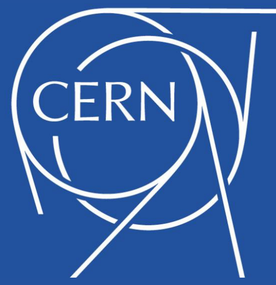
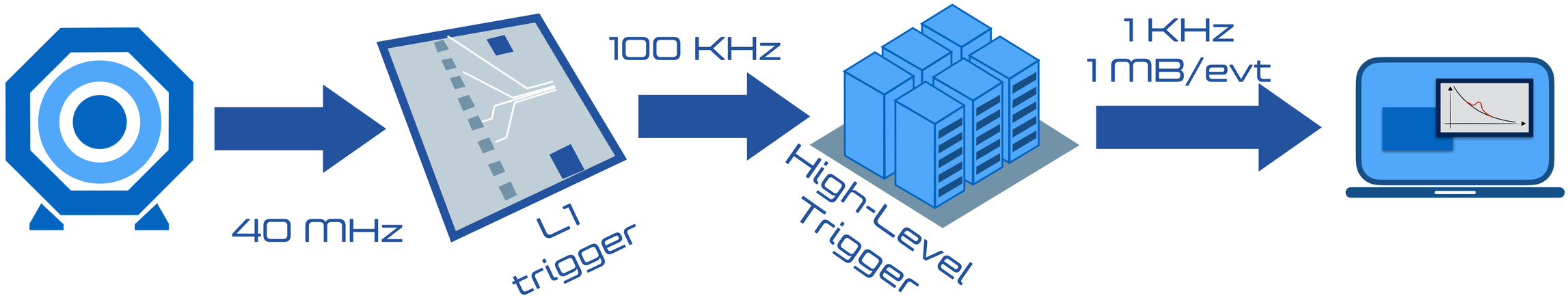
Fast inference of deep neural networks in FPGAs

Maurizio Pierini

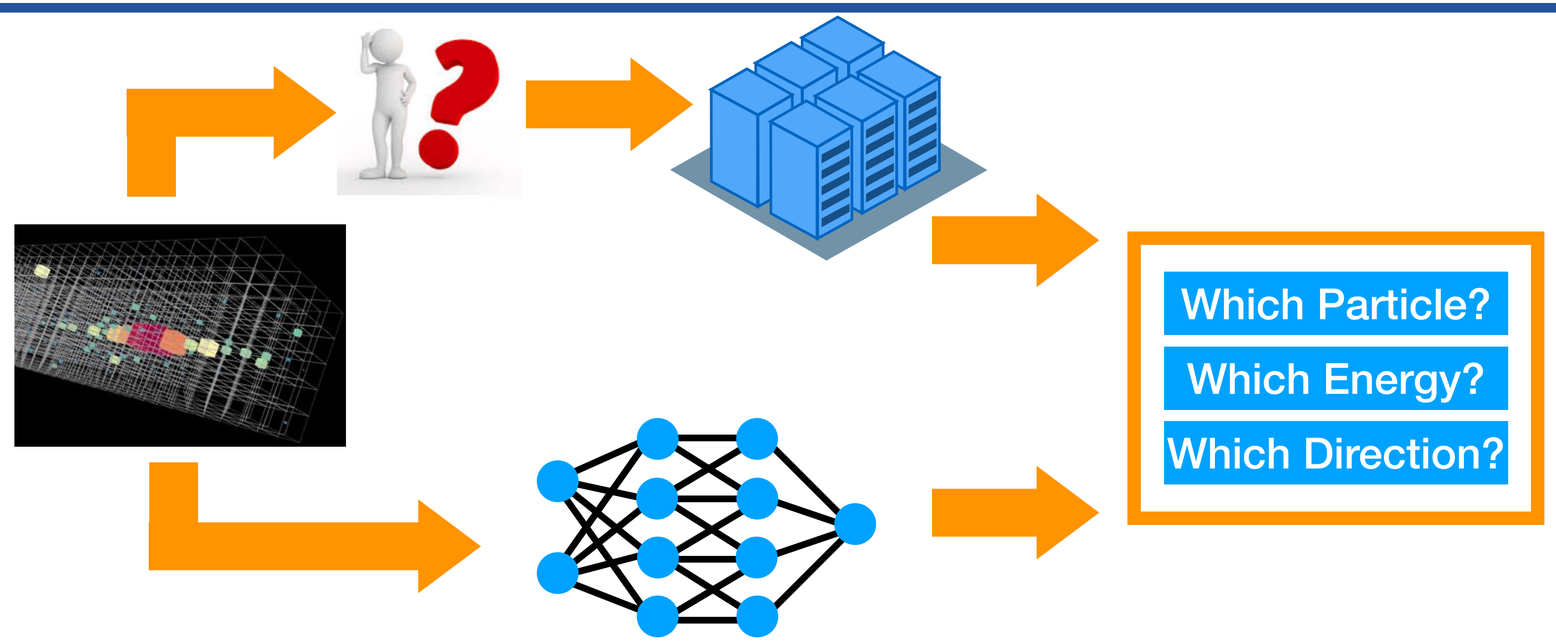


The LHC Big Data Problem



- *The LHC delivers ~40M events/sec. This gives us little time (<10 μsec) to reconstruct and filter these events*
  - *Coarse reconstruction, with worse resolution than offline*
  - *Simple algorithms, e.g., sums and products (at some point, deploying invariant mass trigger was a big deal)*
  - *Extended use of Look Up Tables to parameterise complex functions (with limitation in terms of dimensionality)*

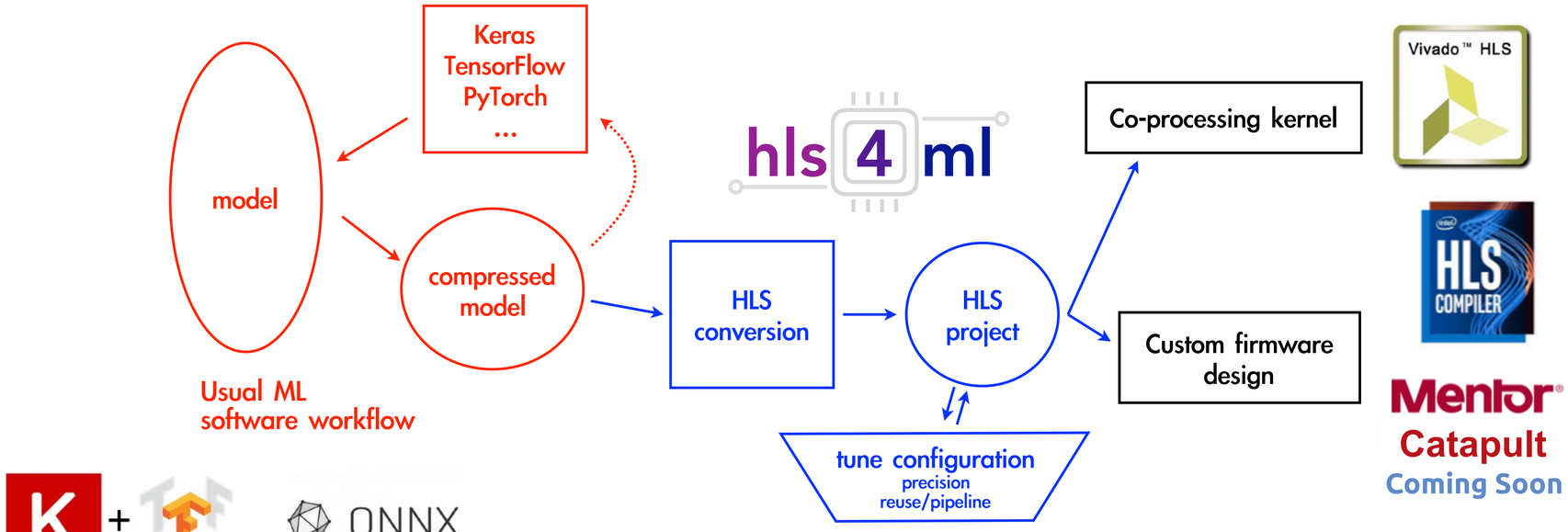
Enter Machine Learning



- *We know how to get from the data the answers we want*
  - *physics + intuition + computing*
- *But the process is slow*
  - *We can use DL solutions as a shortcut: we teach neural networks how to give us the answer we want directly from the raw data*
- *Problem: how do we put a NN on an FPGA?*

HLS4ML: the idea

- *HLS4ML aims to be this automatic tool*
  - *reads as input models trained on standard DeepLearning libraries*
  - *comes with implementation of common ingredients (layers, activation functions, etc)*
  - *Uses HLS softwares to provide a firmware implementation of a given network*
  - *Could also be used to create co-processing kernels for HLT environments*



https://fastmachinelearning.org/hls4ml/
https://arxiv.org/abs/1804.06913

HLS4ML: the implementation

- *Dataflow architecture: each layer is an independent compute unit*
  - *With tunable parallelism and quantization*
- *Fully on-chip: NN must fit within available FPGA resources (pynq-z2 floorplan shown)*
  - *Example: small CNN trained on MNIST*

Conv2D

ReLU

MaxPool2D

Conv2D

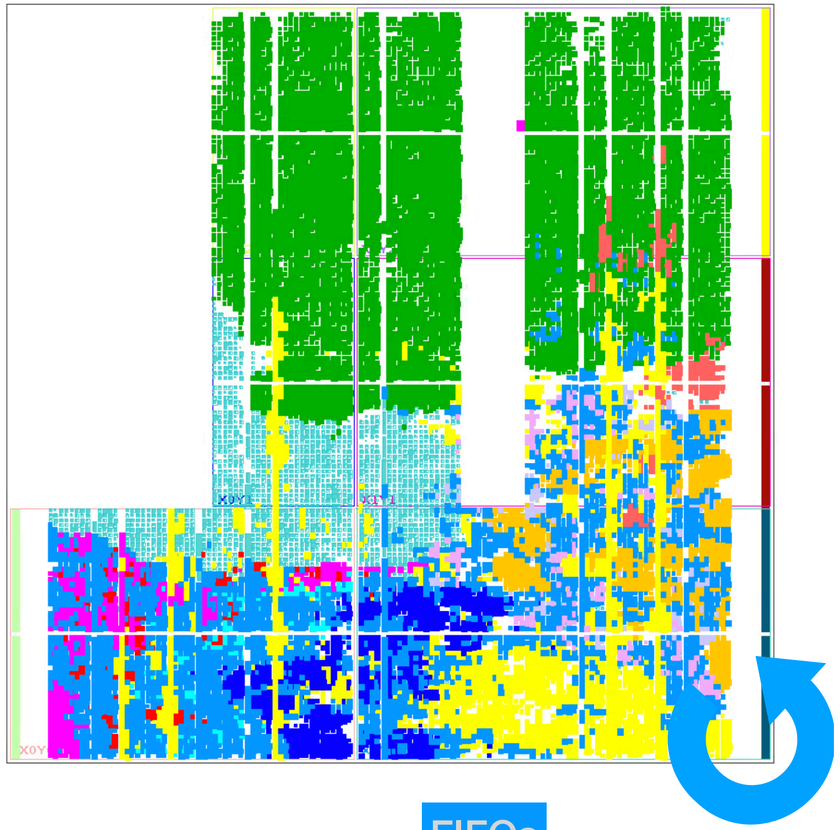
ReLU

MaxPool2D

Flatten

Dense

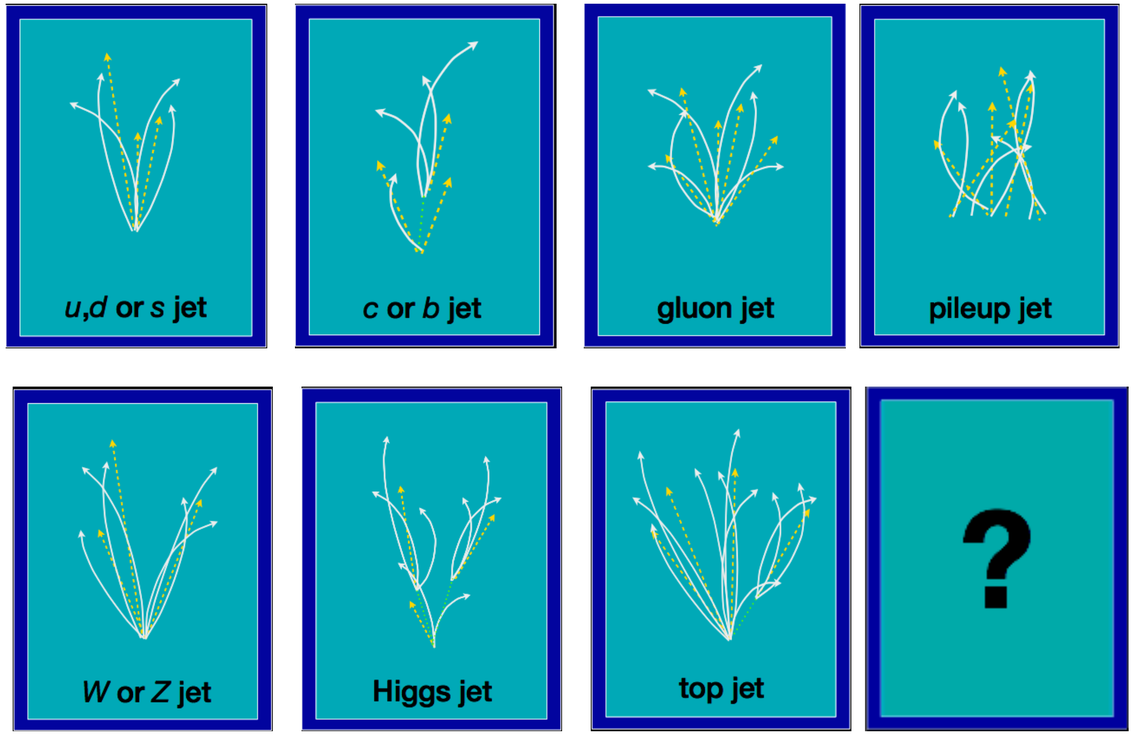
Softmax



FIFOs

Example: fast inference

- *You have a jet at LHC: spray of hadrons coming from a "shower" initiated by a fundamental particle of some kind (quark, gluon, W/Z/H bosons, top quark)*
- *You have a set of jet features whose distribution depends on the nature of the initial particle*
- *You can train a network to start from the values of these quantities and guess the nature of your jet*
- *To do this you need a sample for which you know the answer*



https://arxiv.org/abs/1804.06913

Example: jet tagging

- *Simple DNN based on high-level features (jet masses, multiplicities, energy correlation functions)*

