Hands-on set-up

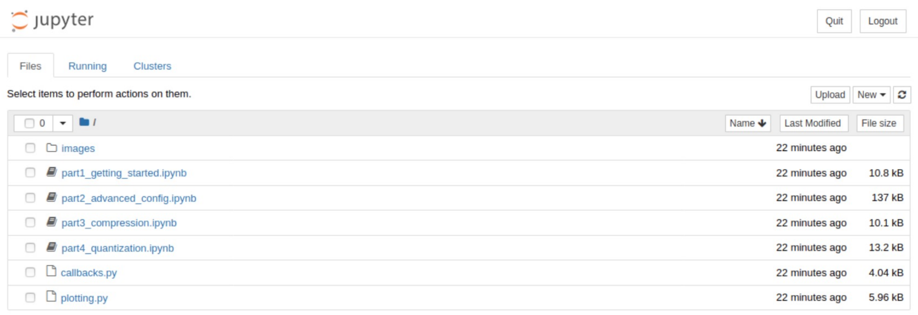
- The interactive part is done using Python notebooks

- Open http://35.194.40.33/ in your web browser
  - Authenticate with your GitHub account (login if necessary)
  - If you haven't shared your GitHub username already, please fill in https://forms.gle/EfvrXykKCMydTvnX9, so that access can be granted

- If you have Vivado install yourself, you might prefer to work locally, see 'conda' section at: https://github.com/fastmachinelearning/hls4ml-tutorial

You should see something like this

if everything worked



Select items to perform actions on them.

Name	Last Modified	File size
images	22 minutes ago	
part1_getting_started.ipynb	22 minutes ago	10.8 kB
part2_advanced_config.ipynb	22 minutes ago	137 kB
part3_compression.ipynb	22 minutes ago	10.1 kB
part4_quantization.ipynb	22 minutes ago	13.2 kB
callbacks.py	22 minutes ago	4.04 kB
plotting.py	22 minutes ago	5.96 kB

hls4ml

tutorial

SMARTHEP Edge Machine Learning School

Benjamin Ramhorst et al. for the **hls4ml** team

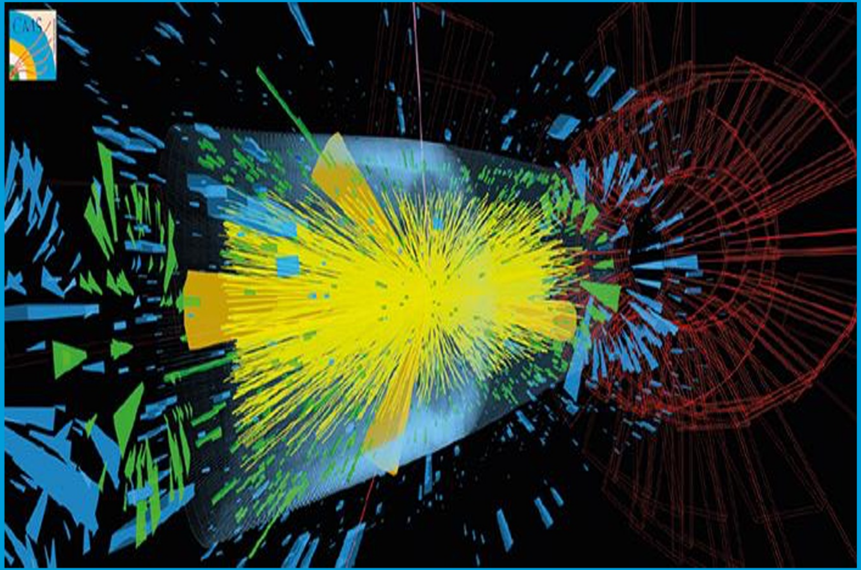
Introduction

- **hls4ml** is a package for translating neural networks to FPGA firmware for inference with extremely low latency on FPGAs

- In this session you will get hands on experience with the **hls4ml** package

- We'll learn how to:
    - Translate high-level models into synthesizable FPGA code
    - Explore the different handles provided by the tool to optimize the inference
    - Make our inference more computationally efficient with quantization

LHC Triggering

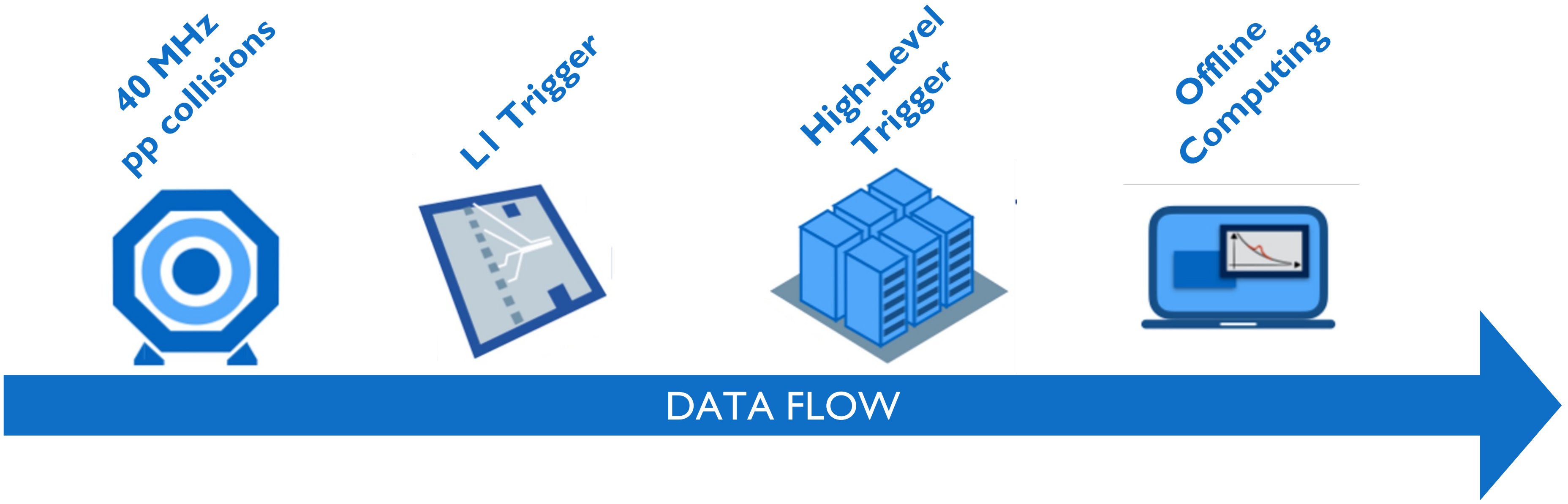


- Extreme collision frequency of **40 MHz** → extreme data rates ~**100 TB/s**
- Most collision "events" don't produce interesting physics

