed?

Hardware Description Languages

HDLs are programming languages which describe electronic circuits

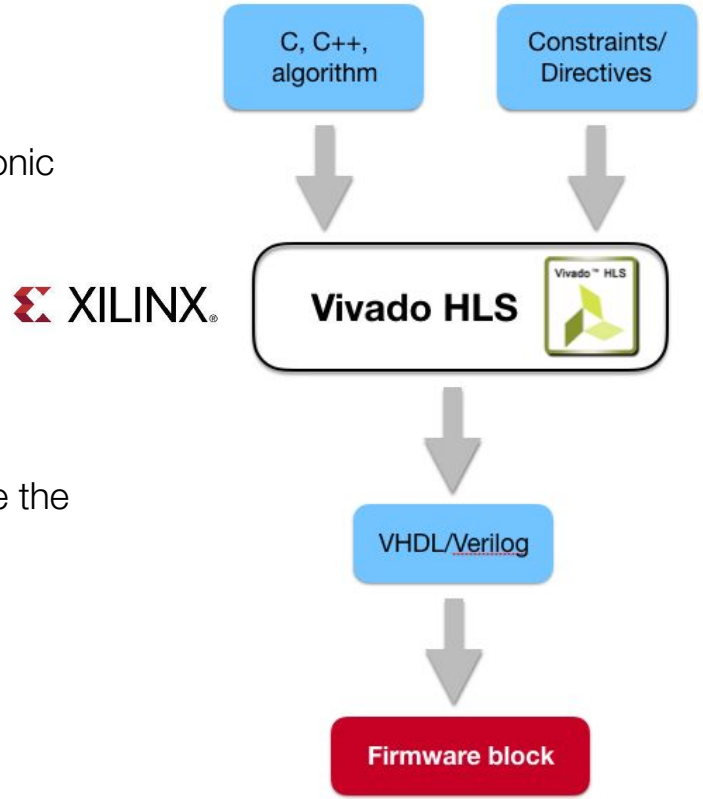
High Level Synthesis

Compile from C/C++ to VHDL

Pre-processor directives and constraints used to optimize the design

Drastic decrease in firmware development time!

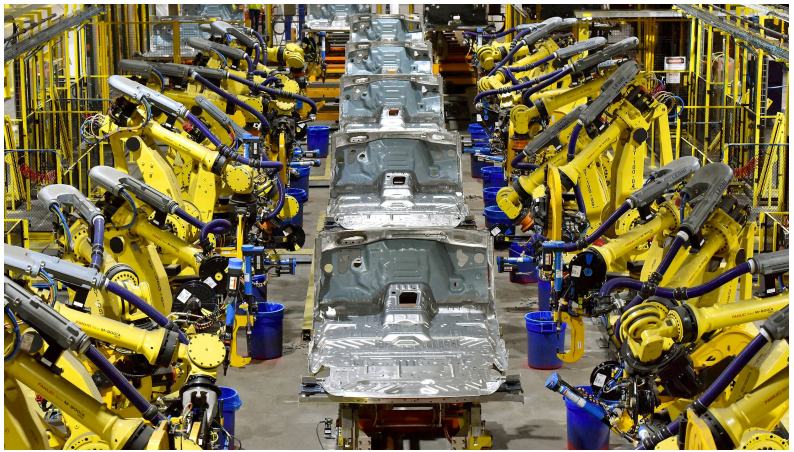
Today we'll use Xilinx Vivado HLS [*]



