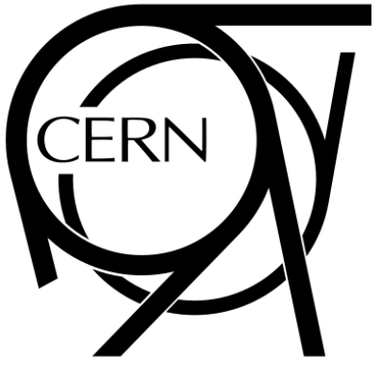
Deep learning on FPGAs for L1 trigger and Data Acquisition

CHEP 2018, 9-13 July 2018, Sofia, Bulgaria

Javier Duarte, Sergo Jindariani, Ben Kreis, Ryan Rivera, Nhan Tran (Fermilab)
Jennifer Ngadiuba, Maurizio Pierini (CERN)
Edward Kreinar (Hawkeye 360)
Phil Harris, Song Han, Dylan Rankin (MIT)
Zhenbin Wu (University of Illinois at Chicago)
Sioni Summers (Imperial College London)

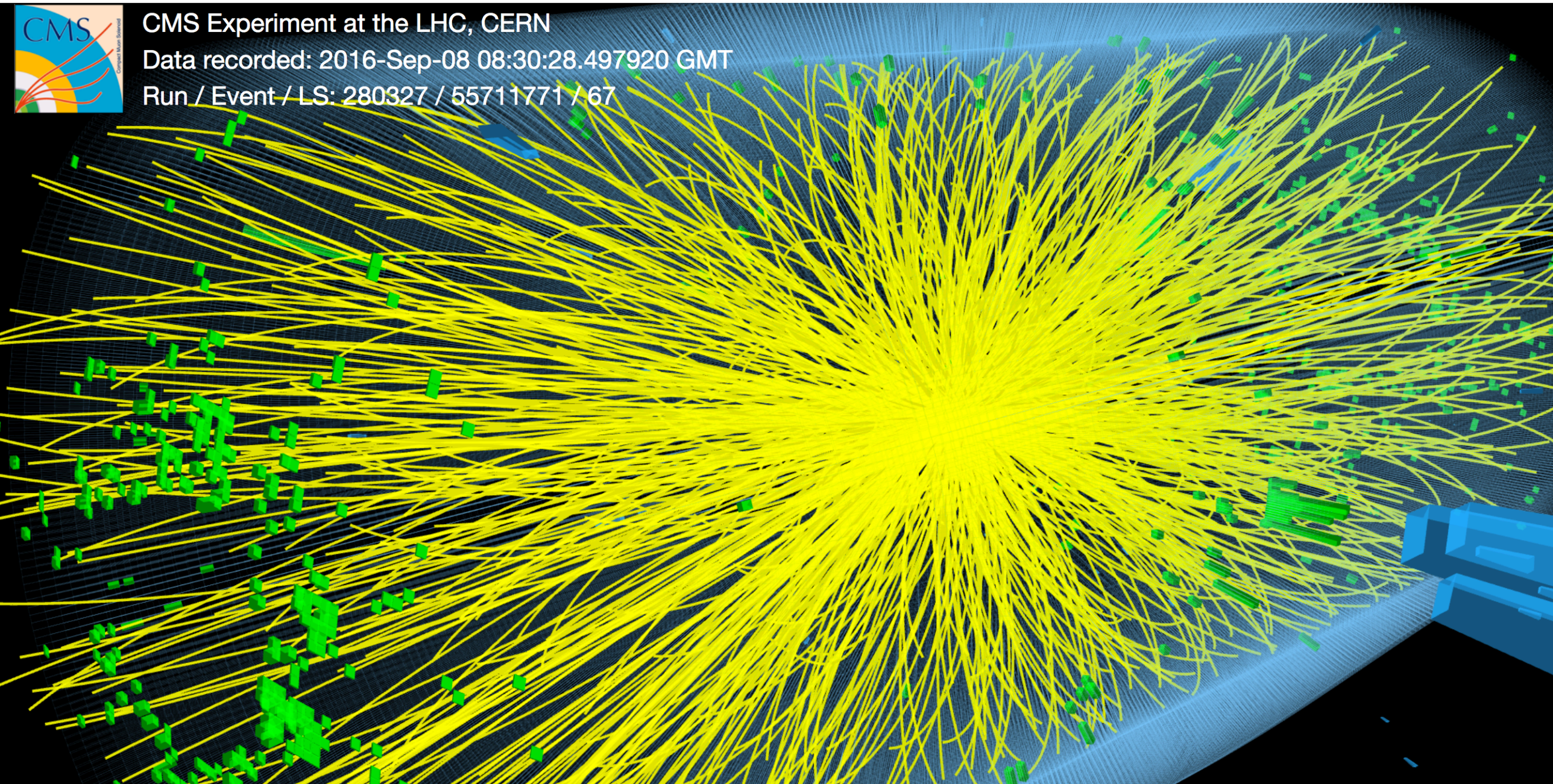


Motivation

The challenge: triggering at (HL-)LHC

The challenge: triggering at (HL-)LHC

Extreme bunch crossing frequency of 40 MHz → extreme data rates O(100 TB/s)
