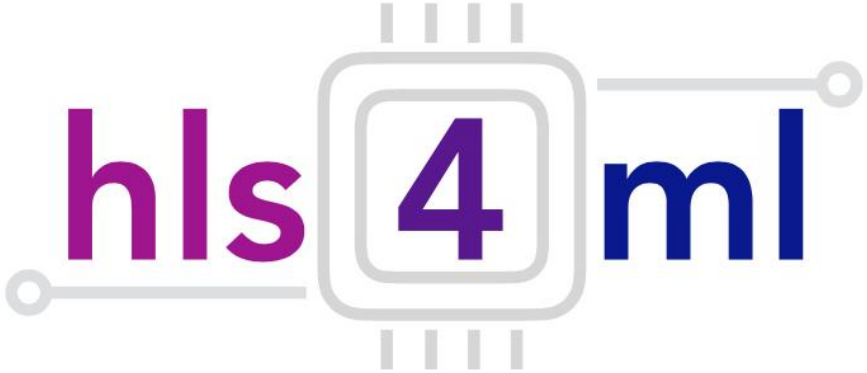
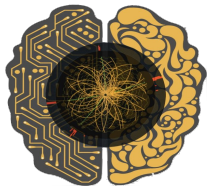


hls4ml: low latency neural network inference on FPGAs

Vladimir Loncar (MIT)
For the FastML team
fastmachinelearning.org



The Large Hadron Collider

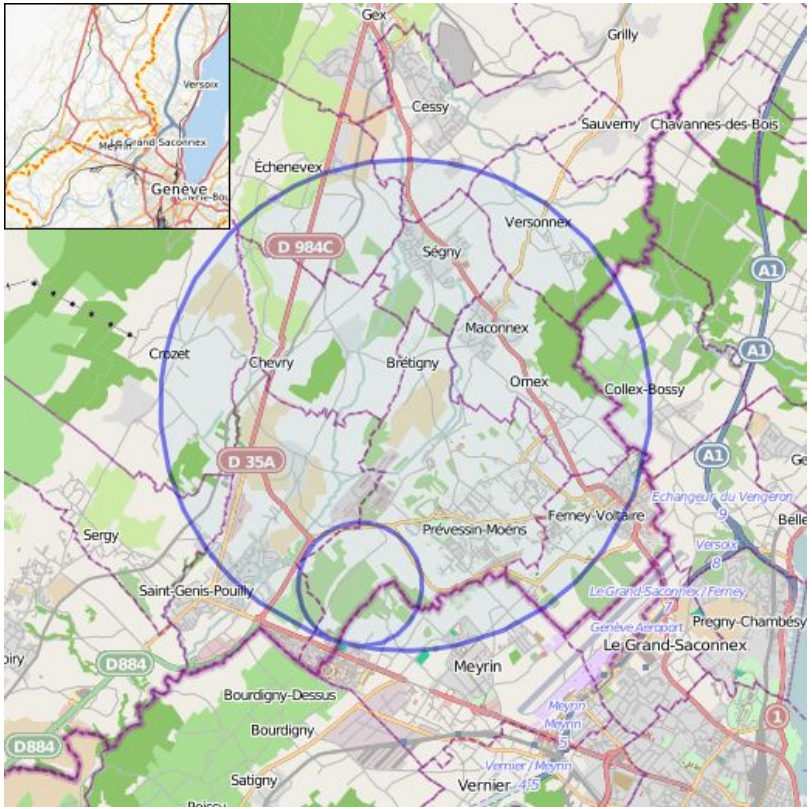
27 km circumference accelerator at CERN on the border of France and Switzerland near Geneva

Accelerates protons close to the speed of light, and collides them at 14 TeV centre of mass energy

Searching for new fundamental physics of the universe!

Collisions happen at 4 points where there are detectors

- We work on one of these: the **CMS** experiment

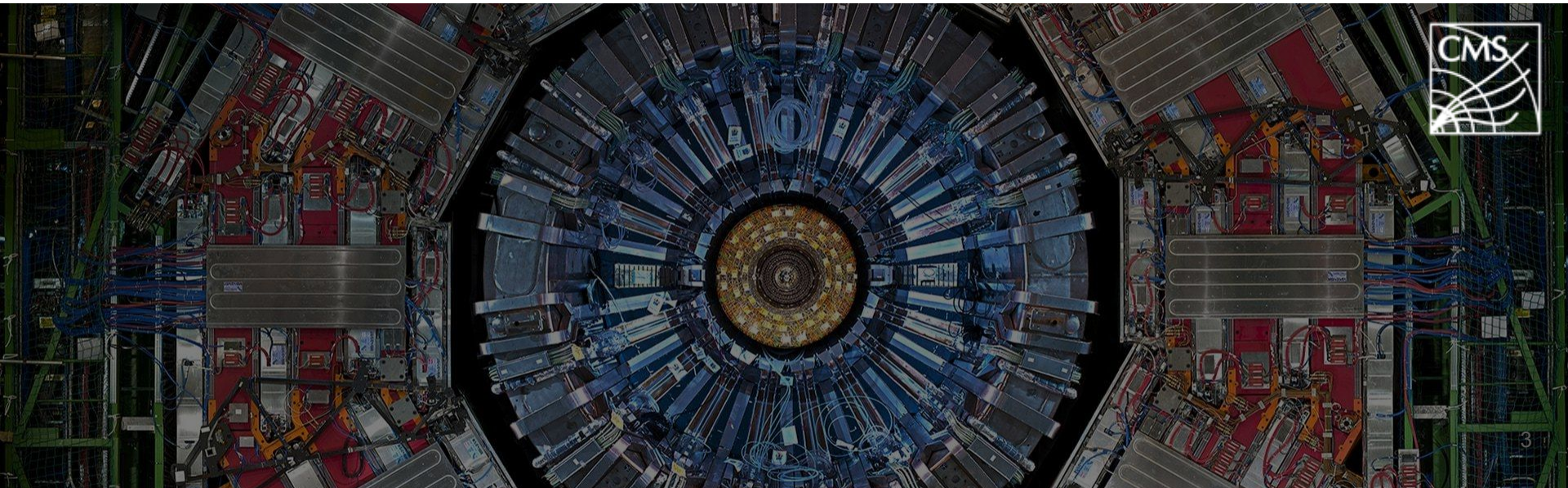


Data acquisition in LHC

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



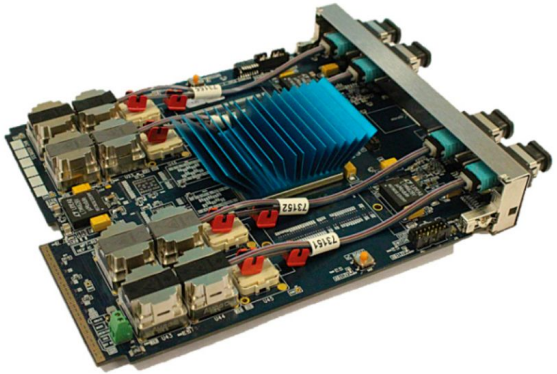
Trigger hardware

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

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

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

- Field-programmable gate arrays (FPGAs)
- Perfect because:
  - Resource parallelism ➔ **low latency**
  - Pipeline parallelism ➔ **high throughput**