Observables

m_{mMDT}

$N_2^{\beta=1,2}$

$M_2^{\beta=1,2}$

$C_1^{\beta=0,1,2}$

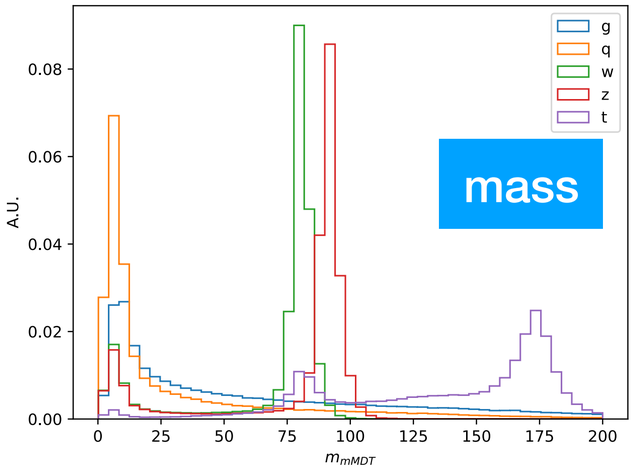
$C_2^{\beta=1,2}$

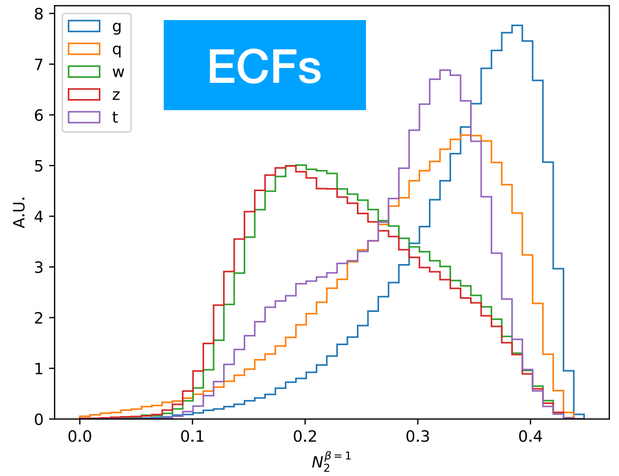
$D_2^{\beta=1,2}$

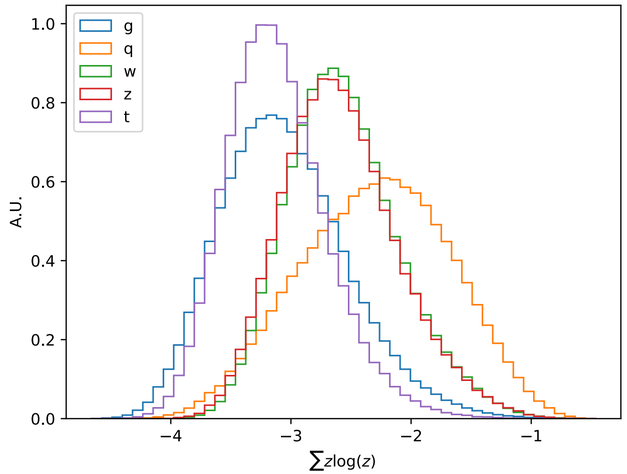
$D_2^{(\alpha,\beta)=(1,1),(1,2)}$

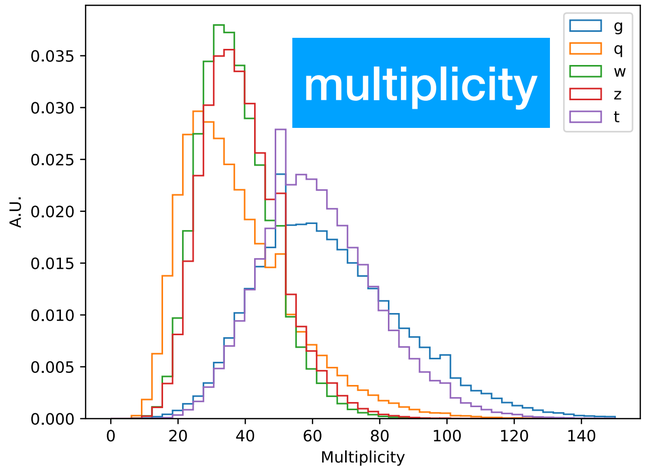
$\sum z \log z$

Multiplicity



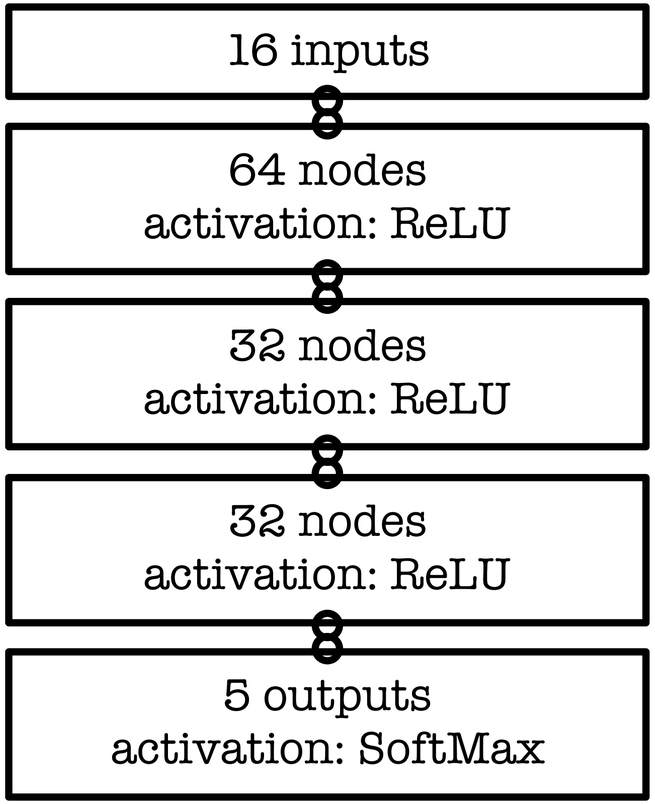


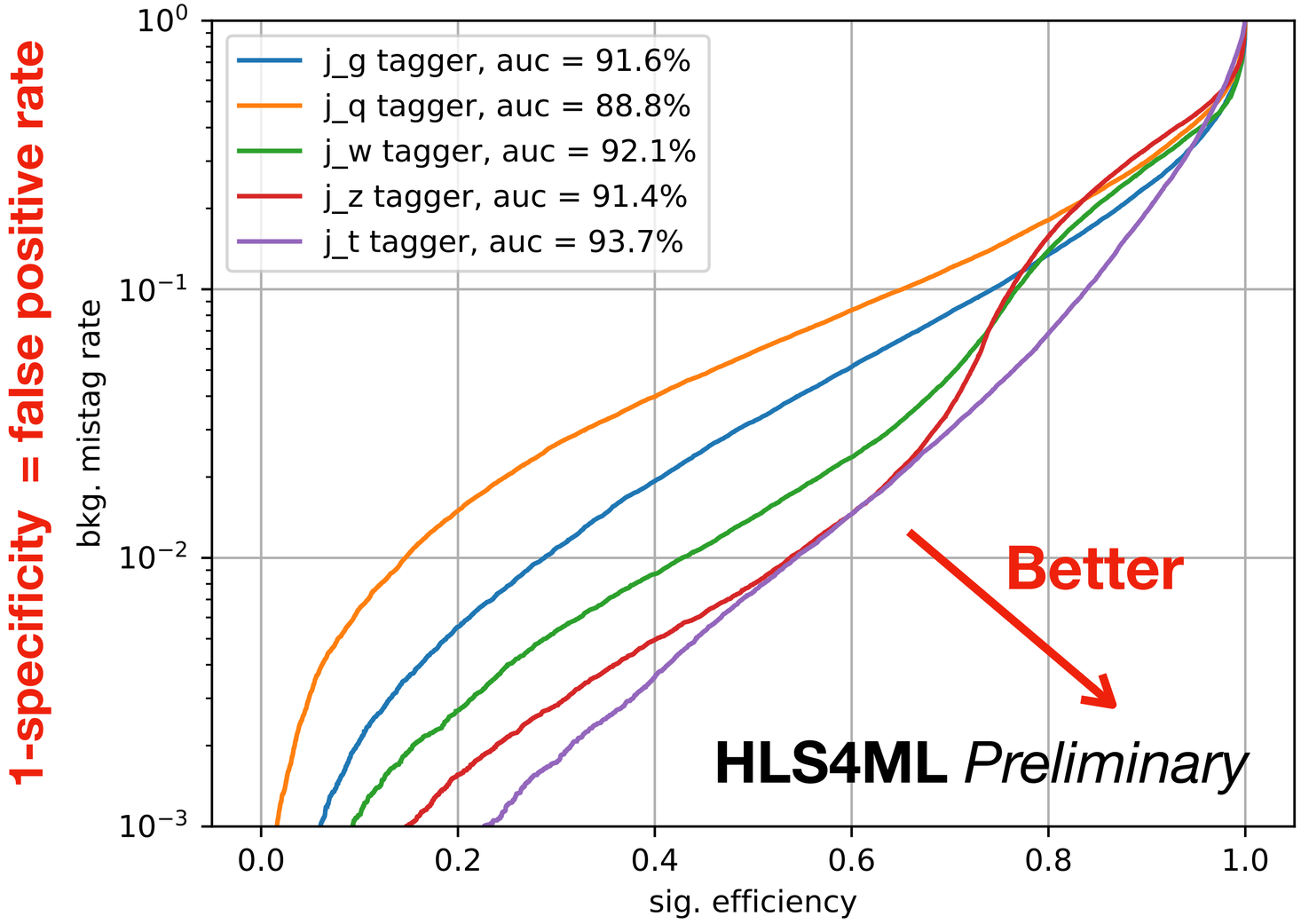




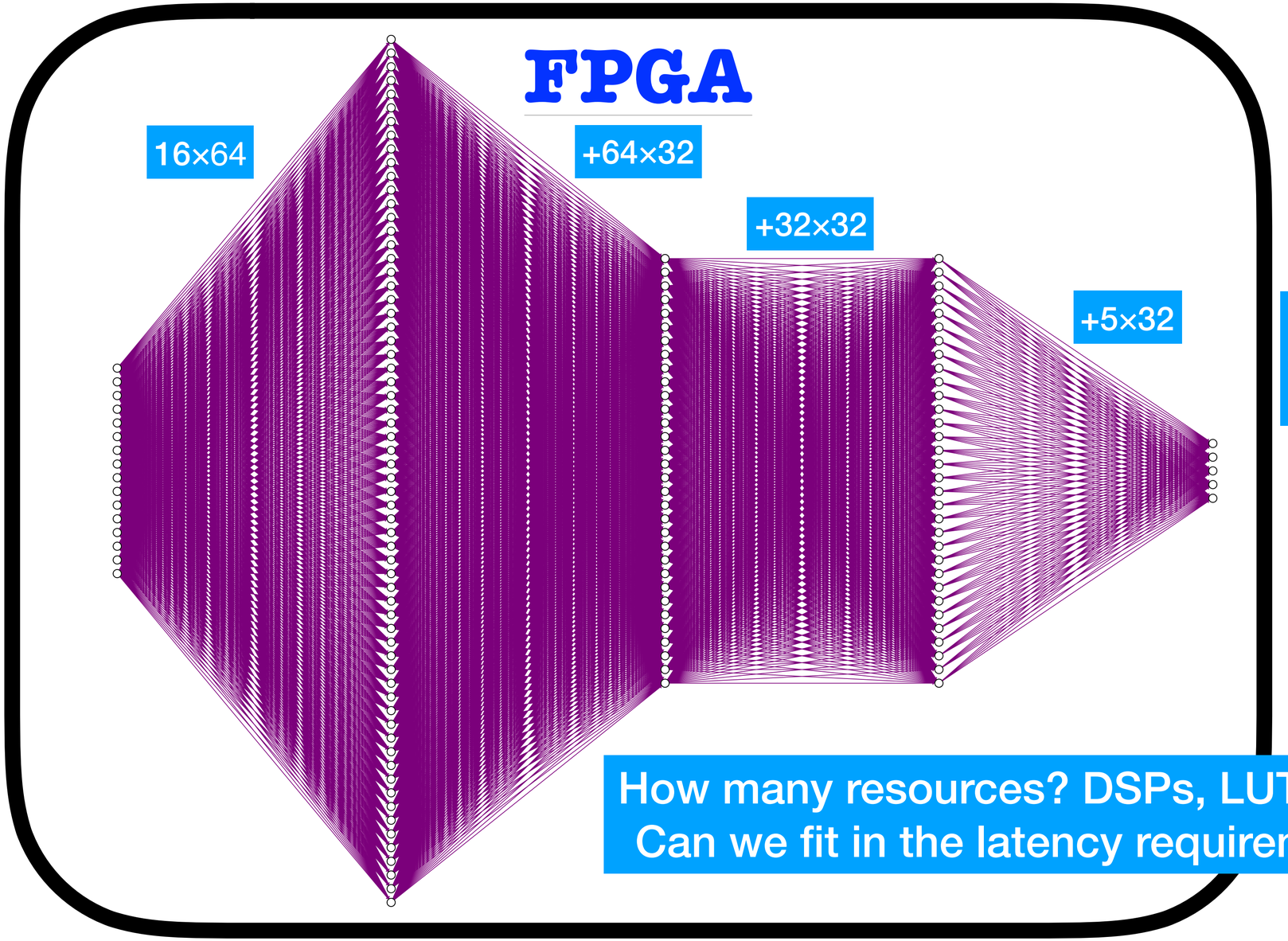
Example: jet tagging

- *Simple DNN based on high-level features (jet masses, multiplicities, energy correlation functions)*





The full model

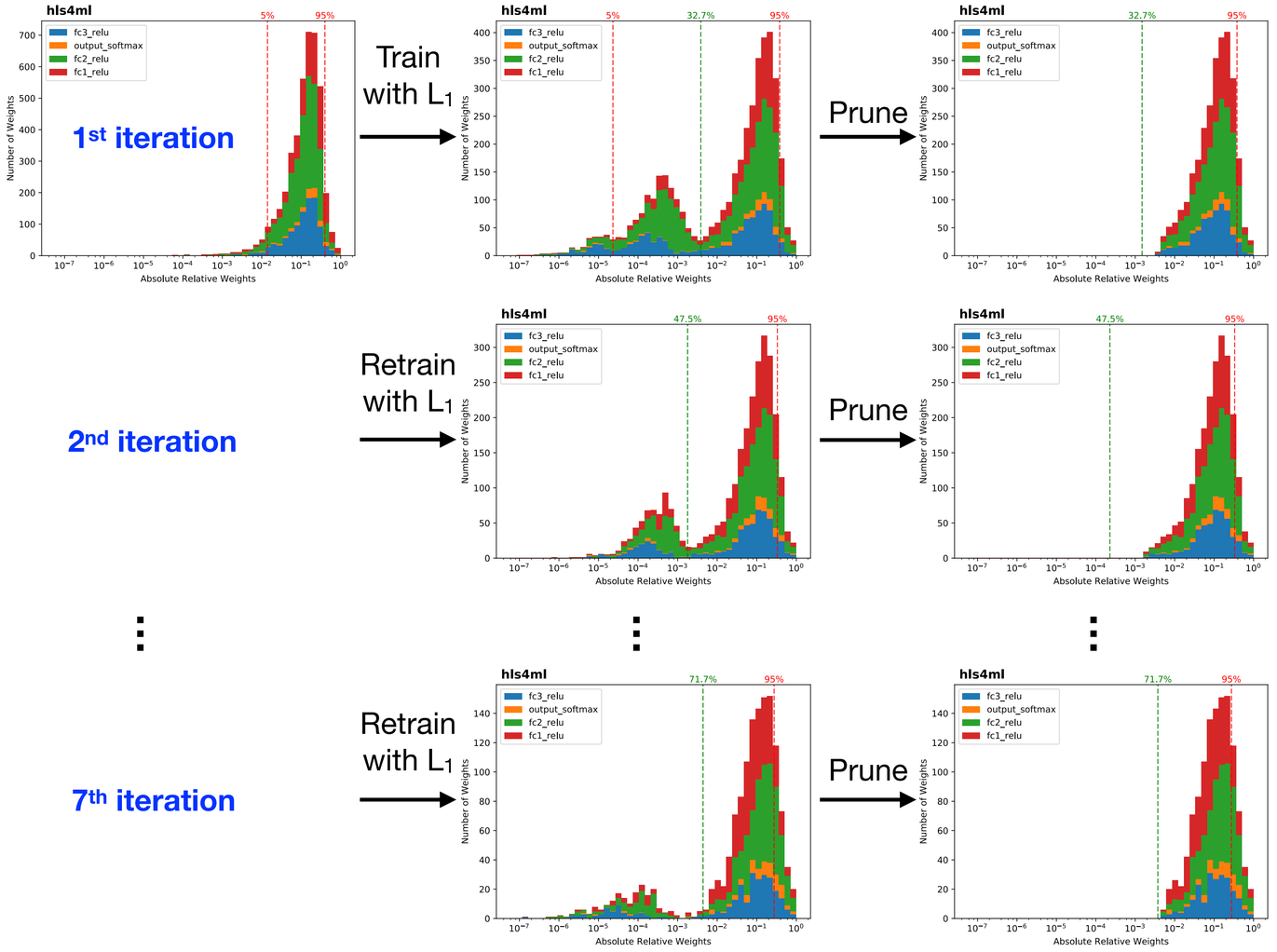


Making the NN smaller: pruning

- ***Pruning:*** *remove parameters that don't really contribute to performances*