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

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

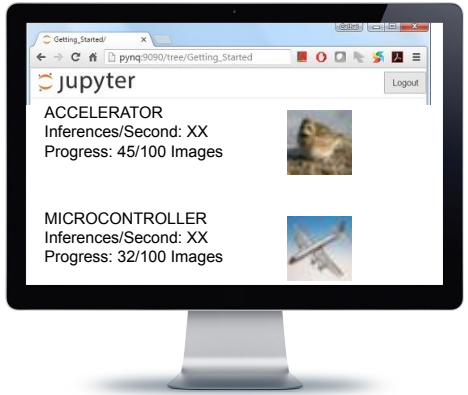


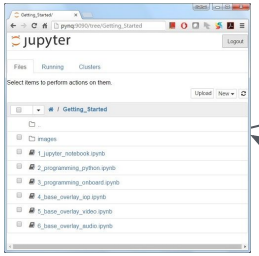
Like a production line for data…

Demo #2 - Inference Race

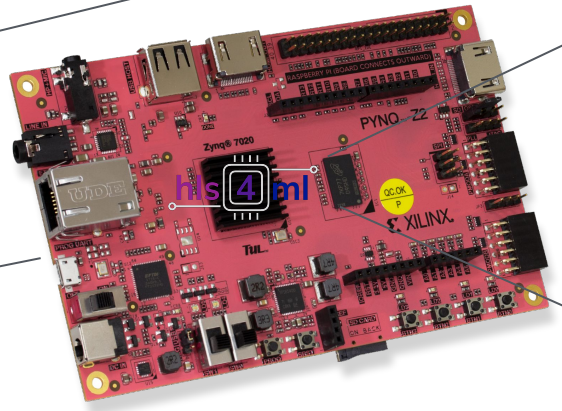
Using the Pynq Software stack (Python api to interact with & program FPGA, hosts Jupyter directly on Pynq)

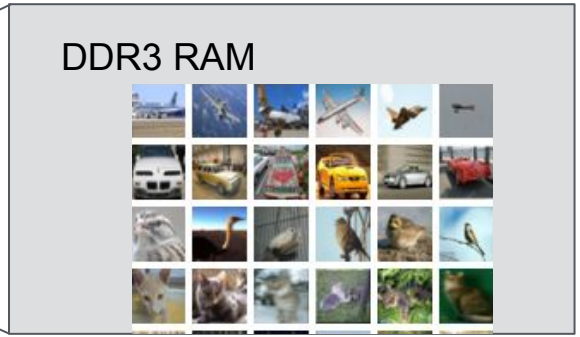
Run a sample model on the accelerator & MCU with live representation on screen to demo speed of accelerator vs a regular MCU