Designing low-latency ML processing pipelines

The design of low latency algorithms differs from other ML implementations

- We must tailor specific processing hardware to the task at hand to increase the overall algorithm performance
- Processor design + the design of algorithms = hardware ML co-design

However, designing hardware is challenging

- Designing efficient parallel algorithms for programmable hardware is even more challenging
- Usually done by domain experts using hardware description languages (HDLs)

➔ **High-Level Synthesis (HLS)**

High-Level Synthesis

An automated design process that takes functional description (usually in C/C++-like language) as input and produces register-transfer level (RTL) abstraction expressed in HDL

- No need to manually write HDL code
- C++ is not ideal language for hardware design, as it lacks definitions of concurrency and timing
- HLS tools extend the specification of the language with compiler directives aimed at guiding the conversion to RTL

HLS tools have advanced significantly in recent years (study), making them a viable option for creating ML tools

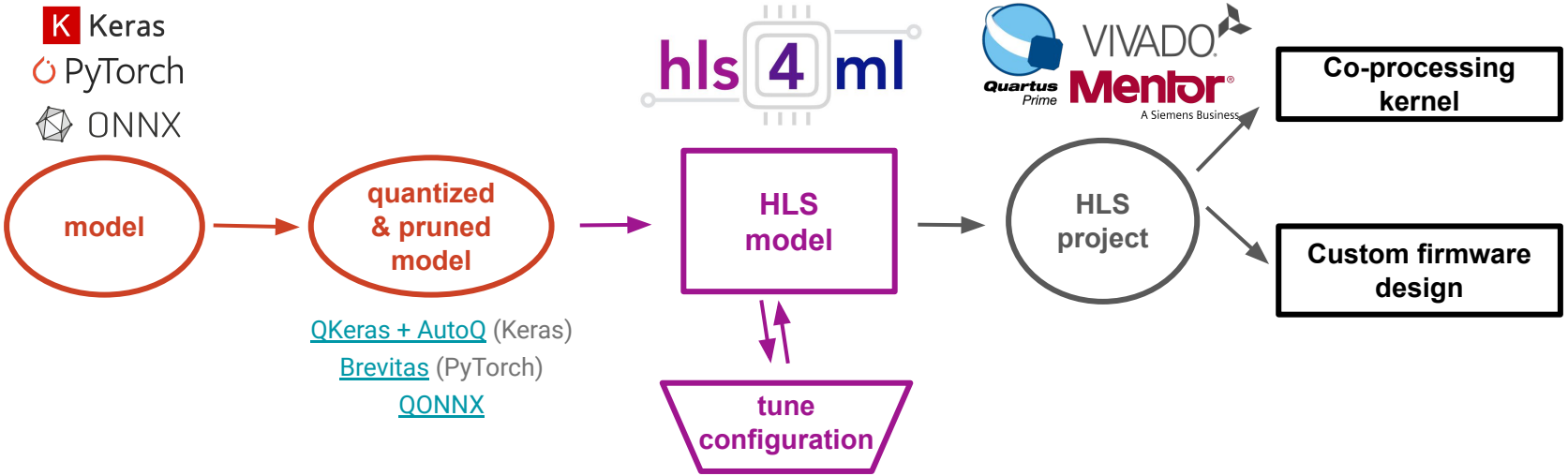
Using HLS we created a tool for converting DL models into high-performance hardware definition

➔ High-Level Synthesis for Machine Learning, **hls4ml**

high level synthesis for machine learning

hls4ml - User-friendly tool to automatically build and optimize DL models for FPGAs:

- Python library, `pip install hls4ml`
- Thriving github ecosystem, 1.1k ⭐



Supported neural networks

Supported architectures:

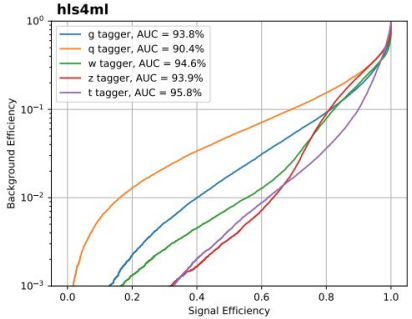
- Fully-connected Networks (MLPs) - arxiv:1804.06913
- Convolutional Neural Networks (CNNs) - arxiv:2101.05108
- Recurrent Neural Networks (RNNs) - arxiv:2207.00559

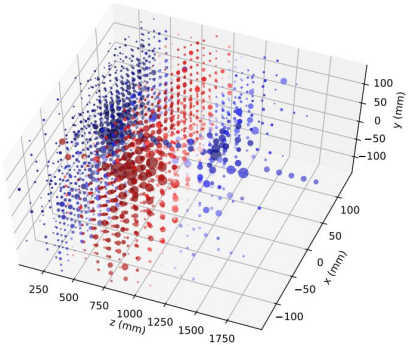
Experimental support:

- Graph NNs
  - GarNet architecture - arxiv:2008.03601
  - JEDI-net - arxiv:2209.14065
- Transformers - arxiv:2402.01047

Other ML

- Symbolic expressions - arxiv:2305.04099





Library features

Distinguishing features:

- Zero-suppressed weights
- Quantization
  - Binary/Ternary layers (computation without using DSPs) - arxiv:2003.06308
  - Google QKeras integration - arxiv:2006.10159
- User-controllable trade-off between resource usage and latency/throughput
  - Tuned via “reuse factor”
- Tunable I/O interface - fully parallel (low latency) or streaming (resource-efficient)

Multiple HLS **“backends”** to support different vendor tools

- AMD/Xilinx Vivado/Vitis HLS
- Intel/Altera Quartus HLS (and oneAPI)
- Siemens/Mentor Catapult HLS

Constraints and limits of hls4ml

- Achievable with **parallel I/O** and low reuse factor
- Requires extensive model compression via pruning and quantization
- Number of model parameters:
  ***O*(100) - *O*(1000)**

- Achievable with **streaming I/O**
- Compression required on lower-end FPGAs, moderate quantization
- Number of model parameters:
  ***O*(1000) - *O*(10 000)**

- Served by vendor tools: **Vitis AI, Intel oneAPI**…
- Based on systolic array design with off-chip storage of weights
- Number of model parameters:
  ***O*(100 000) - *O*(1 million)**

How does hls4ml deploy NNs to FPGAs?