"Triggering" = filter events to reduce data rates to manageable levels

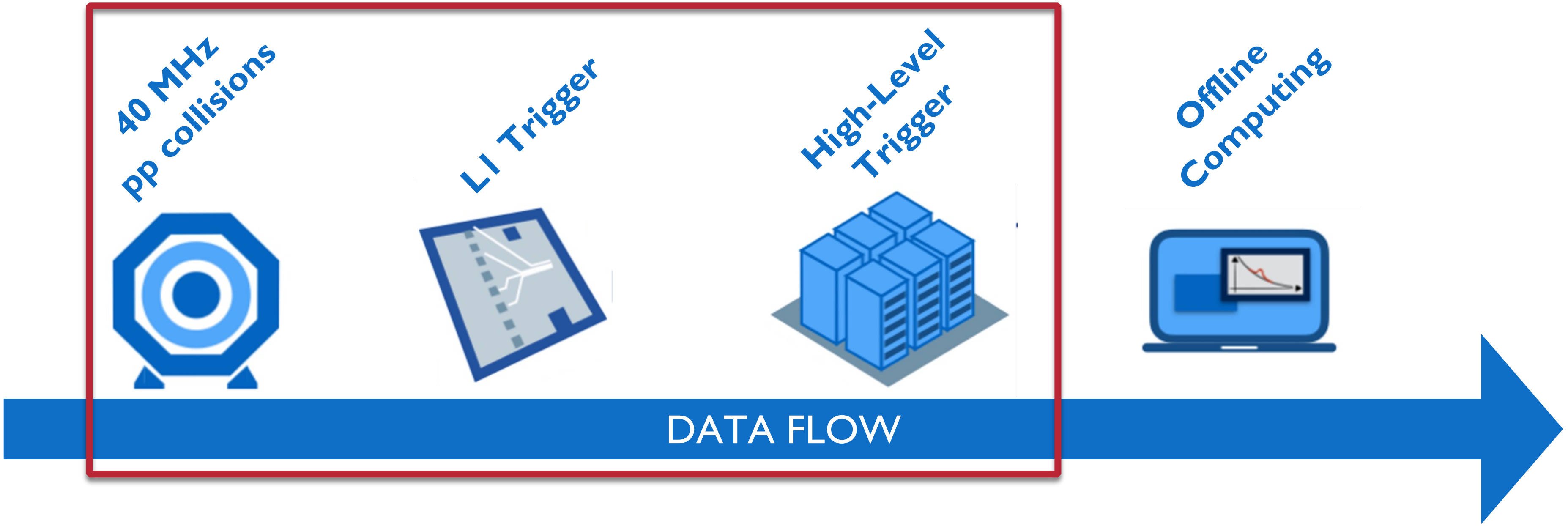
LHC Experiment Data Flow

- L1 trigger: Incoming data rates of **100s TB/s:**



LHC Experiment Data Flow

- Deploy ML algorithms very early, avoiding off-line computation and storage
  - Challenge: Strict latency constraints **~10us**

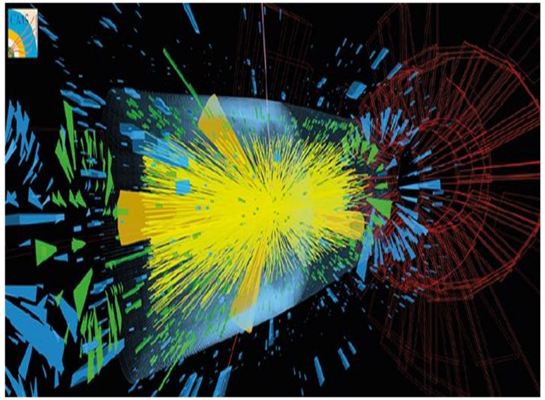


The latency - visualised

~1-3 seconds

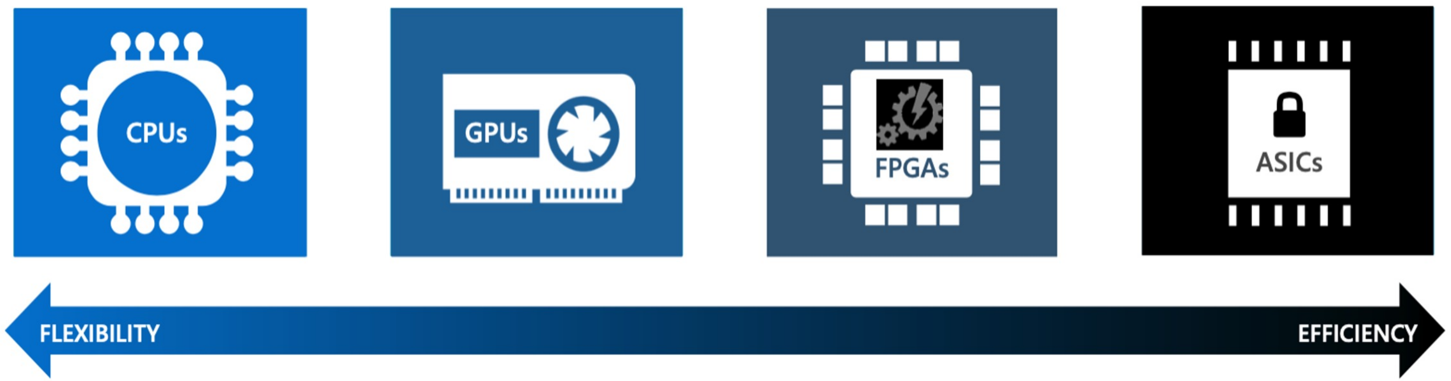
~50ms

~500 ns

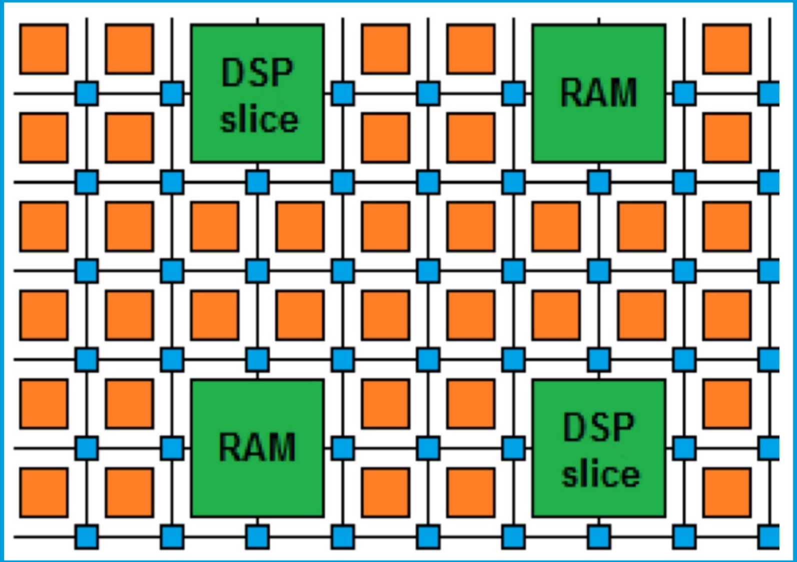


Why FPGAs?