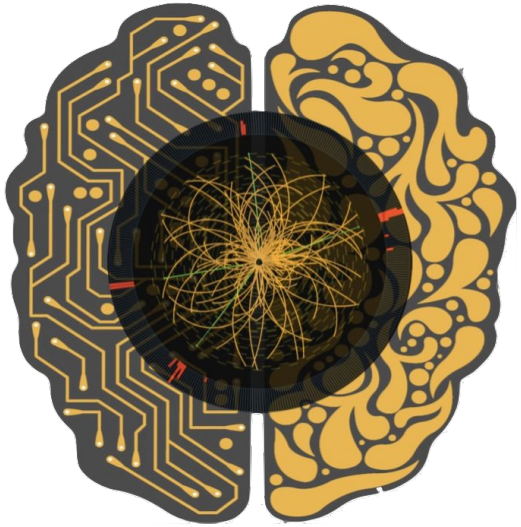




Power







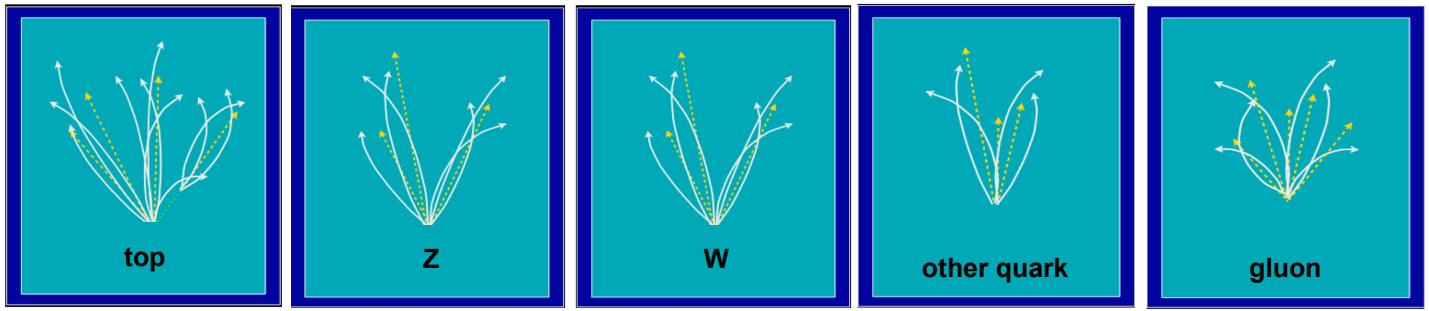
hls4ml tutorial

Part 1: Model Conversion

Physics case: 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



t→bW→bqq 3-prong jet

Z→qq 2-prong jet

W→qq 2-prong jet

Reconstructed as one massive jet with substructure

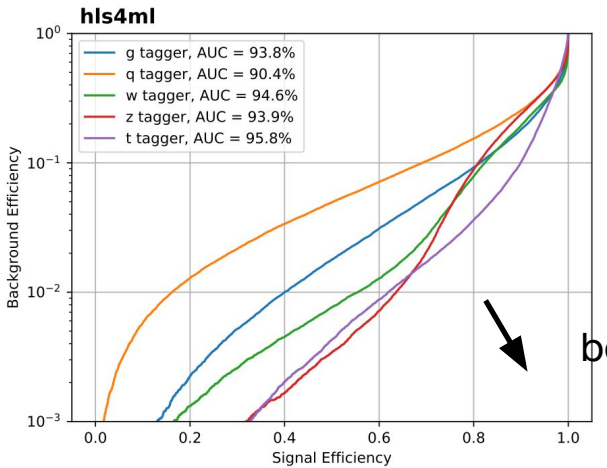
q/g background

no substructure
and/or mass ~ 0

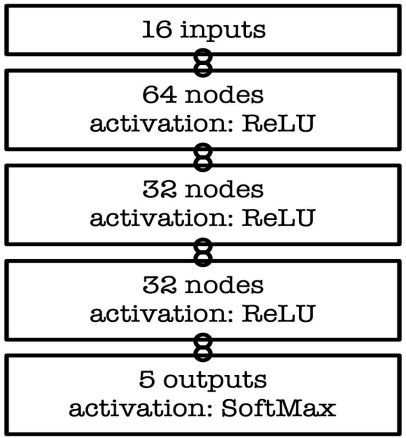
Physics case: jet tagging

- 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

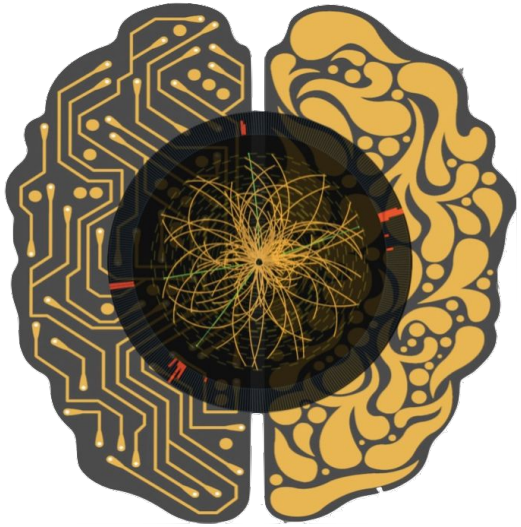
- Fully connected neural network with **16 expert-level inputs**:
  - Relu activation function for intermediate layers
  - Softmax activation function for output layer



better

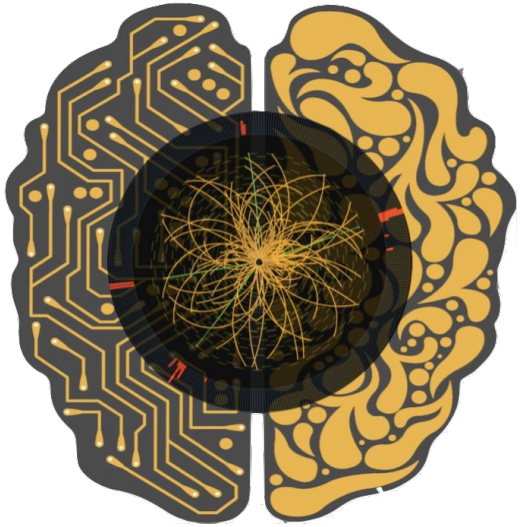


AUC = area under ROC curve
(100% is perfect, 20% is random)



hls4ml Tutorial

Part 2: Advanced Configuration



hls4ml Tutorial

Part 4: Quantization

Would you like to know more?