Efficient use of available resources

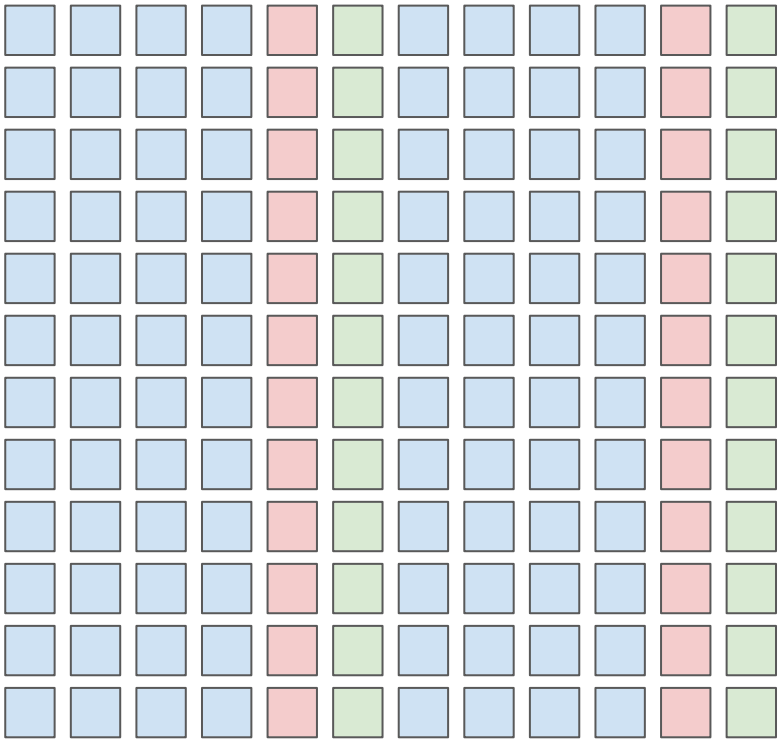
- **hls4ml** performs optimizations on the model to achieve hardware affinity

Everything must be stored on-chip!

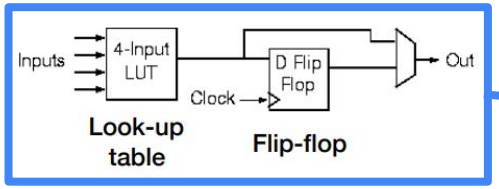
- Better latency/throughput
- Required to meet the constraints of L1 trigger

Parallelize as much as possible

- Pipeline the layers to increase throughput



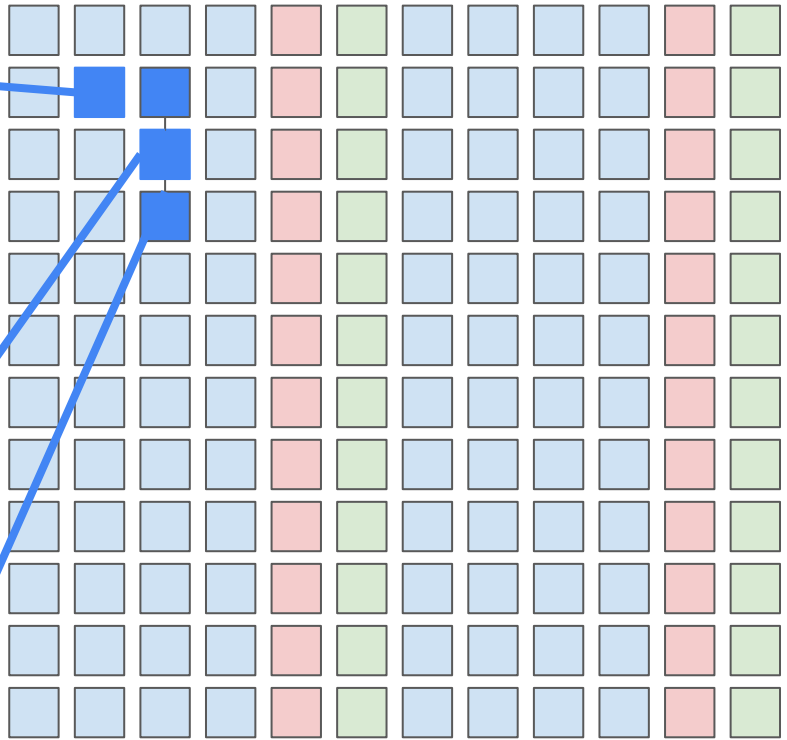
FPGA architecture: Basic Elements



Configured to perform any 1-bit operation

Wider custom operations are implemented by interconnecting Basic Elements

Basic Elements are surrounded with a flexible interconnect

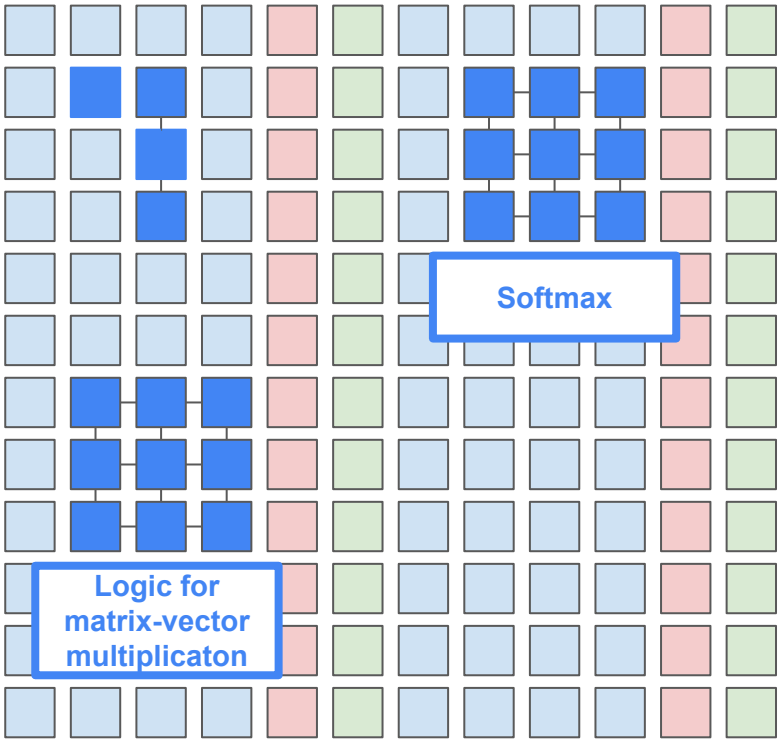


FPGA architecture: Basic Elements

We combine basic elements to build the ingredients of the neural network

For example:

- Matrix-vector multiplication
- Various math functions
- Convolution
- Activations
- …



