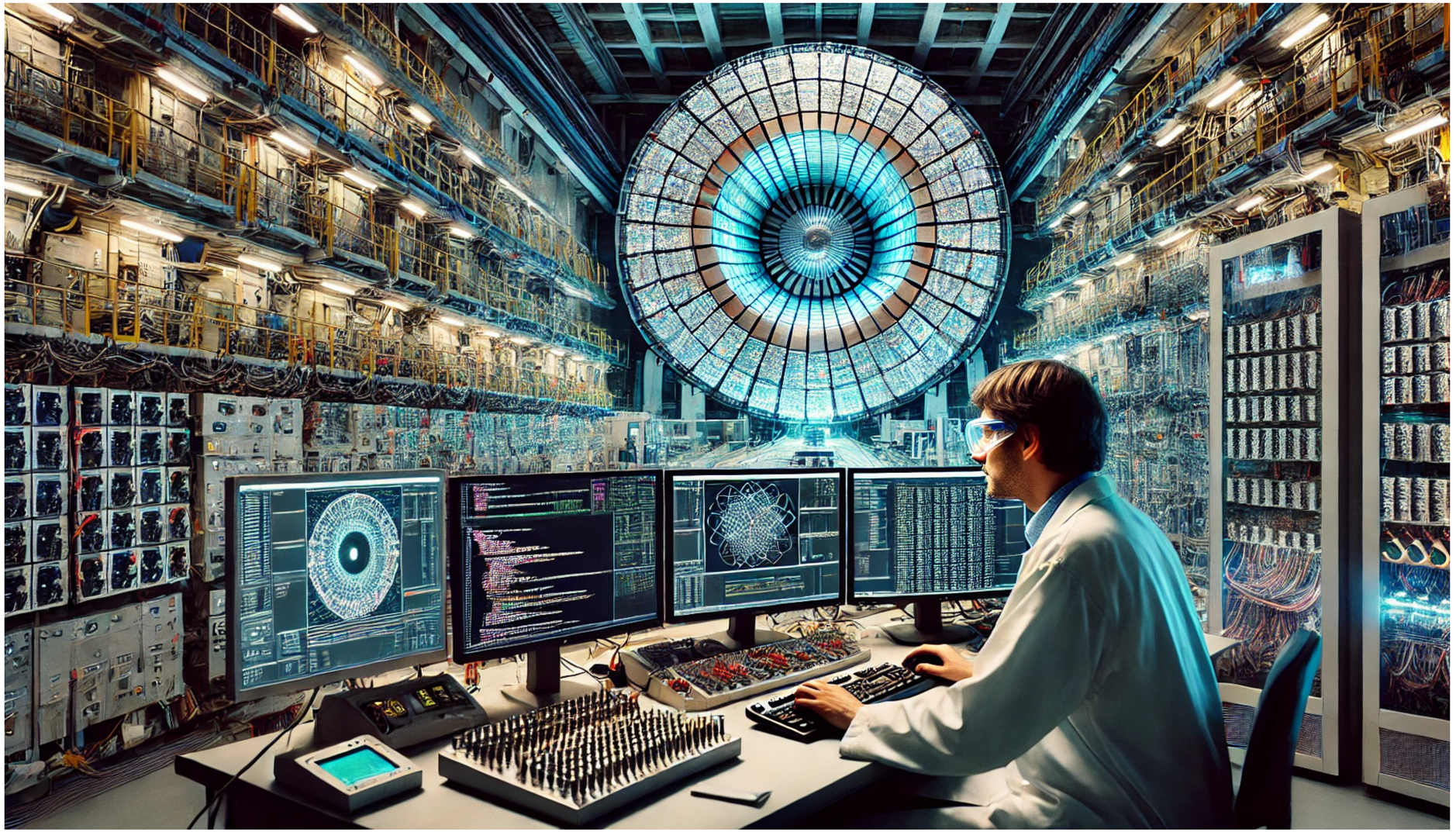
Lecture 11: Network Compression

Maurizio Pierini (CERN), Thea Aarrestad (ETH)

Program for Today

Date	Topic	Tutorial
Sep 16	Intro & class description	
Sep 23	probability Theory + Basic of machine learning + Dense NN	*Basic jupyter + DNN on mnist (give jet dnn as homework)*
Sep 30	Statistics + Convolutional NN	*Convolutional NNs with MNIST*
Oct 7	Training in practice: regularization, optimization, etc	*Free Practice*
Oct 14	Unsupervised learning and anomaly detection	*Autoencoders with MNIST*
Oct 21	Graph NNs	*Tutorial on Graph NNs*
Oct 28	Transformers	
Nov 4	Network compression (pruning, quantization, Knoweldge Distillation)	
Nov 11	Tutorial on hls4ml	
Nov 18	Generative models: GANs, VAEs, etc	*TBD*
Nov 25	Normalizing flows	*TBD*
Dec 2	To Be Decided	
Dec 9	Quantum Machine Learning	
Dec 16	**Exams To Be Confirmed**	

Today We Talk About Real Life



Create an image of a scientist deploying a Neural Network on the trigger system of one of the LHC experiments

Paradigm for AI inference

- Most of AI inference happens on the cloud
  - Some data is collected
  - Data are transferred to a data centre, where they are processed by an AI model
  - The answer is shipped back
- This can happen in real time (e.g., AI assistants on phone) or not (e.g., LHC data analysis)
- In these cases, the relevant metric of performance is accuracy
  - The available computational power is not a concern in most of the cases (reasoning agents being an exception)
  - The latency is dominated by data transfer and can stay between O(msec)-O(sec) without any issue

Inference on Edge

- *Certain applications cannot proceed this way*
  - *Safety demands faster reaction and cannot depend on availability of internet connection (e.g., self-driving cars)*
  - *Data cannot be moved (e.g., privacy concerns on medical data)*
  - *Inference happens in remote locations with costs data transfer (e.g., underground experiments, satellites, refugee camps in combat zones, etc.)*
- *In these cases, the inference has to happen on-edge, i.e., on the device collecting the data*

Pros of on-edge inference

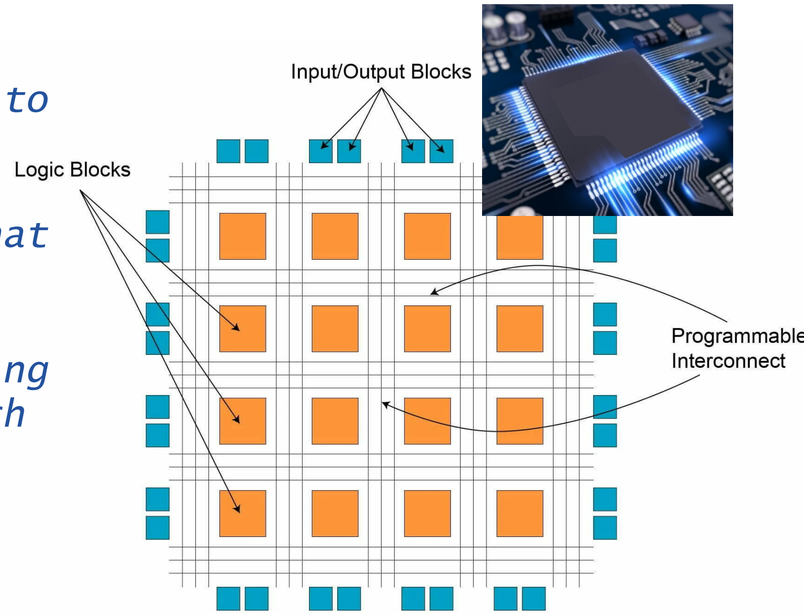
- *Fast response: one avoids the delay due to data transfer*
- *No need for an internet connection: not a negligible issue when dealing with portable devices etc, for what concerns device cost, power consumption & operation costs*
- *User owns the data: not to be neglected in an epoch of data privacy concerns*
- *CO_2 impact: on-edge devices are typically low-power devices. They don't require operating huge cooling plants 24/7, etc.*

Cons of on-edge inference

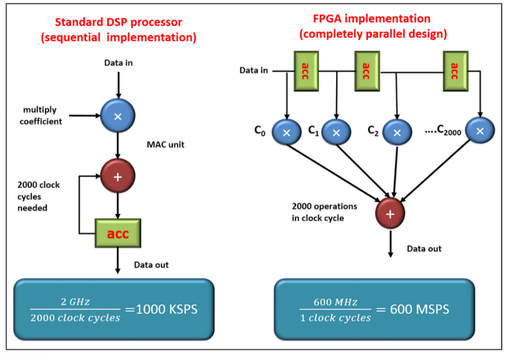
- *On-edge inference comes with computing constraints*
  - *No access to complex processing systems (GPUs, CPUs, etc)*
  - *Mostly relying on low-capacity devices that can do limited calculations (e.g., portable devices, IoT, etc.)*
- *This puts constraints on the kind of applications one can support*
  - *Small models*
  - *Large models, if computation spread across time (i.e., slowing down response)*

FPGAs

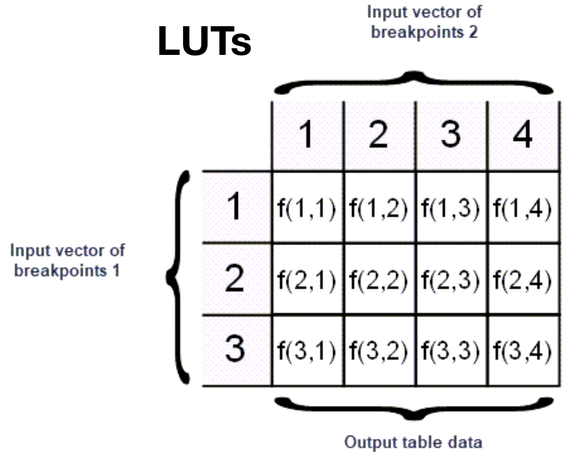
- *Field-programmable gate arrays (FPGAs) are integrated circuits that that can be reconfigured to meet designers' needs*
- *An FPGA consists of an array of computing units that can be integrated on an electronic board*
- *Programming and FPGA consists in setting the routing of data between units and the computation that each unit performs*
  - *DSPs: Digital Signal Processing unit, a programmable block of mathematical computation (e.g., multiplications) on a given input*
  - *LUT: Look up tables, a tabular approximation of some function*
  - *FF: flip flops register a given input (moved to output) only at a clock rising edge. This allows to keep computation in sync (what comes first, what comes after) and makes FPGA processing possible*
  - *BRAM: the memory blocks on the card*



parallel DSPs in an FPGA

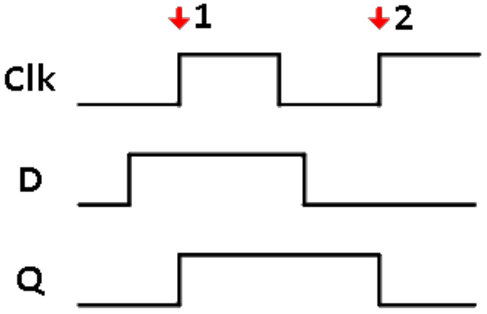


LUTs



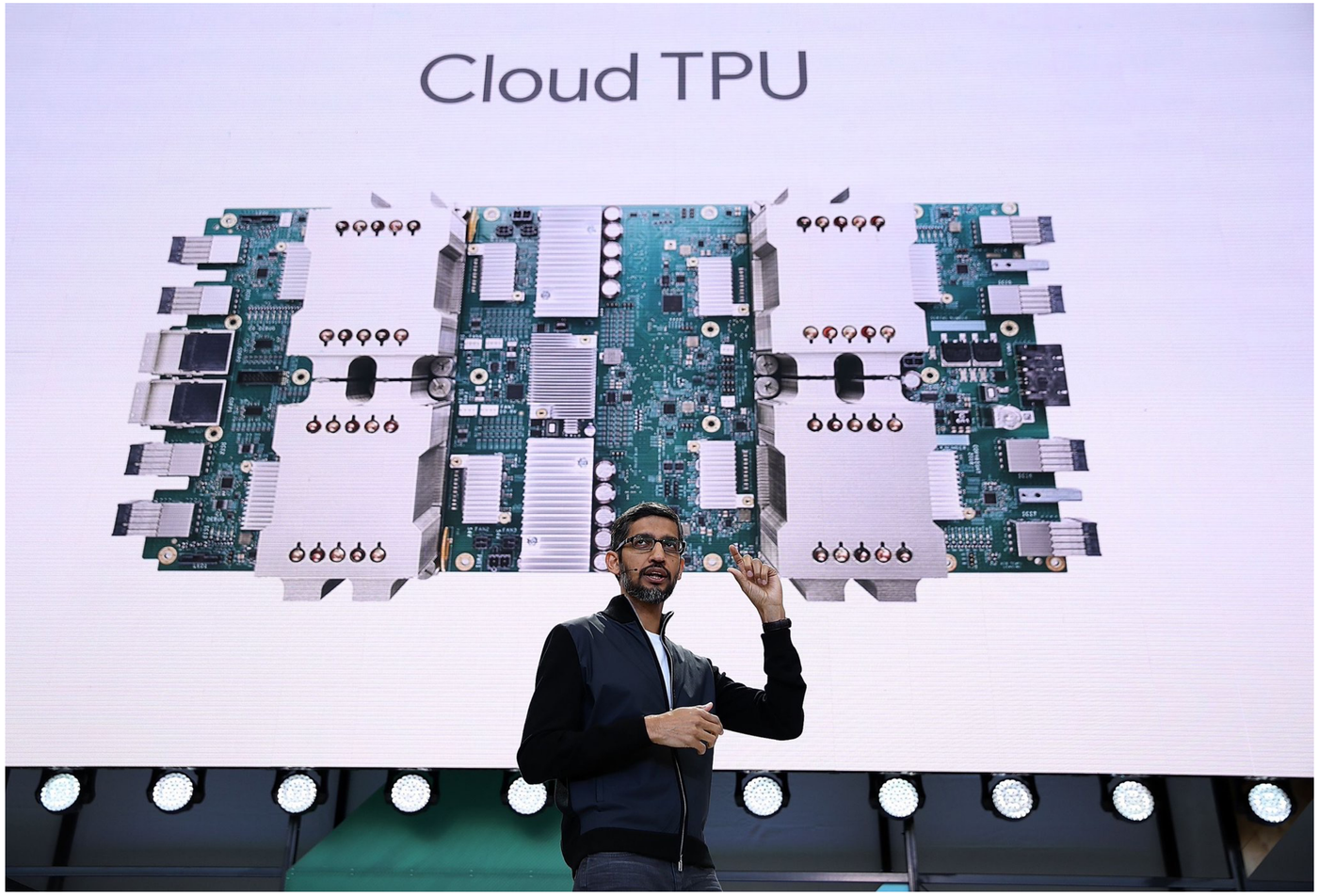
	1	2	3	4
1	f(1,1)	f(1,2)	f(1,3)	f(1,4)
2	f(2,1)	f(2,2)	f(2,3)	f(2,4)
3	f(3,1)	f(3,2)	f(3,3)	f(3,4)

Flip Flops



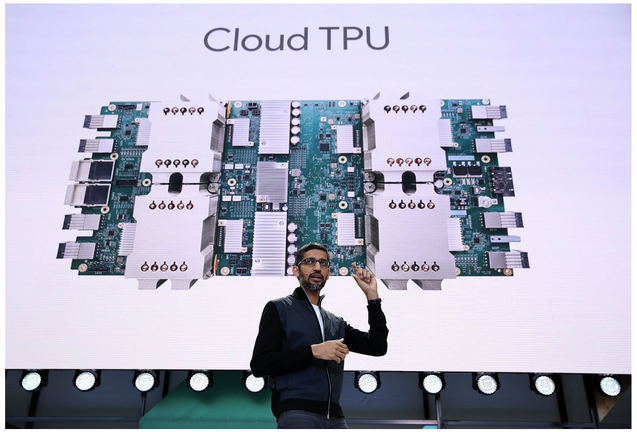
ASICs

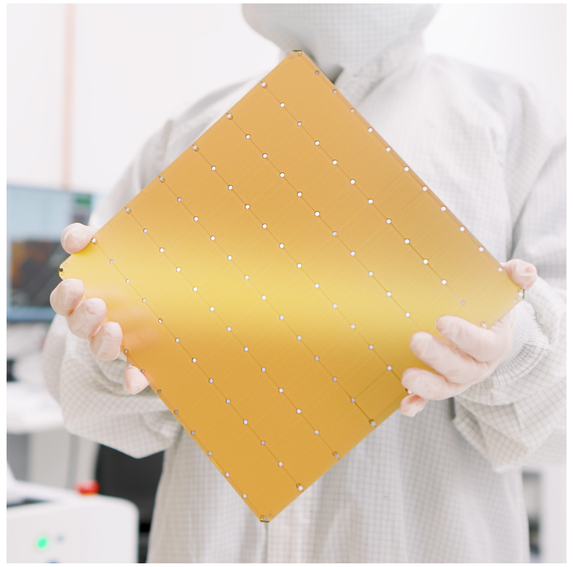
- *FPGAs are emulators of logic circuits*
  - *They can be reprogrammed at need, adding flexibility*
- *But sometimes one does not need that flexibility*
  - *It could be economically convenient to print the logic circuit as an Application-specific integrated circuit (ASIC), e.g., with cars*
- *Advantages:*
  - *Tailored to application, typically faster and more performing*
  - *Logic can be changed (to some extent) by programmed (e.g., one can change parameters in mathematical operations, e.g., a DNN layer)*
- *Disadvantages:*
  - *Design more complicated*
  - *To apply big logic changes, one has to reprint the ASIC (expensive)*

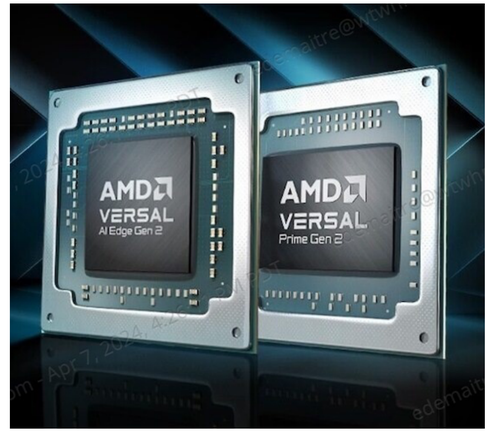


New Technologies

- *The need of AI on edge initiated a proliferation of new technologies dedicated to AI inference*
  - *Google TPUs*
  - *GraphCore IPUs*
  - *Cerebras WSEs*
  - *Xilinx/AMD Versal*
- *Most of these devices are also excellent for training*
- *This new trend might be defining a post-GPU era for NNs*
- *This is calling for hardware-universal approaches to perform on-edge inference (and training?)*