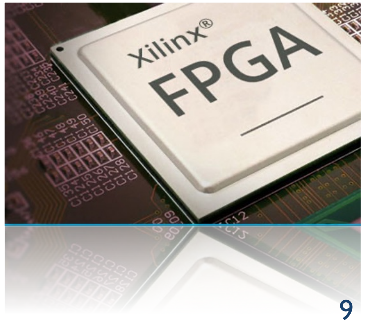
- ❑ Custom hardware acceleration, precisions and memory management
- ❑ Data-flow architecture with no scheduling or control overheads



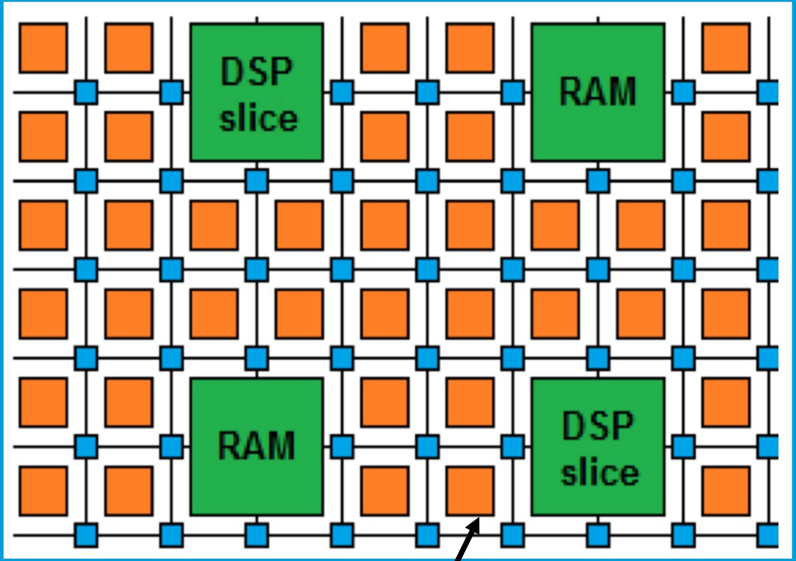
What are FPGAs?



- Field Programmable Gate Arrays are **reprogrammable integrated circuits**
- Contain many different **building blocks** (‘resources’) which are **connected together as you desire**



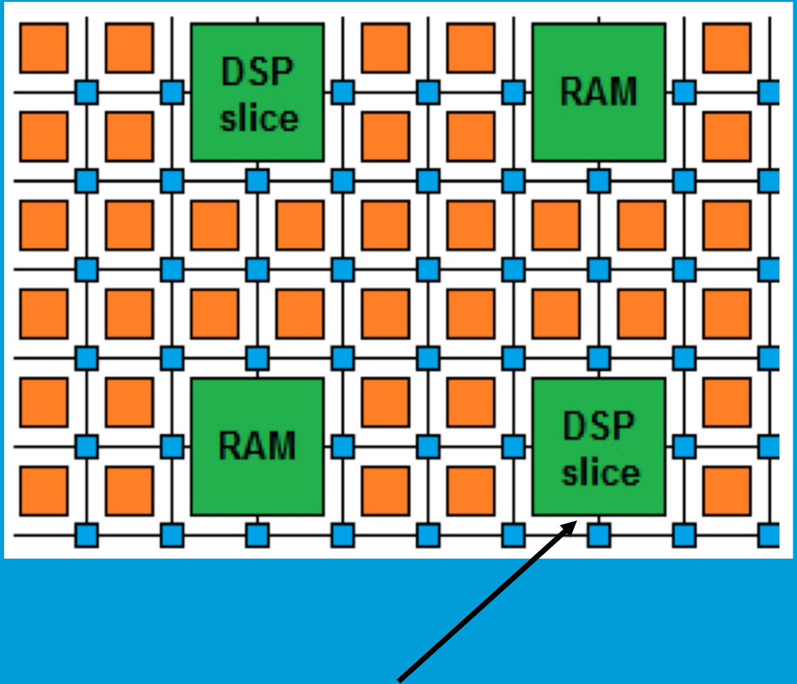
What are FPGAs?



Logic cell

- **Logic cells (Look–up Tables)** perform arbitrary functions on small bit width inputs
- These can be used for Boolean operations, arithmetic, small memories
- **Flip-Flops (registers)** data in time with the clock pulse

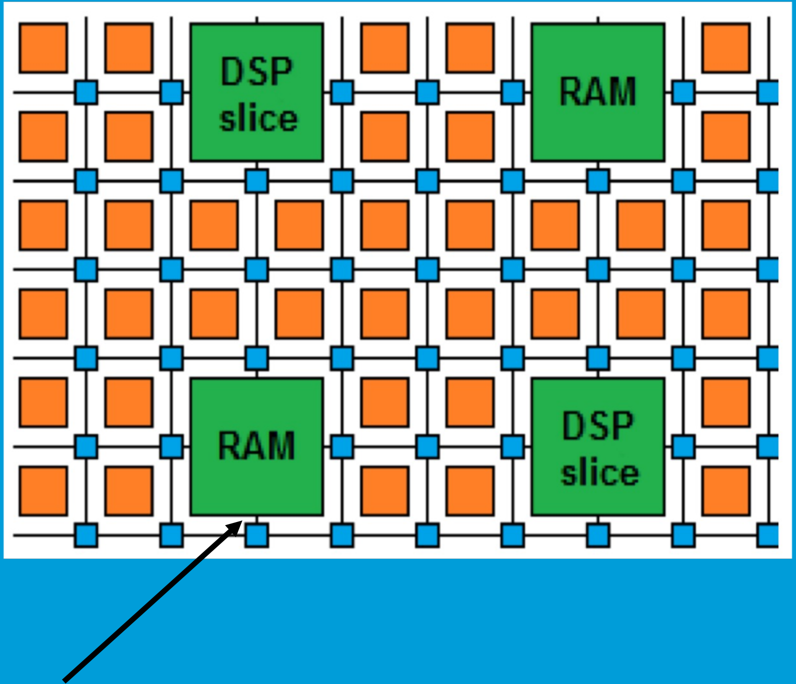
What are FPGAs?