- We have a collection of hands on tutorial Jupyter notebooks!
  - They step through, in much more detail, the process of using hls4ml, along with how to optimize your model even more for hardware implementation!
- You can find these tutorial notebooks to run locally at: https://github.com/fastmachinelearning/hls4ml-tutorial
- You can run the tutorial Docker image yourself like:
  - docker run -p 8888:8888 gitlab-registry.cern.ch/ssummers/hls4ml-tutorial:18.v
  - 15 GB download! Or remove '.v' for a much smaller image but without Xilinx tools (so no 'build')
- Use hls4ml in your own environment: `pip install hls4ml[profiling]`
  - The version of hls4ml used in today's demo is slightly different, and merging the changes into the main branch is currently in progress. If you're trying to replicate the demo, you can use this branch here: https://github.com/hls4ml-finn-mlperftiny/hls4ml

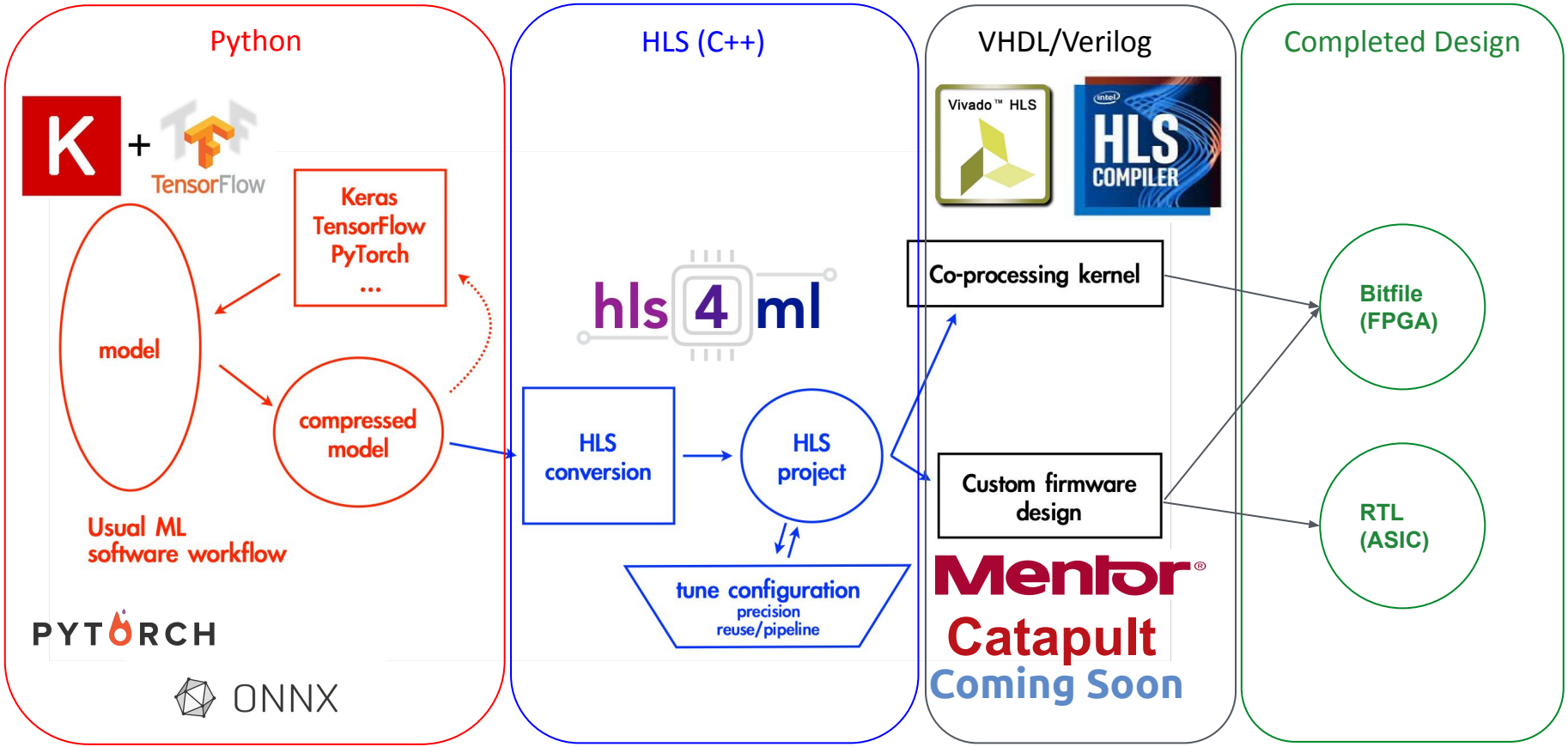
Thanks!