“Triggering” = filter events to reduce data rates to manageable levels



The challenge: triggering at (HL-)LHC

2016: <PU> ~ 20-50

2017 + Run 3: <PU> ~ 50-80

HL-LHC: 140-200

Squeeze the beams to increase data rates
→ multiple pp collisions per bunch crossing (pileup)

CHALLENGE: maintain physics in increasingly complex collision environment
→ untriggered events lost forever!

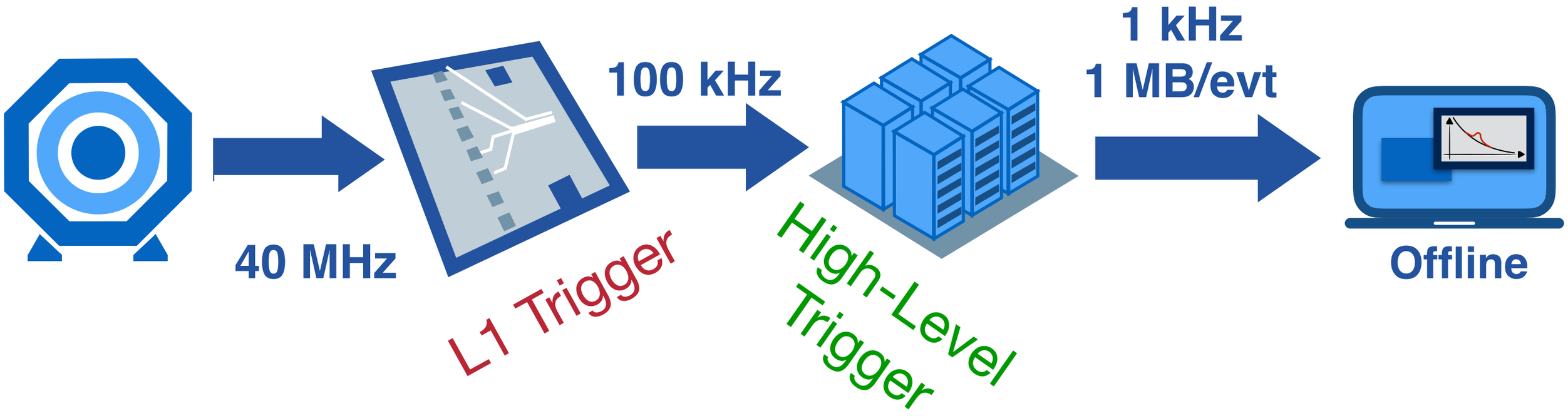
Sophisticated techniques needed to preserve the physics!

Particle flow and PUPPI (see poster by G. Petrucciani)

Machine learning THIS TALK!