  - *force parameters to be as small as possible (regularization)*

$$L_\lambda(\vec{w}) = L(\vec{w}) + \lambda||\vec{w}_1||$$

  - *Remove the small parameters*
  - *Retrain*

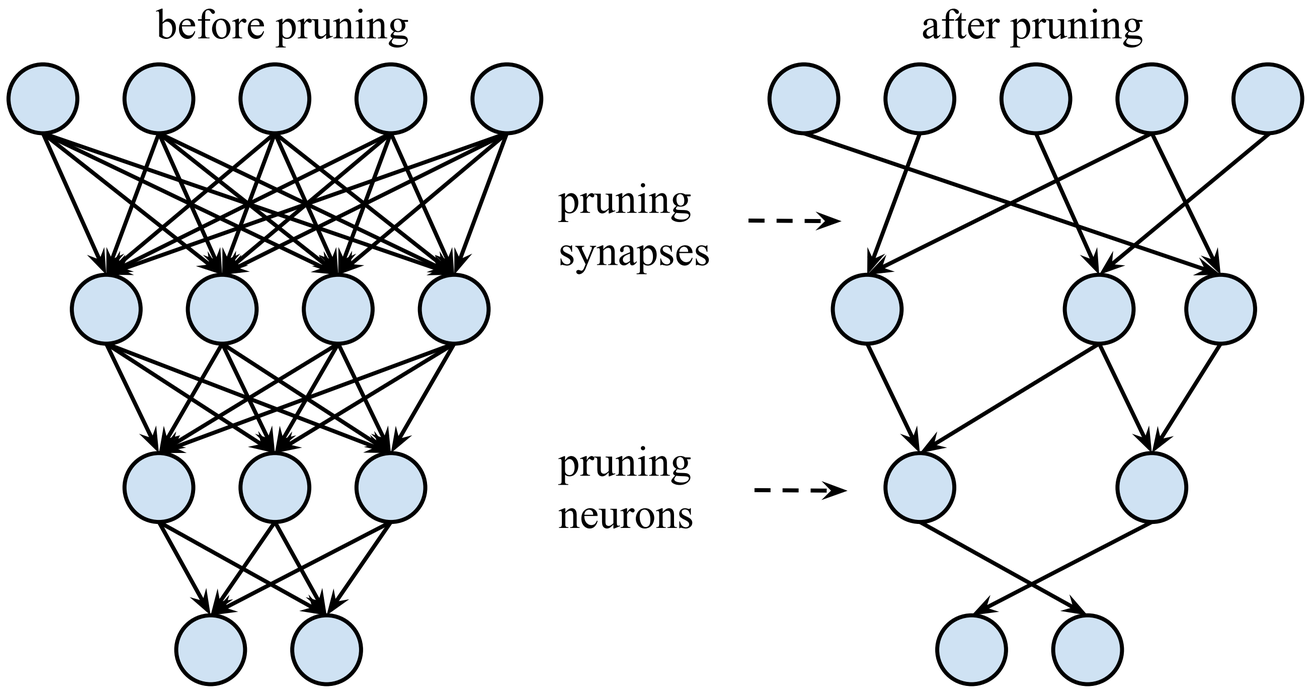


Making the NN smaller: pruning

- ***Pruning:*** *remove parameters that don't really contribute to performances*
  - *force parameters to be as small as possible (regularization)*

$$L_\lambda(\vec{w}) = L(\vec{w}) + \lambda||\vec{w}_1||$$

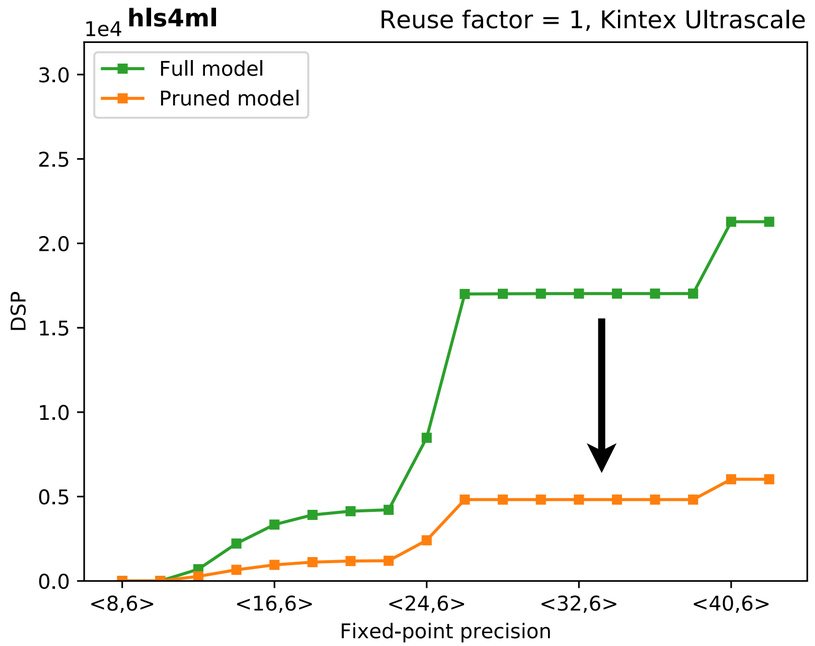
  - *Remove the small parameters*
  - *Retrain*

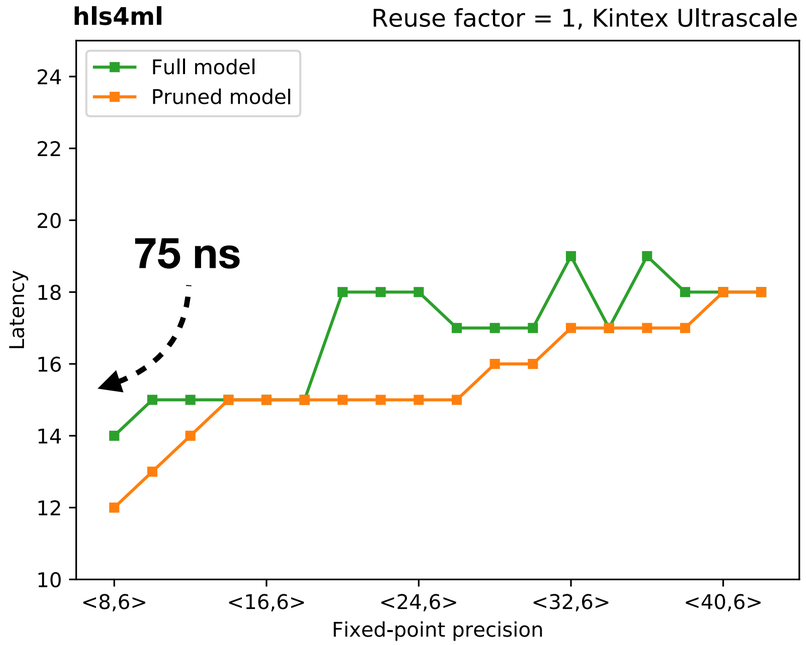


→ 70% reduction of weights and multiplications w/o performance loss

Compression

Xilinx Vivado 2017.2
Clock frequency: 200 MHz
FPGA: Xilinx Kintex Ultrascale
(XCKU115-FLVB2104)