DSPs

- **DSPs (Digital Signal Processors)** are specialized units for multiplication and arithmetic
- Faster and more efficient than using LUTs for these types of operations
- And for neural networks, DSPs are often the most scarce

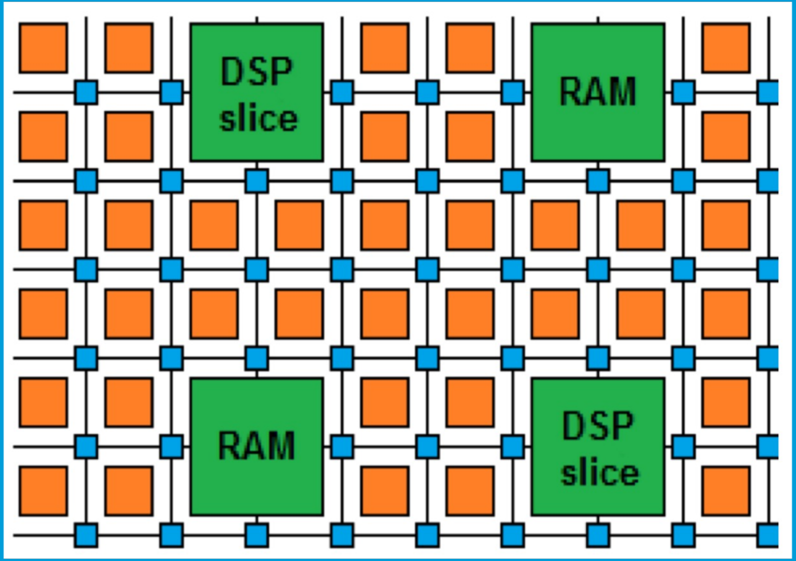
What are FPGAs?



RAM

- **BRAMs** are small, fast memories
  - Access data in one clock cycle
- A big FPGA has nearly 40MB of BRAM, chained together as needed (bandwidth)
  - Even suitable for "larger" models, such as ResNet
- Recent accelerator cards also come equipped with off-chip HBM memory (up to 800 GBps)

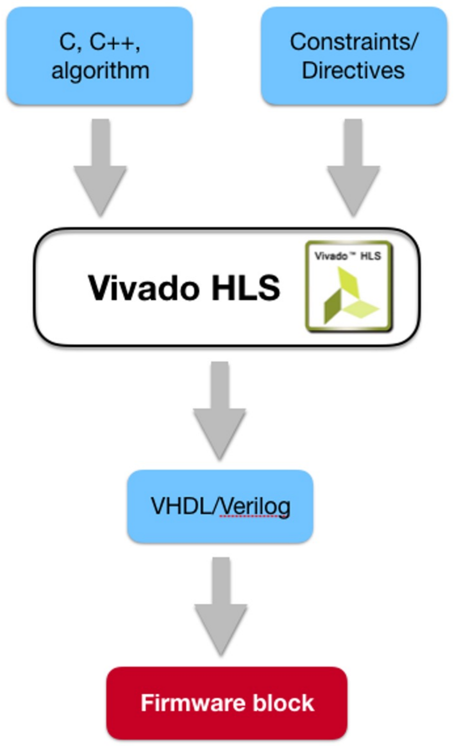
What are FPGAs?



- 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, 100G Ethernet, InfiniBand
- **Low power per Op** (relative to CPU/GPU)

How are FPGAs programmed?

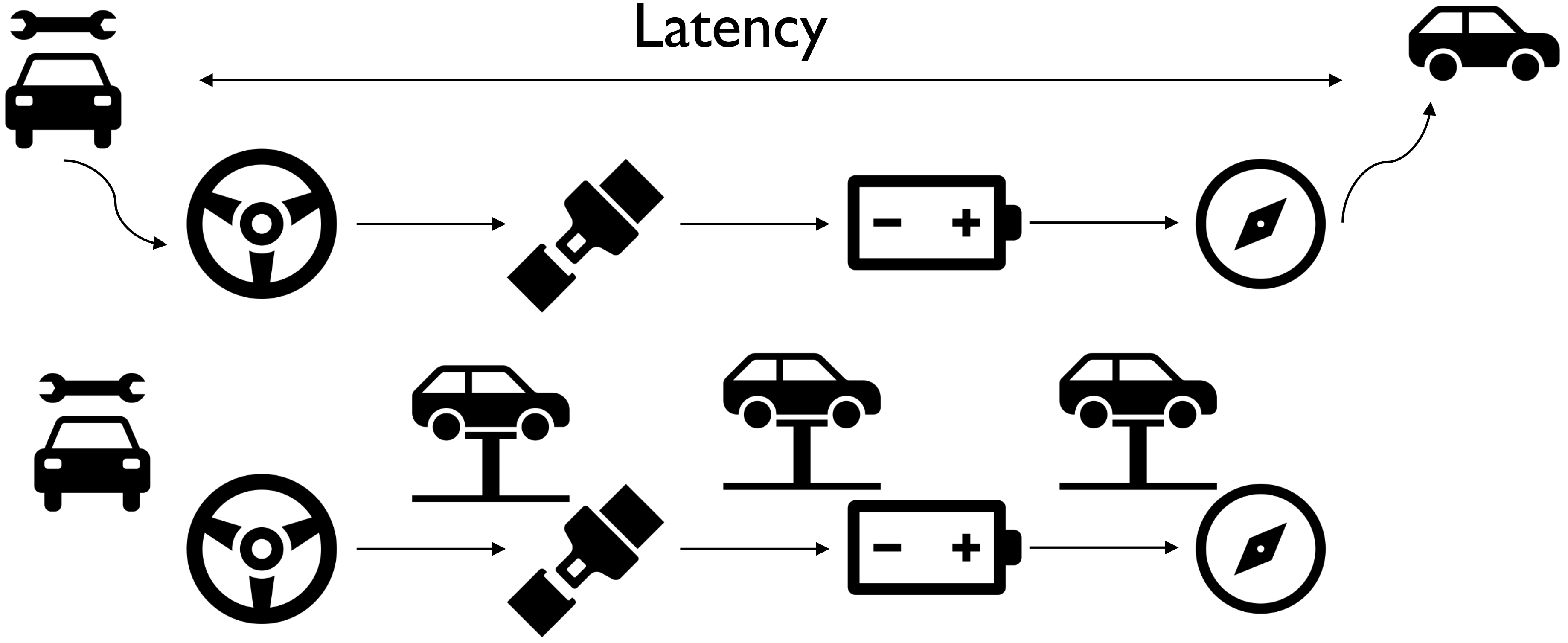
- **Hardware Description Languages**
  - HDLs are programming languages which describe electronic circuits
- **High Level Synthesis**
  - Compile from C/C++ to VHDL
  - Pre-processor directives and constraints used to optimize the design
  - Drastic decrease in firmware development time!
- Today we'll use **Xilinx Vivado HLS**



Jargon

- **LUT** - Look Up Table aka ‘logic’ - generic boolean 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, such as Verilog or VHDL for describing circuits
- **Latency** - time between starting processing and receiving the result
- **II** - Initiation Interval - time from accepting first input to accepting next input (visualize, cars on a production line)