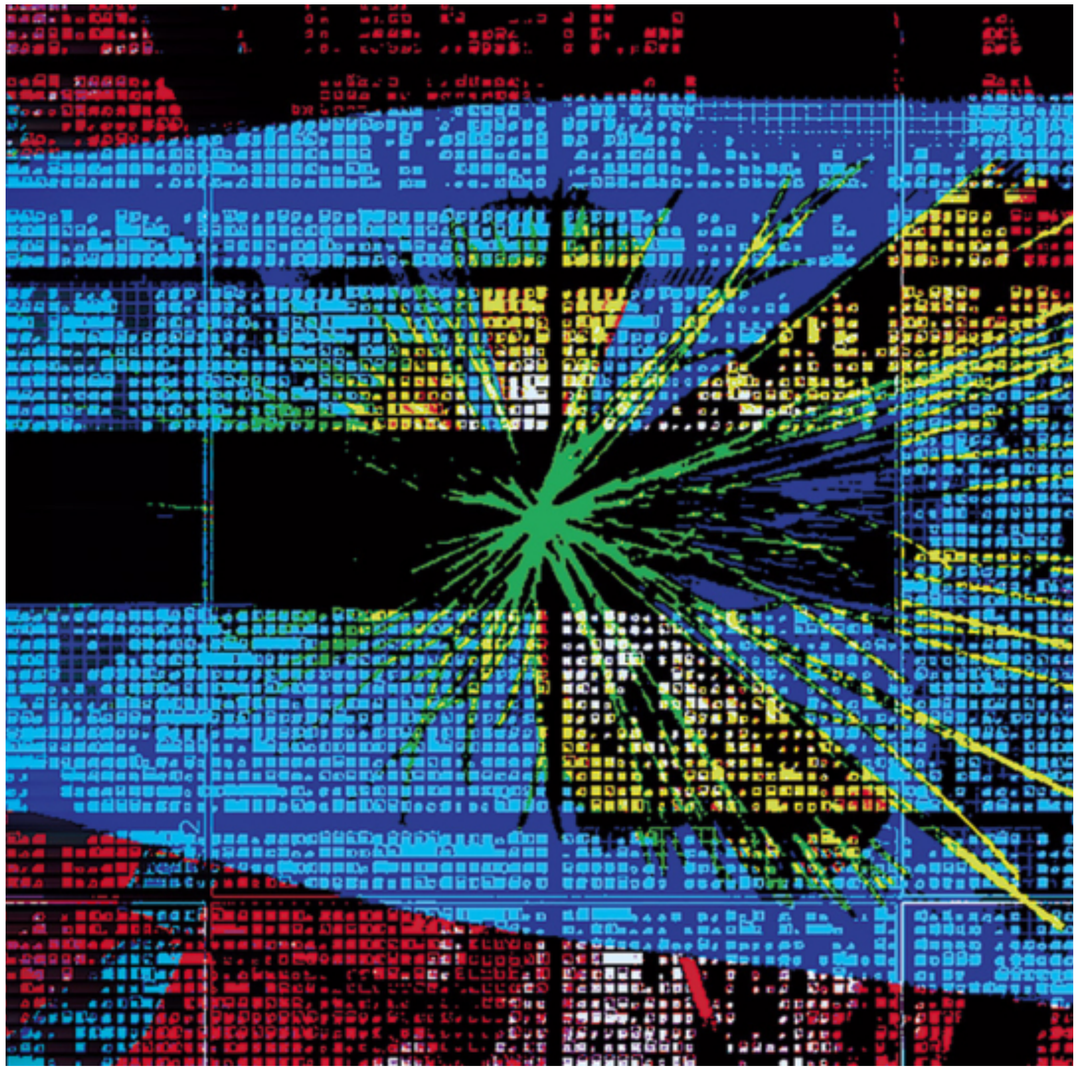




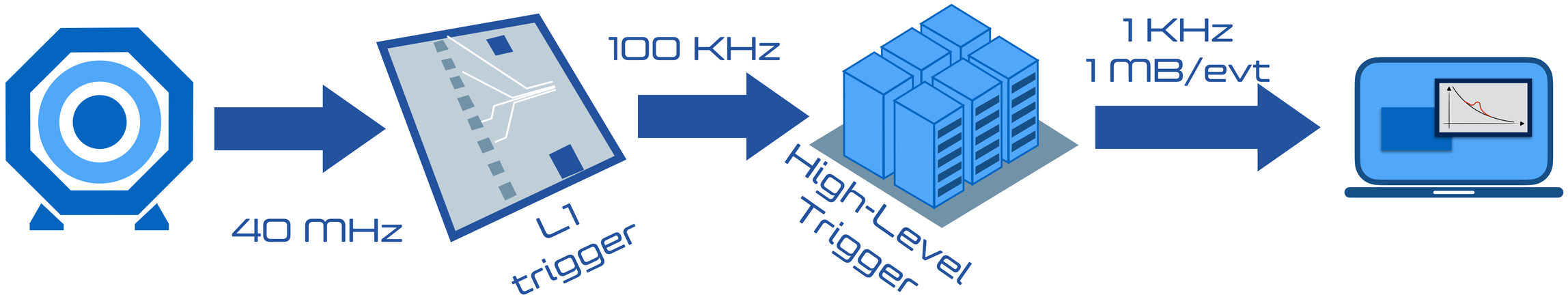


A use case: LHC

- *The problem*
  - *LHC Big Data problem and the need for algorithm complexity on edge*
- *The solution*
  - *Deep Neural networks as shortcut*
- *The Challenge*
  - *Executing Neural Networks on FPGAs as logic circuits*
- *The implementation*
  - *hls4ml network -> firmware conversion*
  - *Model compression techniques*
  - *Applications: Jet Tagging, Momentum regression, Anomaly Detection*



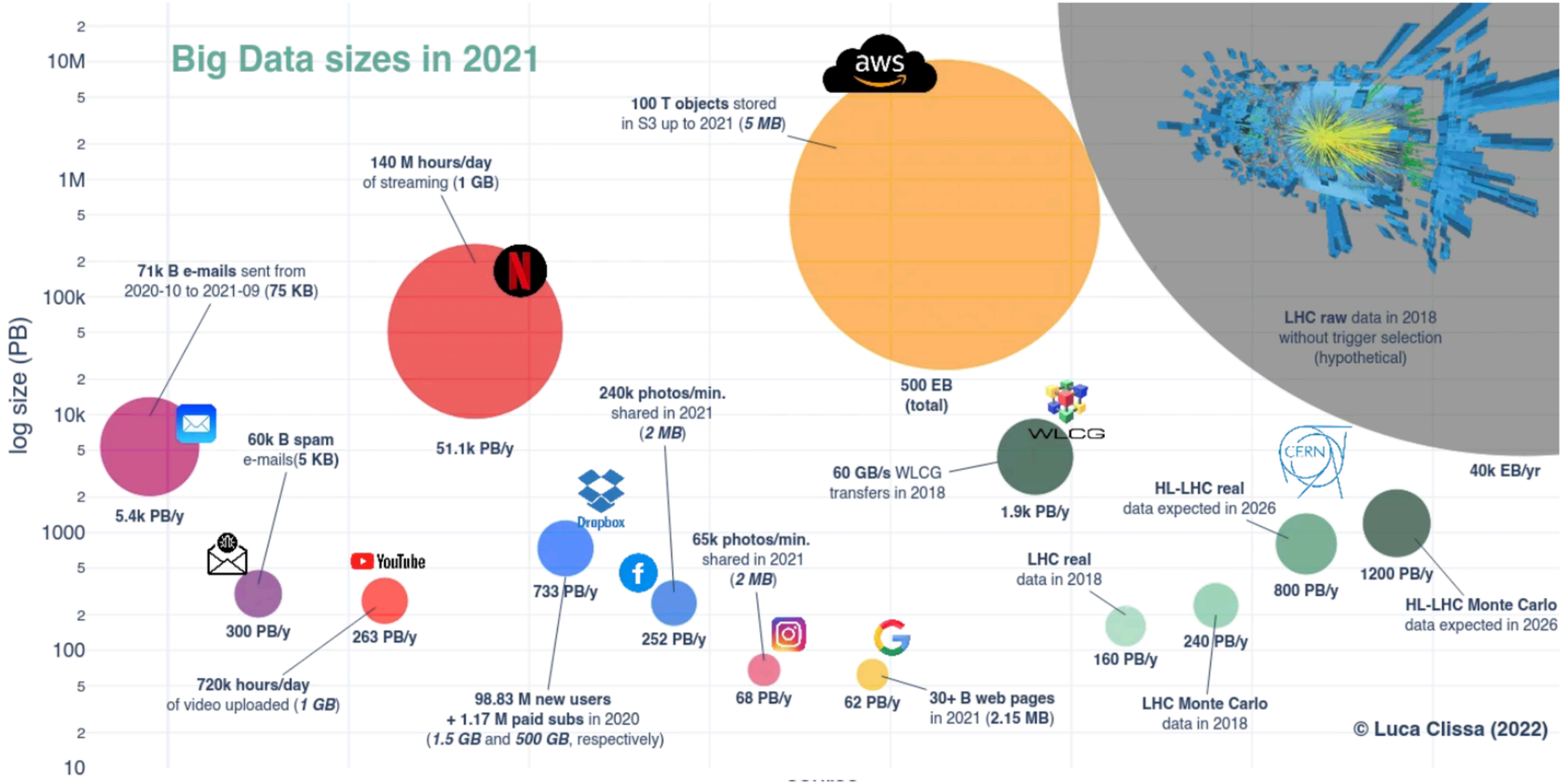
The LHC Big Data Problem



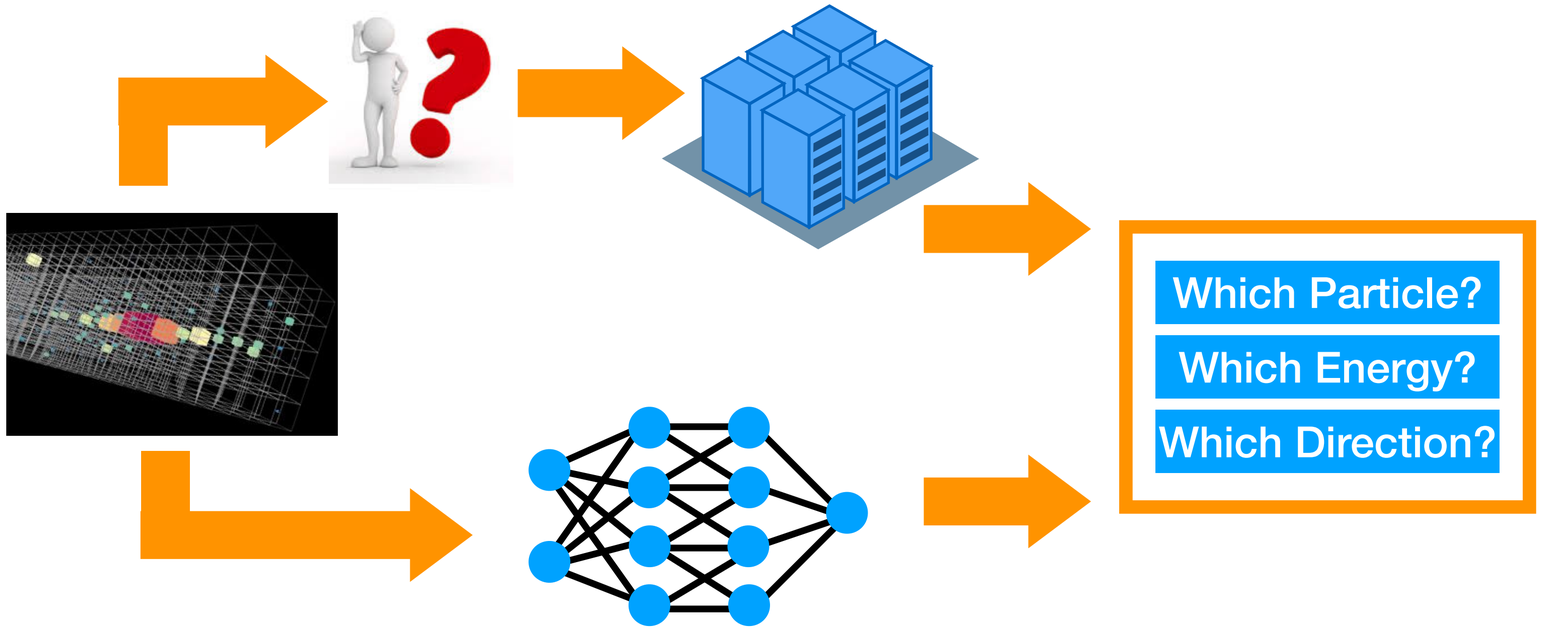
- *The LHC delivers O(10M) events/sec. This gives us little time (<10 μsec) to reconstruct and filter these events*
  - *Coarse reconstruction, with worse resolution than offline*
  - *Simple algorithms, e.g., sums and products (at some point, deploying invariant mass trigger was a big deal)*
  - *Extended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

Two Challenges in one

- *The largest data volume humankind had to handle*
  - *So large that we can't store it*
- *One of the hardest processing latency constraints*
  - *First data processing requires single-task algorithms to run in O(100) nsec*



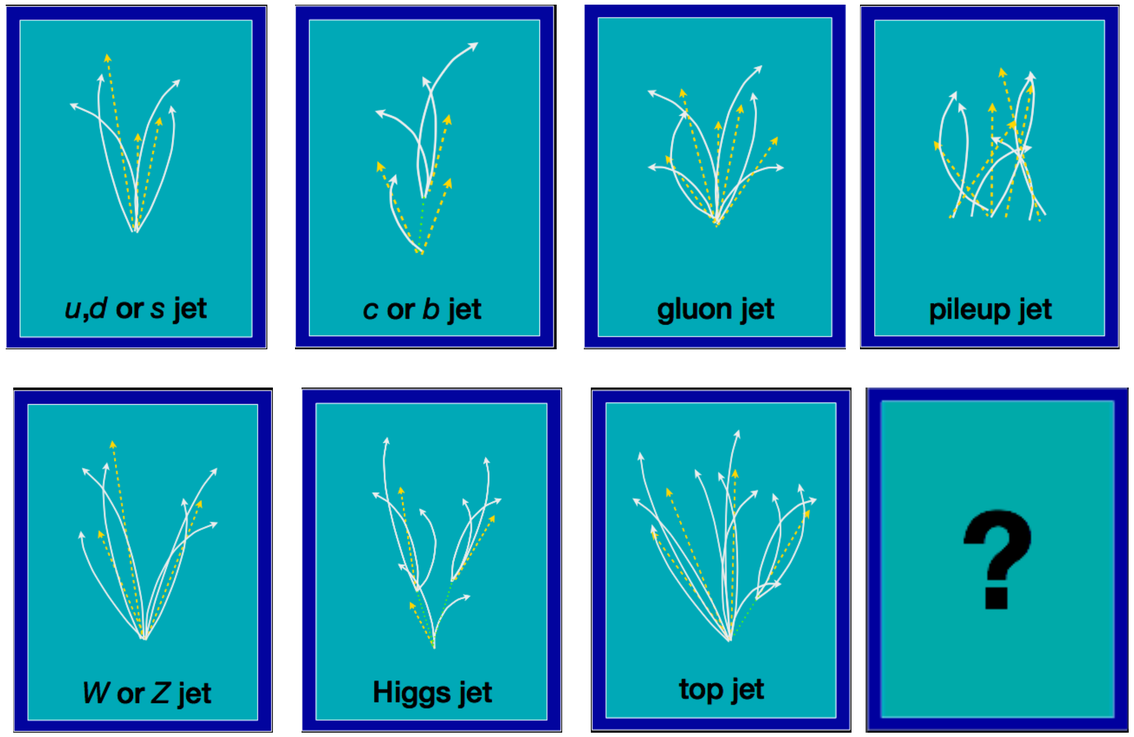
Neural Networks as a Solution



- *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?*

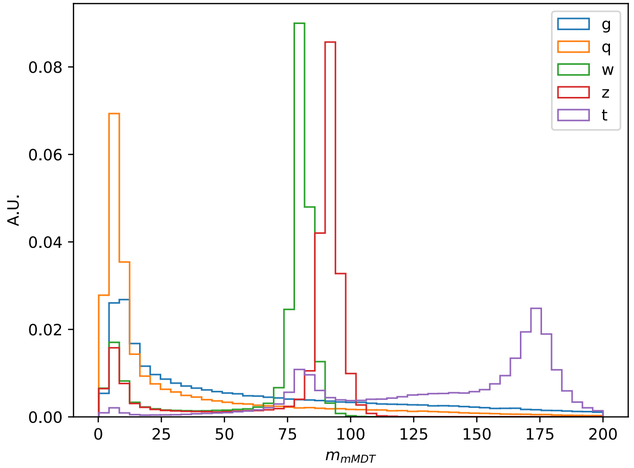
Example: fast inference

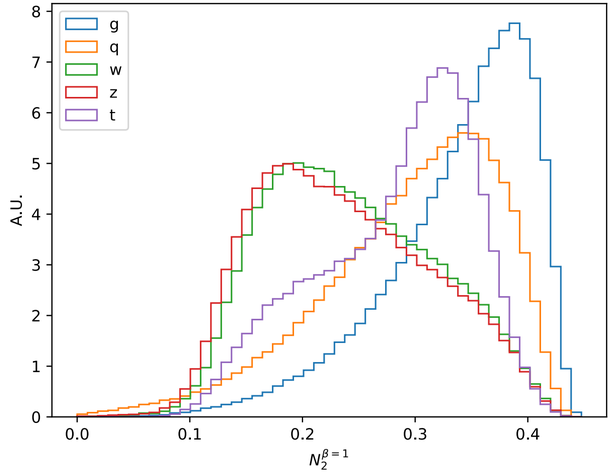
- *You have a jet at LHC: spray of hadrons coming from a "shower" initiated by a fundamental particle of some kind (quark, gluon, W/Z/H bosons, top quark)*
- *You have a set of jet features whose distribution depends on the nature of the initial particle*
- *You can train a network to start from the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