Thank you for giving us some of your time and attention, and we hope you found this interesting and even useful!

- hls4ml on Github: https://github.com/fastmachinelearning/hls4ml
- hls4ml documentation: https://fastmachinelearning.org/hls4ml/
- The FastMachineLearning community: https://fastmachinelearning.org/
- Is there a feature that you want added to hls4ml? We happily welcome contributions, pull requests, and collaboration!
    - If you want to work with the maintaining organizations to directly build a project, add features/support for something you need, and more, many of them offer "Technology/Knowledge Transfer" Programs! You can contact:
        - Us! bhawks@fnal.gov, anmeza@ucsd.edu
        - Fermilab: https://partnerships.fnal.gov/
        - CERN: https://kt.cern/
- Massive thank you to all the people who worked on this demo who couldn't be here today!
  Micol Rigatti, Javier Campos, Giuseppe Di Guglielmo, Jules Muhizi, Jovan Mitrevski, and all of our summer students!

Backup Slides!

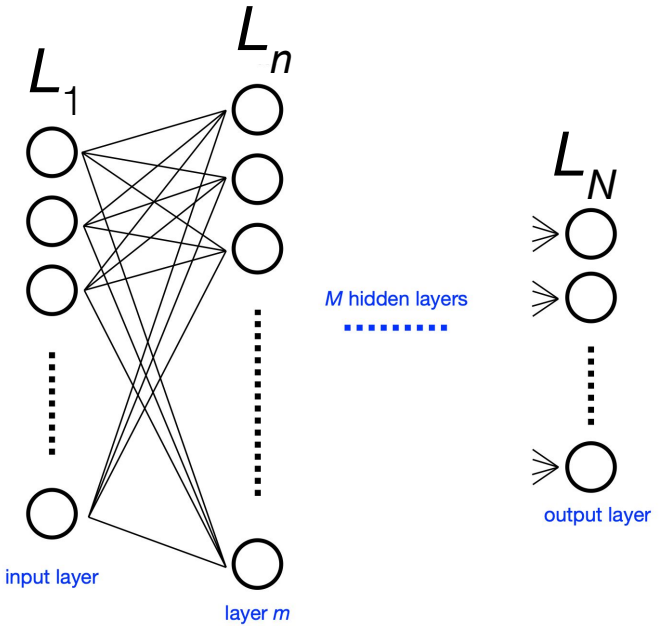
high level synthesis for machine learning



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

Neural network inference



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

precomputed and stored in BRAMs

DSPs

logic cells

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

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

There are multiple ways to optimize a network :