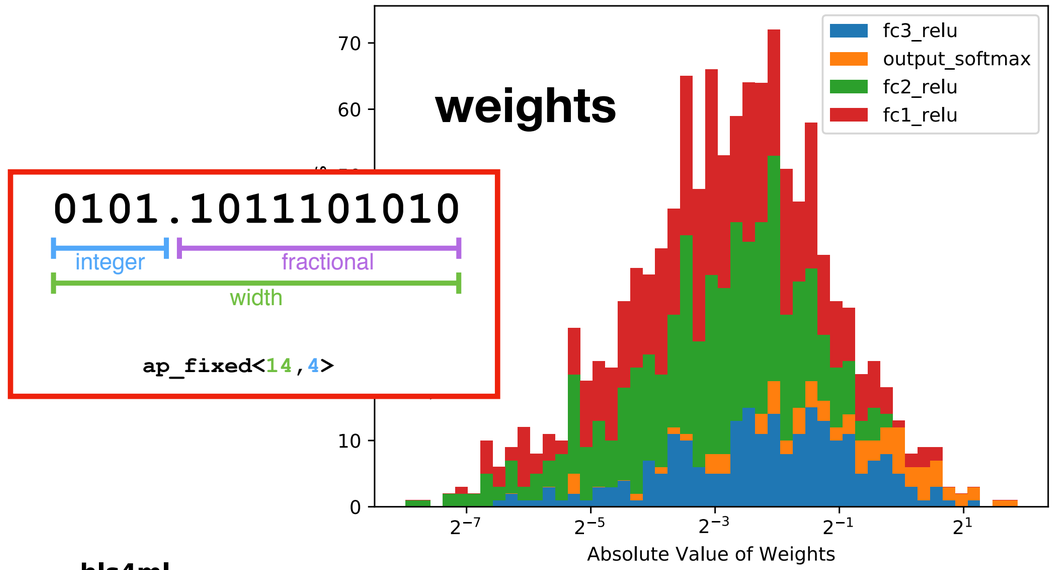


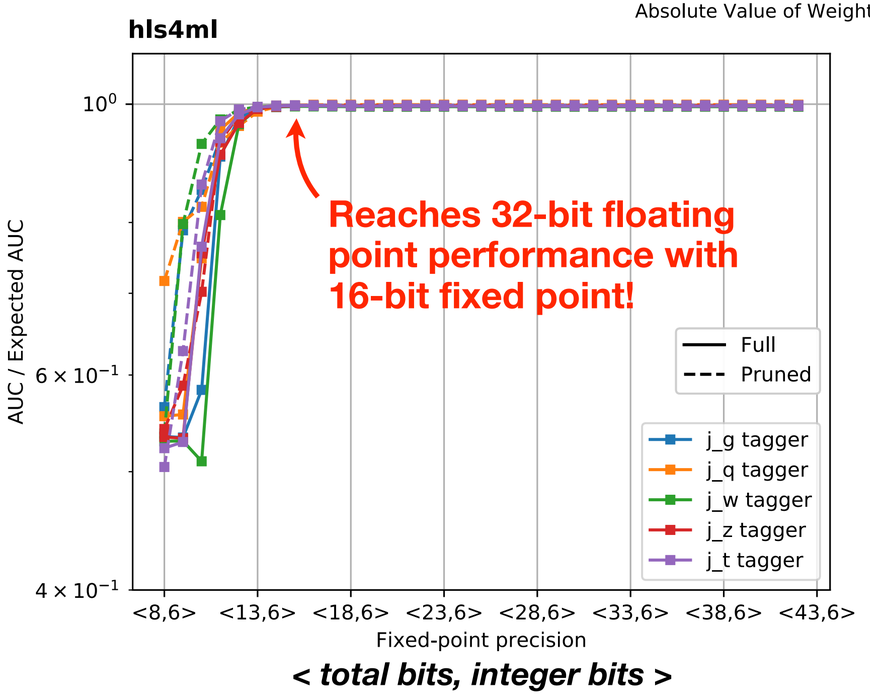


- Big reduction in DSP usage with pruned model!
- ~15 clocks @ 200 MHz = 75 ns inference

Quantisation

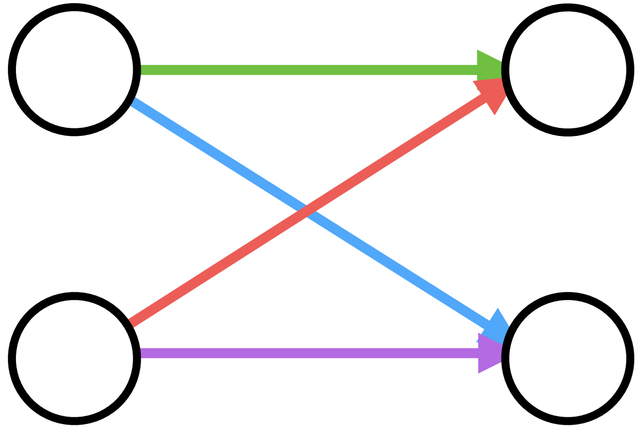
- ***Quantisation***: *reduce the number of bits used to represent numbers (i.e., reduce used memory)*
  - *models are usually trained at 64 or 32 bits*
  - *this is not necessarily needed in real life*
- *In our case, we could reduce to 16 bits w/o loosing precision*
- *Beyond that, one would have to accept some performance loss*

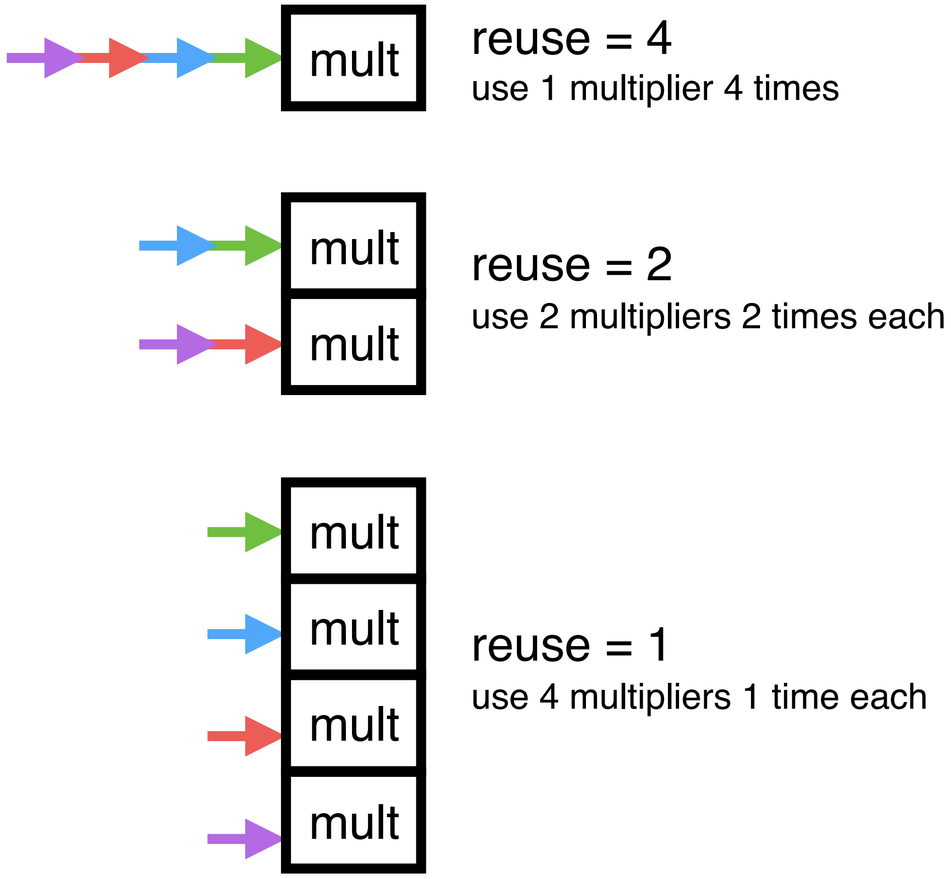




Parallelisation

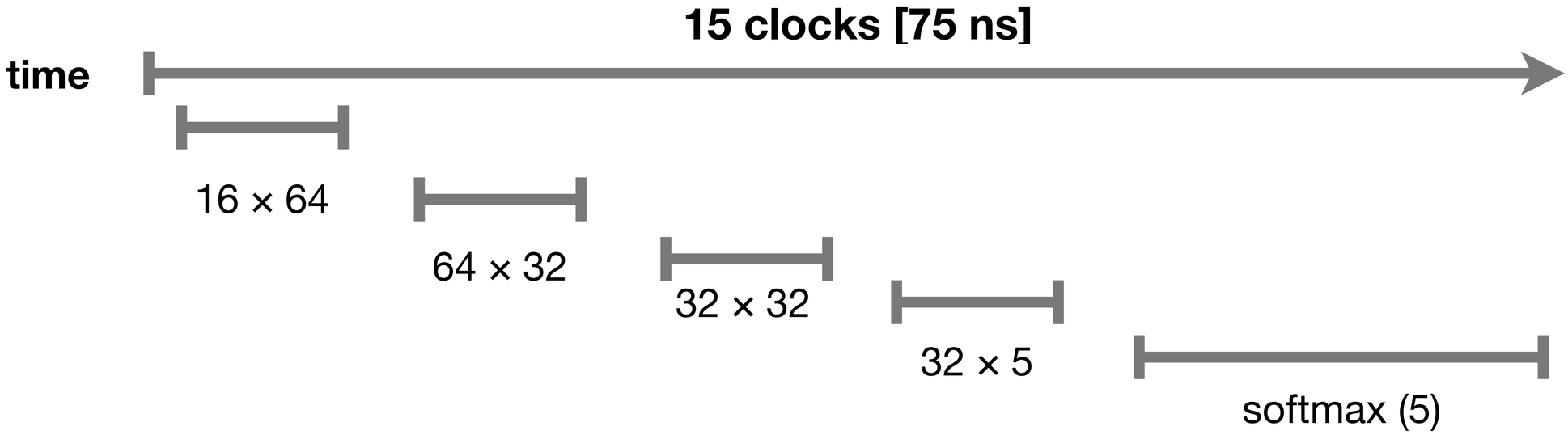
- `ReuseFactor`: how much to parallelize





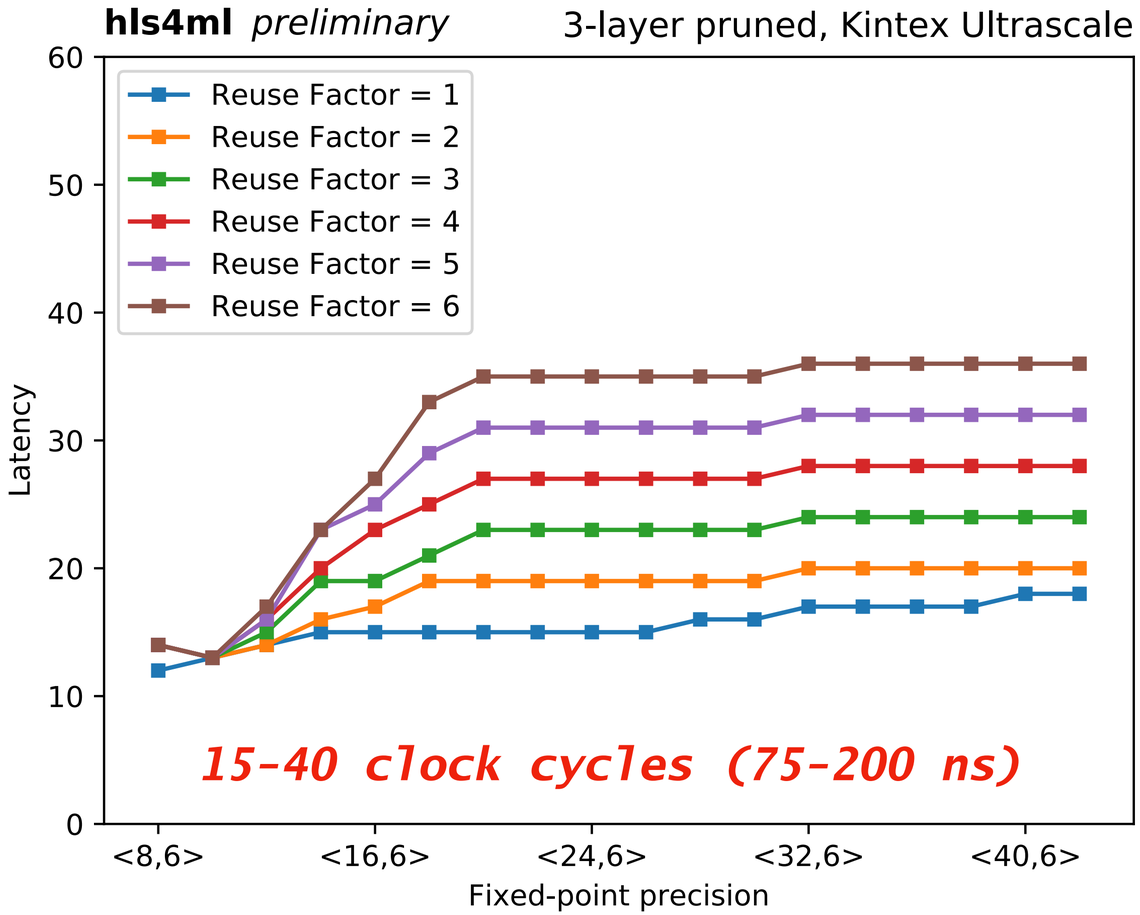
related to the Initiation Interval = when new inputs are introduced to the algo.