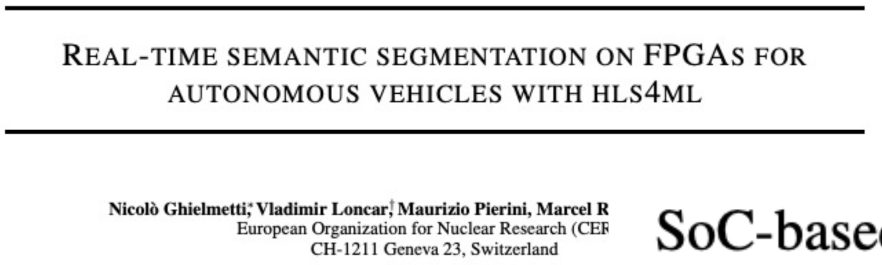
Latency vs initiation interval



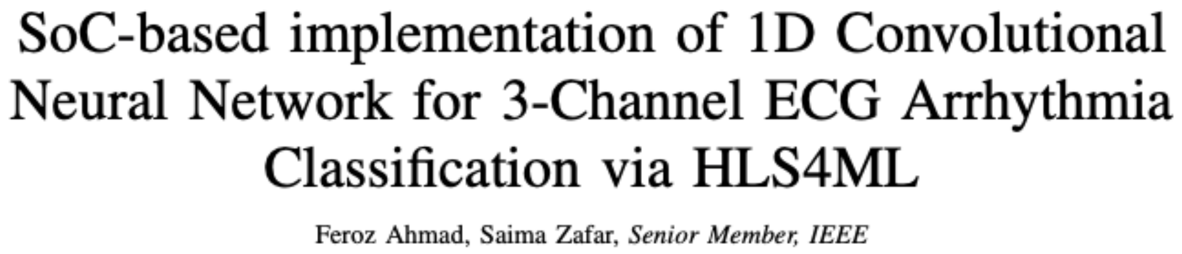
What is hls4ml today?

- A generic framework for FPGA acceleration of neural networks:
    - **Front-end agnostic**: Keras, PyTorch, (Q)ONNX
    - **Back-end agnostic:** Vivado HLS, Vitis HLS, Intel HLS, oneAPI etc.
    - **Many supported layers:** Dense, Conv, Recurrent, Graph etc.
    - **High configurability:** Tune precision, reuse factor, custom layers etc.
    - **An active, open-source community:** Many collaborators from many different fields and institutions

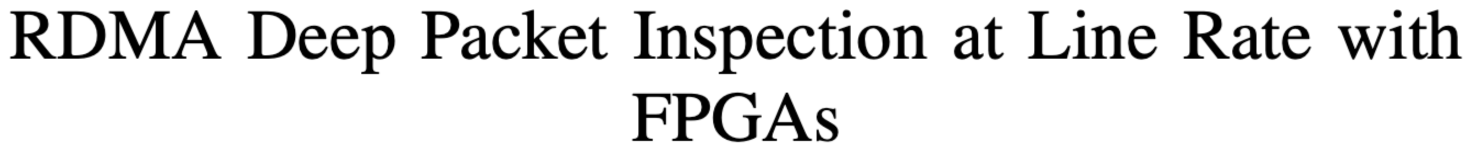
hls4ml use cases


Real-time semantic segmentation on FPGAs for autonomous vehicles with hls4ml

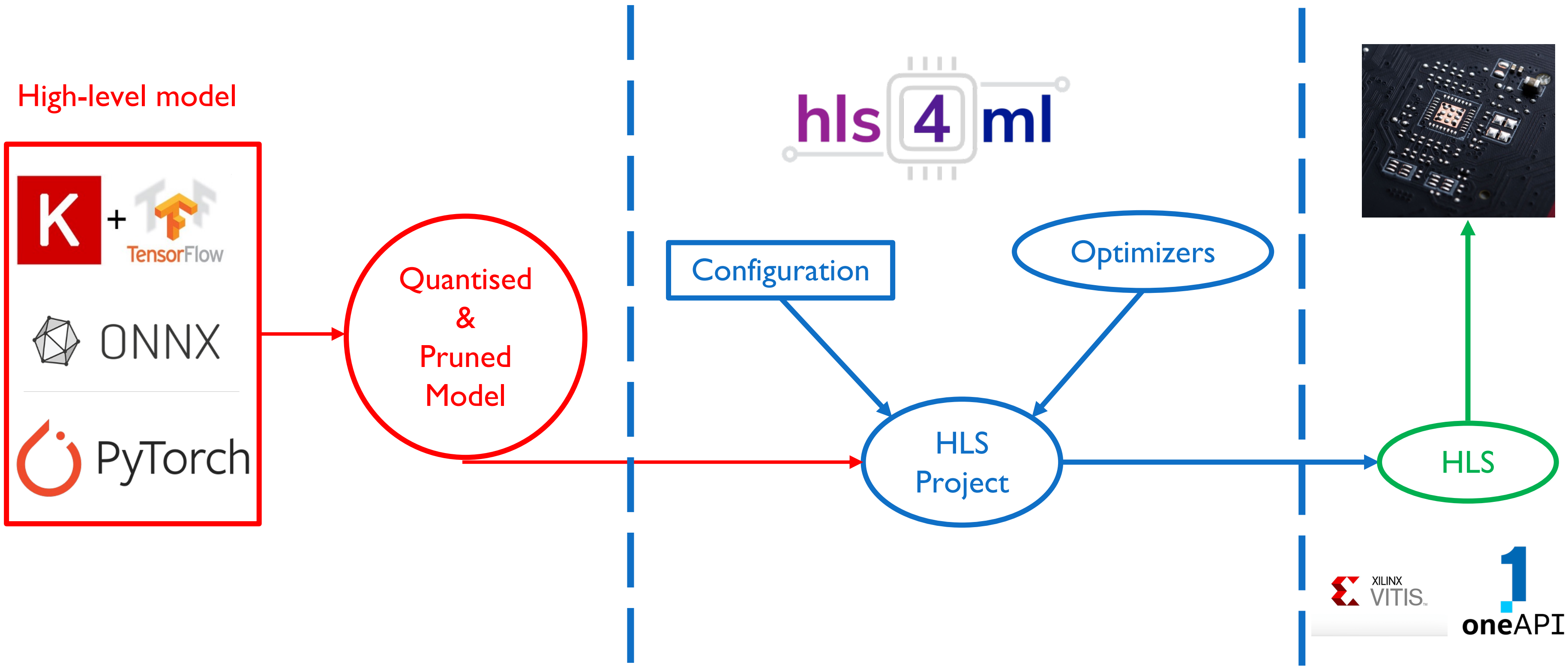
Nicolò Ghielmetti,* Vladimir Loncar,† Maurizio Pierini, Marcel R
European Organization for Nuclear Research (CER
CH-1211 Geneva 23, Switzerland