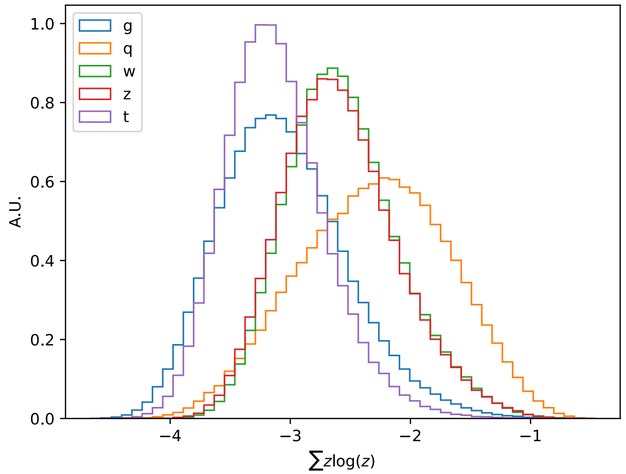


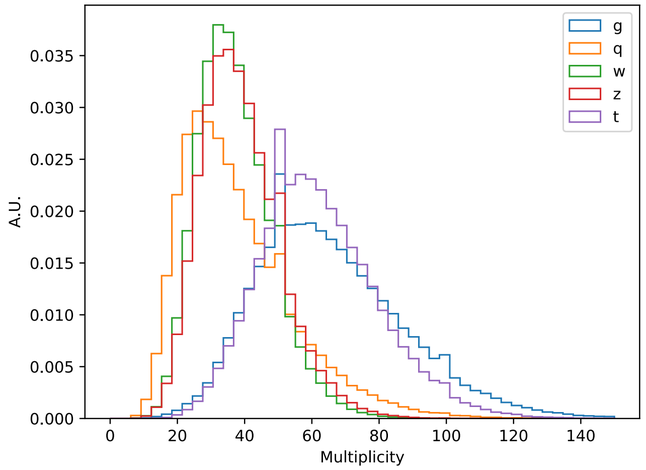
Example: jet tagging

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



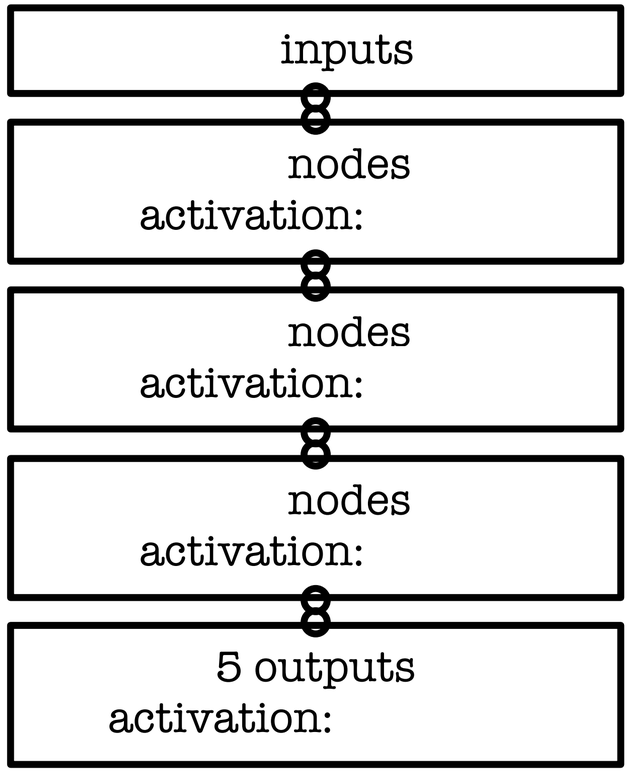


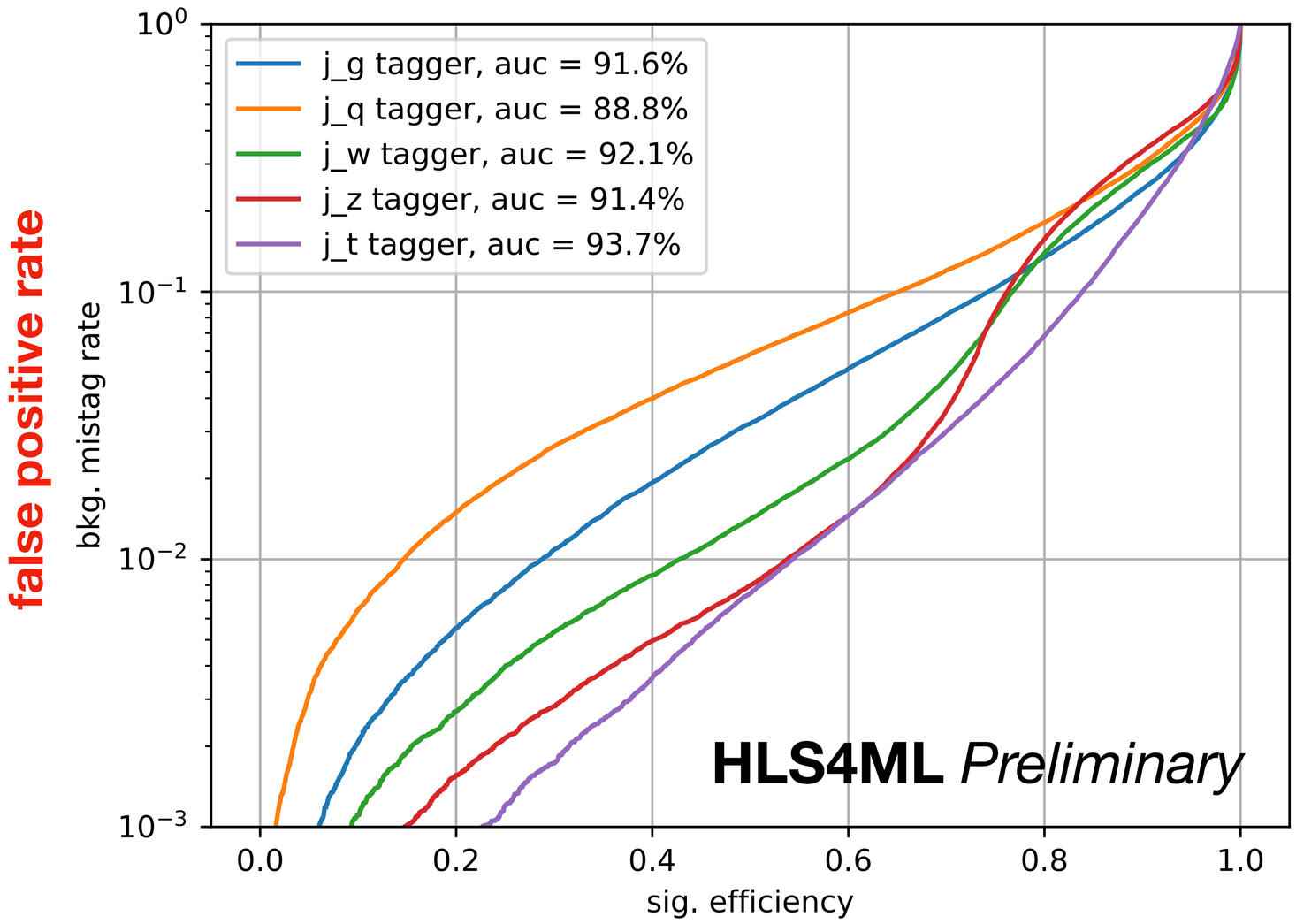




Example: jet tagging

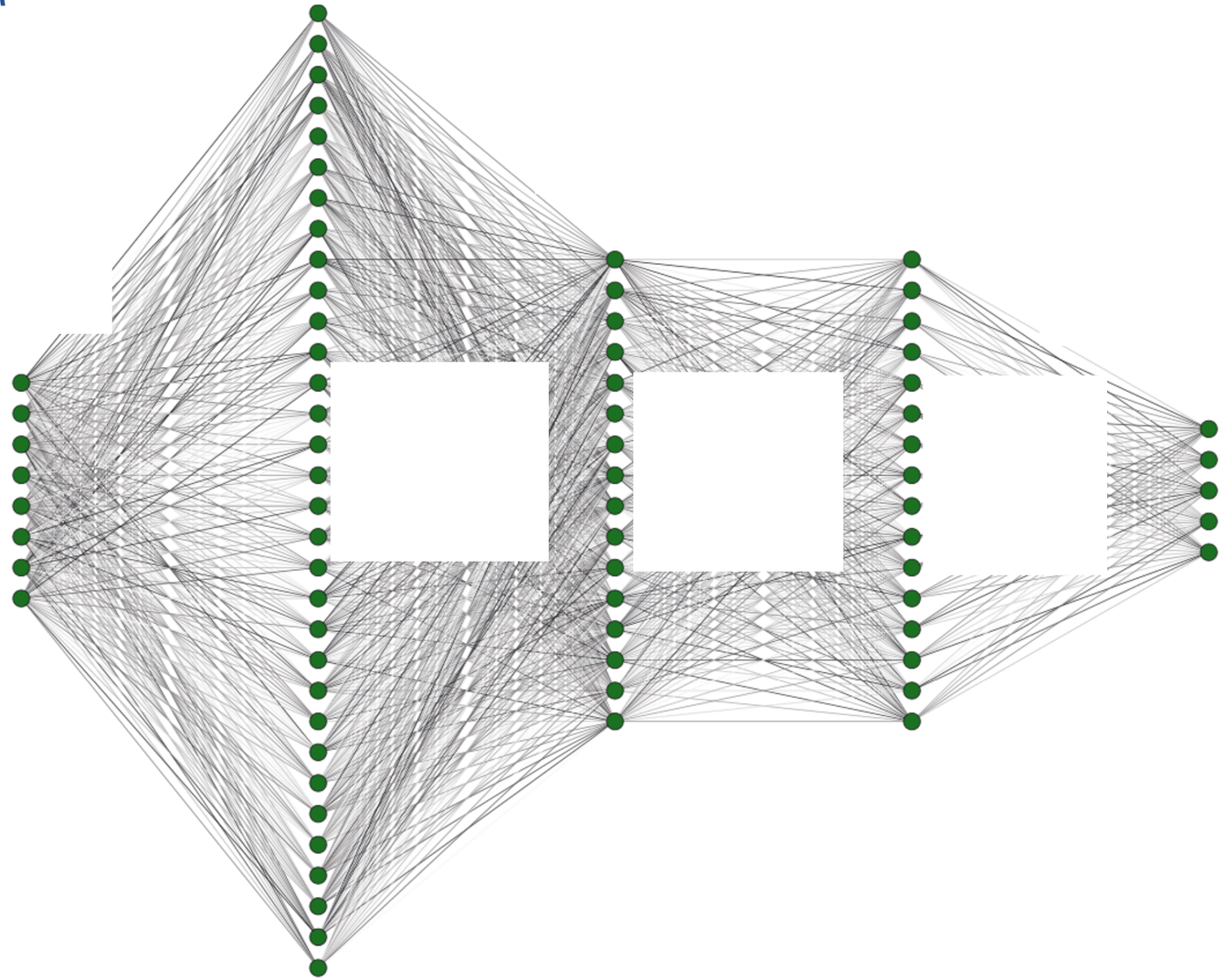
- *Simple DNN based on high-level features (jet masses, multiplicities, energy correlation functions)*





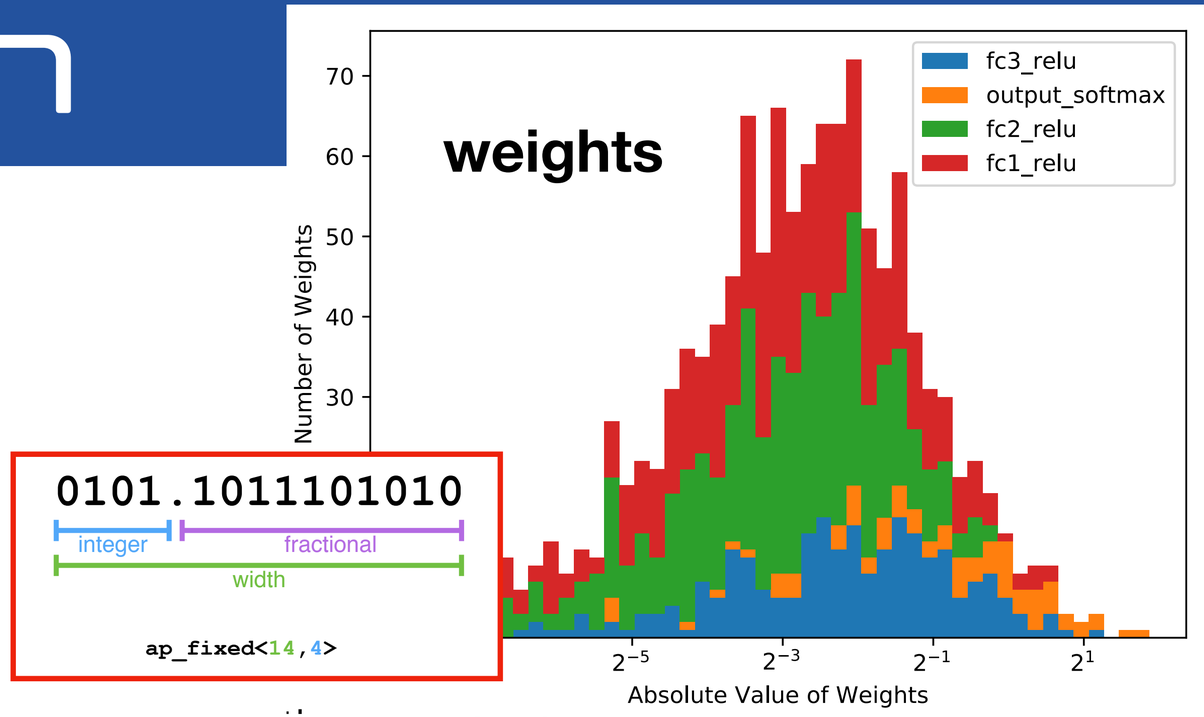
The full model

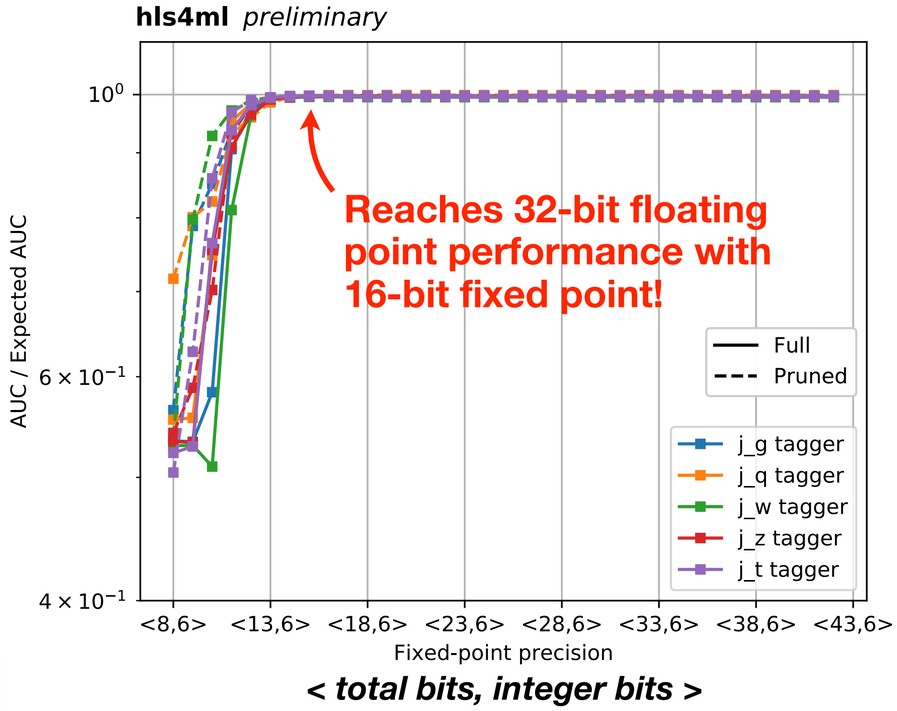
- *Even a relatively small neural network is too big to fit an FPGA*
- *Two solutions*
  - *Use the FPGA as the accelerator device of a CPU, executing the network by pieces*
    - *Can host big models, but execution is slowed down by CPU <-> FPGA communication*
  - *Deploy the whole network on the FPGA*
    - *Fast approach*
    - *But one has to fit the model in the FPGA somehow*



Quantization

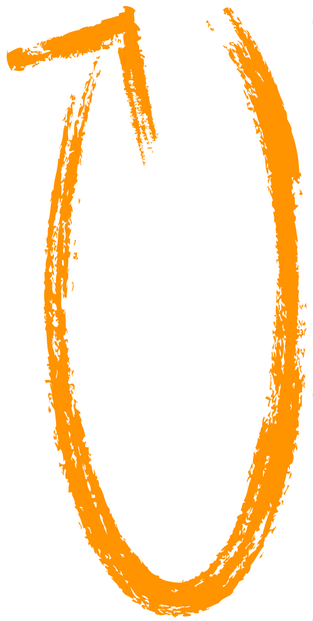
- ***Quantization:*** *reduce the number of bits used to represent numbers (i.e., reduce used memory)*
  - *models are usually trained at 64 or 32 bits*
  - *this is not necessarily needed in real life*
- *In our case, we could reduce to 16 bits w/o loosing precision*
- *Beyond that, one would have to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



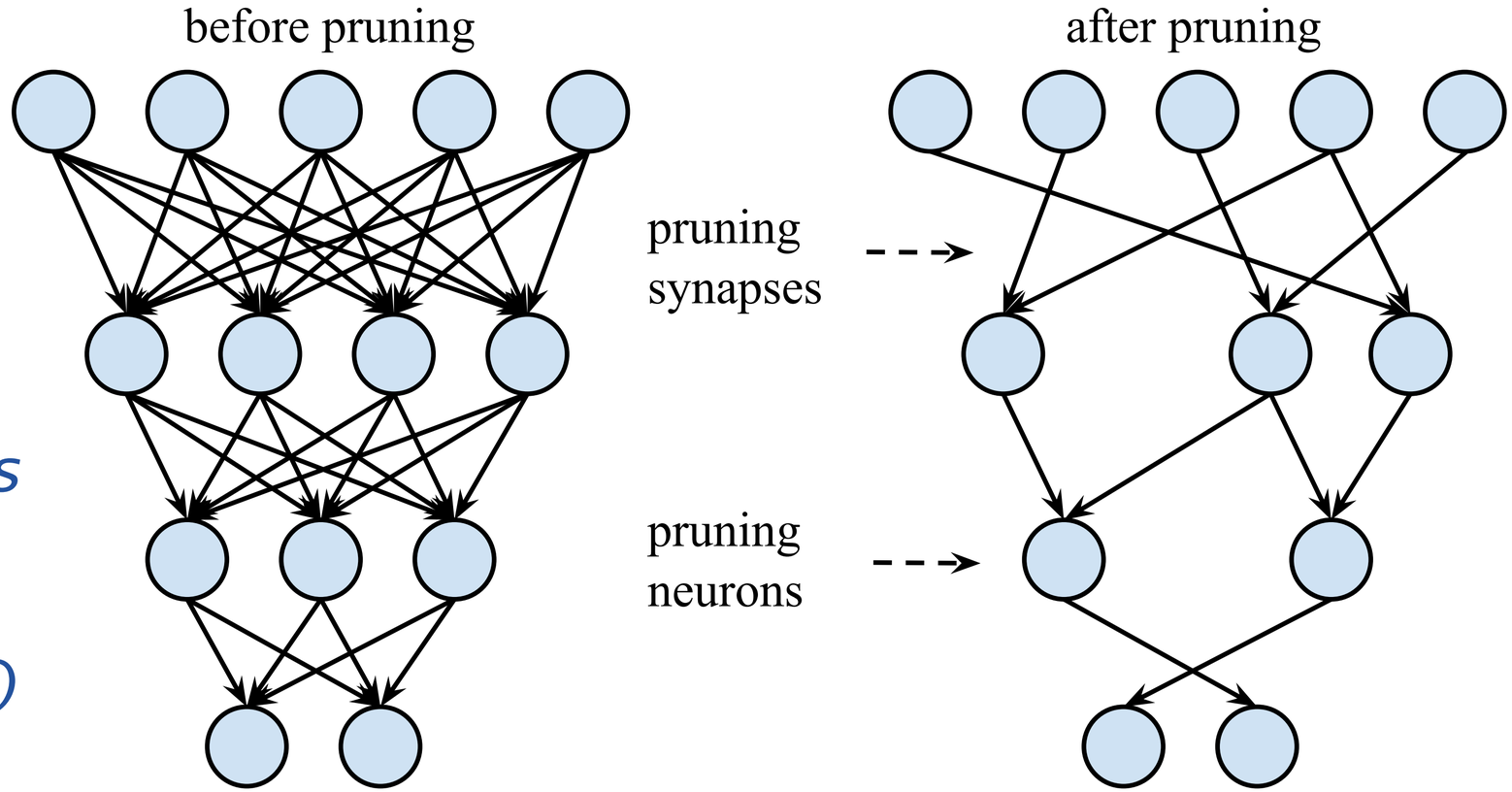


Making the NN smaller: pruning

- ***Pruning:*** *remove parameters that don't really contribute to performances*



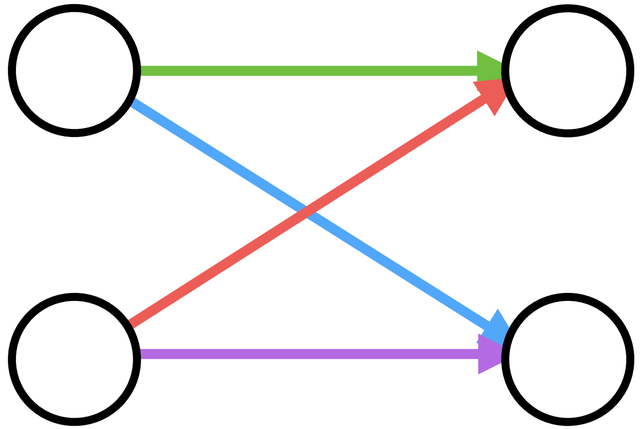
- *force parameters to be as small as possible (regularization)*
- *Remove the small parameters*
- *Retrain*

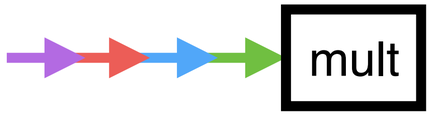


70% reduction of weights and multiplications w/o observing any performance loss

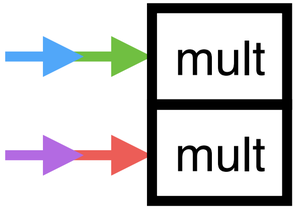
Parallelization

- `ReuseFactor`: how much to parallelize

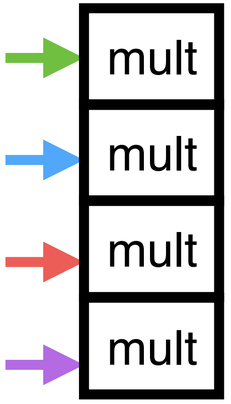




reuse = 4
use 1 multiplier 4 times



reuse = 2
use 2 multipliers 2 times each

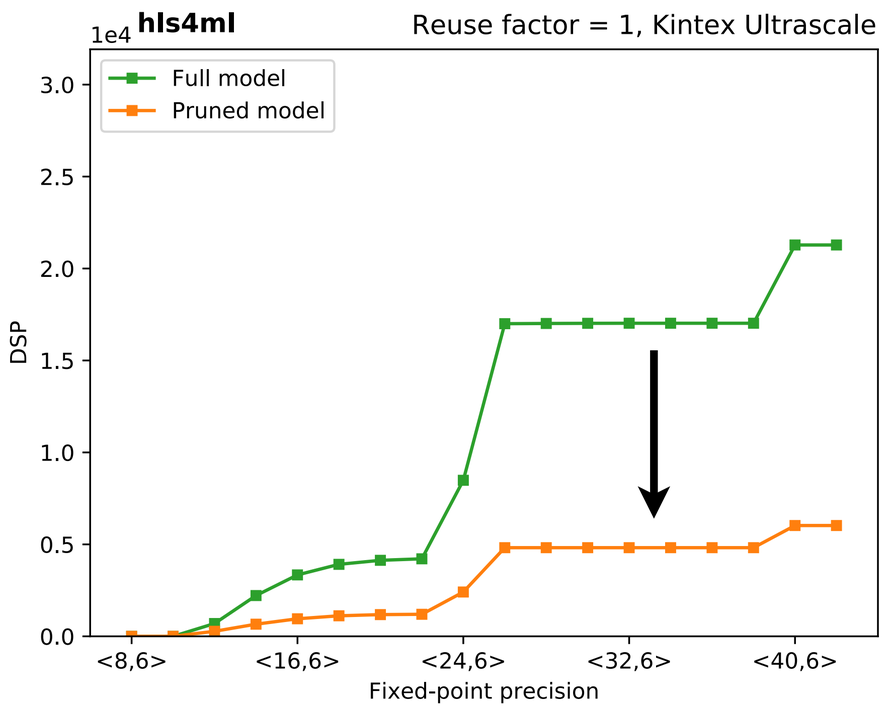


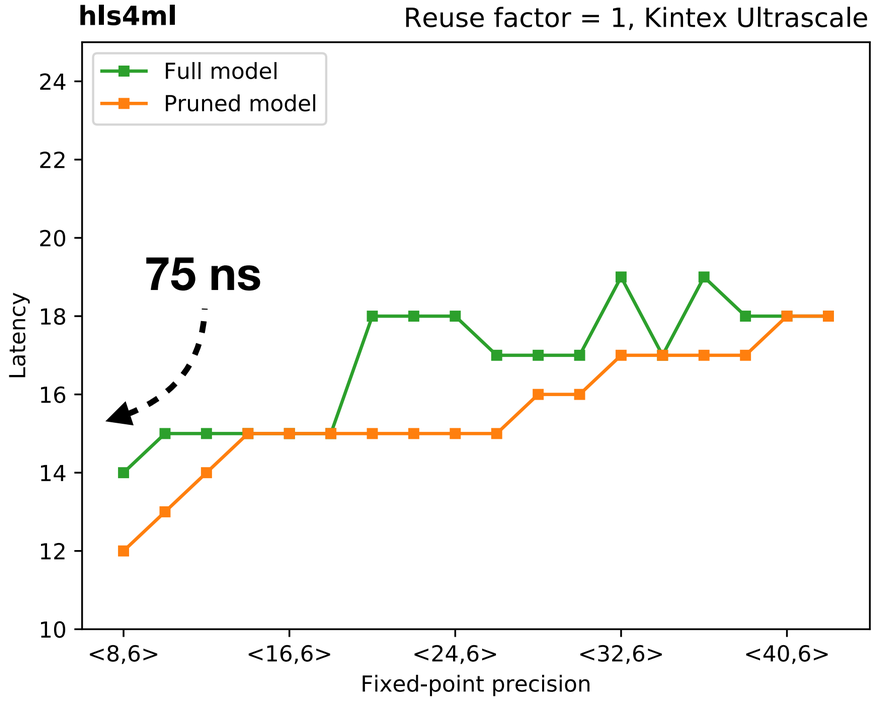
reuse = 1
use 4 multipliers 1 time each

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

Impact on Resource consumption

- *Resource consumption reduced to manageable level*
- *Network inference within 15 clocks @ 200 MHz clock = 75 sec*