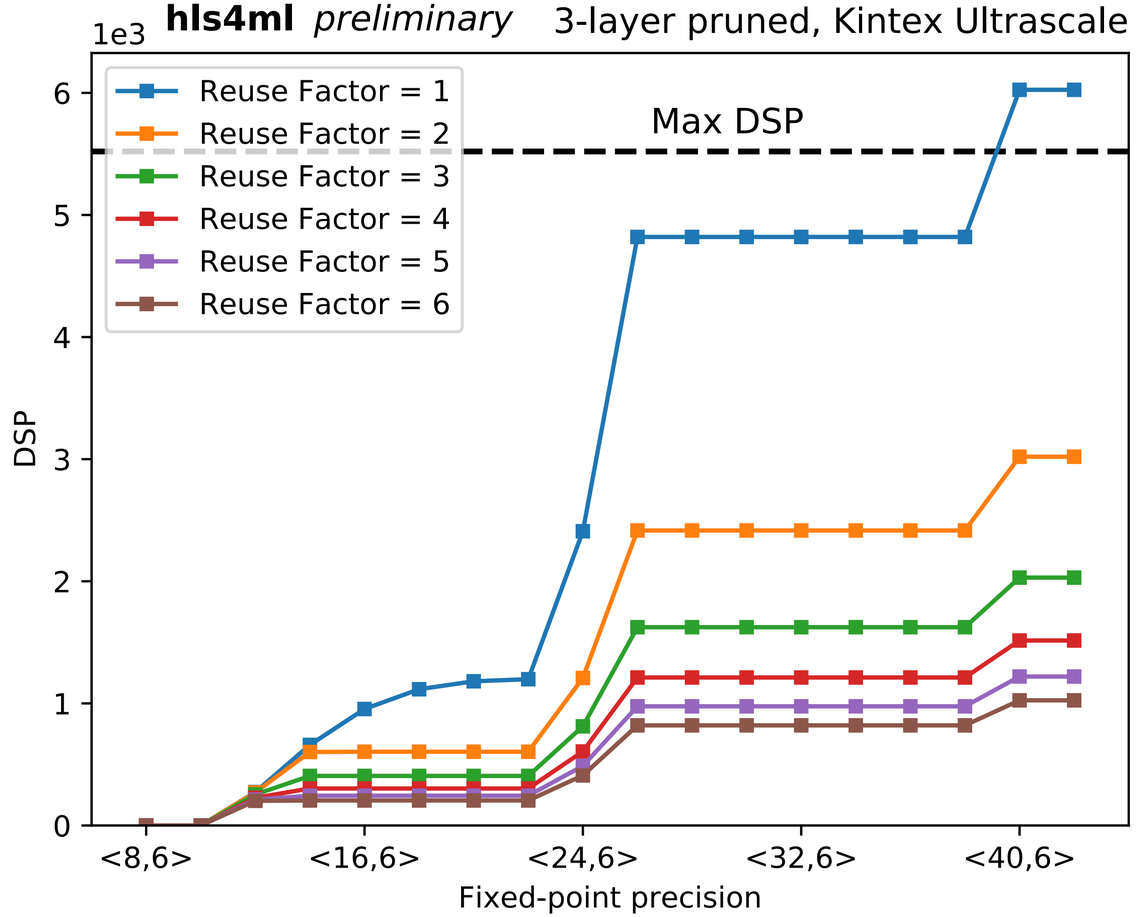
Parallelisation



reuse = 1 <16, 6> bits	BRAM	DSP	FF	LUT
Total	13	954	53k	36k
% Usage	~0%	17%	3%	5%

Parallelisation

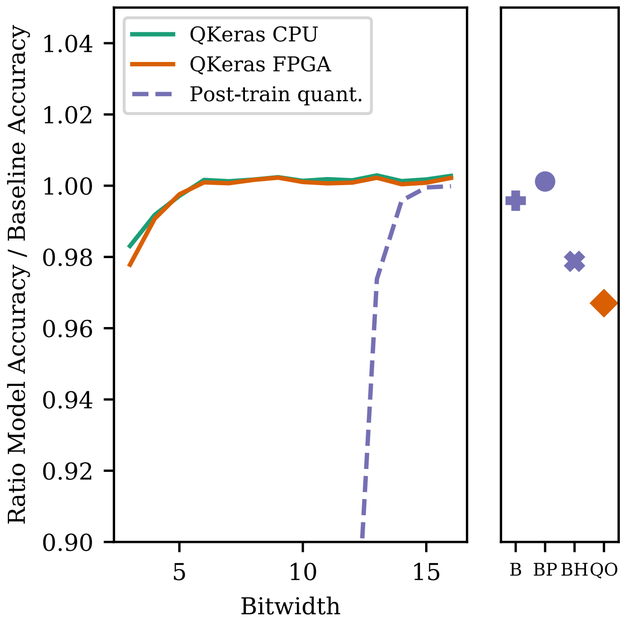


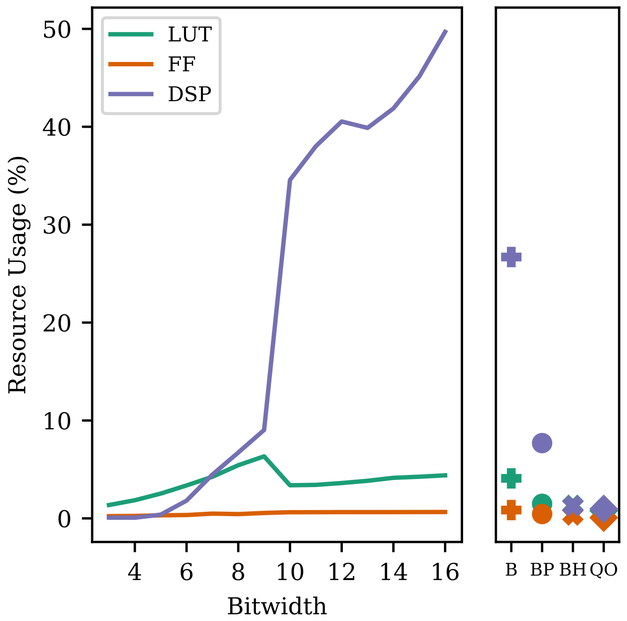


Foreseen architecture (FPGAs) will handle these networks
Inference-optimized GPUs could break the current paradigm
Looking forward to R&D projects with nVidia & E4 on this

Quantization-aware Training

- *Post-training quantisation can affect accuracy*
  - *for a given bit allocation, the loss minimum at floating-point precision might not be the minimum anymore*
- *One could specify quantisation while look for the minimum*
  - *Maximize accuracy for minimal FPGA resources*
- *We teamed up with Google to exploit this strategy in a QKeras+hls4ml bundle*



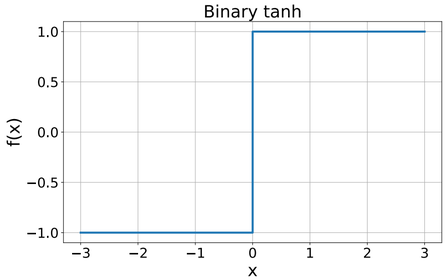


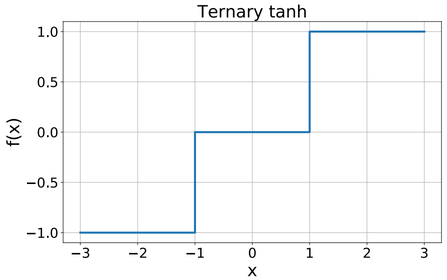
Model	Accuracy [%]	Latency [ns]	Latency [clock cycles]	DSP [%]	LUT [%]	FF [%]
Baseline	74.4	45	9	56.0 (1826)	5.2 (48321)	0.8 (20132)
Baseline pruned	74.8	70	14	7.7 (526)	1.5 (17577)	0.4 (10548)
Baseline heterogeneous	73.2	70	14	1.3 (88)	1.3 (15802)	0.3 (8108)
QKeras 6-bit	74.8	55	11	1.8 (124)	3.4 (39782)	0.3 (8128)
QKeras Optimized	72.3	55	11	1.0 (66)	0.8 (9149)	0.1 (1781)

https://arxiv.org/abs/2006.10159

European Research Council

Extreme Quantization

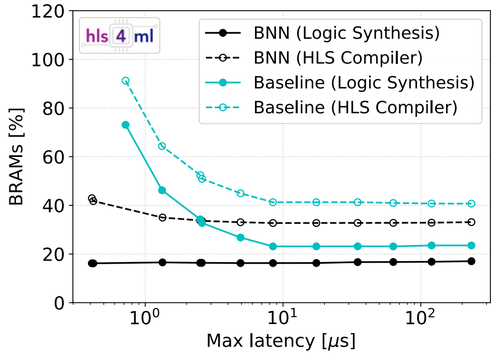


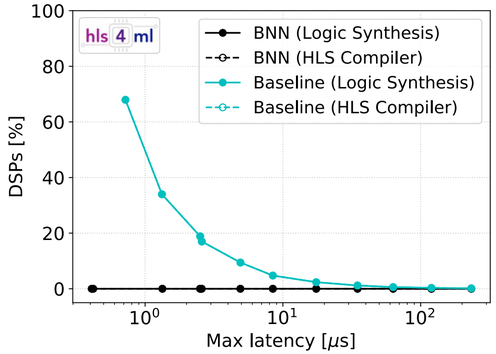


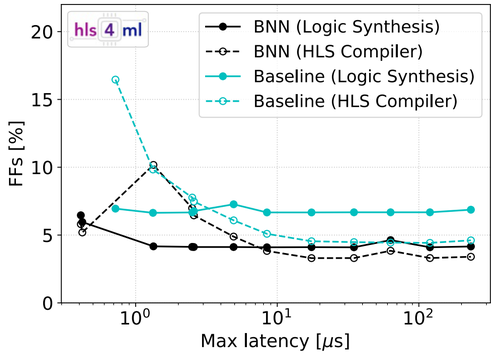
A	B	$A \times B$
-1	-1	1
-1	1	-1
1	-1	-1
1	1	1

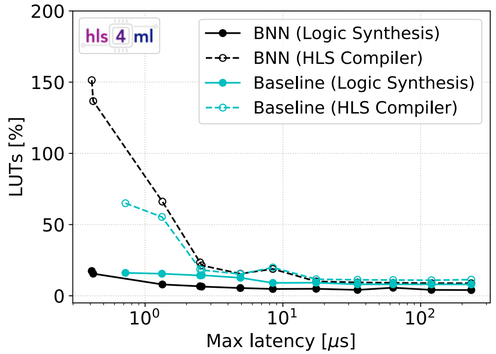
A	B	$\overline{A \oplus B}$
0	0	1
0	1	0
1	0	0
1	1	1

- *One can push quantisation to extremes*
  - *binary & ternary networks*
- *Multiplications can be replaced by bit manipulations, saving resources*
- *Can achieve low latencies at small accuracy cost and minimal resource consumption*









https://arxiv.org/abs/2003.06308

Fast CNN inference on FPGAs

Input: 32x32x3

Block 1: Conv 0 (f=16, k=3), Max Pooling (2,2), Batch Norm., ReLU

Block 2: Conv 1 (f=16, k=3), Max Pooling (2,2), Batch Norm., ReLU

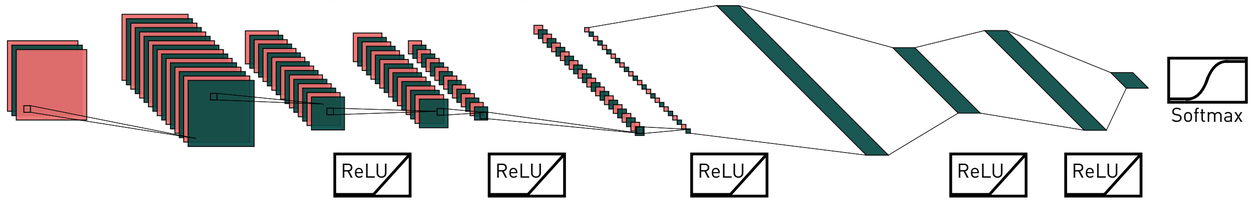
Block 3: Conv 2 (f=24, k=3), Max Pooling (2,2), Batch Norm., ReLU