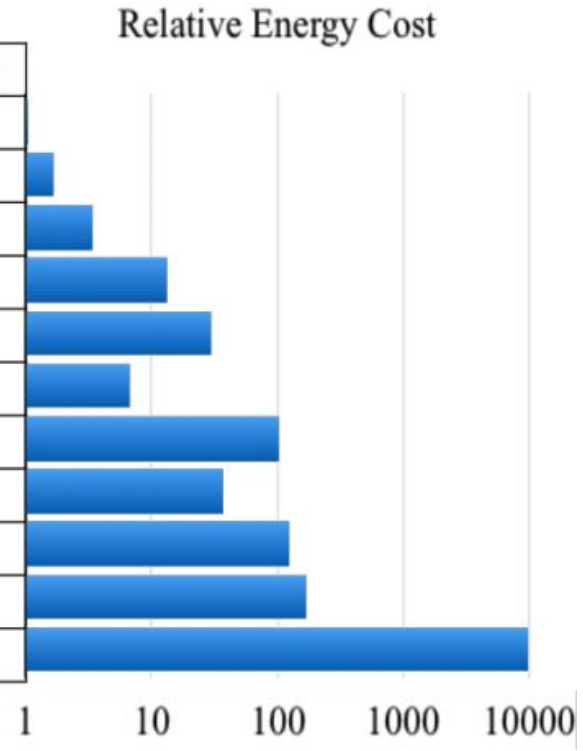
- **compression:** reduce number of synapses or neurons
- **quantization:** reduces the precision of the calculations (inputs, weights, biases)
- **parallelization**: tune how much to parallelize to make the inference faster/slower versus FPGA resources

NN training

FPGA project design

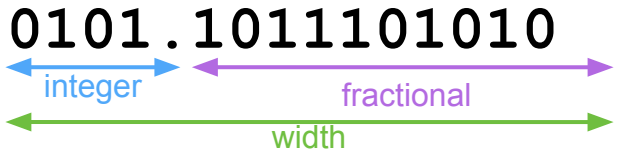
Efficient NN design: quantization

Operation:	Energy (pJ)
8b Add	0.03
16b Add	0.05
32b Add	0.1
16b FP Add	0.4
32b FP Add	0.9
8b Mult	0.2
32b Mult	3.1
16b FP Mult	1.1
32b FP Mult	3.7
32b SRAM Read (8KB)	5
32b DRAM Read	640



Efficient NN design: quantization

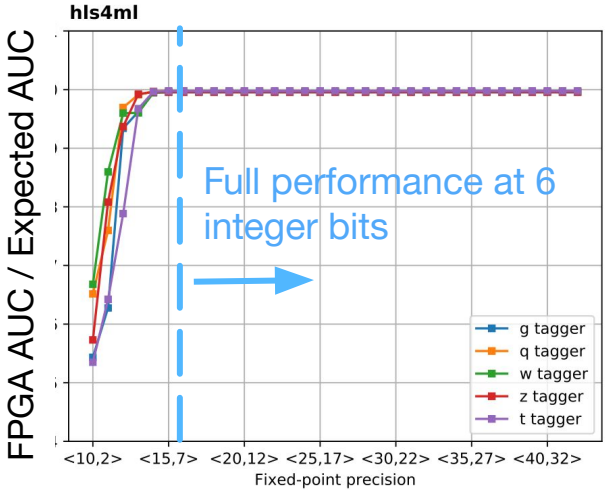
ap_fixed<width bits, integer bits>



- 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!