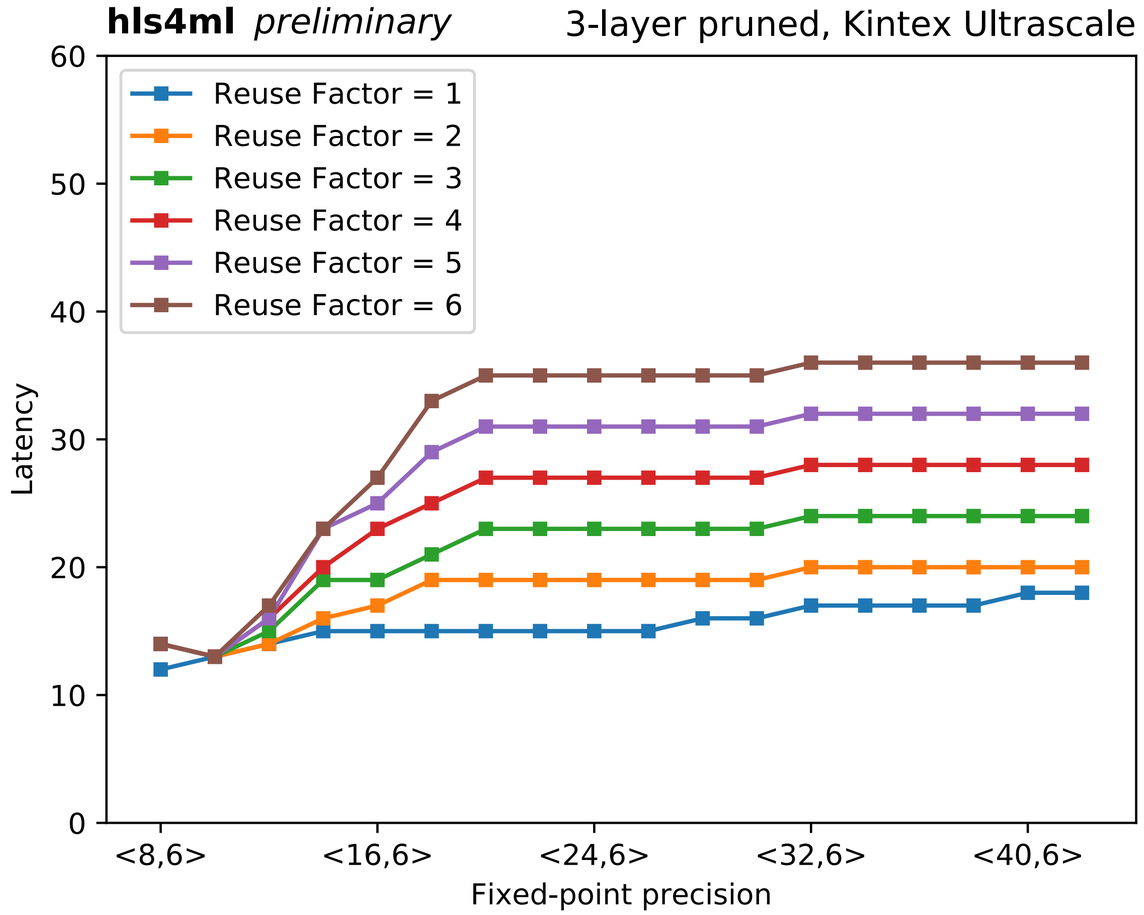


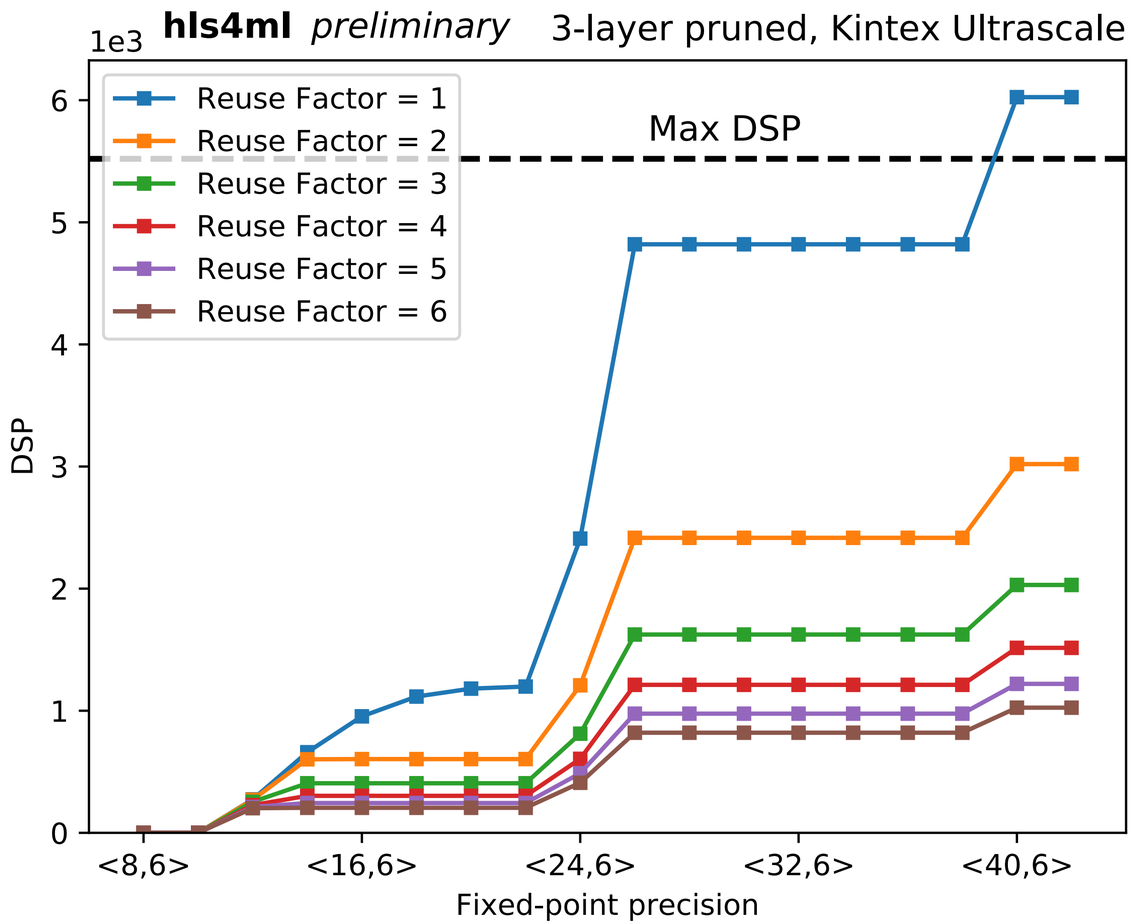


On Xilinx Kintex Ultrascale

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

Impact on Resource consumption

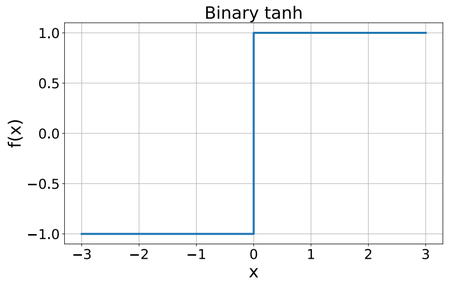


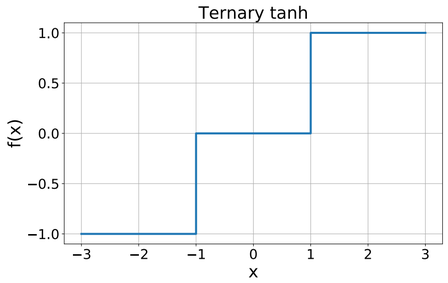


FPGAs architectures on HL-LHC time scale can handle these networks
Challenging but still possible with current FPGAs (at some performance loss cost)
But we can do better than this…

Extreme Quantization

- *One can push quantization to extremes*
  - *binary & ternary networks*

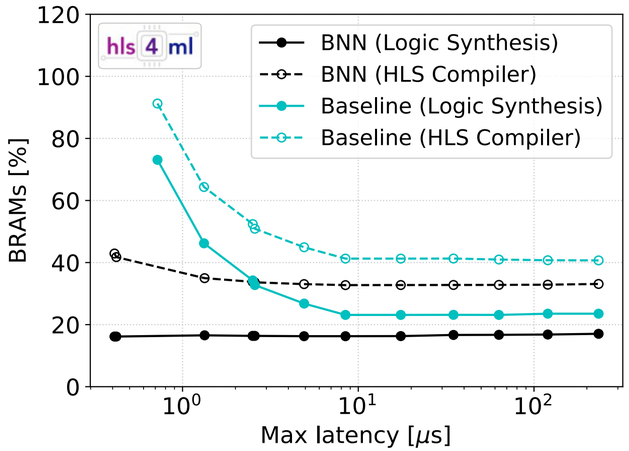


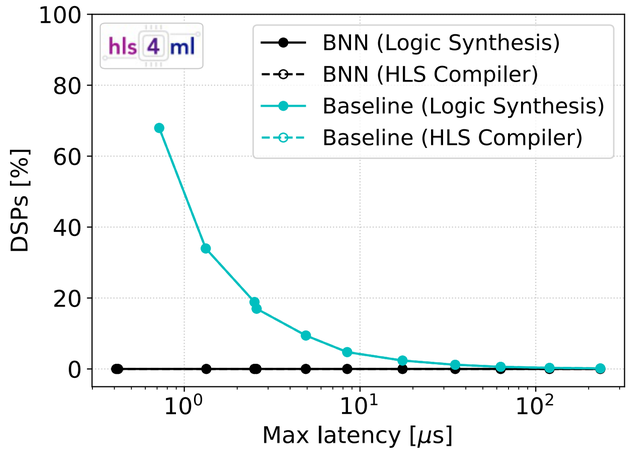


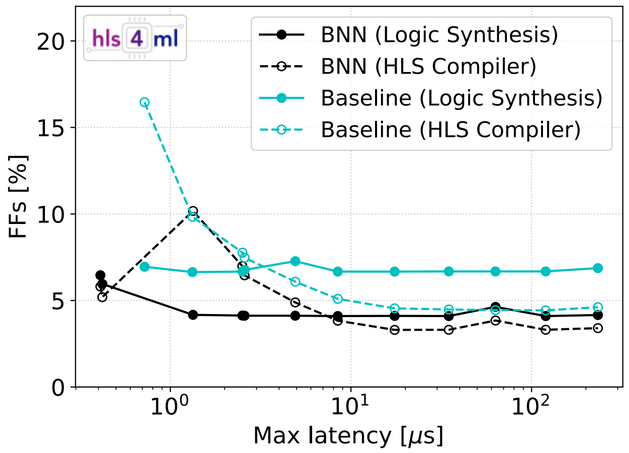
A	B	$A \times B$
-1	-1	1
-1	1	-1
1	-1	-1
1	1	1

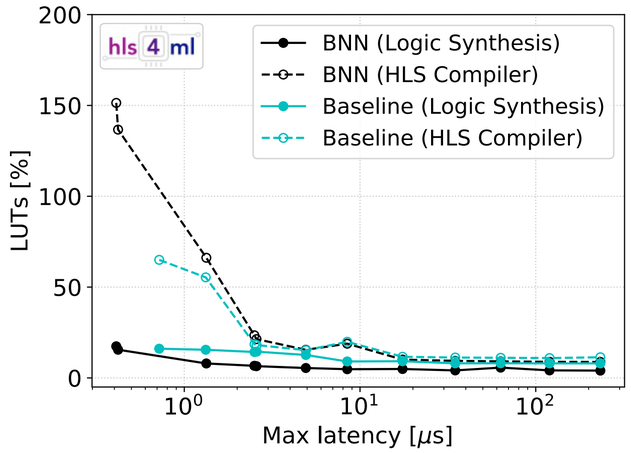
A	B	$\overline{A \oplus B}$
0	0	1
0	1	0
1	0	0
1	1	1

- *Multiplications can be replaced by bit manipulations, saving resources*
- *Can achieve low latencies at small accuracy cost and minimal resource consumption*





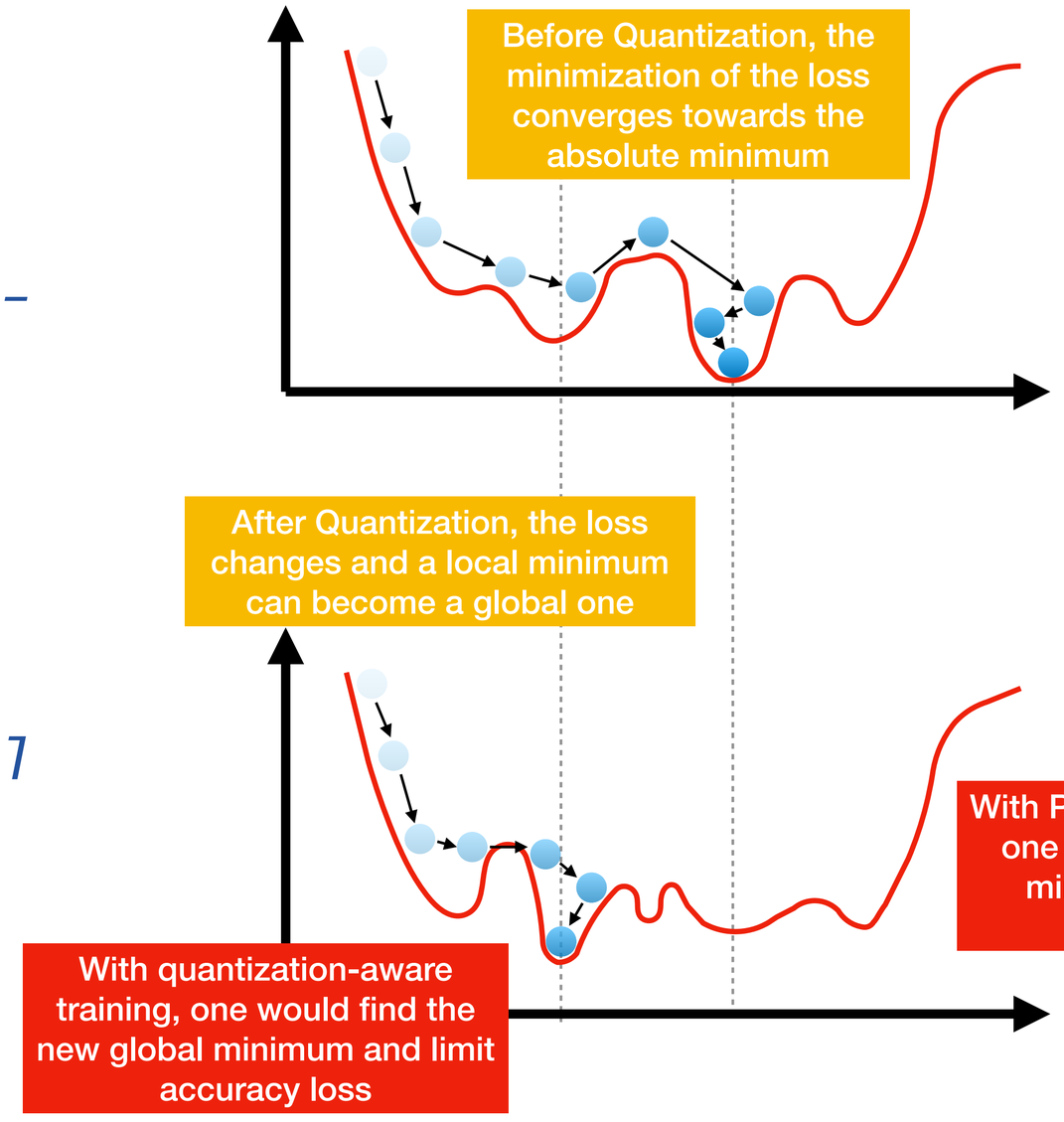




https://arxiv.org/abs/2003.06308

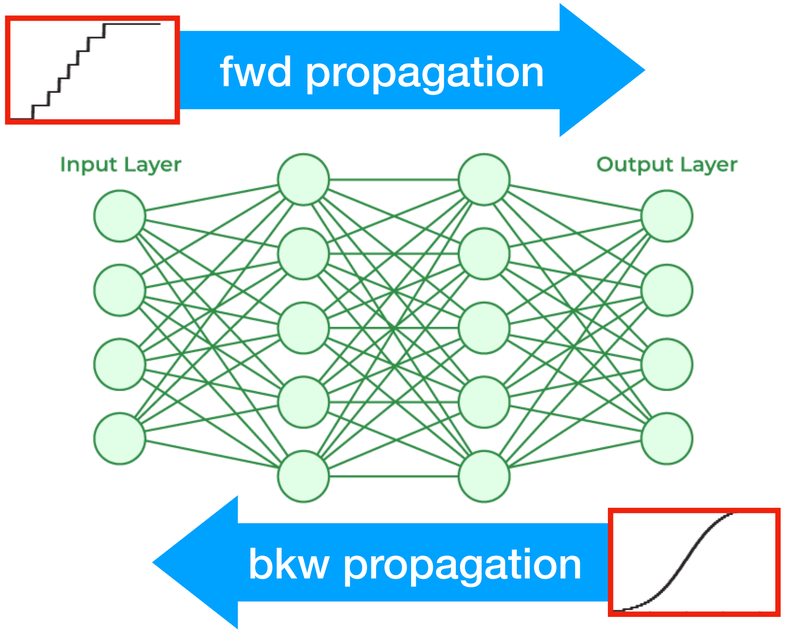
Quantization-aware Training

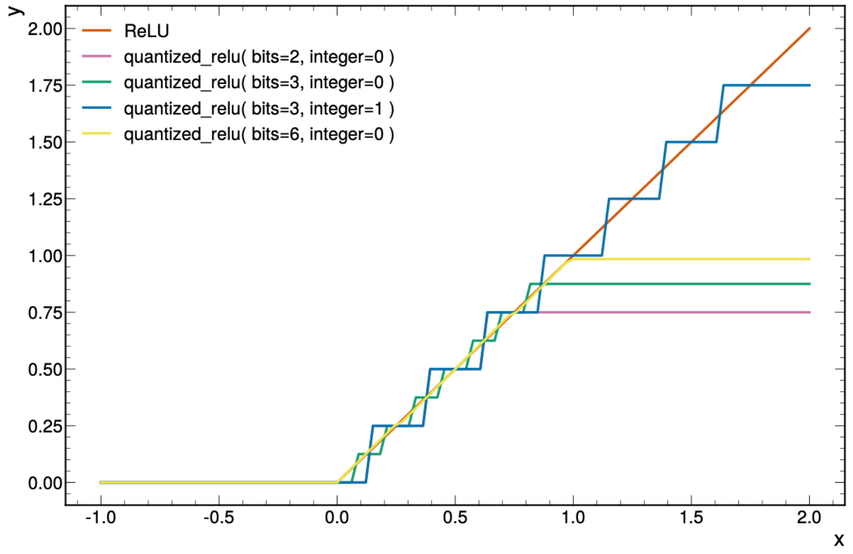
- *Post-training quantization can affect accuracy*
  - *for a given bit allocation, the loss minimum at floating-point precision might not be the minimum anymore*
- *One could specify quantization while look for the minimum*
  - *Maximize accuracy for minimal FPGA resources*
- *We teamed up with Google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