Flatten (96)

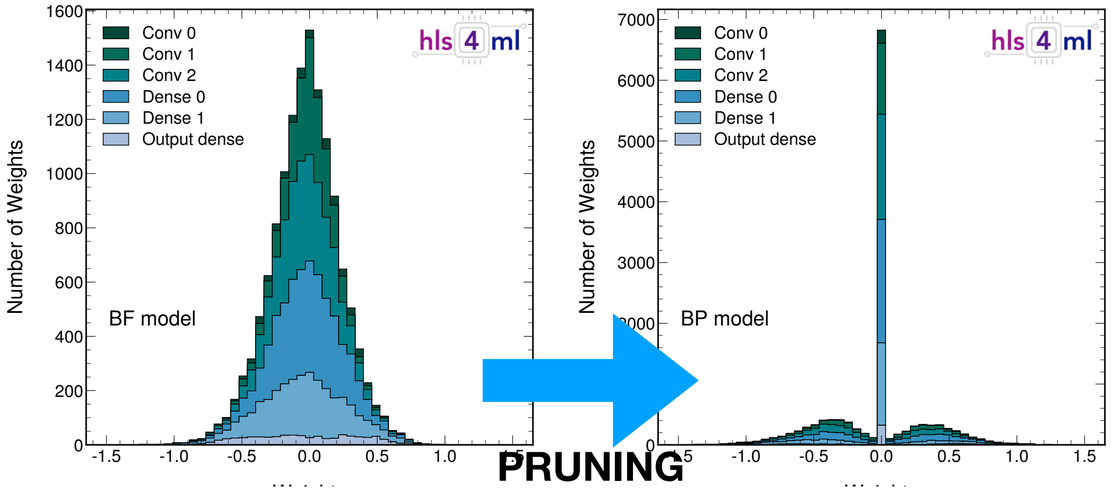
Block 4: Dense 0 (n=42), Batch Norm., ReLU

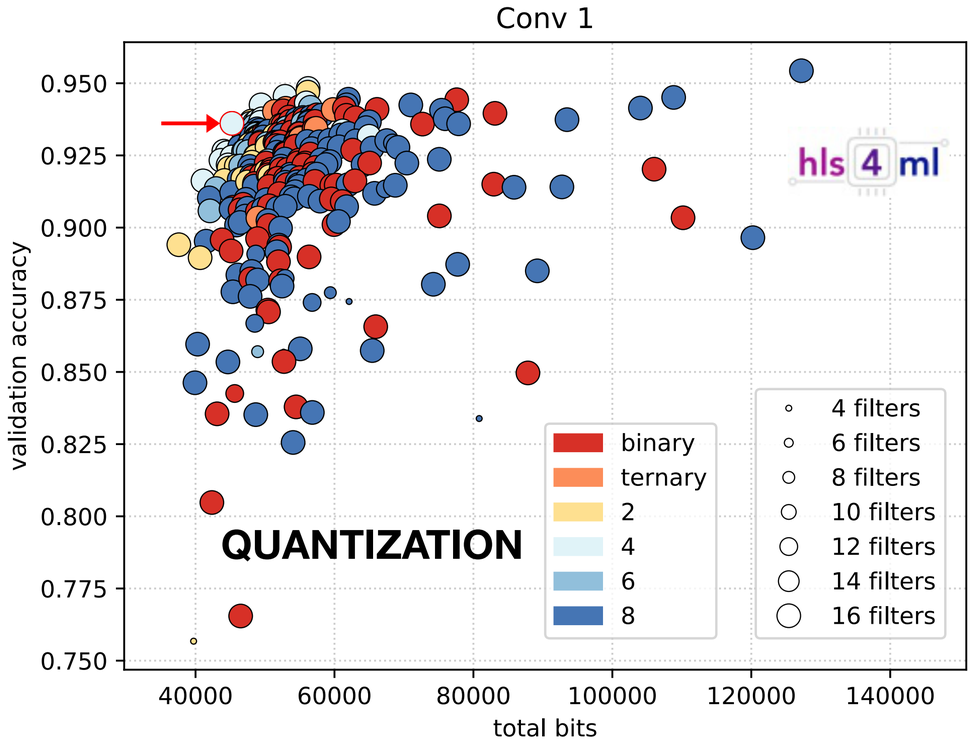
Block 5: Dense 1 (n=64), Batch Norm., ReLU

Output: Dense output (n=10), Softmax



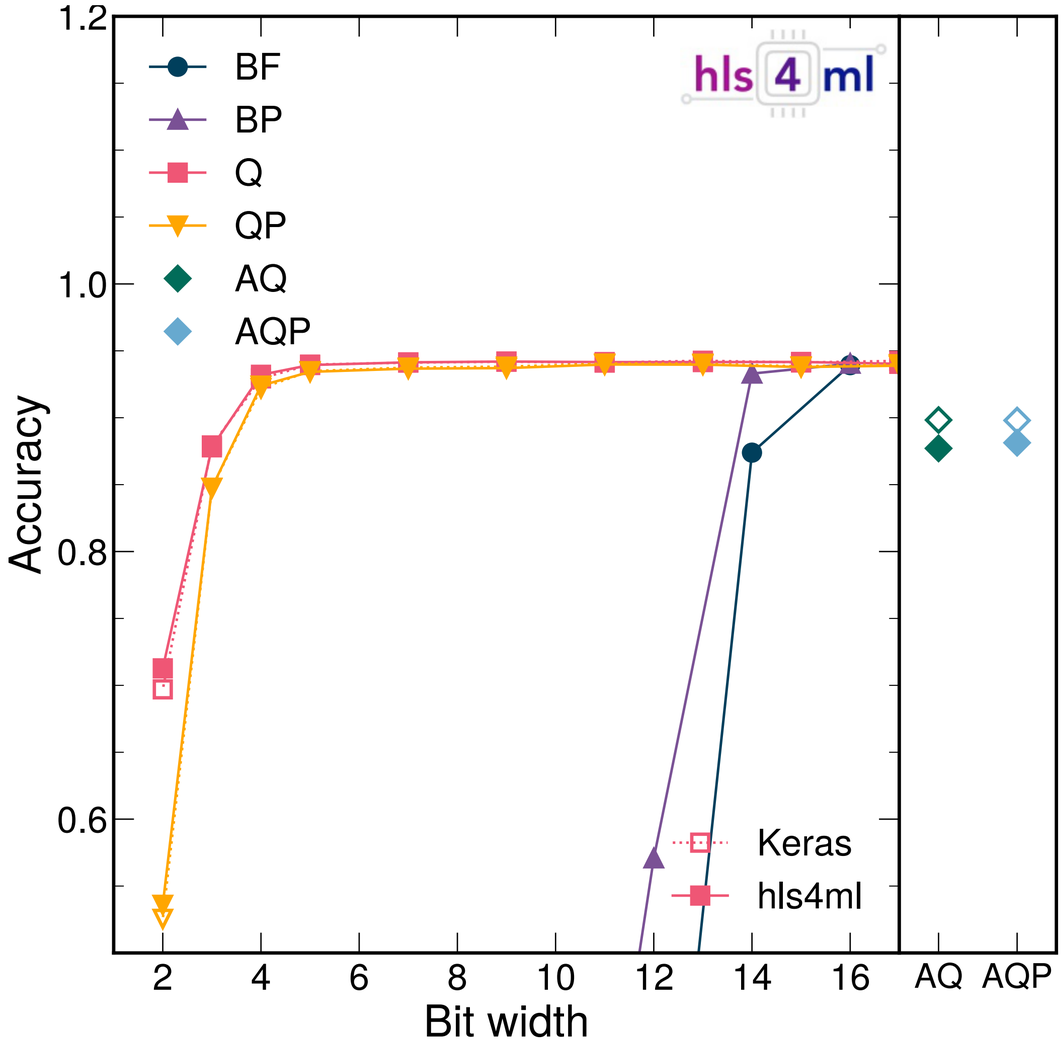
Layer name	Layer type	Input shape	Weights	MFLOPs	Energy [nJ]	Bit size
Conv 0	Conv2D	(32, 32, 3)	432	0.778	1,795	3,456
Conv 1	Conv2D	(15, 15, 16)	2,304	0.779	1,802	18,432
Conv 2	Conv2D	(6, 6, 16)	3,456	0.110	262	27,648
Dense 0	Dense	(96)	4,032	0.008	26	32,256
Dense 1	Dense	(42)	2,688	0.005	17	21,504
Output	Dense	(64)	65	0.001	4	5,200
Model total			**12,858**	**1.71**	**3,918**	**170,816**