SoC-based implementation of 1D Convolutional Neural Network for 3-Channel ECG Arrhythmia Classification via HLS4ML

Feroz Ahmad, Saima Zafar, *Senior Member, IEEE*


RDMA Deep Packet Inspection at Line Rate with FPGAs

high level synthesis for machine learning



Today's tutorial

- **Part 1:**
  - Get started with **hls4ml**: train a basic model and run the conversion, simulation & C-synthesis steps
- **Part 2:**
  - Learn how to tune inference performance with quantization & ReuseFactor
- **Part 3:**
  - Train using QKeras "quantization-aware training" and study impact on FPGA metrics

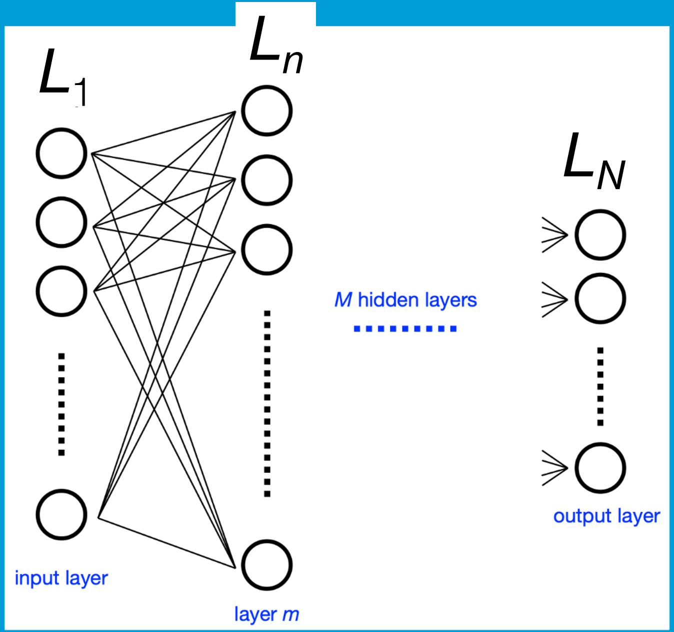
What's not covered today?

- **Boosted decision trees:** implemented in a companion package to hls4ml
  - https://github.com/thesps/conifer - **see the talk tomorrow!**
- **High-granularity quantisation:** heterogenous layer quantisation **(covered yesterday)**
- **Convolutional neural networks**
  - Notebooks available on GitHub, however, synthesis takes long
- **What comes after hls4ml…** you would need to integrate the 'IP core' into a larger design
  - For a custom board, you'd need to do this by hand (e.g. CMS L1 Trigger)
  - For more off-the-shelf boards, integration with system-on-chip or host CPU can be more straightforward, using tools such as XRT

hls4ml

Part 1: Model Conversion

Neural network inference



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

logic cells OR precomputed and stored in BRAMs

DSPs

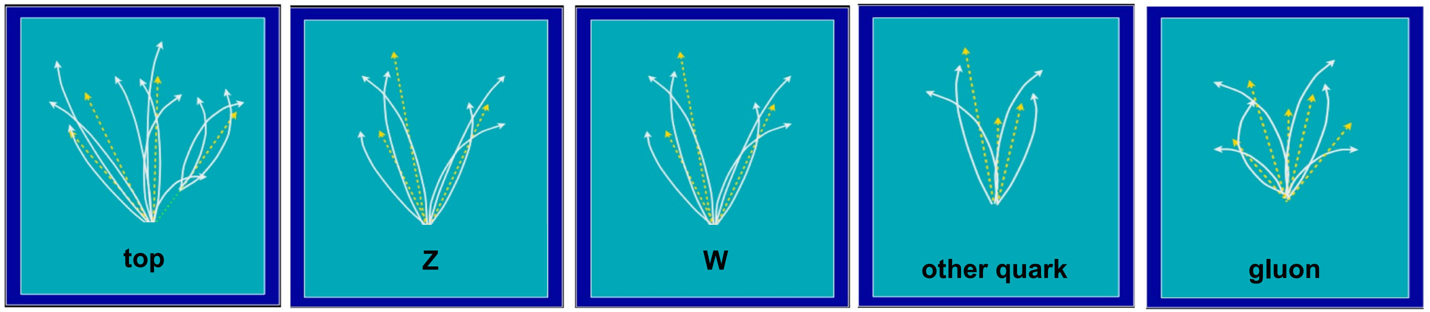
logic cells

How many resources? DSPs, LUTs, FFs?

Does the model fit in the latency requirements?

Physics use 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



t→bW→bqq 3-prong jet

Z→qq 2-prong jet

W→qq 2-prong jet

Reconstructed as one massive jet with substructure

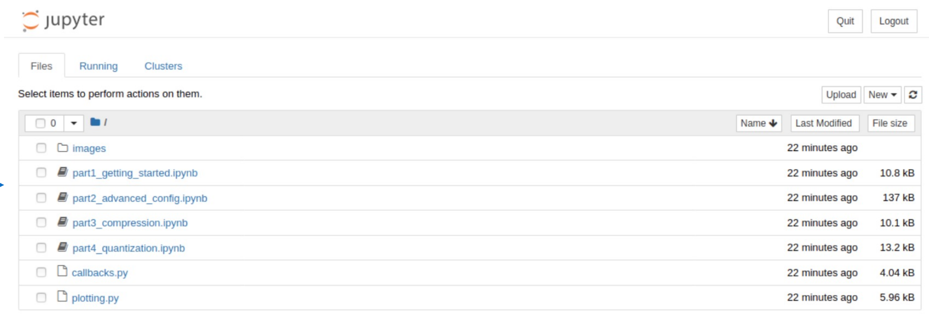
q/g background no substructure and/or mass ~ 0

Hands-on set-up

- ❑ The interactive part is done using Python notebooks
- ❑ Open http://35.194.40.33/ in your web browser
  - ❑ Authenticate with your GitHub account (login if necessary)
  - ❑ If you haven't shared your GitHub username already, please fill in https://forms.gle/EfvrXykKCMydTvnX9, so that access can be granted
- ❑ If you have Vivado install yourself, you might prefer to work locally, see 'conda' section at: https://github.com/fastmachinelearning/hls4ml-tutorial