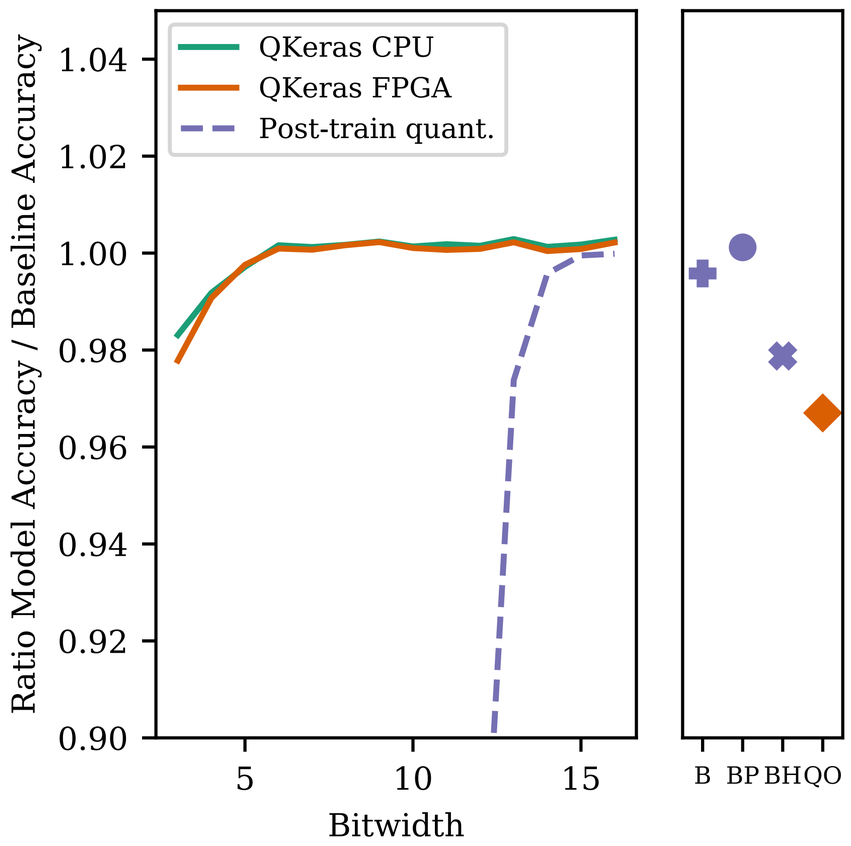
QAT in practice

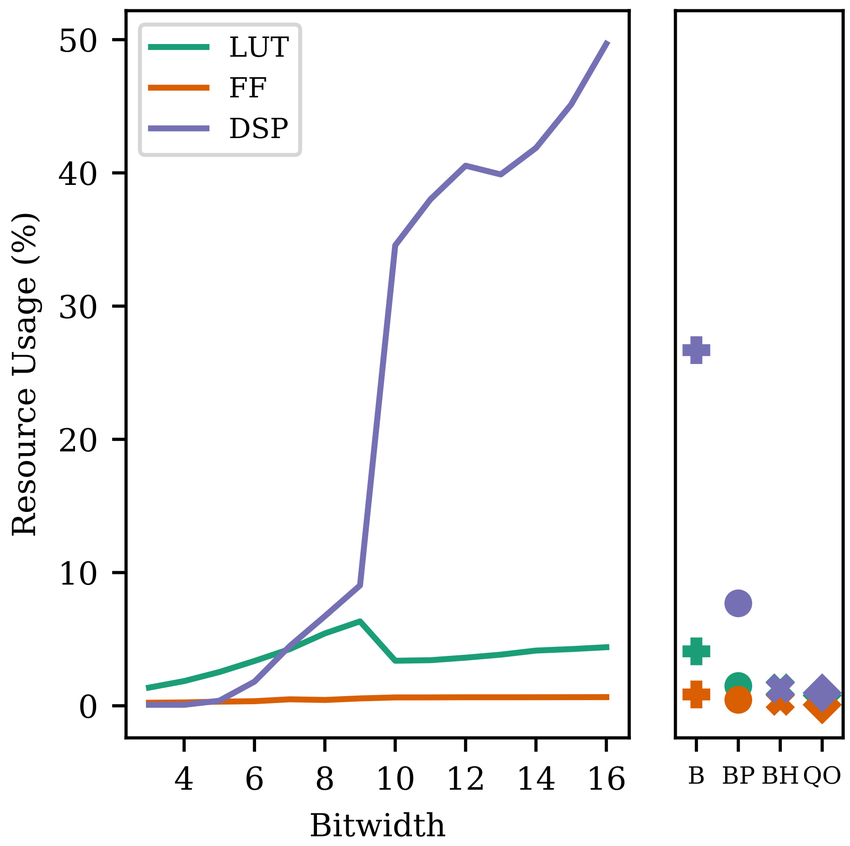
- During fwd propagation
  - Data and weights are represented as quantized numbers
  - Activation functions are binned so that their response is also quantized
  - Quantization can be global (same for all network) or customized on specific components
- During back propagation
  - Each component is represented in full precision, so that the differentiation in loss minimization is not affected
- Libraries exist to do this: QKeras (for TensorFlow), Brevity's (for PyTorch)





Quantization-aware Training





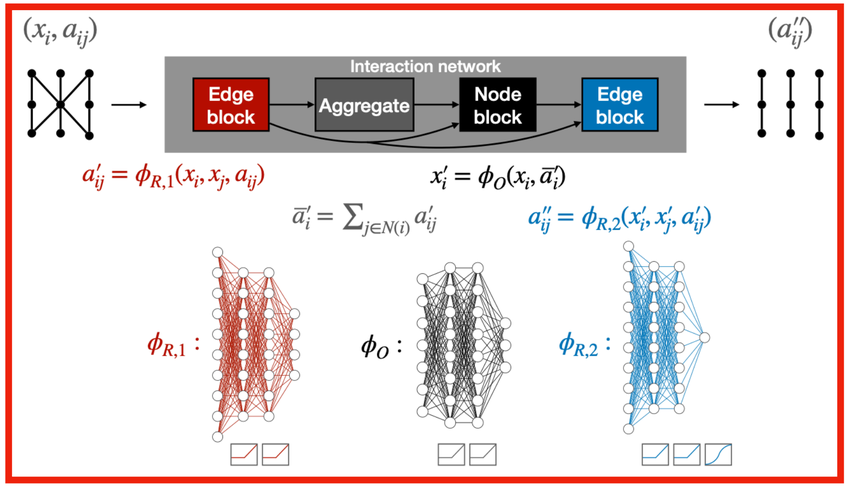
Model	Accuracy [%]	Latency [ns]	Latency [clock cycles]	DSP [%]	LUT [%]	FF [%]
Baseline	74.4	45	9	56.0 (1826)	5.2 (48321)	0.8 (20132)
Baseline pruned	74.8	70	14	7.7 (526)	1.5 (17577)	0.4 (10548)
Baseline heterogeneous	73.2	70	14	1.3 (88)	1.3 (15802)	0.3 (8108)
QKeras 6-bit	74.8	55	11	1.8 (124)	3.4 (39782)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

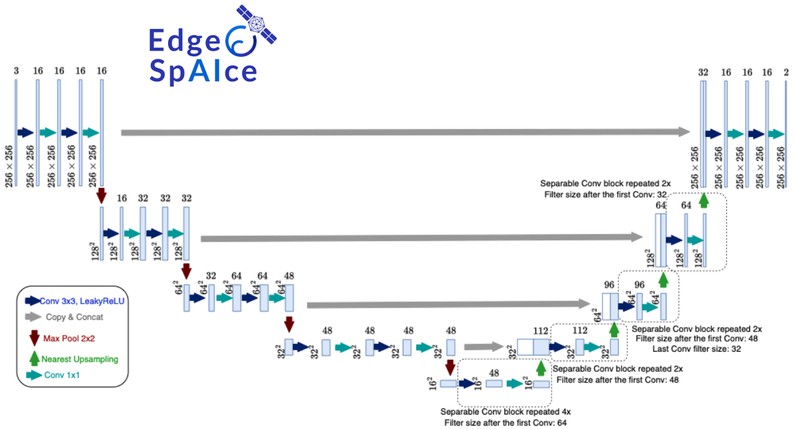
Quantization-aware Training

- *Thanks to this procedure, we can now deploy on FPGAs relatively large models*

Semantic Segmentation for Automotive

Graph Networks for Tracking



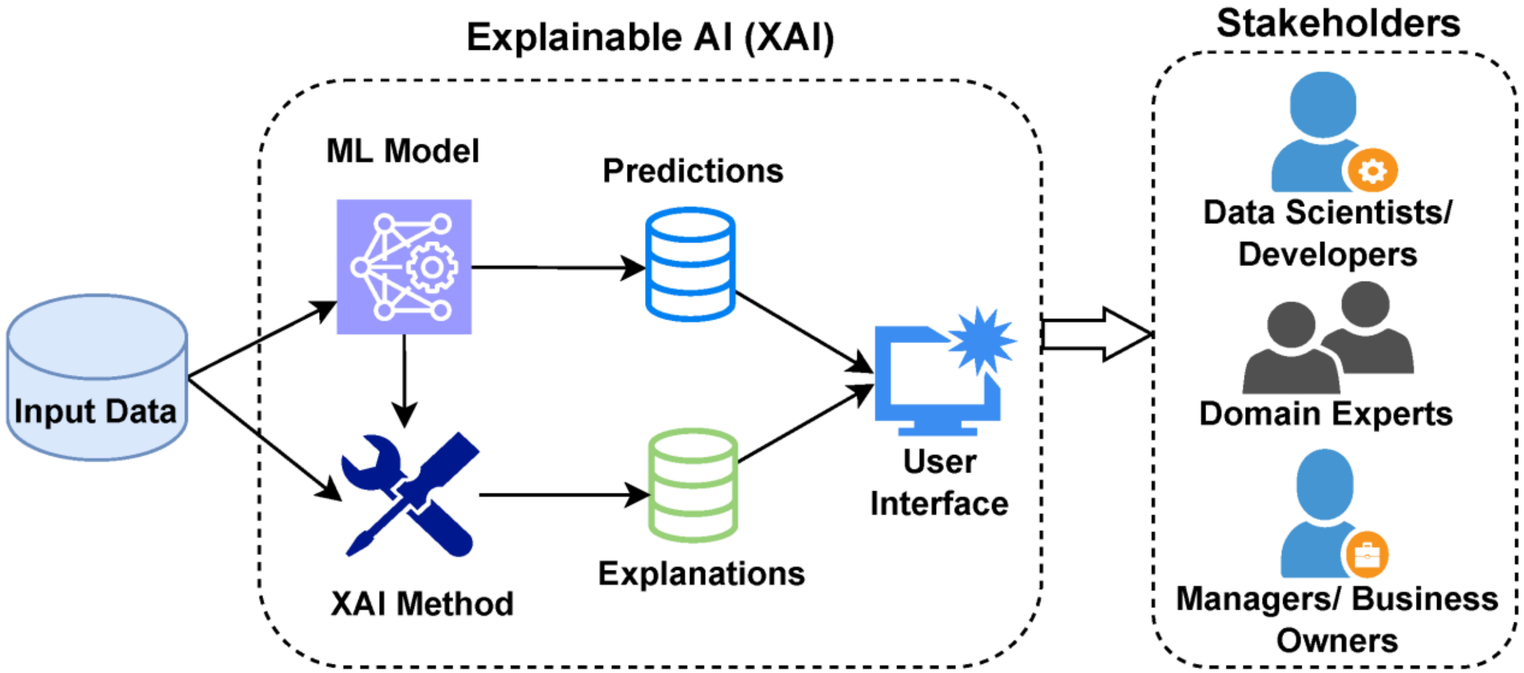


U-Net for image processing on satellites

- *Still, we cannot yet deliver the big ones*
  - *One has to look for even more effective compression strategies*

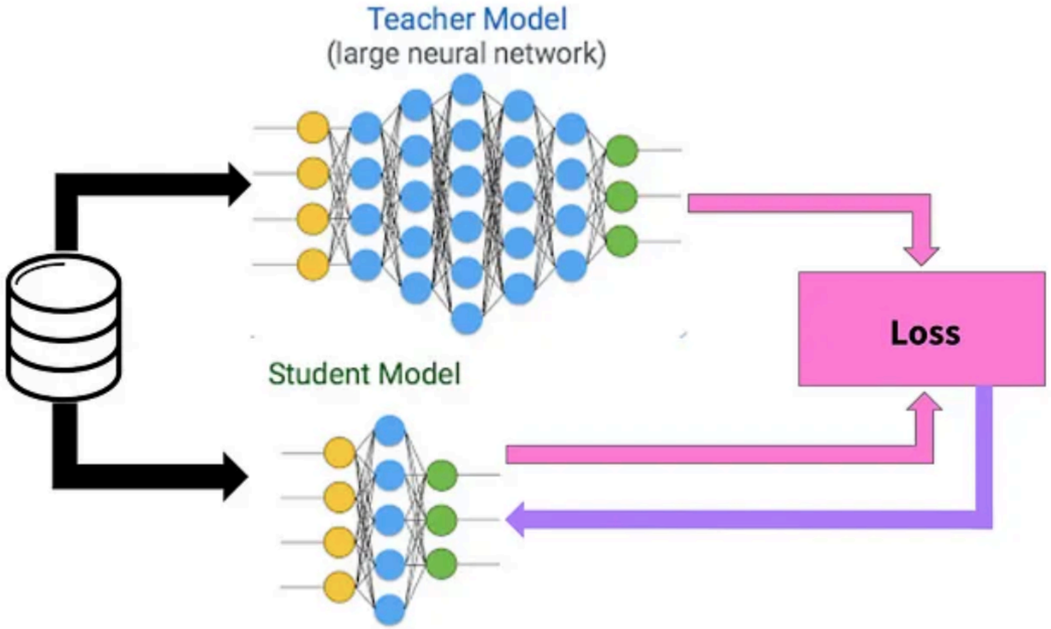
The next step with compression

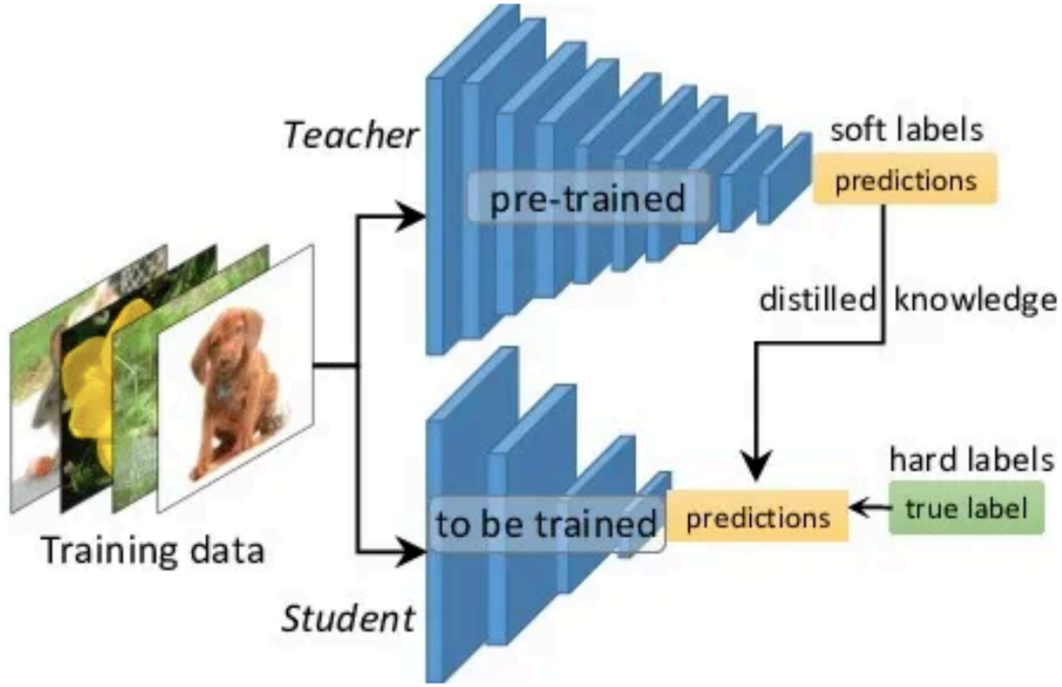
- *Compressing a NN means replacing it with a smaller network doing the same job with similar accuracy*
- *This is basically what people do with Explainable AI (XAI)*
  - *For a different reason: they want to open the black box, which is easier if the model is smaller*
- *We can then look at trends in XAI and use those approaches for model compression*
  - *Knowledge Distillation*
  - *Symbolic Regression*
  - *…*



Knowledge Distillation

- *Replace the main model (the teacher) with a smaller one (the student), trained to reproduce the result of the teacher*



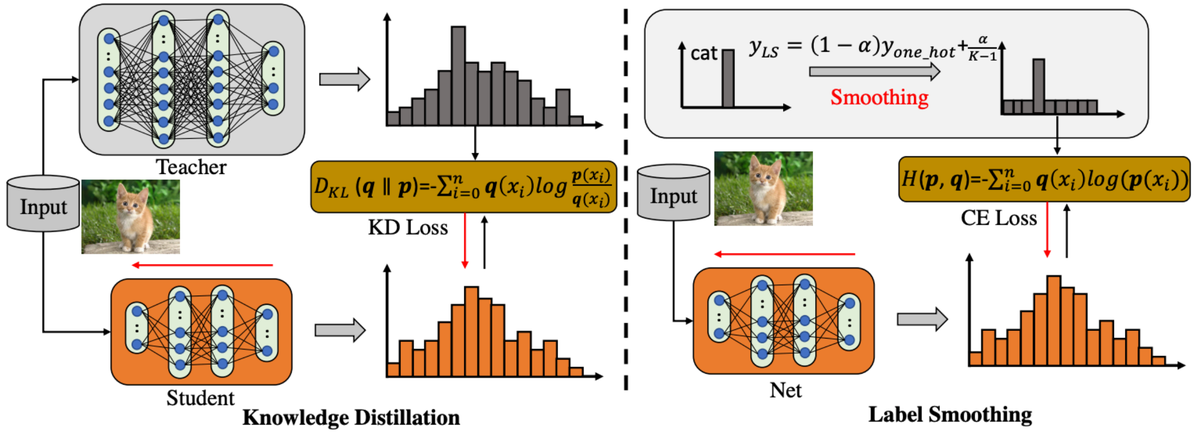


Does KD make sense?

- *In principle if a student classifier can mimic the teacher, most likely it can learn to solve the task directly*
- *In practice, KD works very well (empirical fact w/o clear theoretical explanation).*
  - *The advantage could be numerical (e.g., training vs scores of another classifier is similar to label-smoothing regularization, but the smoothing carries knowledge and it is not from noise)*

Does Knowledge Distillation Really Work?

Samuel Stanton NYU
Pavel Izmailov NYU
Polina Kirichenko NYU
Alexander A. Alemi Google Research
Andrew Gordon Wilson NYU



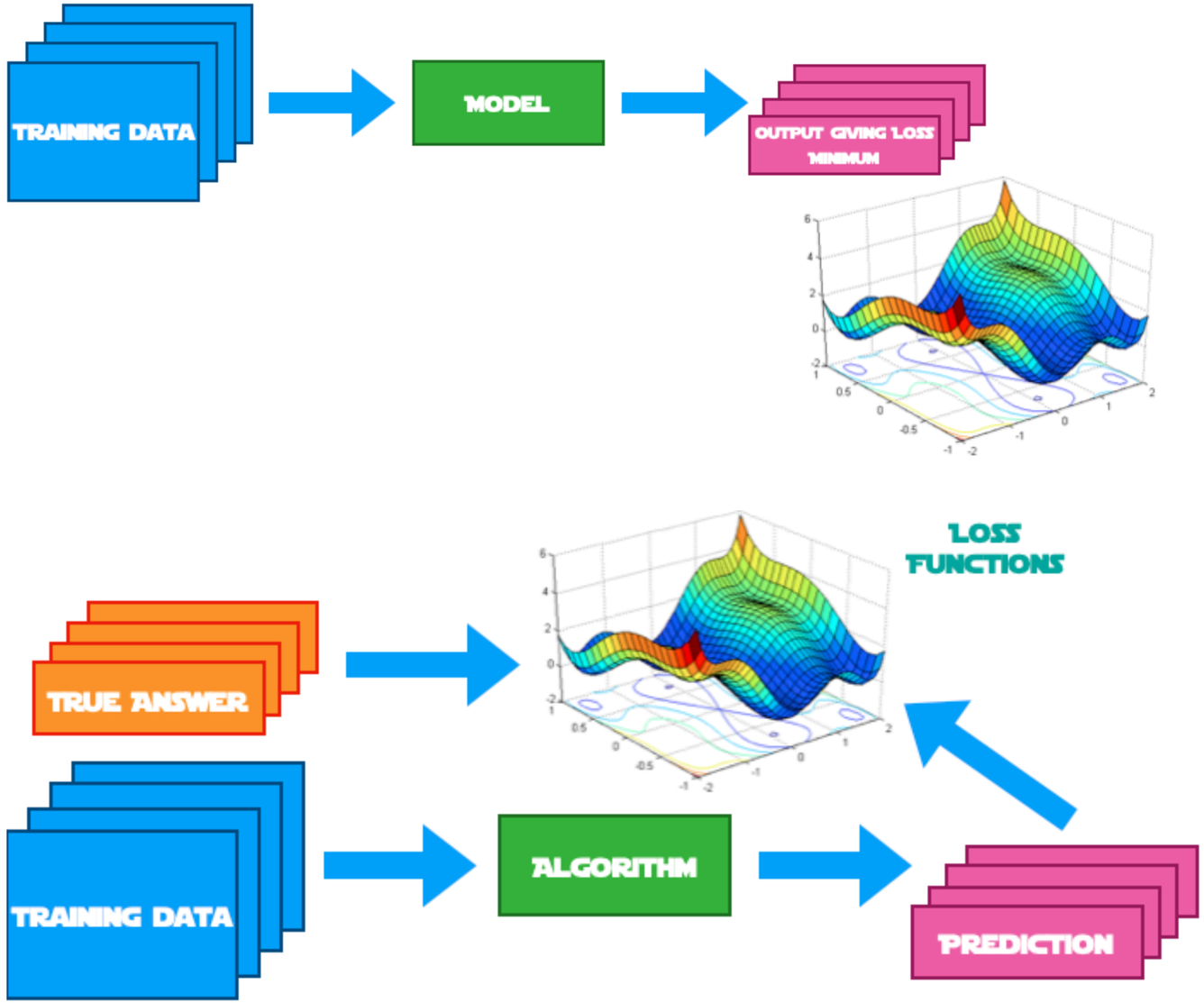
Revisiting Knowledge Distillation via Label Smoothing Regularization

Li Yuan[1] Francis EH Tay[1] Guilin Li[2] Tao Wang[1] Jiashi Feng[1]

[1]National University of Singapore [2]Huawei Noah's Ark Lab
{ylustcnus, twangnh}@gmail.com, {mpetayeh,elefjia}@nus.edu.sg, guilinli2@huawei.com