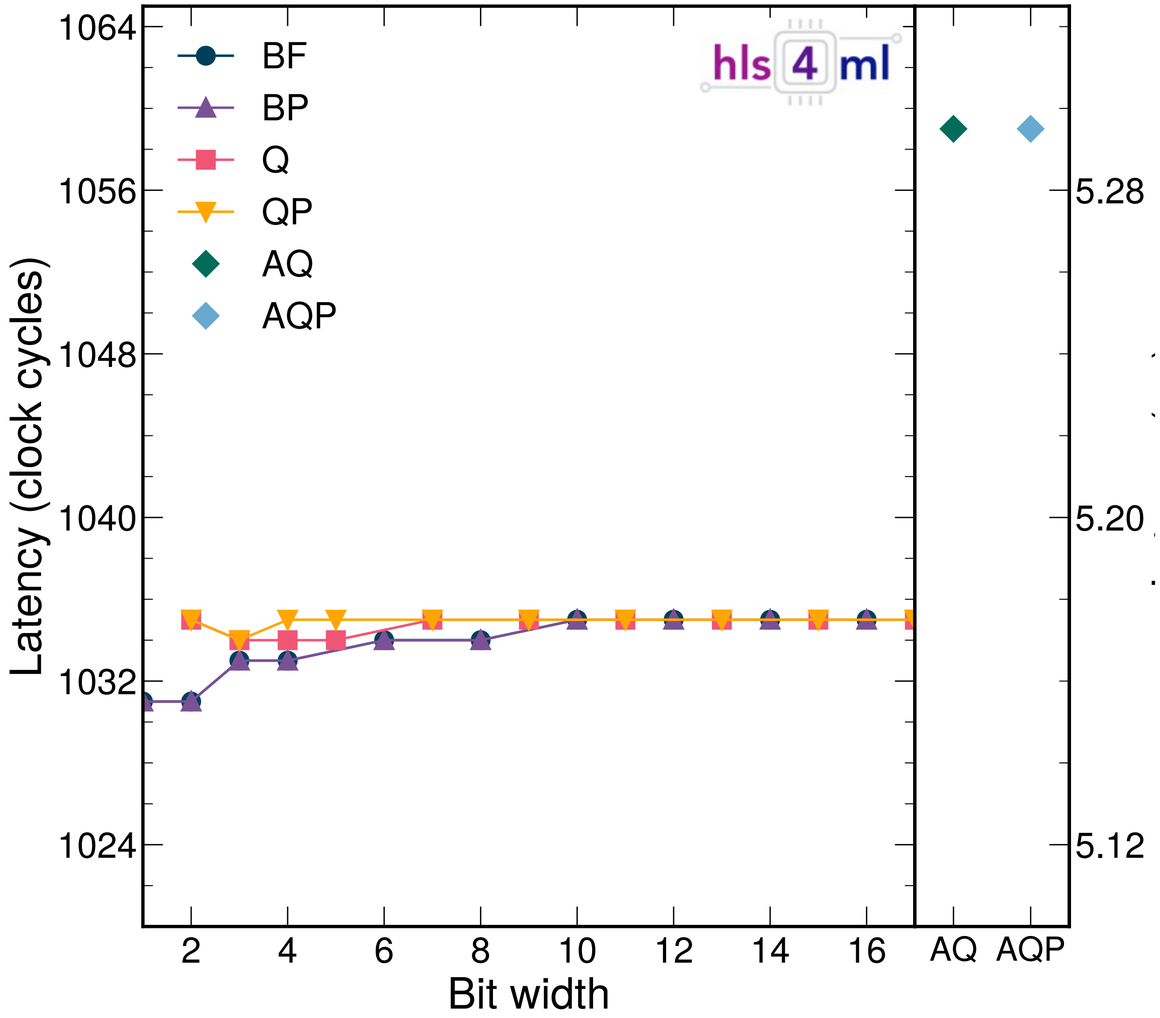




European Research Council

Fast CNN inference on FPGAs





Execution time reduced to 5 µsec to basically no accuracy loss down to 6 bits

GraphNets on FPGAs

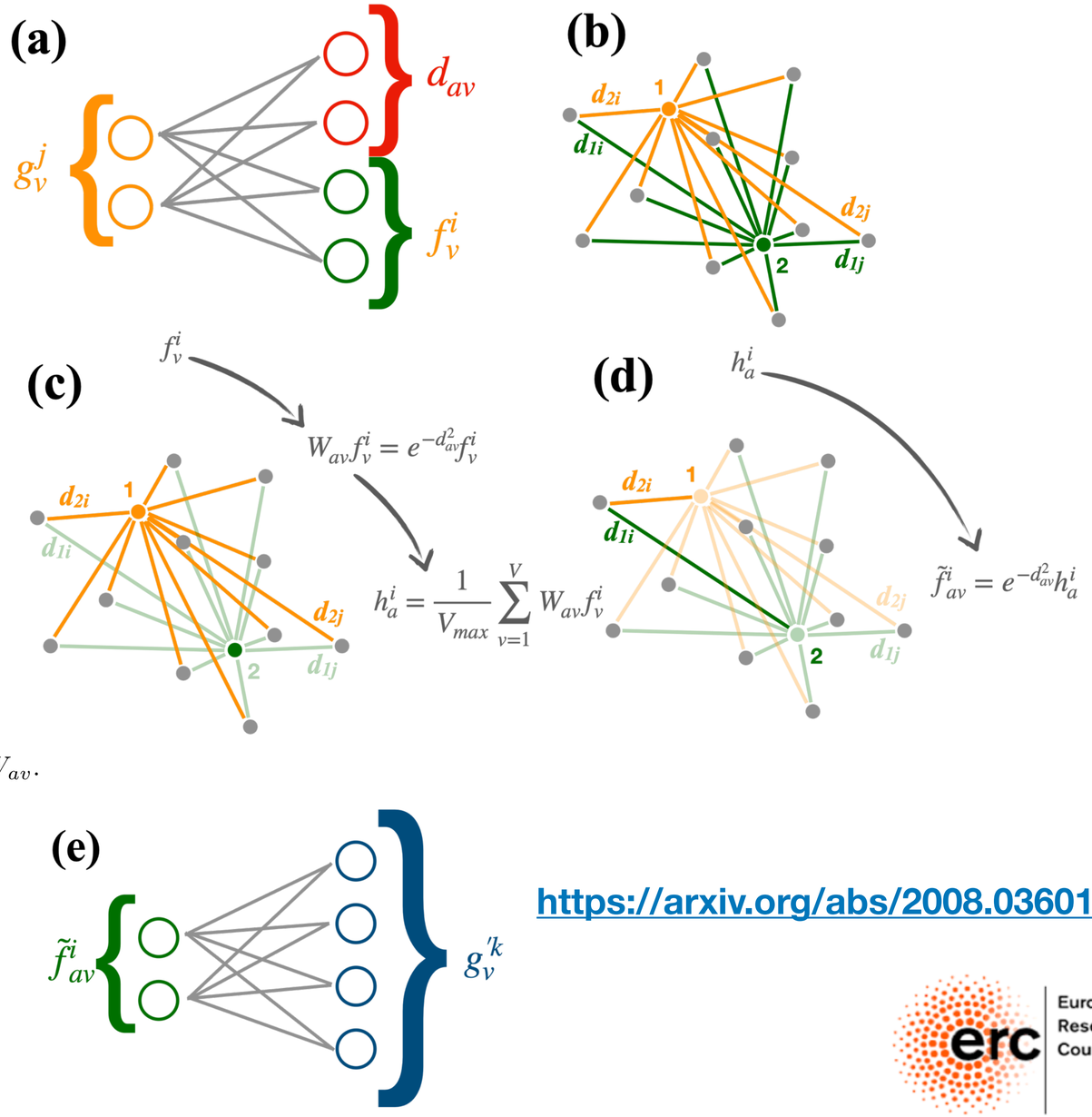
- *Slightly simplified version of GarNet*
- *Double message passing (vertices <-> aggregators) can be condensed in FPGA-friendly math*

$$g_v'^k = \sum_{a=1}^{S} W_{av} \left(\sum_{j=1}^{F_{\text{in}}} \tilde{w}_{ja}^k G_a^j + \tilde{b}_a^k L_a \right) + c^k,$$

$$\tilde{w}_{ja}^k = \sum_{i=1}^{F_{\text{LR}}} u_{ia}^k w_j^i, \quad \tilde{b}_a^k = \sum_{i=1}^{F_{\text{LR}}} u_{ia}^k b^i, \quad G_a^j = \frac{1}{V_{\max}} \sum_{v=1}^{V} W_{av} g_v^j, \text{ and } L_a = \frac{1}{V_{\max}} \sum_{v=1}^{V} W_{av}.$$

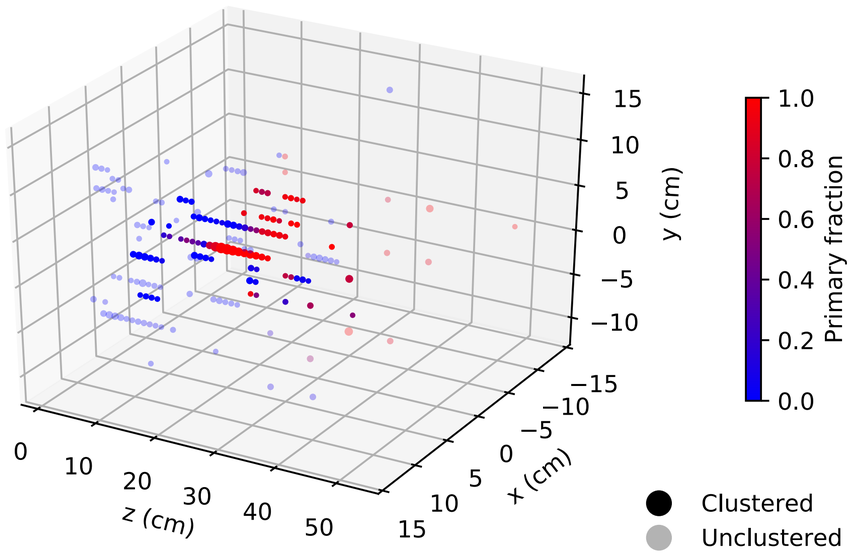
- *Non-linearity in distance weighting*

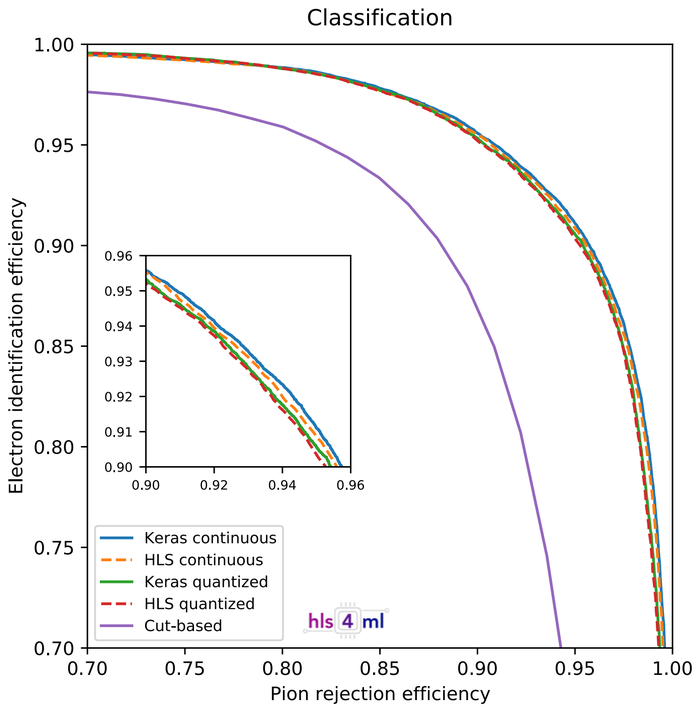
$$W_{av}' = \exp(-d_{av}^2)$$

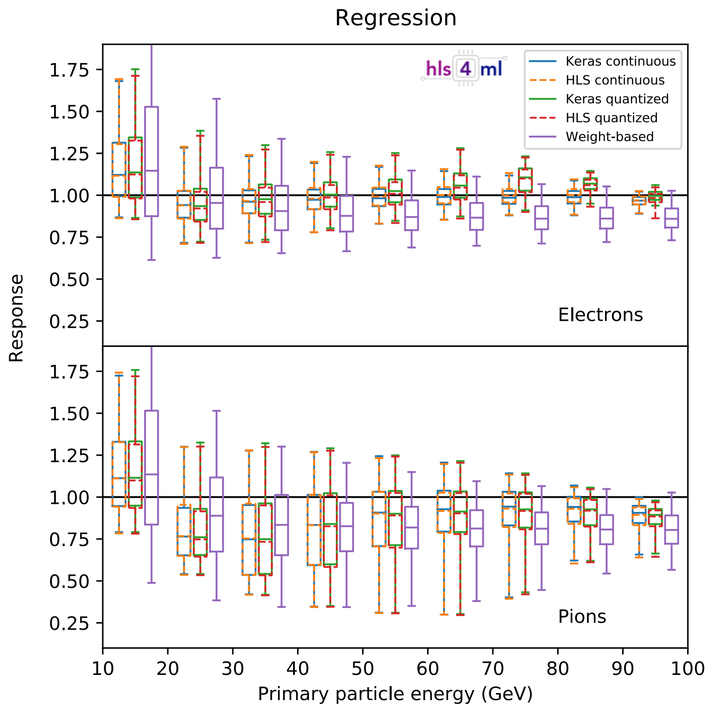


https://arxiv.org/abs/2008.03601

GraphNets on FPGAs







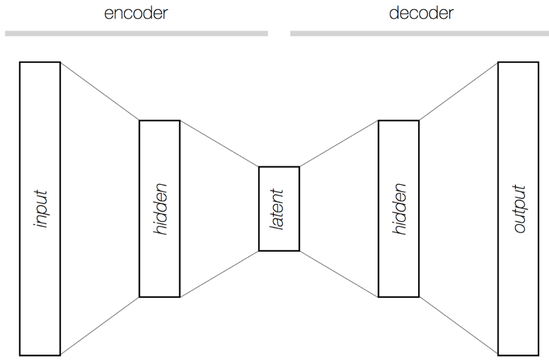
Model	$V_{\max}$	R_{reuse}	Latency (cycles)	Interval (cycles)	DSP (10^3)	LUT (10^3)	FF (10^3)	BRAM (Mb)	ROC AUC	Response RMS
Continuous	128	32	155	55	3.1 [56%]	57 [9%]	39 [2.9%]	1.8 [2.3%]	0.98	0.23
Quantized	128	32	148	50	1.6 [29%]	70 [11%]	41 [3.1%]	1.9 [2.4%]	0.98	0.24
Quantized	64	16	99	34	1.6 [29%]	63 [9%]	38 [2.9%]	1.8 [2.3%]	0.96	0.24
Quantized	32	8	75	26	1.4 [25%]	52 [8%]	33 [2.5%]	1.8 [2.3%]	0.86	0.37
Quantized	16	4	63	22	1.5 [27%]	57 [9%]	37 [2.8%]	1.8 [2.3%]	0.64	0.36

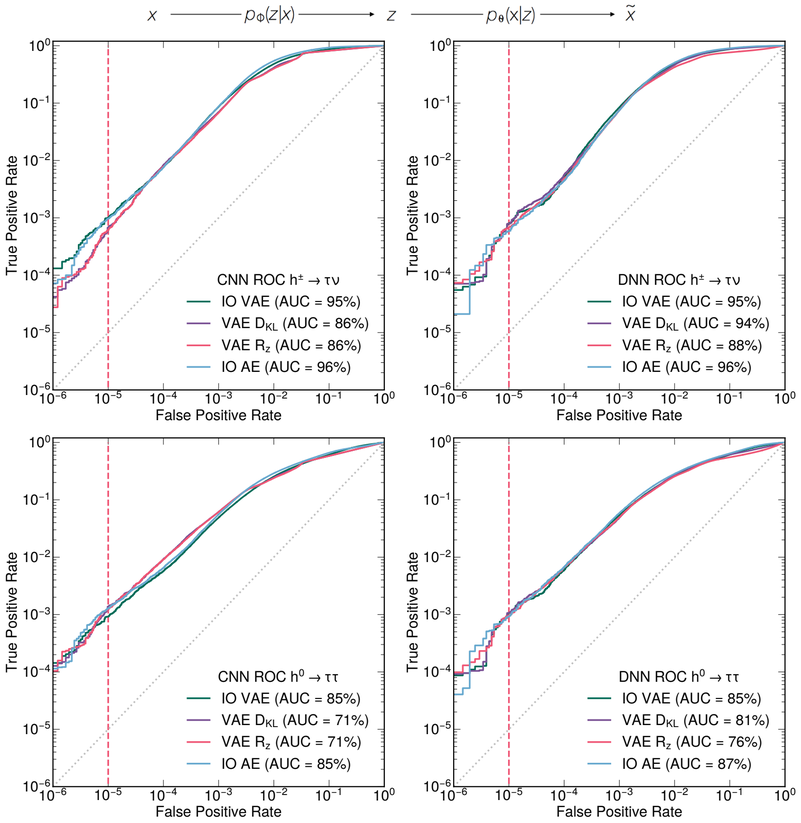
https://arxiv.org/abs/2008.03601

European Research Council

Anomaly Detection on FPGA

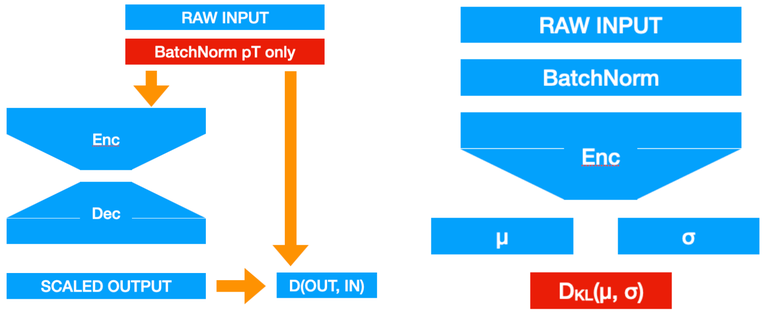
- *Autoencoders are compression-decompression algorithms that learn to describe a given dataset in terms of points in a lower-dimension latent space*
- *Can be used for anomaly detection: compression less effective when anomalous events are given*
- *In principle, same compression techniques apply*
- *In practice, special considerations apply*
  - *Figure of merit for anomaly detection might be different than loss -> QAT might run in trouble*
  - *Variational autoencoders require sampling from random numbers (possible, but with some resource utilisation)*
  - *Might need custom layers to evaluate custom AD figures of merit*