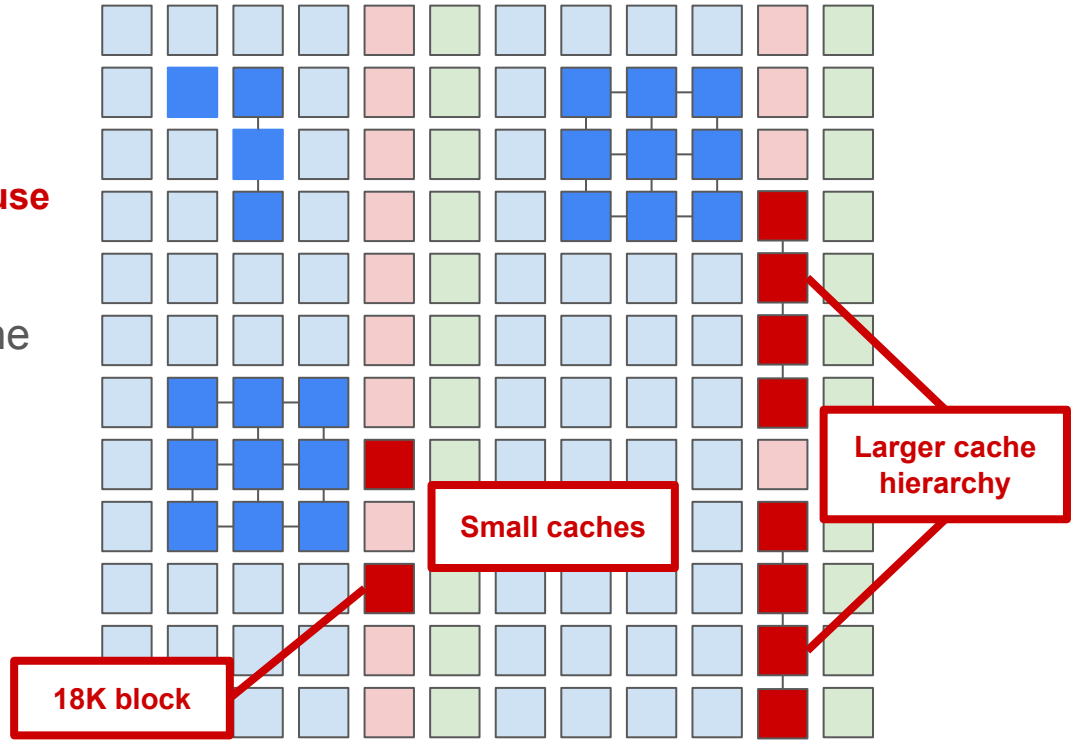
FPGA architecture: Memory blocks

On-chip memory blocks provide fast access to data

- **Due to latency constraints, we can't use off-chip memory**

Can be configured and grouped using the interconnect to create various cache architectures

- We use memory blocks to store the (immutable) weights of the network



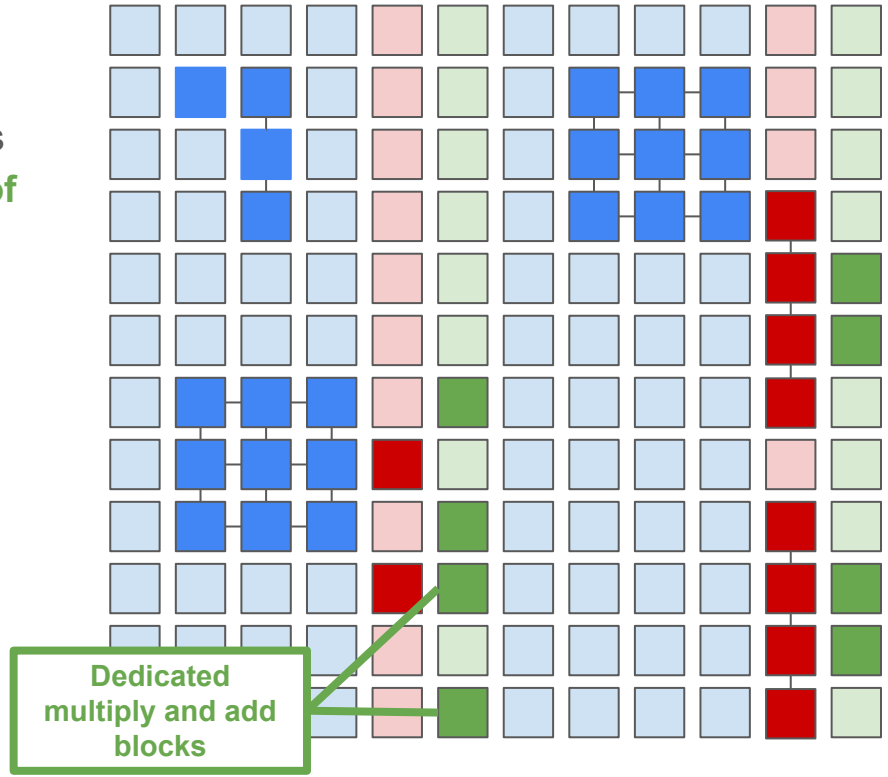
FPGA architecture: Multiply/Add blocks

Digital Signal Processing (DSP) Blocks

- Up to 12 000 blocks in high-end FPGAs
- **Gives us the theoretical upper limit of parallel operations per clock cycle**

Support variable precision fixed-point multipliers

- 27 × 18 two's complement multiplier
- Support for rounding/saturation
- Can emulate floating-point operations



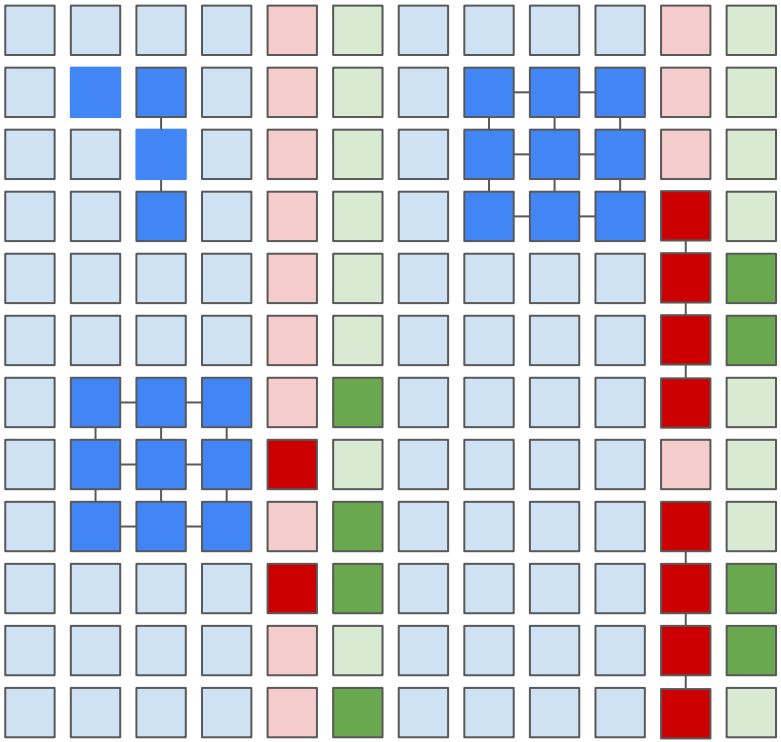
How to meet the requirements of LHC trigger?