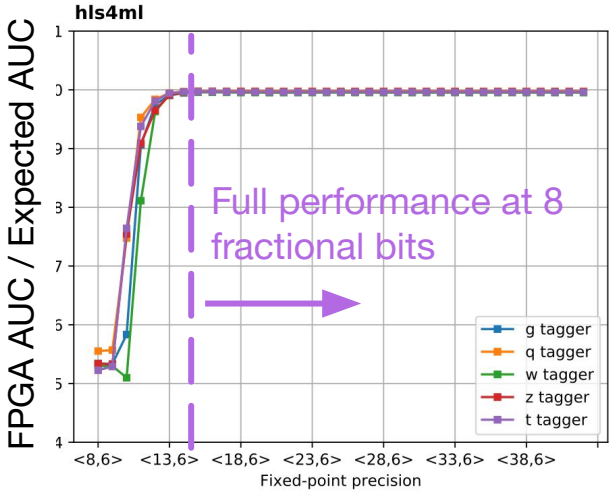
Scan integer bits

Fractional bits fixed to 8



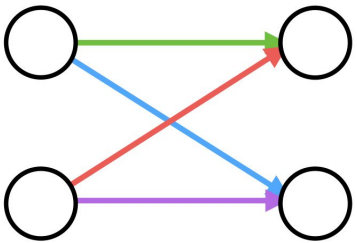
Scan fractional bits

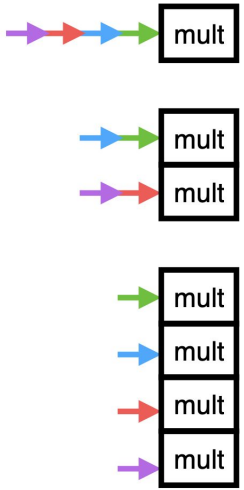
Integer bits fixed to 6



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





Fully serial

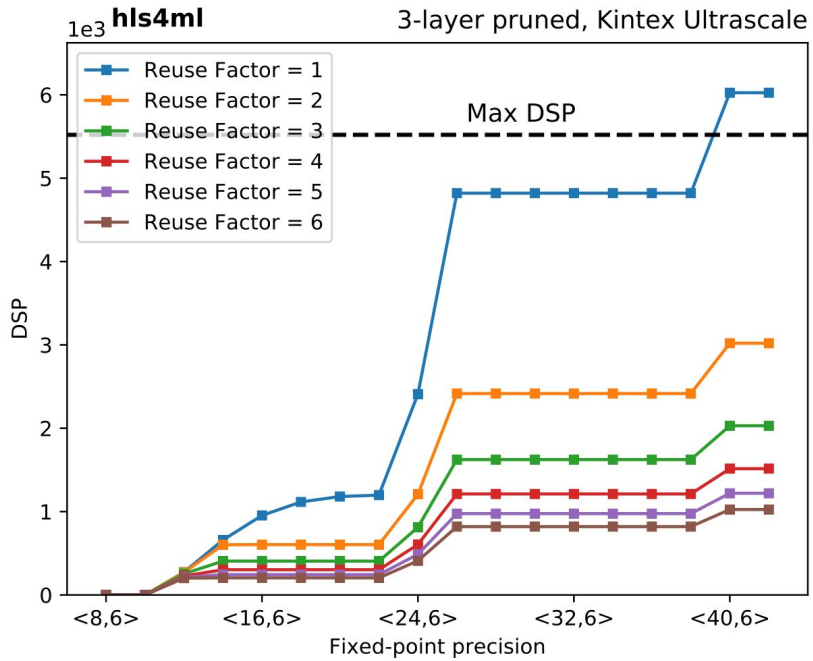
Fewer resources, Lower throughput, Higher latency