Where KD certainly helps

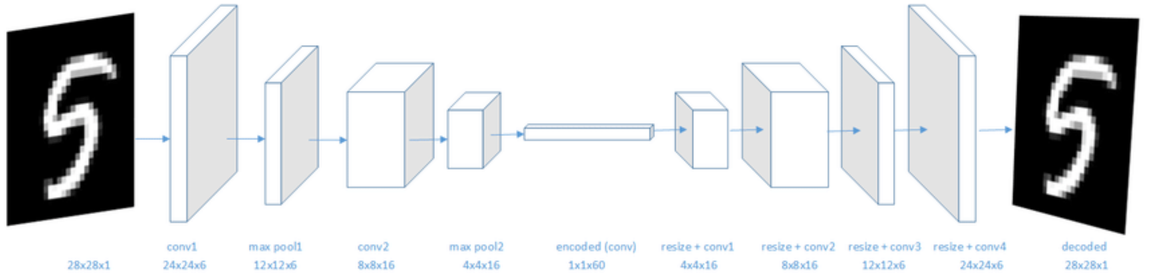
- *Supervised learning: learn to predict a given ground truth (e.g., labels), given the inputs*
- *Unsupervised learning: extract knowledge from a dataset in a label-less training process*
- *With KD, one can solve an unsupervised task with a big teacher, from the trained teacher derive a ground truth, and then train a simple student in a supervised manner*

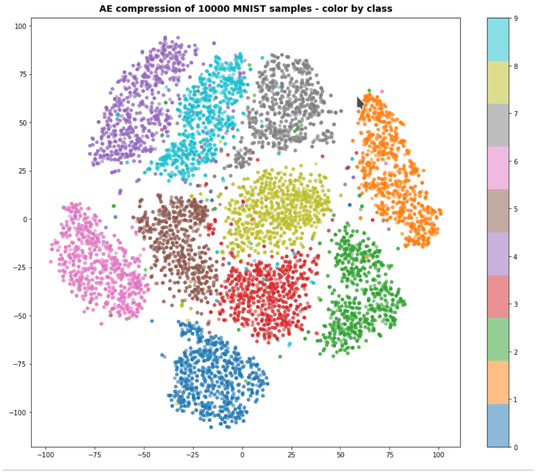


Where KD certainly helps

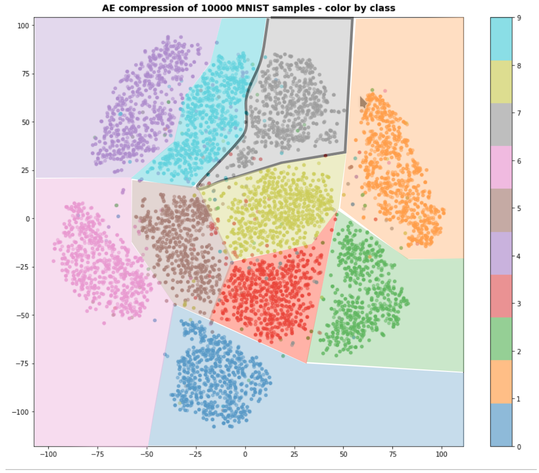
- *Supervised learning: learn to predict a given ground truth (e.g., labels), given the inputs*
- *Unsupervised learning: extract knowledge from a dataset in a label-less training process*
- *With KD, one can solve an unsupervised task with a big teacher, from the trained teacher derive a ground truth, and then train a simple student in a supervised manner*

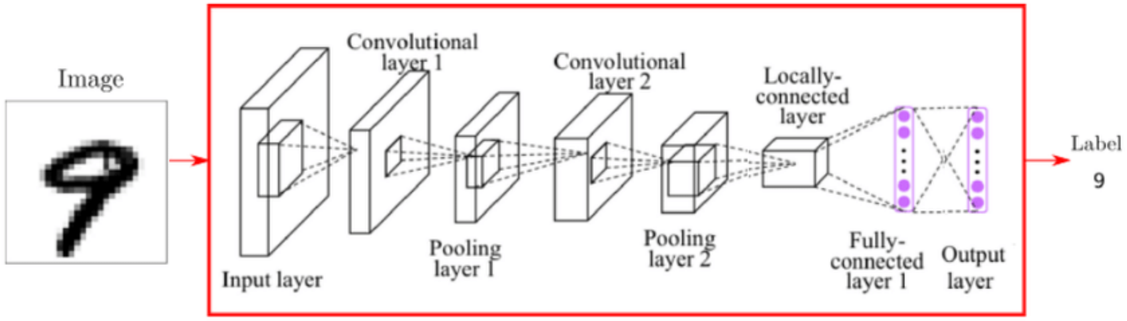
Teacher: Train a clustering algorithm to identify alike images





Using Teacher score: one can define labels



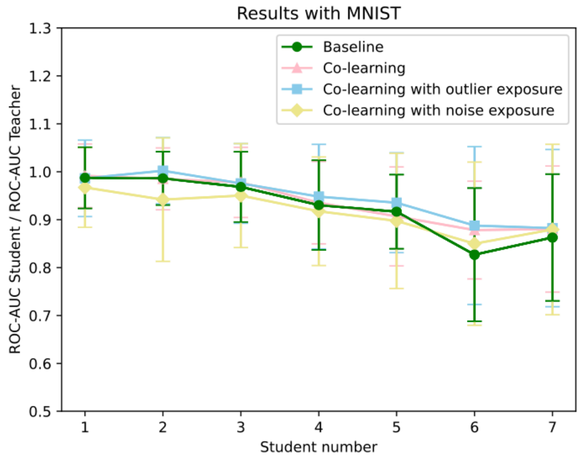


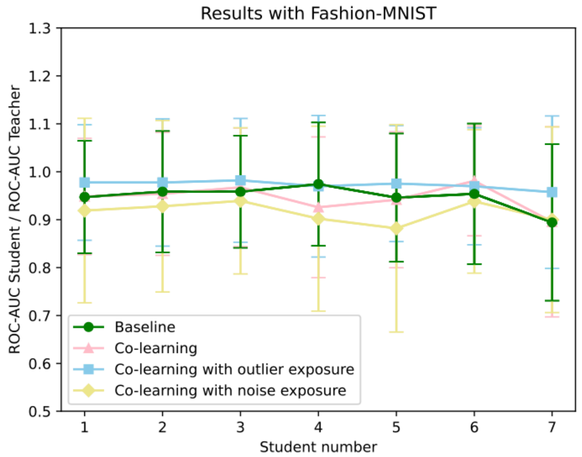
Student: is a classifier on the labels defined from the teacher. Simpler problem to solve -> smaller network

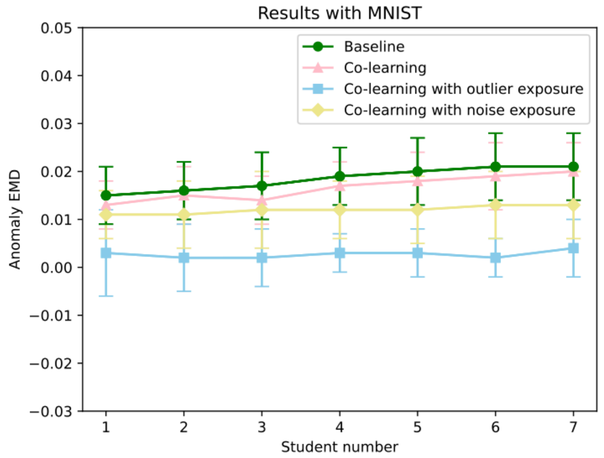
Student Training

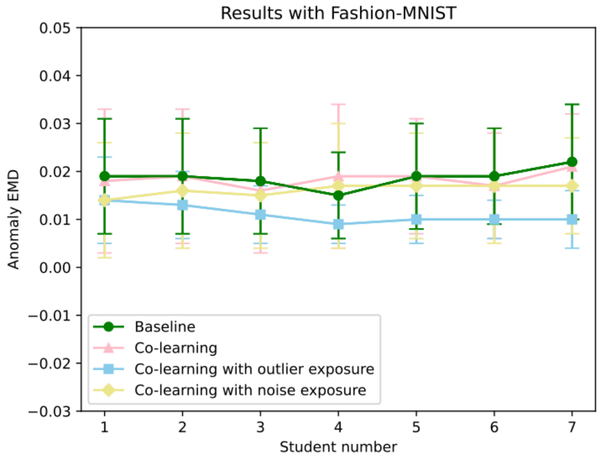
- *Baseline: the student is trained to learn the score of the teacher, which was previously trained*
- *Co-learning: the students and the teacher are trained at once, minimizing the sum of the two losses*
- *For unsupervised learning: one needs to make sure that the students learns also to identify outlier behaviors*
  - *Outlier exposure: use unrelated data for student learning*
  - *Noise exposure: add random noise to the data to create outliers*

	Teacher	S_1	S_2	S_3	S_4	S_5	S_6	S_7
Params	19,360	7,180	2190	1060	409	225	133	77
FLOPS	18.91M	6.37M	2.25M	432k	131k	114k	58k	53k



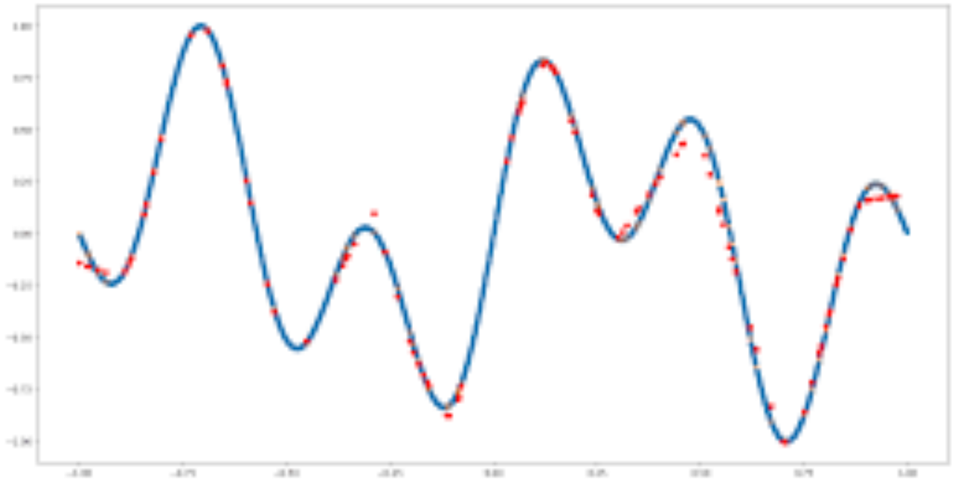




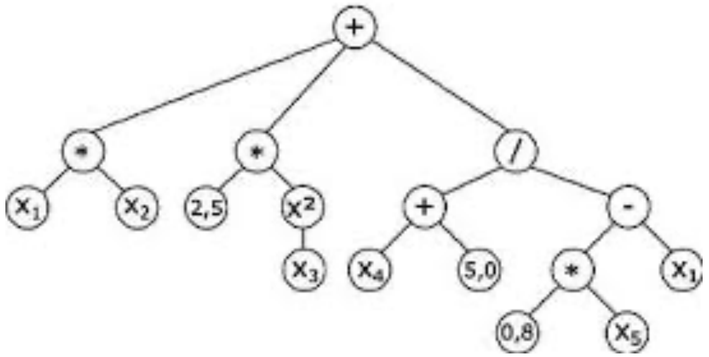


Symbolic Regression

- *Neural networks are universal function approximators*
  - *There is an unknown function f that relates the input to the answer. a NN can learn to mimic that function if smooth enough*
- *What if we try to learn directly the function?*
  - *The outcome would be an analytical expression that we could*
    - *Interpret*
    - *Deploy on FPGAs at minimal cost*

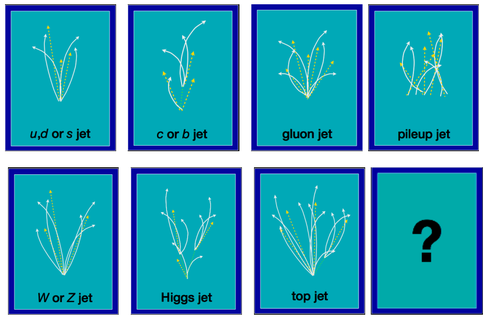


$$f(x) = x_1 x_2 + 2.5\,x_3^2 + \frac{x_4 + 5.0}{0.8\,x_5 - x_1}$$



Symbolic Regression

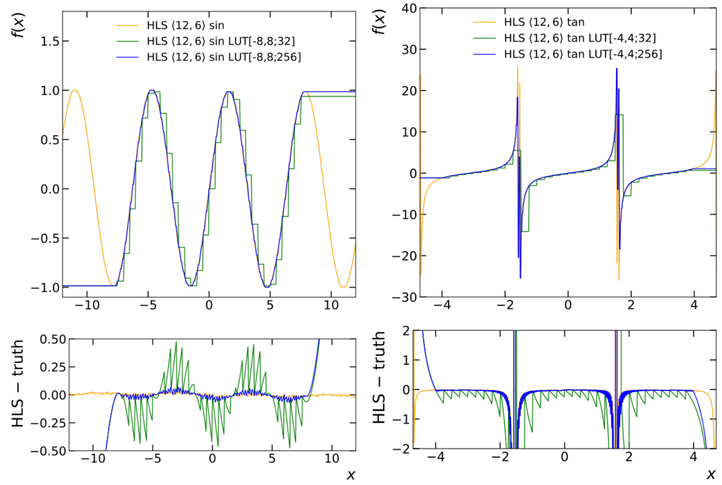
- *Expression implemented as a tree*
- *Possibilities explored using a random forest*
- *Result evaluated using loss*
- *Regularization with model "complexity": pre-assign complexity to each expression and sum them at training time*



Each function has complexity 1
Take any function with c< Max complexity that minimizes the loss

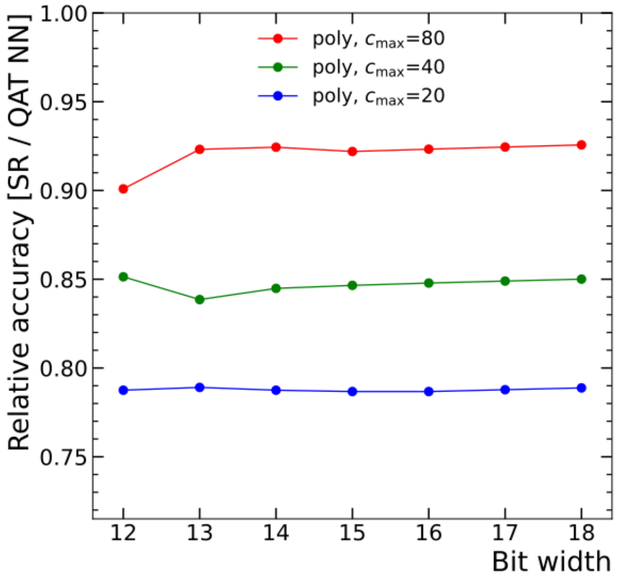
$$h_g^*, h_q^*, h_t^*, h_W^*, h_Z^* = \underset{h_g,h_q,h_t,h_W,h_Z \in \mathcal{S}_{c_{\max}}}{\arg\min} \sum_i \sum_{f\in\{g,q,t,W,Z\}} \ell(h_f(\mathbf{x}^i), y_f^i),$$

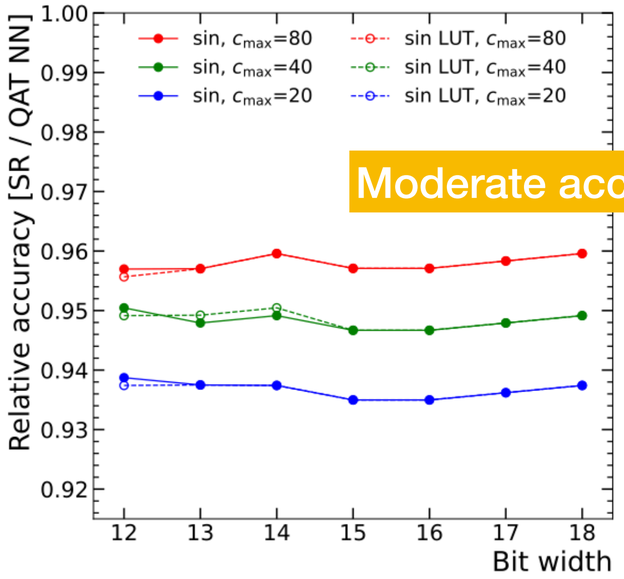
The result is then ported to hls4ml, where all functions are implemented via Xilinx backend. Deployment as in NNs (quantization etc.)



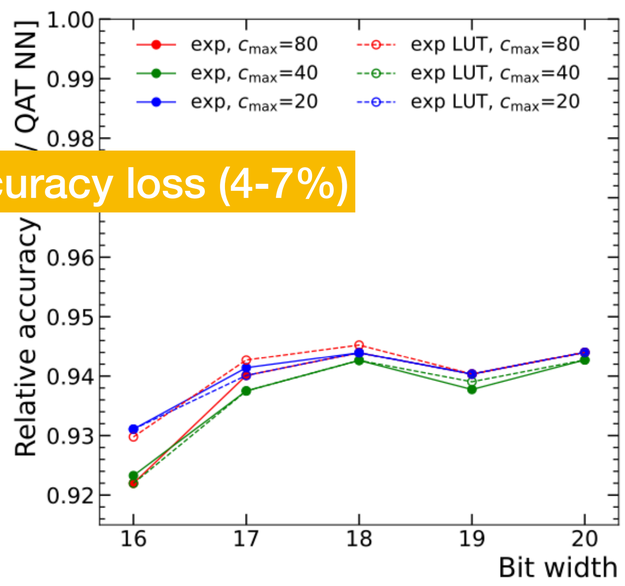
Model	Expression for the t tagger with $c_{\max} = 40$	AUC
Polynomial	$C_1^{\beta=2} + 0.09m_{\text{mMDT}}(2C_1^{\beta=1} + M_2^{\beta=2} - m_{\text{mMDT}} - \text{Multiplicity} - (1.82C_1^{\beta=1} - M_2^{\beta=2})(C_1^{\beta=2} - 0.49m_{\text{mMDT}}) - 3.22) - 0.53$	0.914
Trigonometric	$\sin(0.06(\sum z \log z)M_2^{\beta=2} - 0.25C_1^{\beta=2}(-C_1^{\beta=1} + 2C_1^{\beta=2} - M_2^{\beta=2} + \text{Multiplicity} - 8.86) - m_{\text{mMDT}} + 0.06\text{Multiplicity} - 0.4)$	0.925
Exponential	$0.23C_1^{\beta=1}(-m_{\text{mMDT}} + \text{Gauss}(0.63\text{Multiplicity}) + 1) - \text{Gauss}(C_1^{\beta=1}) + 0.45C_1^{\beta=2} - 0.23m_{\text{mMDT}}$ $+ 0.23\text{Gauss}((4.24 - 1.19C_2^{\beta=1})(C_1^{\beta=2} - m_{\text{mMDT}})) + 0.15$	0.920
Logarithmic	$C_1^{\beta=2} - 0.1m_{\text{mMDT}}(\text{Multiplicity} \times \log(\text{abs}(\text{Multiplicity})) + 2.2) - 0.02\log(\text{abs}(\text{Multiplicity}))$ $- 0.1(C_1^{\beta=2}(C_1^{\beta=1} - 1.6M_2^{\beta=2} + m_{\text{mMDT}} + 1.28) - m_{\text{mMDT}} - 0.48)\log(\text{abs}(C_1^{\beta=2})) - 0.42$	0.923

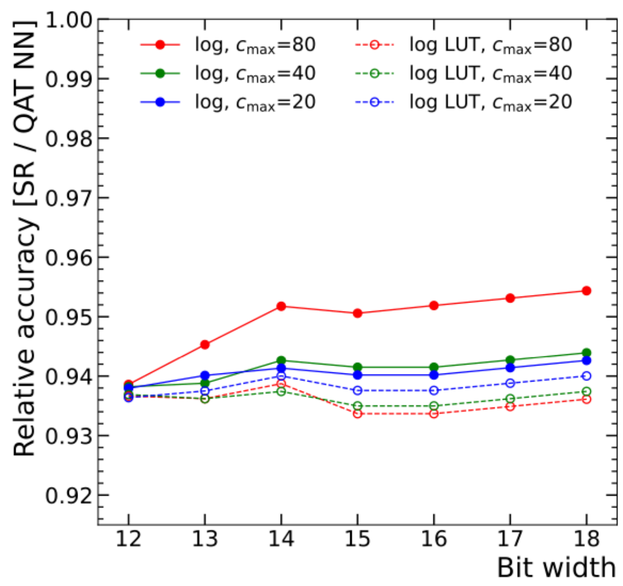
Symbolic Regression

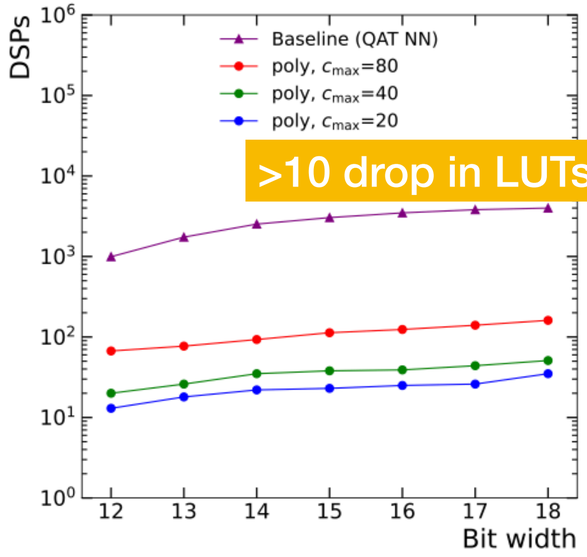




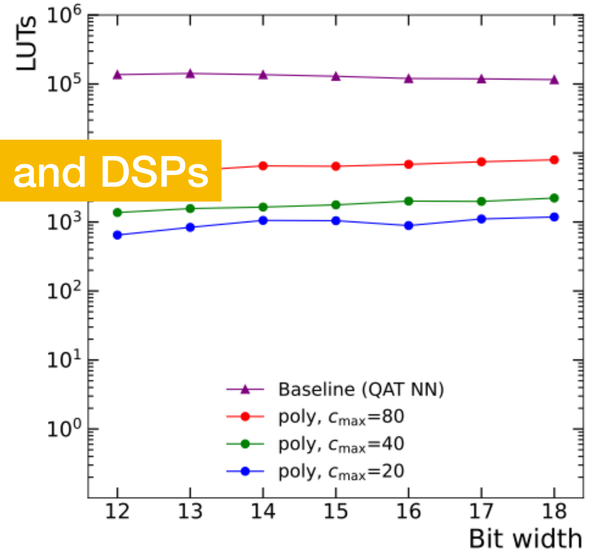
Moderate accuracy loss (4-7%)