Efficient network design via model compression:

- Pruning - removing weights with low magnitude
- Quantization - using fewer bits
- Knowledge distillation - producing smaller models

Tuning I/O and parallelism to meet latency demands

- Reuse factor - controls the amount of unrolling
- Array or streaming I/O



Efficient network design: Pruning & Quantization

Pruning weights to zero allows us to remove them from the computation

- Directly saves memory and multiplier resources

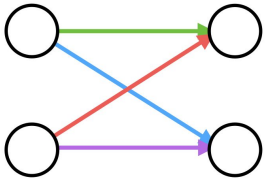
Quantization

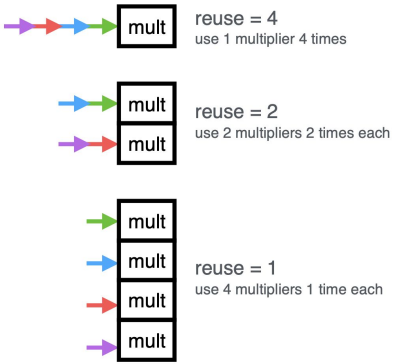
- Using fewer bits for computation saves us logic, memory and multiplier resources
- For best results, we need to use quantization-aware training (QAT)
- Developed **QKeras** for QAT - a collaboration with **Google**, **QONNX** with **AMD** **Xilinx**
- More about quantization in a talk tomorrow

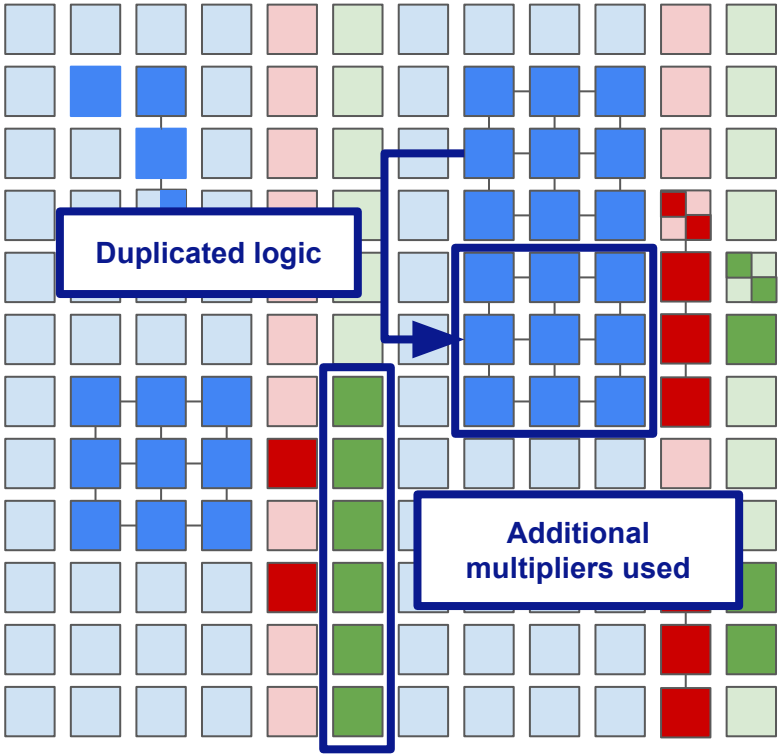
Efficient network design: Parallelism

Trade-off between latency and resource usage determined by the parallelization of the calculations in each layer

- Configure the **“reuse factor”** = number of times a multiplier is used in a computation
- We aim for sub-1μs latency (e.g., <200 cycles @ 200MHz)