You should something like this

if everything worked



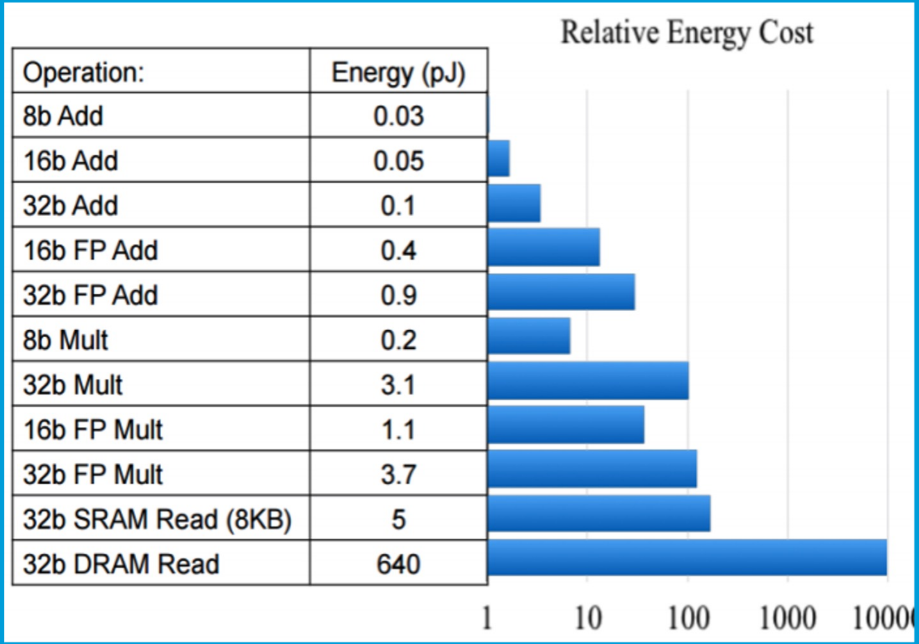
Select items to perform actions on them.

Name	Last Modified	File size
images	22 minutes ago	
part1_getting_started.ipynb	22 minutes ago	10.8 kB
part2_advanced_config.ipynb	22 minutes ago	137 kB
part3_compression.ipynb	22 minutes ago	10.1 kB
part4_quantization.ipynb	22 minutes ago	13.2 kB
callbacks.py	22 minutes ago	4.04 kB
plotting.py	22 minutes ago	5.96 kB

hls4ml

Part 2: Advanced configuration

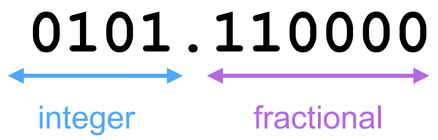
Efficient inference: quantisation



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

[Horowitz @ ISSCC'14]

- ❑ Floating point operations are expensive
- ❑ On FPGAs, we can use fixed-point precision
  - ❑ Implemented using integer logic, so very fast
  - ❑ Acts like "limited-precision" floating-point, so need to ensure sufficient bits

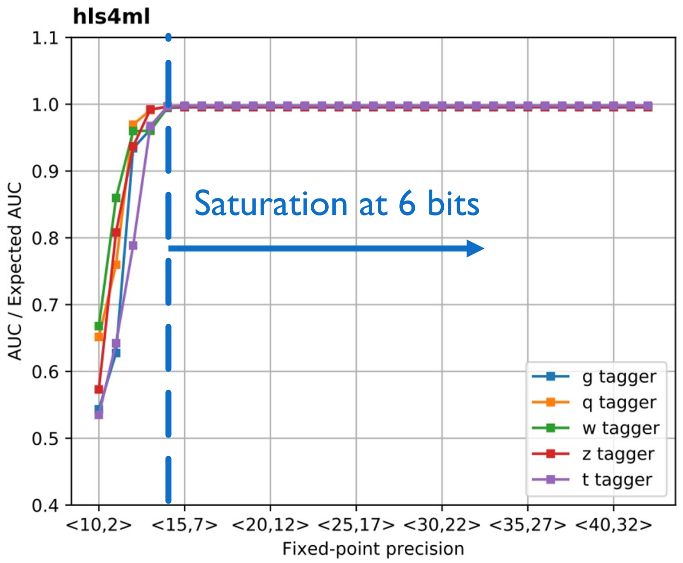


- ❑ Integer value: 1 + 4 = 5
- ❑ Fractional value: ½ + ¼= ¾

Efficient inference: quantisation

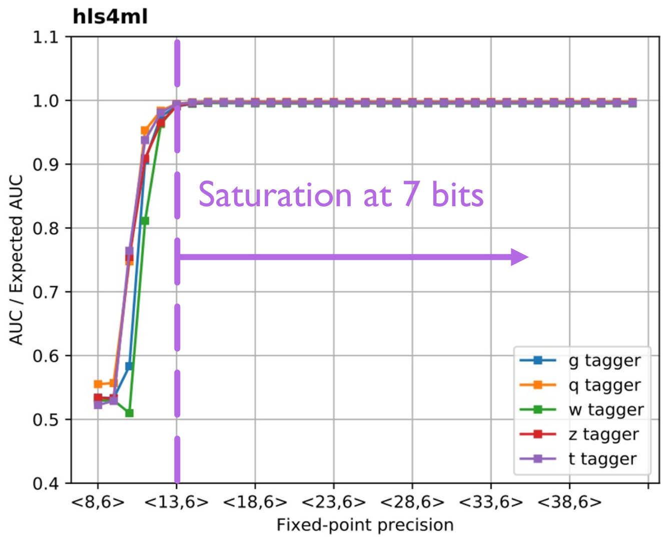
Scan integer bits

Fractional bits fixed to 8



Scan fractional bits

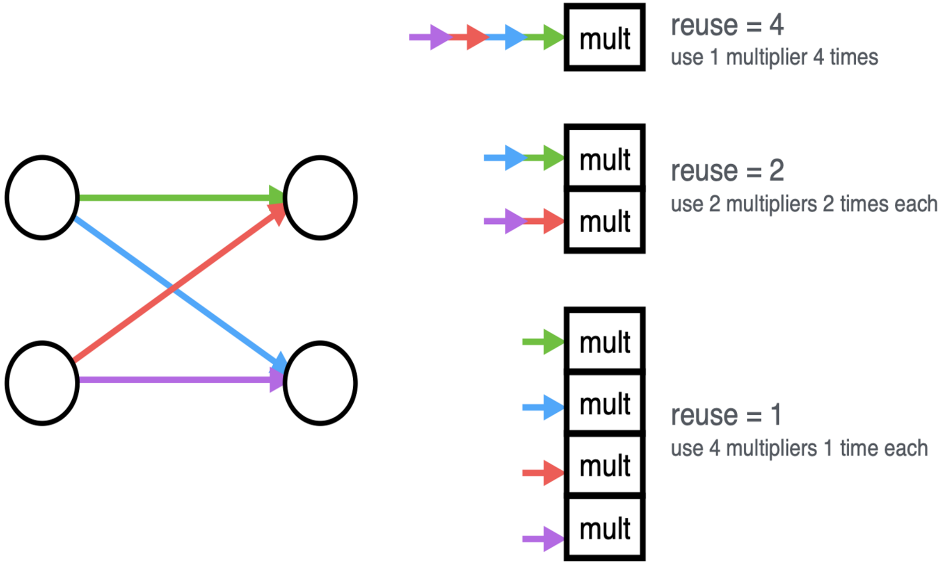
Integer bits fixed to 6



Efficient inference: parallelisation

- 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 re-used to do a computation**