Fully parallel

More resources, Higher throughput, Lower latency

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

Parallelization: DSP usage



More resources

Fully parallel
Each mult. used 1x

Each mult. used 2x

Each mult. used 3x

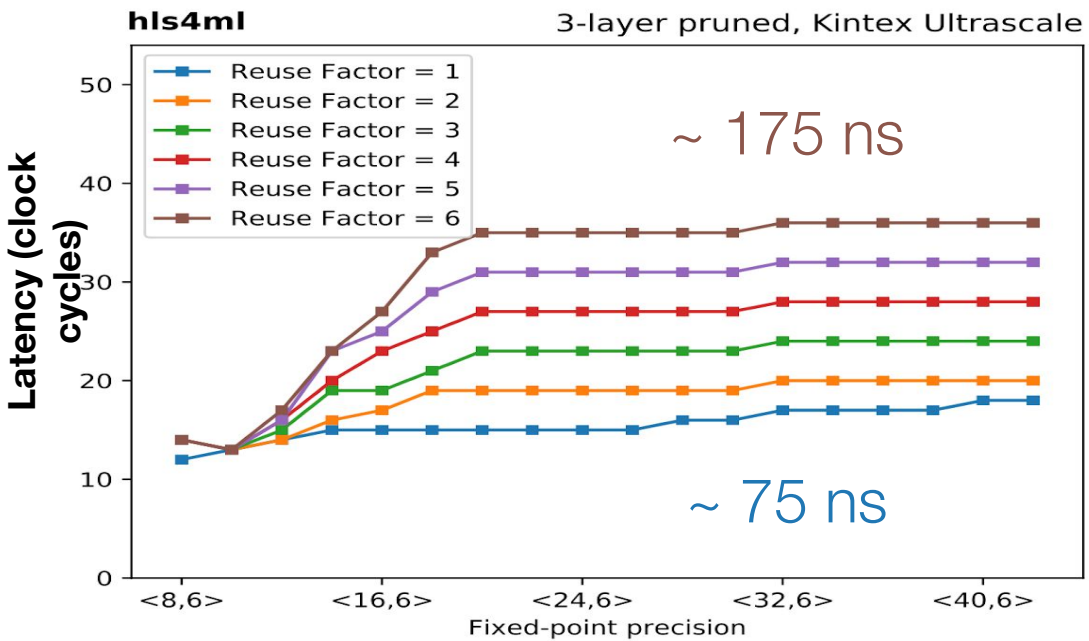
⋮

Longer latency

Parallelization: Timing

Latency of layer m

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



Each mult. used 6x

Each mult. used 3x

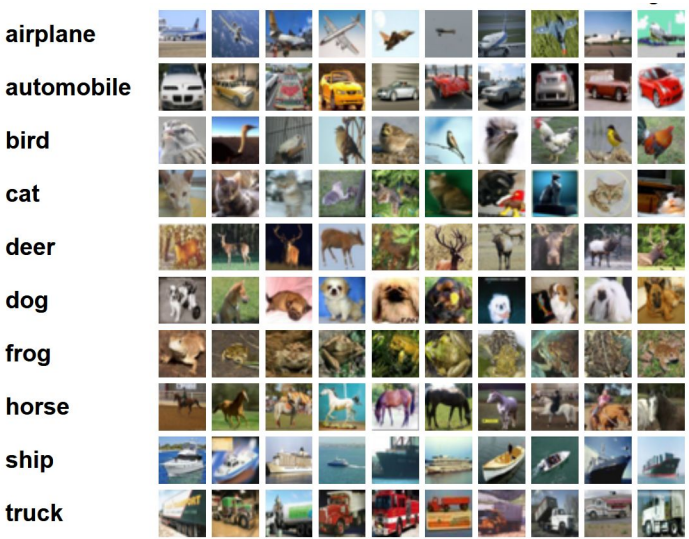
Fully parallel
Each mult. used 1x

Longer latency

More resources

Dataset - CIFAR 10

- The CIFAR-10 and CIFAR-100 are labeled subsets of the 80 million tiny images dataset. They were collected by Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton.
- The CIFAR-10 dataset consists of 60000 32x32 colour images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images.
    - The dataset is divided into five training batches and one test batch, each with 10000 images.
    - The test batch contains exactly 1000 randomly-selected images from each class.
    - The training batches contain the remaining images in random order, but some training batches may contain more images from one class than another.
    - Between them, the training batches contain exactly 5000 images from each class.
    - The classes are completely mutually exclusive. There is no overlap between automobiles and trucks. "Automobile" includes sedans, SUVs, things of that sort. "Truck" includes only big trucks. Neither includes pickup trucks.



Example images of each class

Dataset, images, and text from:
https://www.cs.toronto.edu/~kriz/cifar.html

Dataset - Pokémon

- Training Dataset, ~23k images for 151 pokemon
  - using 0.25 Train/Val split during training
  - ~2600 images for 10 class set
    - Bulbasaur, Charmander, Eevee, Gengar, Jigglypuff, Mewtwo, Onix, Pikachu, Snorlax, Squirtle
  - https://www.kaggle.com/datasets/unexpectedscepticism/11945-pokemon-from-first-gen
  - https://www.kaggle.com/datasets/thedagger/pokemon-generation-one
  - https://www.kaggle.com/datasets/lantian773030/pokemonclassification
- Test Dataset, ~525 Images for 10 classes, downloaded pokemon card images from online, light processing (cropping)
  - Code to reproduce here: https://github.com/ben-hawks/pokedex_scraper

Example images of each (test) class