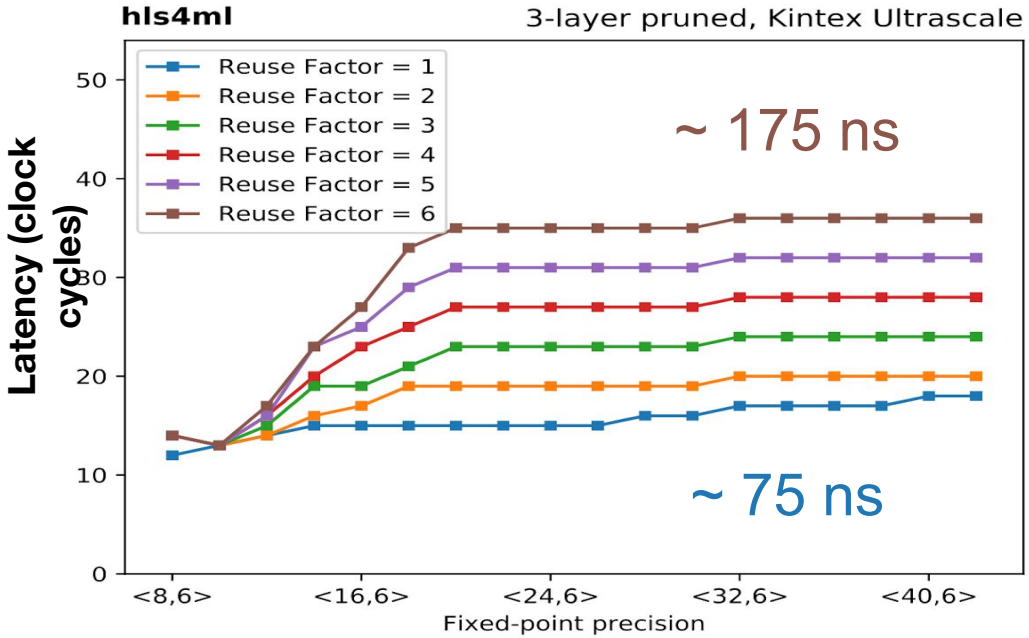




Parallelization: Timing

Latency of layer m

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



Longer latency

Each mult. used 6x

⋮

Each mult. used 3x

⋮

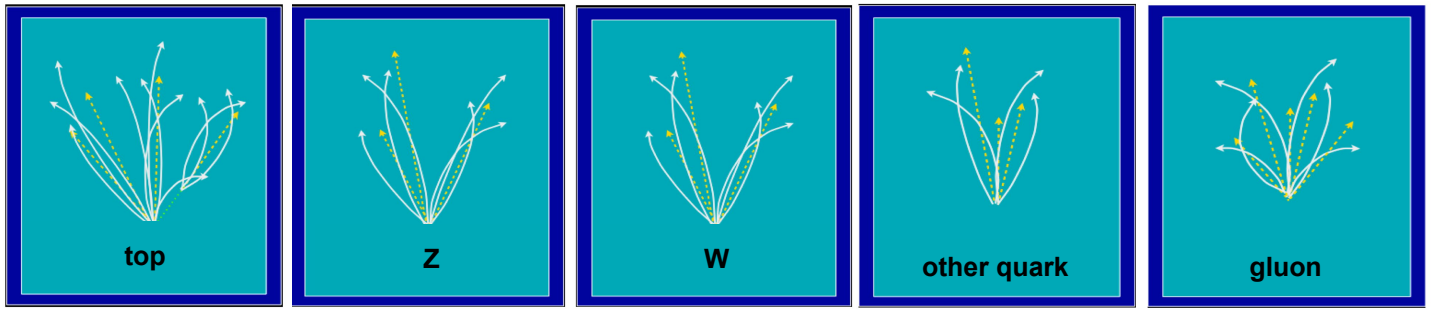
Fully parallel
Each mult. used 1x

More resources

Example: Jet tagging with a DNN

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

Example: Jet tagging with a DNN

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

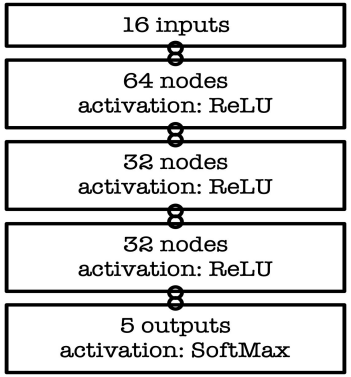
- D. Guest at al. PhysRevD.94.112002, G. Kasieczka et al. JHEP05(2017)006, J. M. Butterworth et al. PhysRevLett.100.242001, etc..

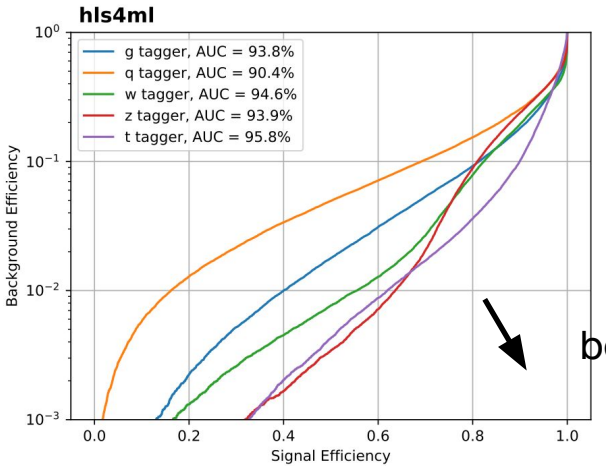
Train a **five class multi-classifier** on a sample of ~ 1M events with two boosted WW/ZZ/tt/qq/gg anti-k_T jets

- Dataset DOI: 10.5281/zenodo.3602254
- OpenML: https://www.openml.org/d/42468

Fully connected neural network with 16 inputs:

- Relu activation function for intermediate layers
- Softmax activation function for output layer



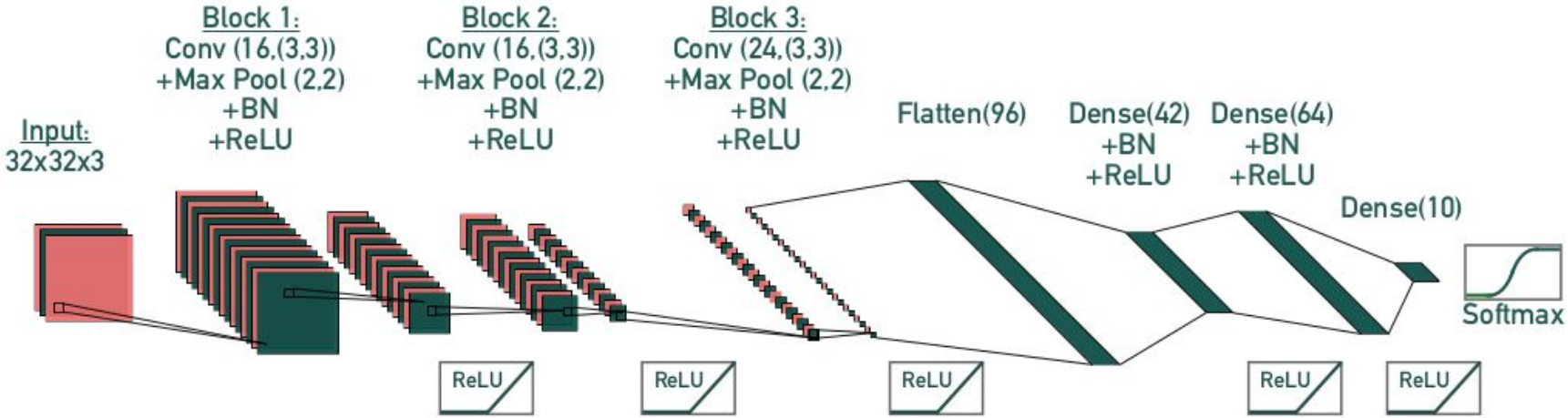


Example: SVHN digit classification with CNNs

Street-view house numbers dataset (SVHN)

- 32x32x3 images
- A tougher MNIST

Model architecture:



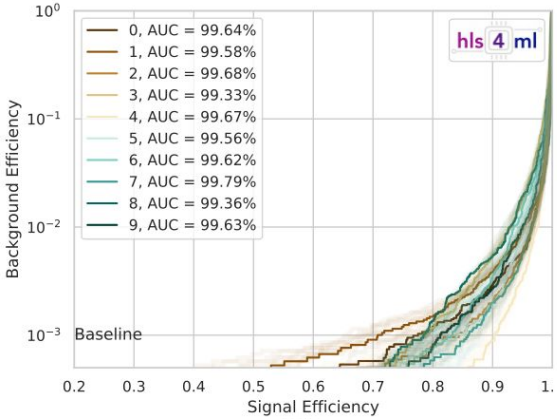
Example: SVHN model performance

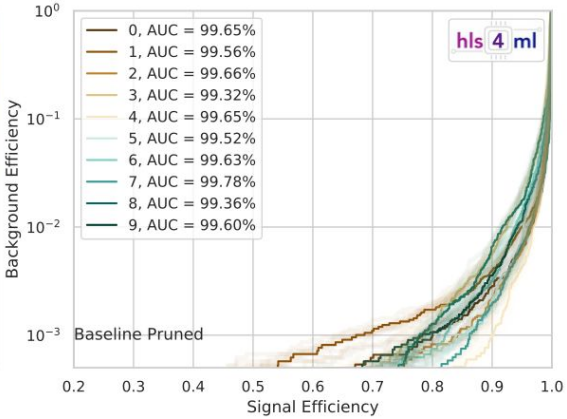
Baseline models

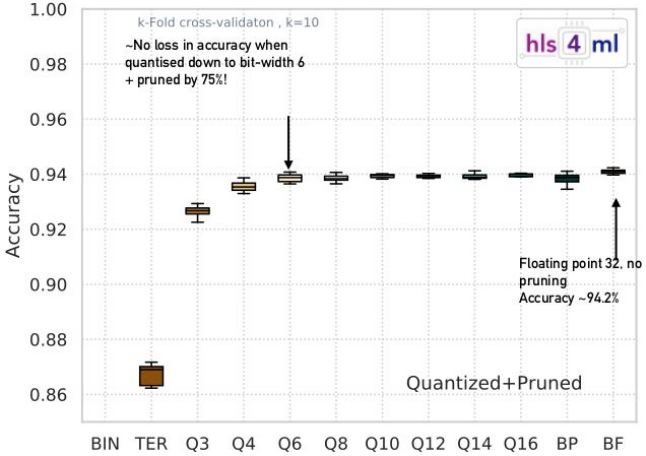
- Full 32-bit precision (BF)
- Full 32-bit precision, pruned (BP)
  - 75% sparsity
  - Polynomial decay

QKeras models

- Quantized (Q)
  - Binary (1-bit)
  - Ternary (2-bit)
  - Quantized to 3-16 bits
- Pruned (QP)
  - 75% sparsity







Example: SVHN model performance on an FPGA

Targeting a high-end Xilinx Virtex UltraScale+ VU9P series FPGA

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!