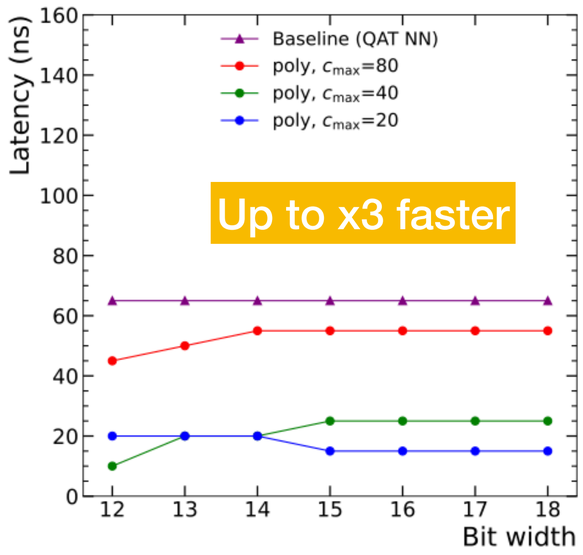






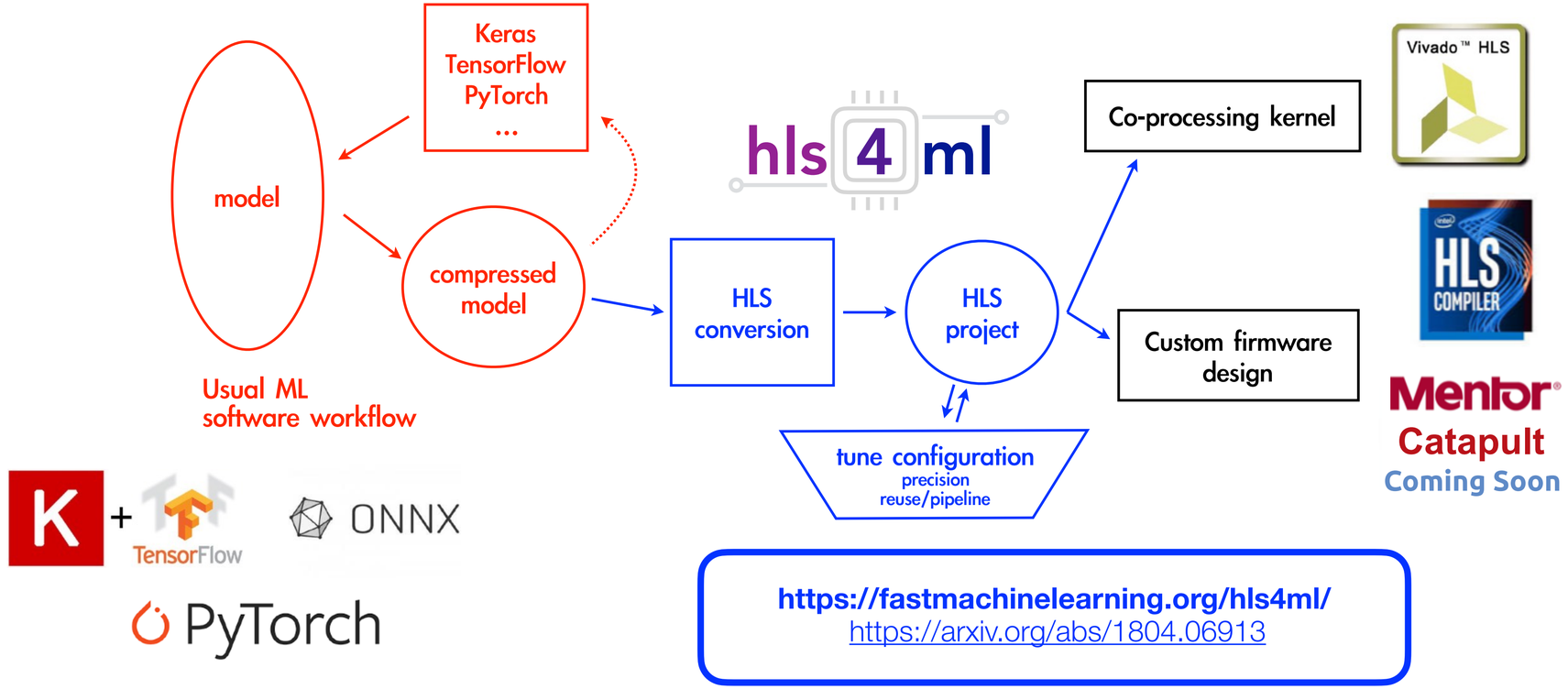
>10 drop in LUTs and DSPs





Up to x3 faster

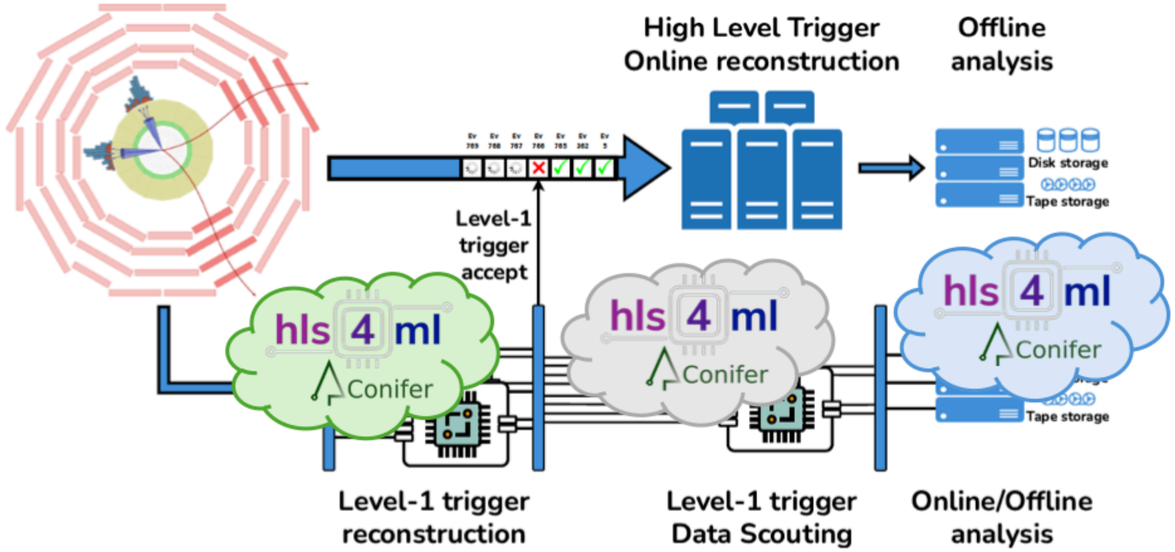
Software Pipeline

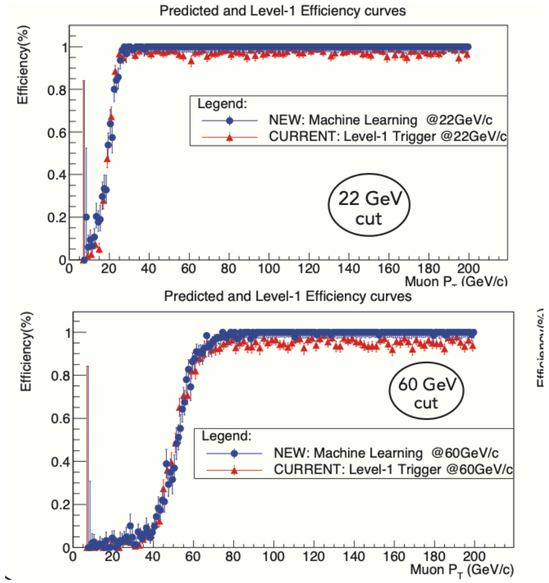


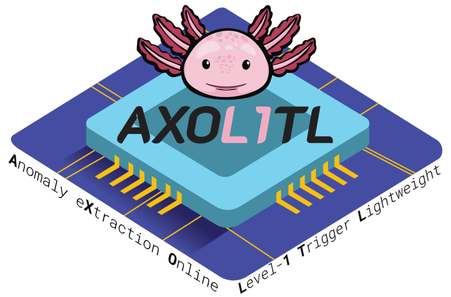
- hls4ml *(tutorial next week) aims to be this automatic tool*
- *reads as input models trained on standard DeepLearning libraries*
- *comes with implementation of common ingredients (layers, activation functions, etc)*
- *Uses HLS softwares to provide a firmware implementation of a given network*
- *Could also be used to create co-processing kernels for HLT environments*

Applications

- *On the long term: this approach will play a major role at HL-LHC*
  - *Several ongoing projects in ATLAS and CMS to exploit more powerful FPGAs after upgrade*
  - *NGT project will contribute to fuel this effort, with expansion of hls4ml library*
- *On the short term: applications are limited by old FPGAs in the L1 system*
  - *Still, non trivial things can be done*
  - *In CMS: effort started with muon regression and it is now concentrated on anomaly detection*



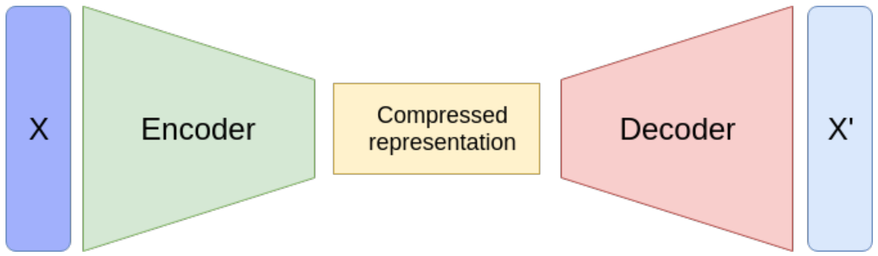


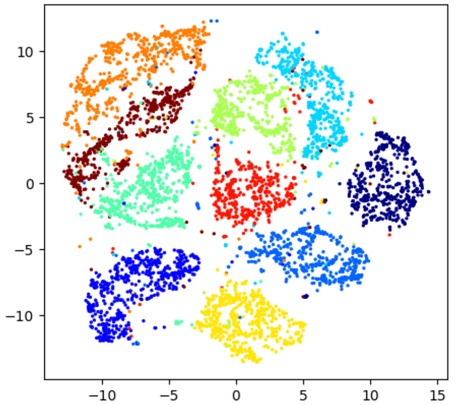


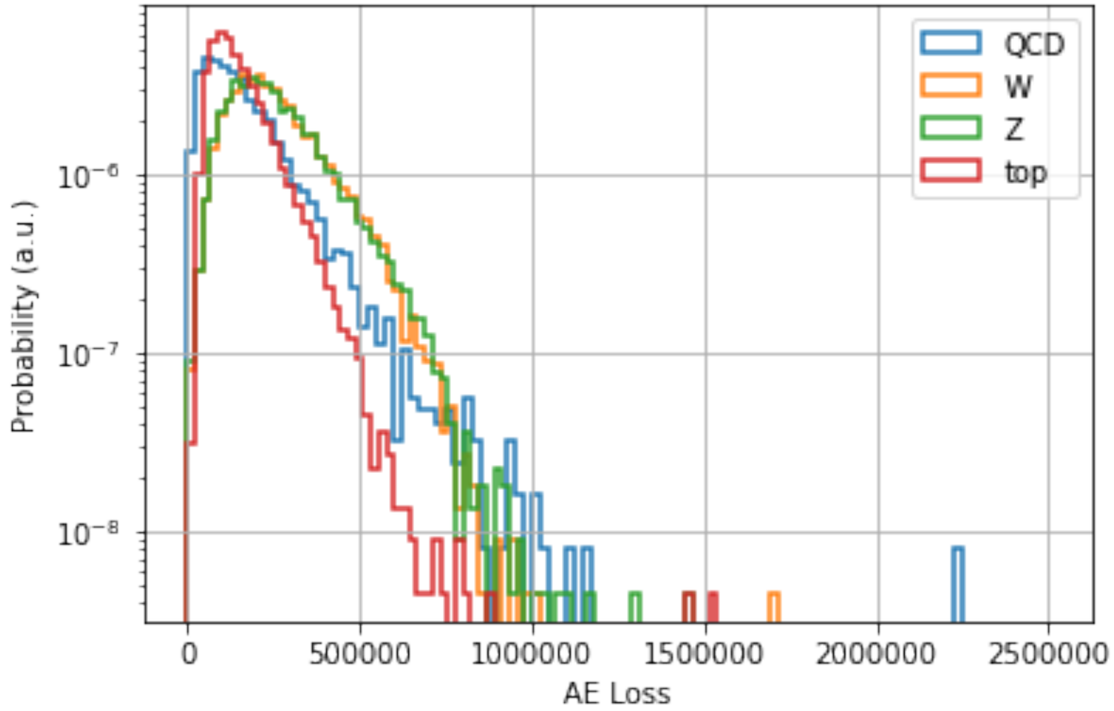
CICADA

Anomaly Detection

- *The idea is to learn a dataset pdf so that one can assign a p-value to each event*
- *This is done using autoencoders*
  - *Learn to compress and decompress data with high fidelity*
  - *Clustering: in the bottle neck (latent space) one has a compressed representation of the dataset*
  - *Metric: the loss can be interpreted as a distance between the input and the output*
  - *Anomaly Detection: when an outlier event (e.g., not belonging to training dataset) is given, compression-decompression might fails, giving a big loss*





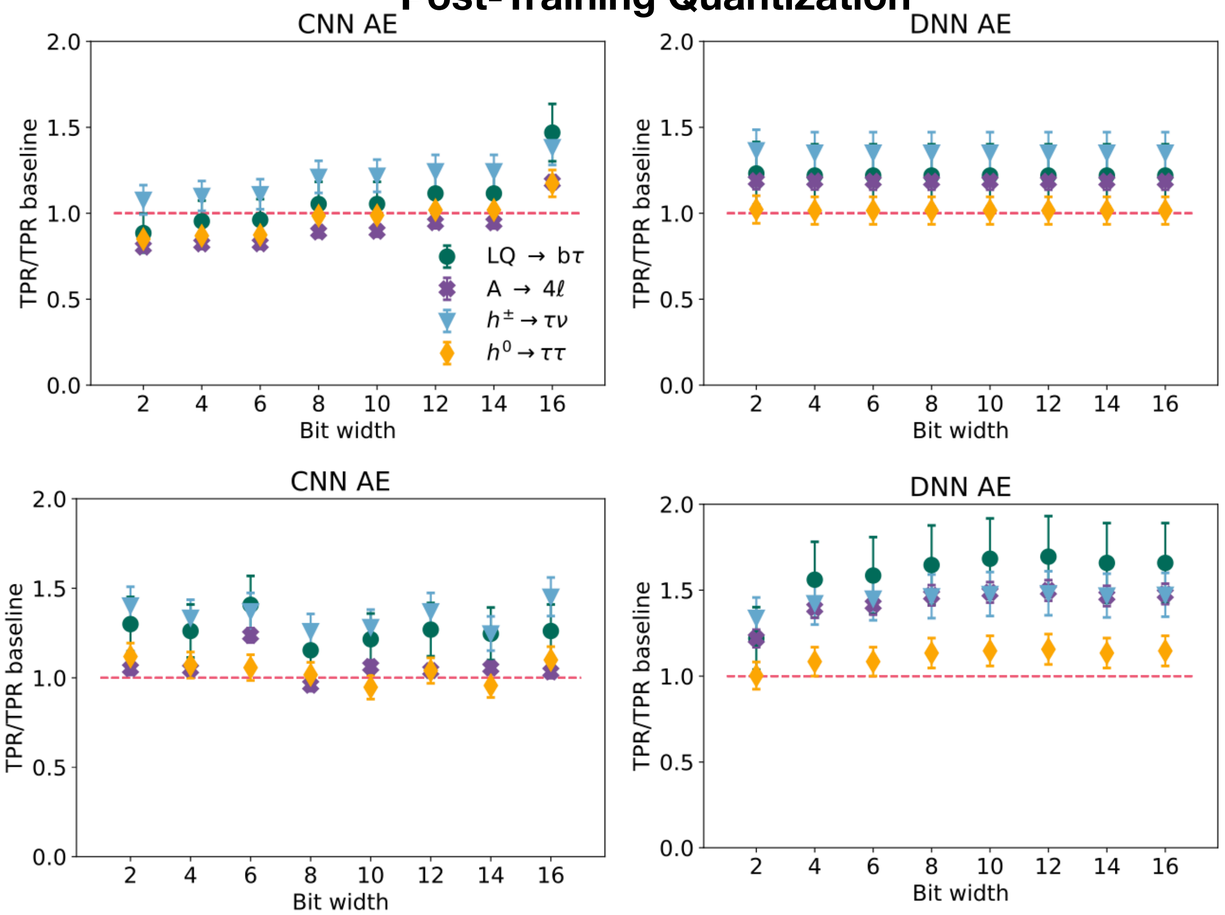


A few complications

- *Autoencoders as anomaly detection are double-task algorithms*
  - *Training: optimized to minimize loss*
  - *Inference: used to detect anomaly score above threshold*
- *Often, then loss is used as anomaly score*
  - *But this is not always the optimal choice*
  - *When a different score is used, compression algorithms might create problems*
    - *Quantization optimized on the loss, could spoil the anomaly detection*

Using Loss as Anomaly Score

Post-Training Quantization



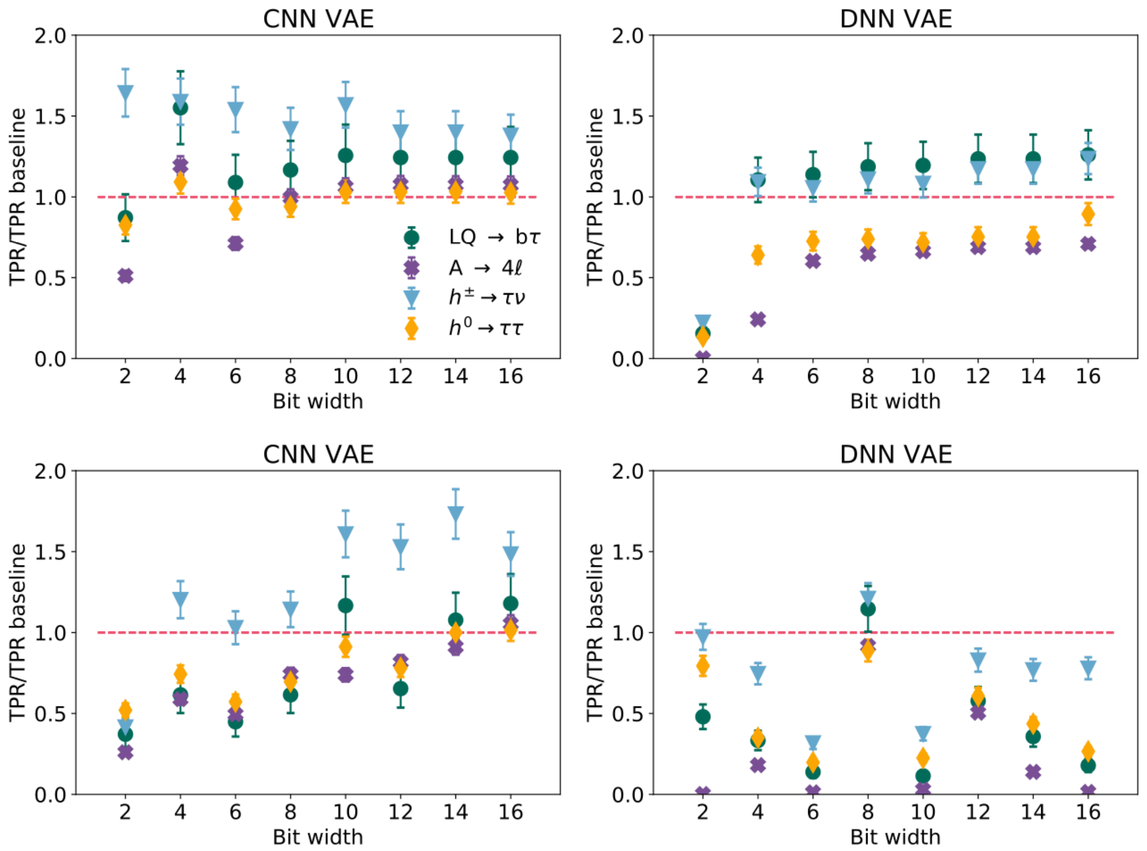
Quantization-aware training

A few complications

- *Autoencoders as anomaly detection are double-task algorithms*
  - *Training: optimized to minimize loss*
  - *Inference: used to detect anomaly score above threshold*
- *Often, then loss is used as anomaly score*
  - *But this is not always the optimal choice*
  - *When a different score is used, compression algorithms might create problems*
    - *Quantization optimized on the loss, could spoil the anomaly detection*