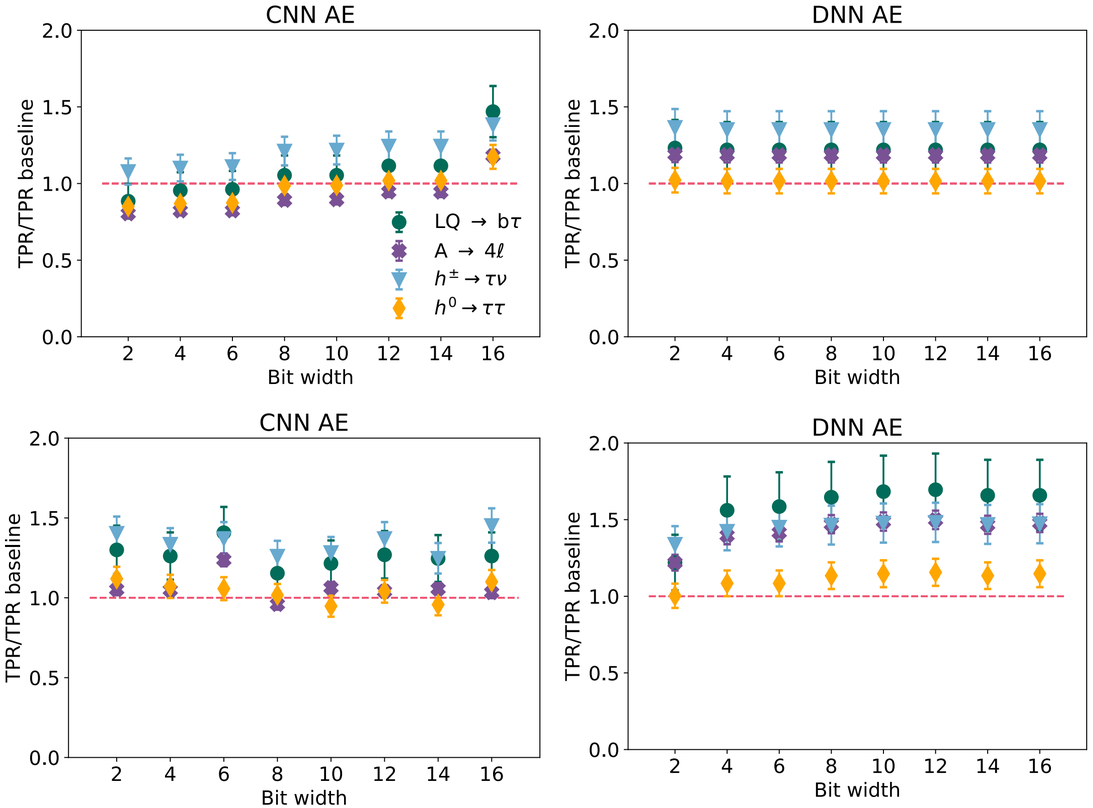


European Research Council

Anomaly Detection on FPGA

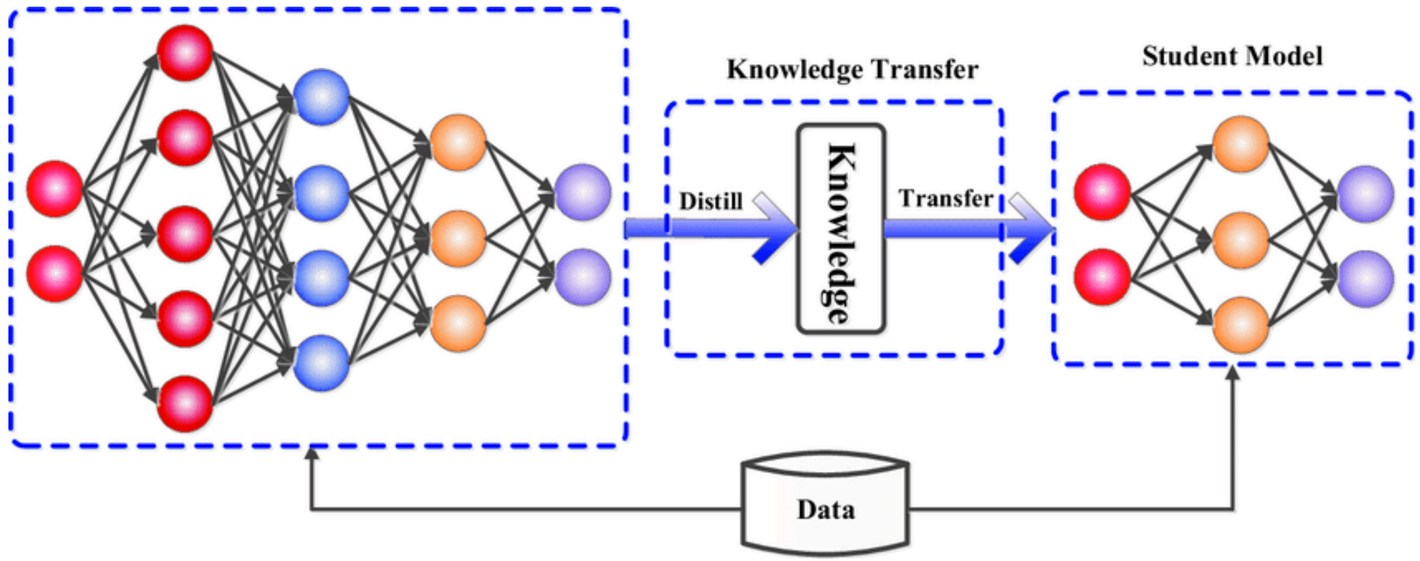


Model	DSP [%]	LUT [%]	FF [%]	BRAM [%]	Latency [ns]	II [ns]
DNN AE QAT 8 bits	2	5	1	0.5	130	5
CNN AE QAT 4 bits	8	47	5	6	1480	895
DNN VAE PTQ 8 bits	1	3	0.5	0.3	80	5
CNN VAE PTQ 8 bits	10	12	4	2	365	115



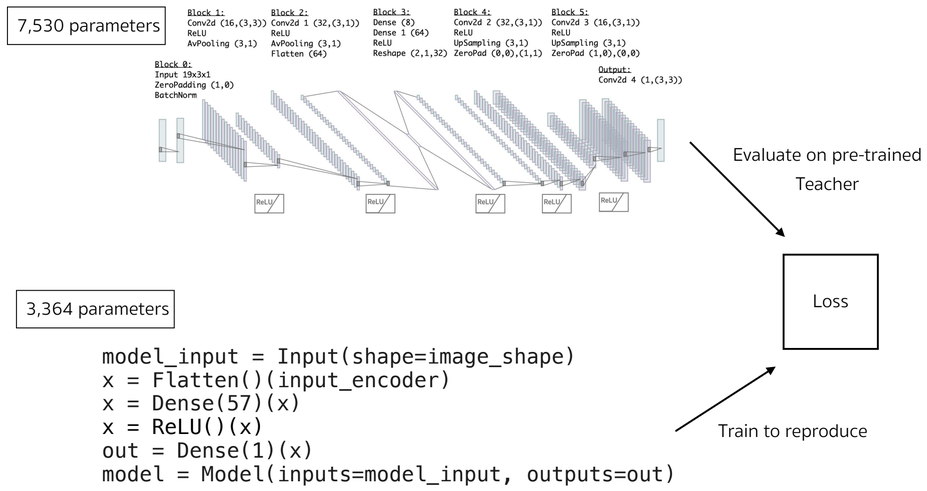
Knowledge Distillation

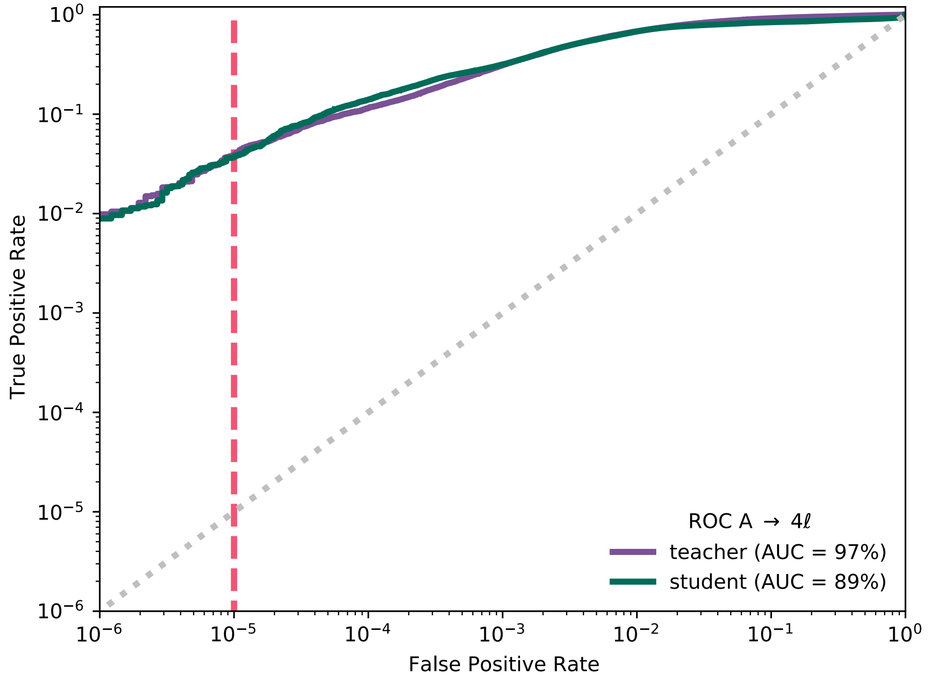
- *Sometimes the model is too big to be deployed*
- *Still, a network is a function that can be approximated by … a neural network (the student model)*
- *Every problem is turned into a regression*
  - *Easier to compress than unsupervised models*
  - *Does not require resource-consuming output activation for classifications*



Knowledge Distillation

- *Sometimes the model is too big to be deployed*
- *Still, a network is a function that can be approximated by … a neural network (the student model)*
- *Every problem is turned into a regression*
  - *Easier to compress than unsupervised models*
  - *Does not require resource-consuming output activation for classifications*





hls4ml: An Open-Source Codesign Workflow to Empower Scientific Low-Power Machine Learning Devices

Farah Fahim*
Benjamin Hawks
Christian Herwig
James Hirschauer
Sergo Jindariani
Nhan Tran*
Fermilab
Batavia, IL, USA

Luca P. Carloni
Giuseppe Di Guglielmo
Columbia University
New York, NY, USA

Philip Harris
Jeffrey Krupa
Dylan Rankin
MIT
Cambridge, MA, USA

Manuel Blanco Valentin
Josiah Hester
Yingyi Luo
John Mamish
Seda Orgrenci-Memik
Northwestern University
Evanston, IL, USA

Thea Aarrestad
Hamza Javed
Vladimir Loncar
Maurizio Pierini
Adrian Alan Pol
Sioni Summers
European Organization for Nuclear Research (CERN)
Geneva, Switzerland

Javier Duarte
UC San Diego
La Jolla, CA, USA
jduarte@ucsd.edu

Scott Hauck
Shih-Chieh Hsu
University of Washington
Seattle, WA, USA

Jennifer Ngadiuba
Caltech
Pasadena, CA, USA

Mia Liu
Purdue University
West Lafayette, IN, USA

Duc Hoang
Rhodes College
Memphis, TN, USA

Edward Kreinar
HawkEye360
Herndon, VA, USA

Zhenbin Wu
University of Illinois at Chicago
Chicago, IL, USA

https://github.com/hls-fpga-machine-learning/hls4ml

https://hls-fpga-machine-learning.github.io/hls4ml/

https://github.com/thesps/conifer

arXiv.org Full list of papers here

European Research Council