Possibility to scale to much smaller FPGAs!

- Potentially much lower cost

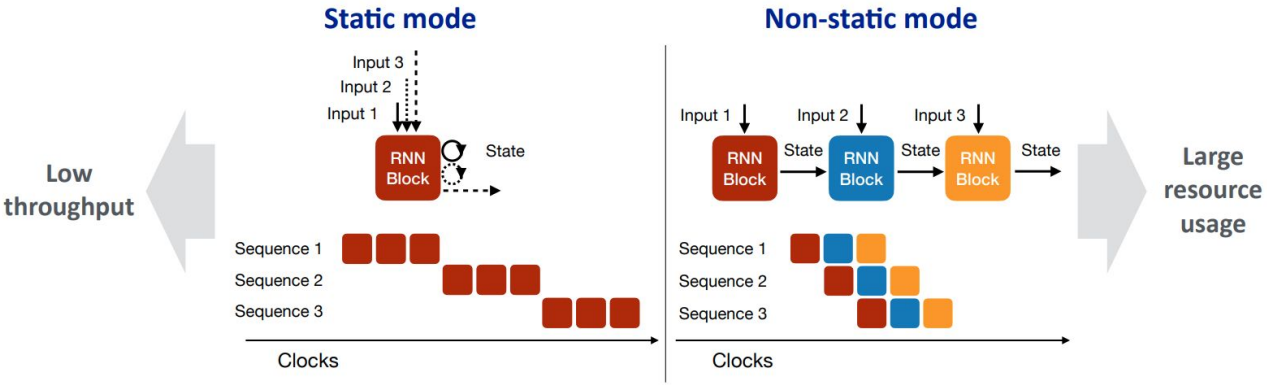
Example: Top tagging with LSTM/GRU

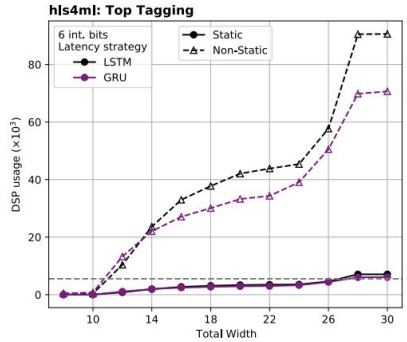
Implementation of LSTM/GRU layers commonly used in RNNs

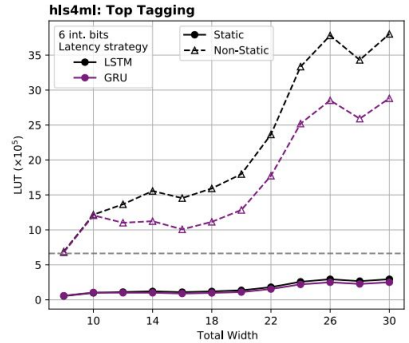
Two “modes” of operation:

- Static mode: a single RNN block processes every input for every sequence
- Non-Static mode: RNN blocks for each input in the sequence

Non-Static mode is expected to give larger throughput (lower II) but at much higher resource usage







Example: Jet tagging with symbolic expressions

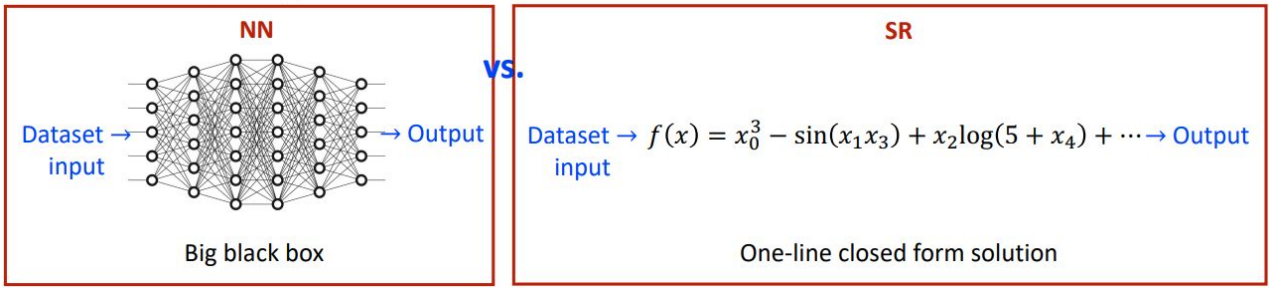
Symbolic Regression (SR): a ML technique that seeks to discover analytic functions that approximate a dataset

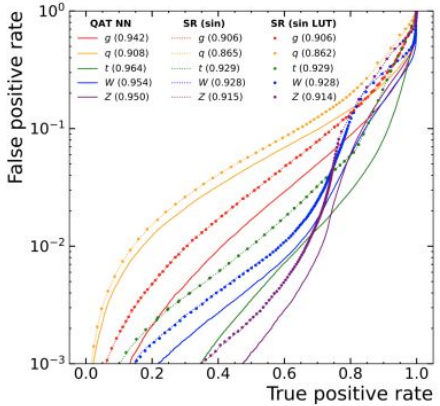
- Offer interpretable results for the underlying problem

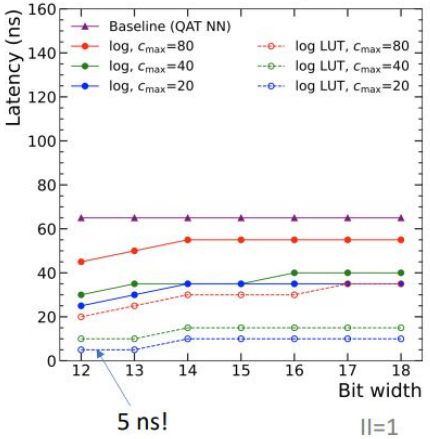
Expressions can be “discovered” via PySR or a neural network

hls4ml takes care of code generation, data types and approximation of math functions with lookup tables