Using Latent-Space Quantity as Anomaly Score

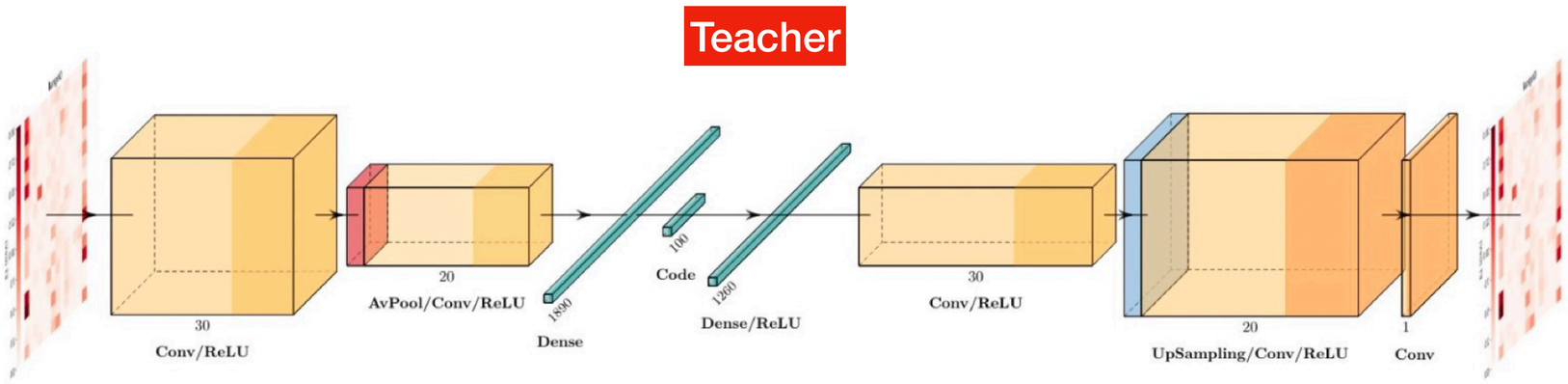
Post-Training Quantization

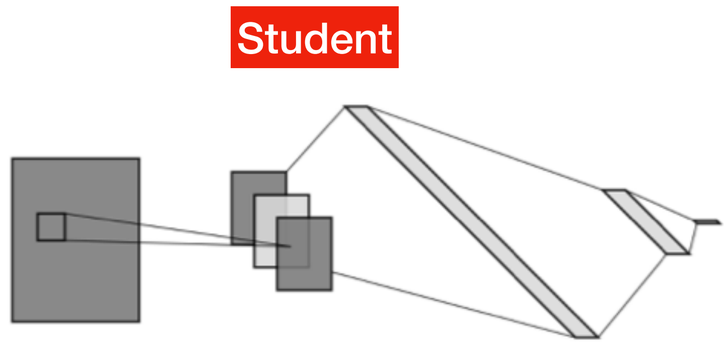


Quantization-aware training

Mitigation measures

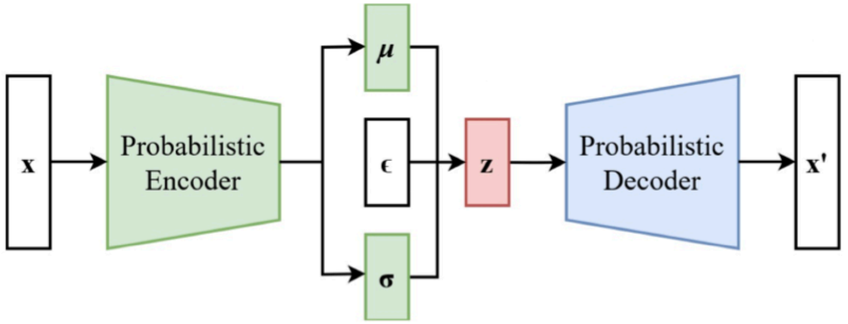
- *Use the loss as score*
- *Use alternative compression strategies*
  - *We saw before Knowledge Distillation*
    - *Train a student regression to learn the anomaly score of some Teacher auto encoder*
    - *Quantize the student regression, to avoid issues with compression of the auto encoder itself*
    - *IMPORTANT: train with outliers to make sure that the regression generalizes to anomalies*





CMS Anomaly Detection Triggers

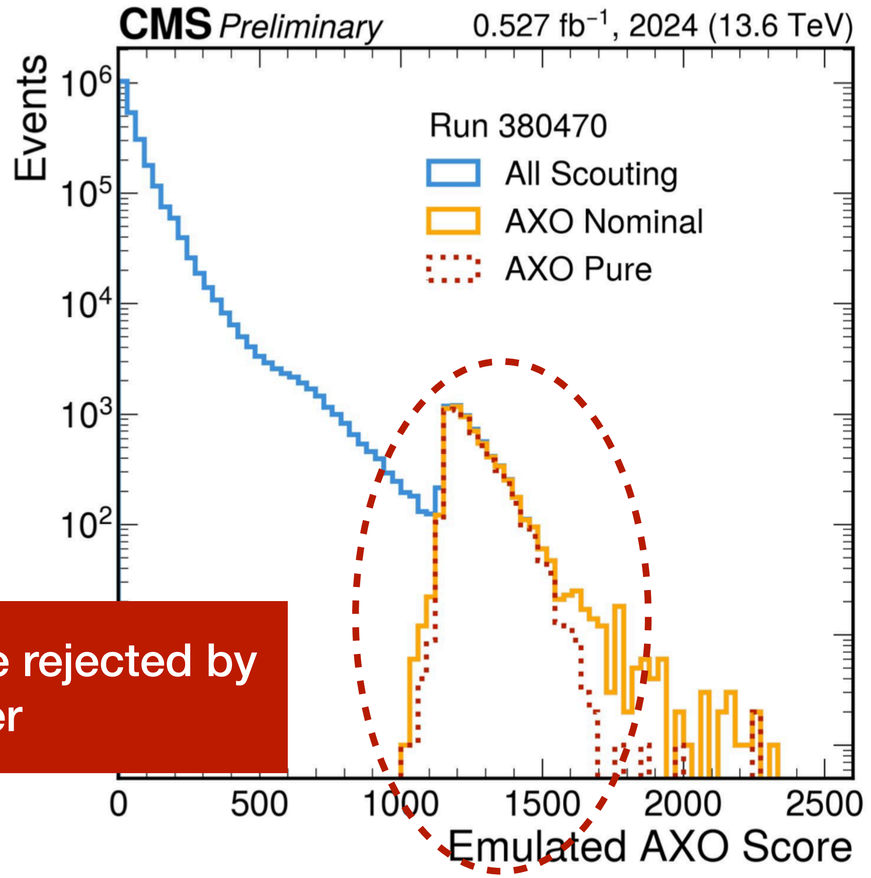
- *AXOL1TL:*
  - *Running since 2023, added to trigger decision in 2024*
  - *Encoder of a Dense (Variational) auto encoder*
  - *Uses the pull of the Gaussians in the latents space as anomaly score*



$$\mathscr{L} = (1-\beta)\,||x-\hat{x}||^2 +$$

$$+\beta\,\frac{1}{2}(\mu^2 + \sigma^2 - 1 - \log\sigma^2)$$

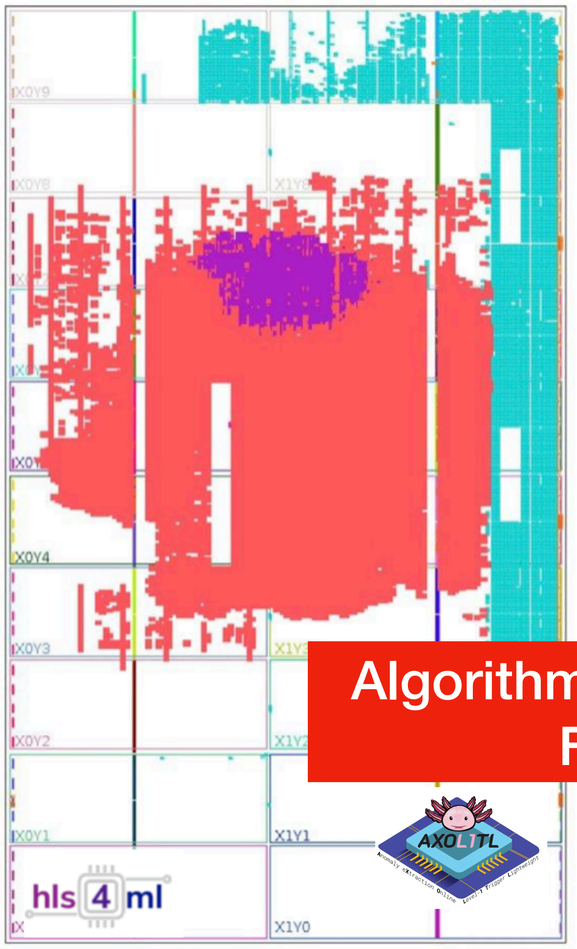
The Anomaly Score



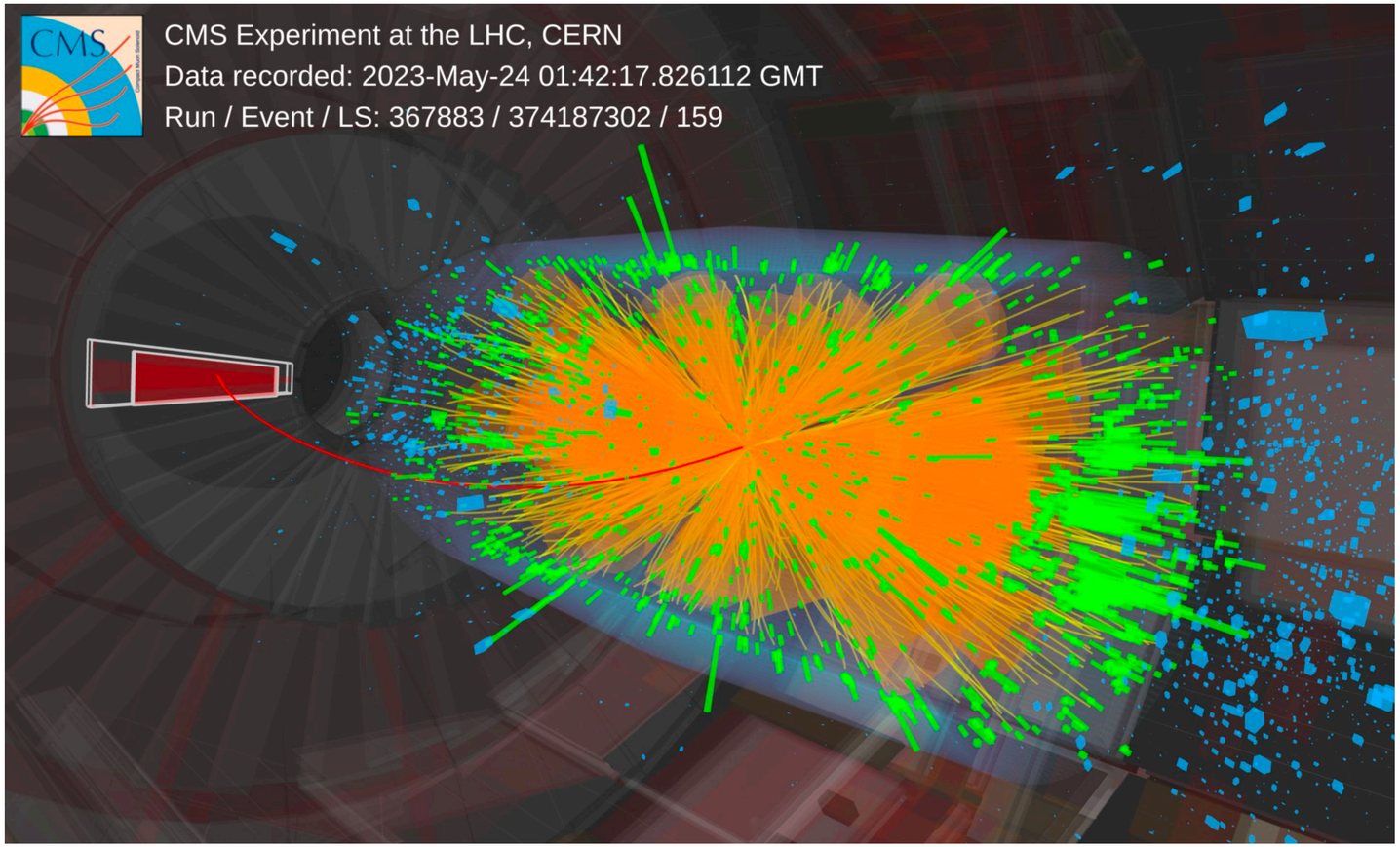
Events otherwise rejected by trigger

CMS Anomaly Detection Triggers

Latency	LUTs	FFs	DSPs	BRAMs
2 ticks 50 ns	2.1%	~0	0	0



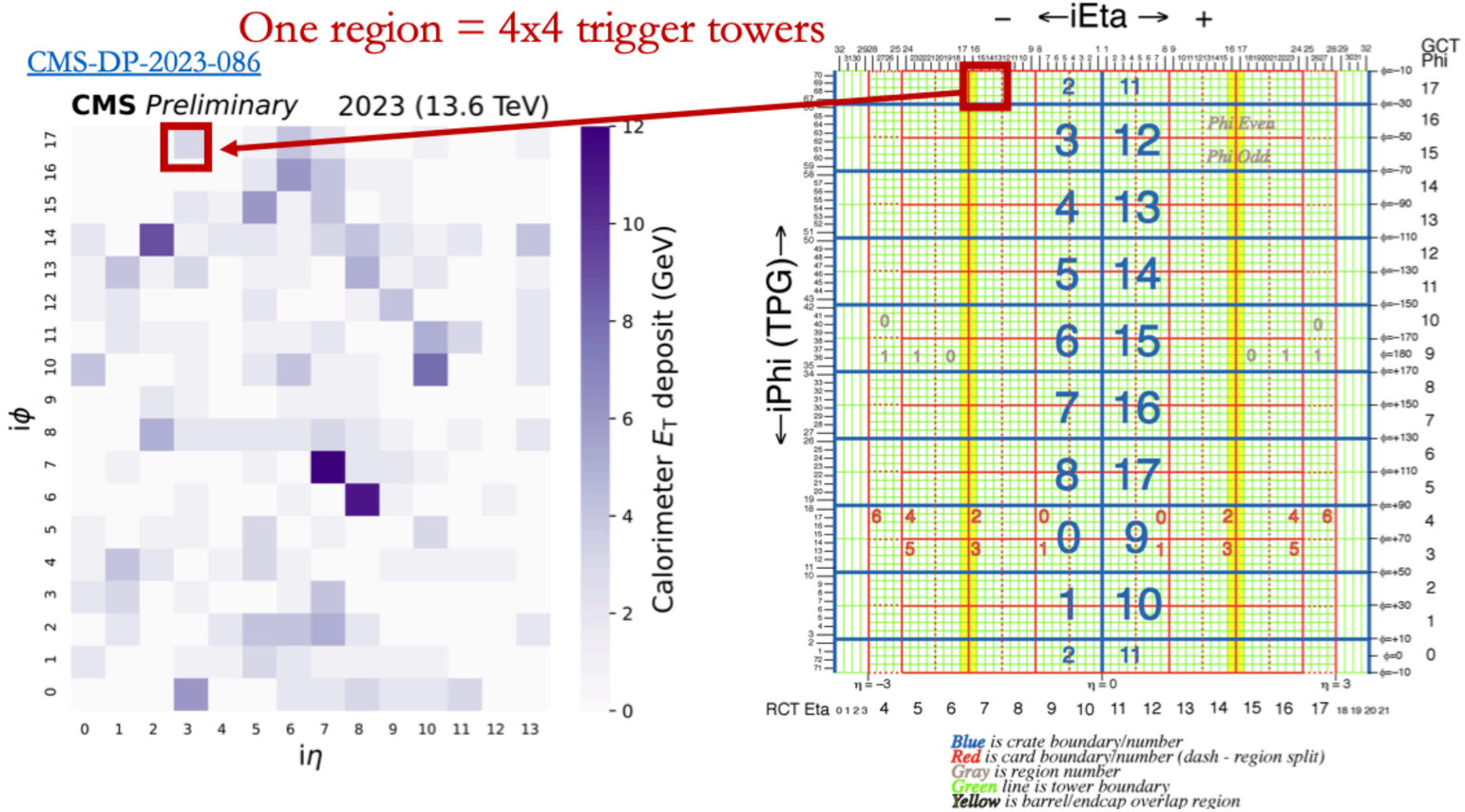
Algorithm Footprint on FPGA



An otherwise untriggered high-multiplicity event

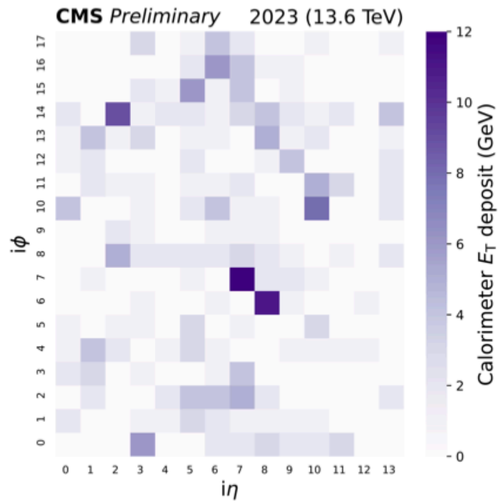
CMS Anomaly Detection Triggers

- *CICADA is a convolutional auto encoder running on a coarse map of calorimeter energy deposits*
  - *Too big for current FPGAs*
  - *Compressed with KD: we deploy a regression of the loss, quantized and pruned*
  - *Went online in 2024, added to trigger decision in 2025*

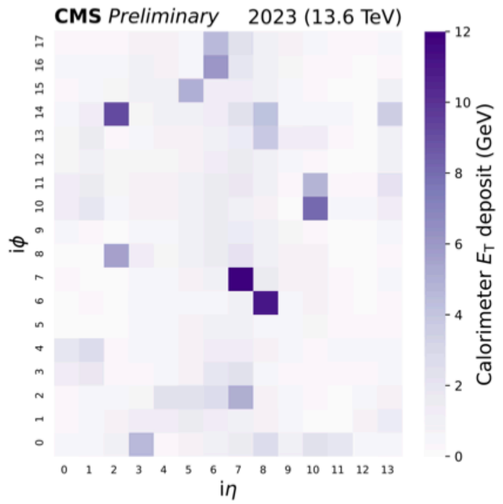


CMS Anomaly Detection Triggers

ZeroBias data

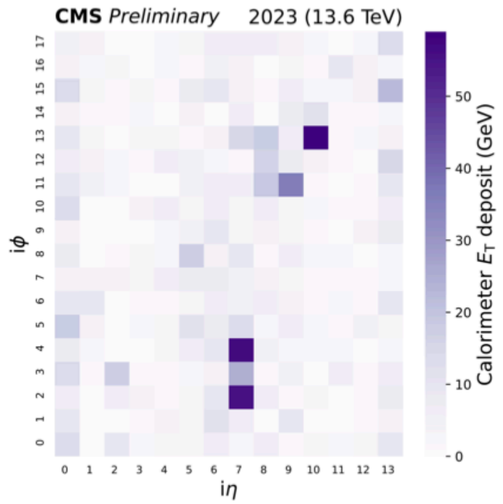


Input

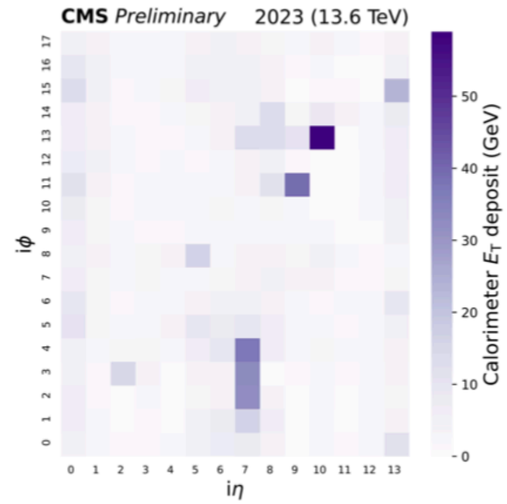


Reconstruction

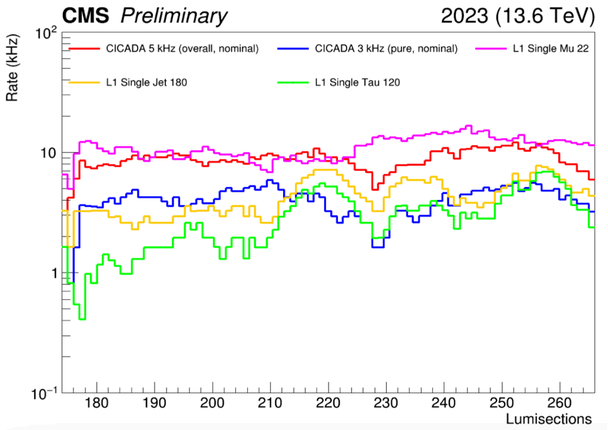
BSM MC signal



Input



Reconstruction



- *Stable performance in the L1 system*
- *Will start taking data next year*