Efficient inference: parallelisation

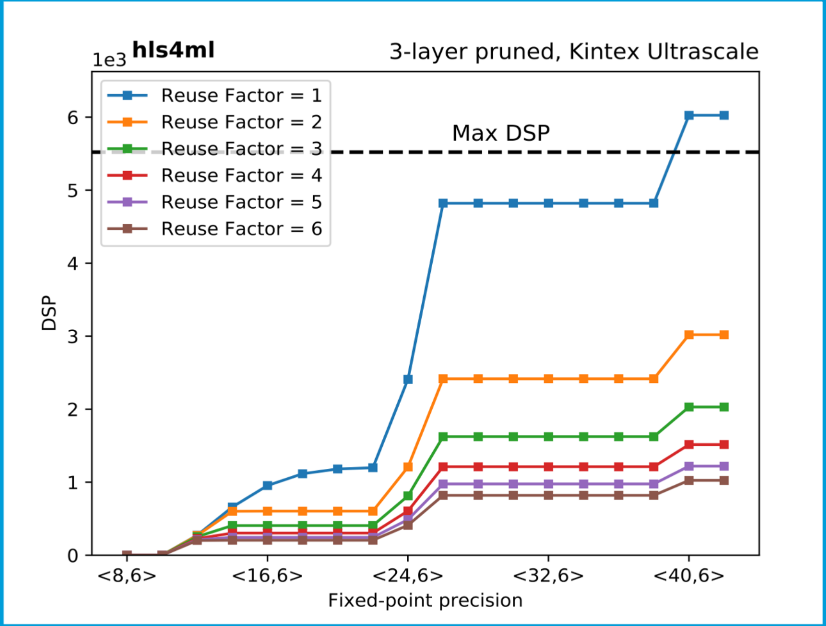
Fully serial

Fewer resources
Lower throughput
Higher latency

Fully parallel

More resources
Higher throughput
Lower latency

Parallelisation: DSP usage



More resources

Fully parallel
Each mult. used 1x

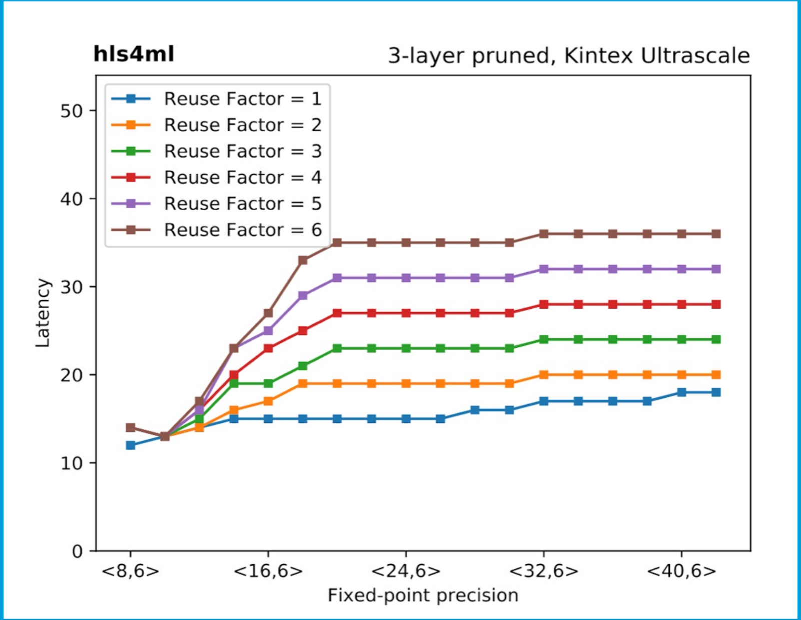
Each mult. used 2x

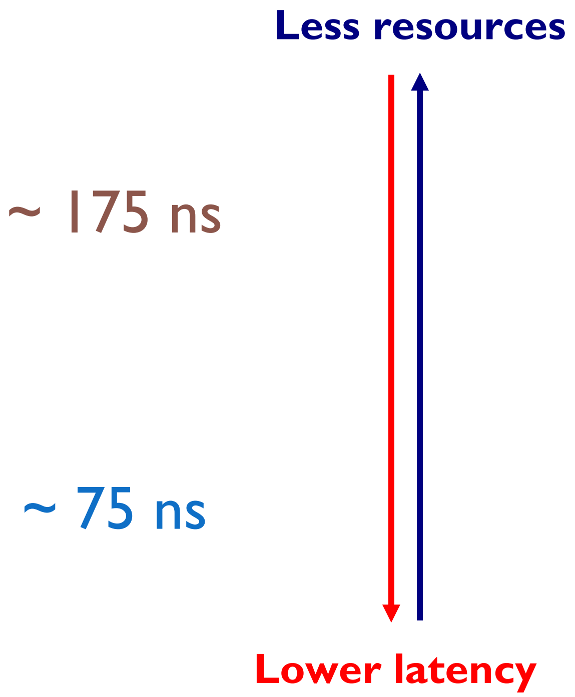
Each mult. used 3x

⋮

Longer latency

Parallelisation: Latency





hls4ml

Part 3: Quantisation

Quantisation-aware training

- **hls4ml** allows us to use different data types everywhere, we saw how to tune that in Part 2
- Now, we will also try quantization-aware training with QKeras
- Quantisation-aware training enables training models with very low precision:
  - Out-performs post-training quantisation significantly
  - At a high level, it performs the forward pass with reduced precision and the backward pass in floating point precision
  - Possible to achieve very precisions (binary and ternary models)

QKeras

- QKeras is a library to train models with quantization during training
  - Developed & maintained by Google
- Easy to use, drop-in replacements for Keras layers
  - e.g. Dense → QDense, Conv2D → QConv2D
  - Use 'quantizers' to specify how many bits to use where
  - Can achieve good performance with very few bits
- Stable support for QKeras-trained models to **hls4ml**
  - The number of bits used in training is automatically parsed for conversion & inference

Summary

- ❑ After this session you've gained some hands-on experience with **hls4ml**
  - ❑ Translated neural networks to FPGA firmware, run simulation and synthesis
  - ❑ Tuned network inference performance with precision and ReuseFactor
  - ❑ Trained a quantized model using QKeras, and use the same model for inference with hls4ml
- ❑ The tutorials to run locally are at: https://github.com/fastmachinelearning/hls4ml-tutorial
- ❑ Use **hls4ml** locally: `pip install hls4ml`

Questions?

Special thanks to the Fast Machine Learning Community for the on-going efforts in hls4ml development and its accompanying tutorials. The materials used today are based on: https://github.com/fastmachinelearning/hls4ml-tutorial and were used and edited with permission of the authors.