More details in Ho-Fung’s talk at CHEP







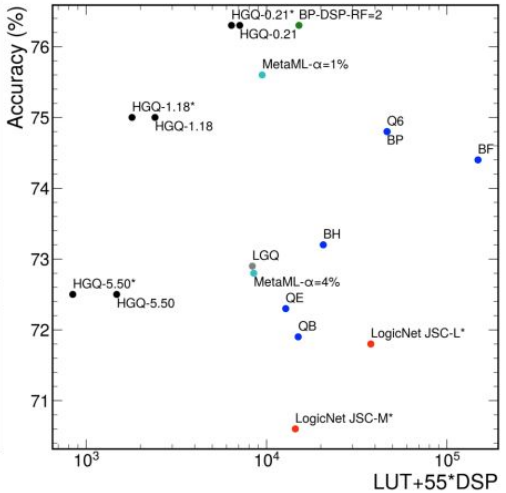
Ongoing work: High-Granularity Quantization

High Granularity Quantization (HGQ) - A novel method on optimizing bitwidths with gradients

- Unlike QKeras where all weights of a layer have the same bitwidth, in HGQ each weight of the same multiplier gets unique bitwidth, perfect for unrolled designs that we use in L1T
- Significantly reduced resource usage, out-of-the-box bit accuracy, accurate resource estimation…

See Chang's talk at FastML, soon fully supported in **hls4ml**

Model	Accuracy (%)	Latency (cc)	DSP (%)	LUT (%)	FF (%)	II (cc)
BF [19]	74.4	9 (45 ns)	56.0 (1,826)	4.09 (48,321)	0.8 (20,132)	
BP [19]	74.8	14 (70 ns)	7.7 (526)	1.49 (17,577)	0.4 (10,548)	
BH [19]	73.2	14 (70 ns)	1.3 (88)	1.34 (15,802)	0.3 (8,108)	
Q6 [19]	74.8	11 (55 ns)	1.8 (124)	3.36 (39,782)	0.3 (8,128)	
QE [19]	72.3	11 (55 ns)	1.0 (66)	0.77 (9,149)	0.1 (1,781)	
QB [19]	71.9	14 (70 ns)	1.0 (69)	0.95 (11,193)	0.1 (1,771)	
LogicNets JSC-M [50]	70.6	N/A	0 (0)	1.22 (14,428)	0.02 (440)	
LogicNets JSC-L [50]	71.8	5 (13 ns)	0 (0)	3.21 (37,931)	0.03 (810)	
BP-DSP-RF=2 [28]	76.3	21 (105 ns)	2.6 (175)	0.47 (5,504)	0.13 (3,036)	2
MetaML-α_q=1% [20]	75.6	9 (45 ns)	0.7 (50)	0.57 (6,698)	N/A	1
MetaML-α_q=4% [20]	72.8	8 (40 ns)	0.2 (23)	0.57 (7,224)	N/A	1
LGQ	72.9	6 (30 ns)	0 (0)	0.70 (8,341)	0.09 (2,059)	1
HGQ-0.21	76.3	5 (25 ns)	0.15 (10)	0.49 (5,843)	0.04 (1,036)	1
(w/ softmax)		9 (45 ns)	0.22 (15)	0.53 (6,276)	0.06 (1,402)	1
HGQ-1.18	75.0	4 (20 ns)	0.04 (3)	0.14 (1,631)	0.02 (375)	1
(w/ softmax)		8 (40 ns)	0.12 (8)	0.17 (1,965)	0.02 (528)	1
HGQ-5.50	72.5	3 (15 ns)	0 (0)	0.07 (843)	0.01 (198)	1
(w/ softmax)		7 (35 ns)	0.07 (5)	0.10 (1,201)	0.01 (340)	1



Ongoing work: Hardware-aware pruning

In fully unrolled designs (ReuseFactor = 1), HLS compiler optimizes any multiplications by zero

- Any unstructured pruning works if RF=1
- But what about RF>1? We don't get the benefit unless all weights that share a DSP are zero

Hardware-aware pruning (from Ben) allows us to structure pruning by targeting weights that are shared by a DSP and set them to 0 (can also be applied to save BRAMs as well)

RF	Model	Quantised accuracy [%]	Latency [ns]	LUT	FF	BRAM (reduction)	DSP (reduction)
2	BM	**76.39**	168	42,103	25,790	951	2,133
	BP-DSP	76.29	**105**	**5,504**	**3,036**	246 (3.9x)	**175 (12.2x)**
	BP-MO	76.23	**105**	9,971	3,682	**182 (5.2x)**	217 (9.8x)
4	BM	**76.39**	210	25,274	21,583	478	1,069
	BP-DSP	75.84	**161**	**6,484**	4,232	138 (3.5x)	**90 (11.9x)**
	BP-MO	75.83	**161**	6,835	**3,736**	**111 (4.3x)**	92 (11.6x)
8	BM	**76.39**	315	20,949	19,613	241	537
	BP-DSP	75.96	**252**	**9,632**	**5,488**	89 (2.7x)	**68 (7.9x)**
	BP-MO	75.76	259	10,368	5,841	**70 (3.4x)**	83 (6.5x)
16	BM	**76.39**	539	19,141	19,598	124	271
	BP-DSP	76.06	**392**	**6,693**	**5,322**	54 (2.3x)	**47 (5.8x)**
	BP-MO	75.90	413	10,701	7,630	**53 (2.3x)**	71 (3.9x)

Effects of pruning on jet classification post P&R with 7ns clock period

BM = Baseline

BP-DSP = DSP-optimised model

BP-MO = BRAM- & DSP optimised

Ongoing work: Transformers

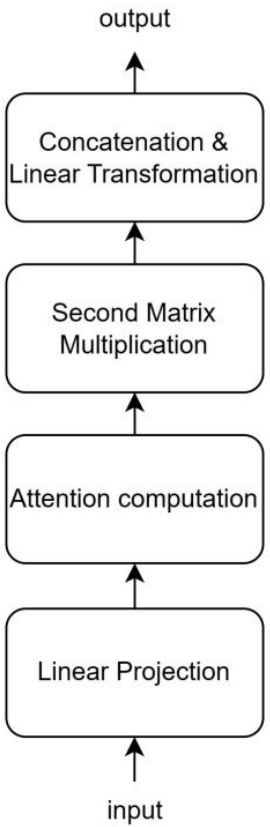
Goal - Transformer architectures with microsecond latencies

A promising demonstration of an implementation of MultiHeadAttention layer in Keras was created, see talk by Elham et al

- Multi-stage pipelined design integrated with **hls4ml**'s existing features (RF)
- + LayerNorm and other building blocks

Tested on a few models, e.g., for B-tagging (9k parameters)

Reuse and clk Period	Interval (cycle)	Latency (cycles)	Latency (time)
Reuse 1 (7.720 ns)	49	269	2.077 us
Reuse 2 (7.720 ns)	65	449	3.467 us
Reuse 4 (7.621 ns)	100	768	5.853 us



Moving on to next stage and investigating efficient transformer architectures

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 and neural network architectures

- Graph NNs, Transformers…

More information:

- Website
- Code
- Tutorial