A typical trigger system

Triggering typically performed in multiple stages @ ATLAS and CMS



Absorbs 100s Tb/s

Trigger decision to be made in O(μs)

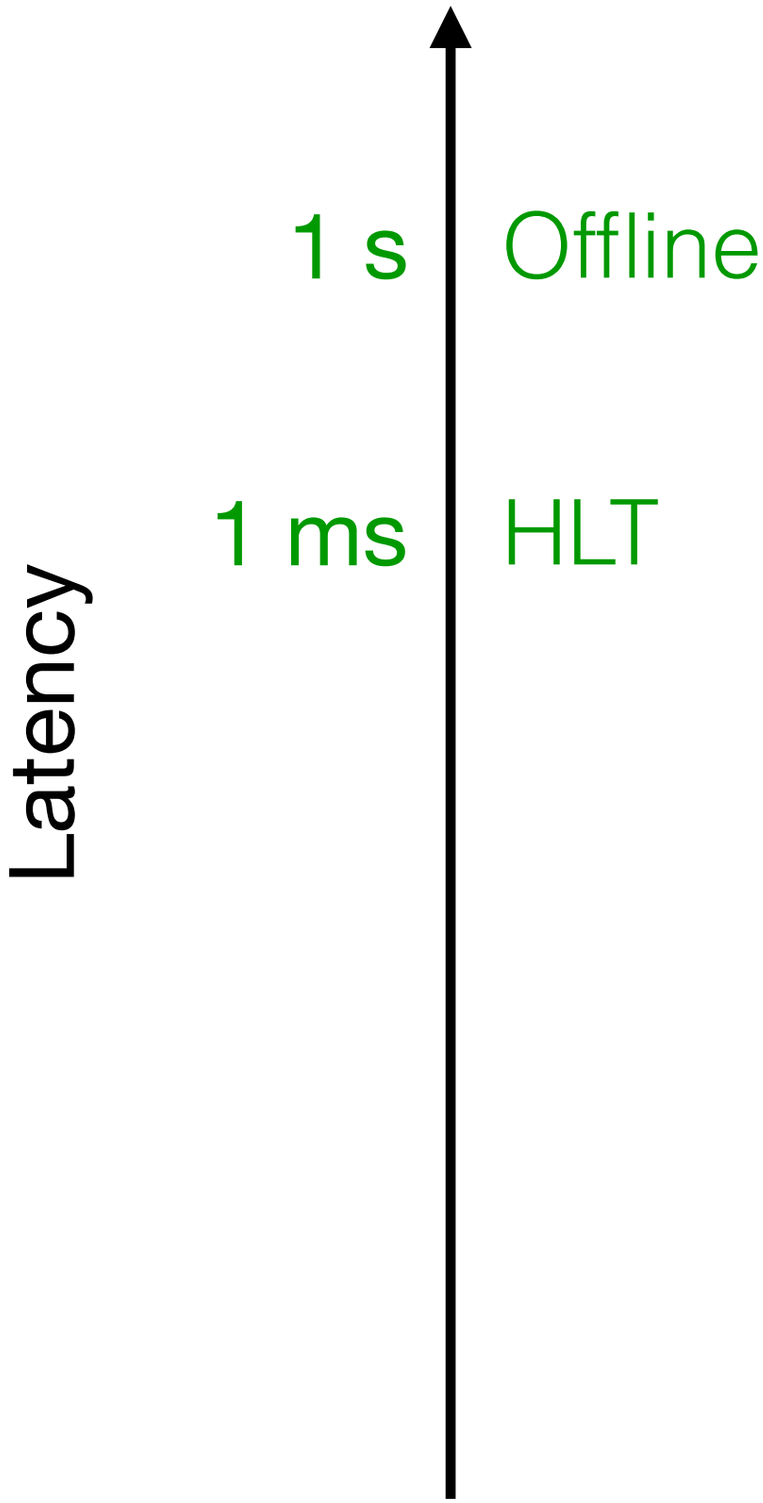
Latencies require all-FPGA design

Computing farm for detailed analysis of the full event

Latency O(100 ms)

For HL-LHC upgrade: latency and output rates will increase by ~ 3 (ex: for CMS 3.8 → 12.5 μs @ L1)

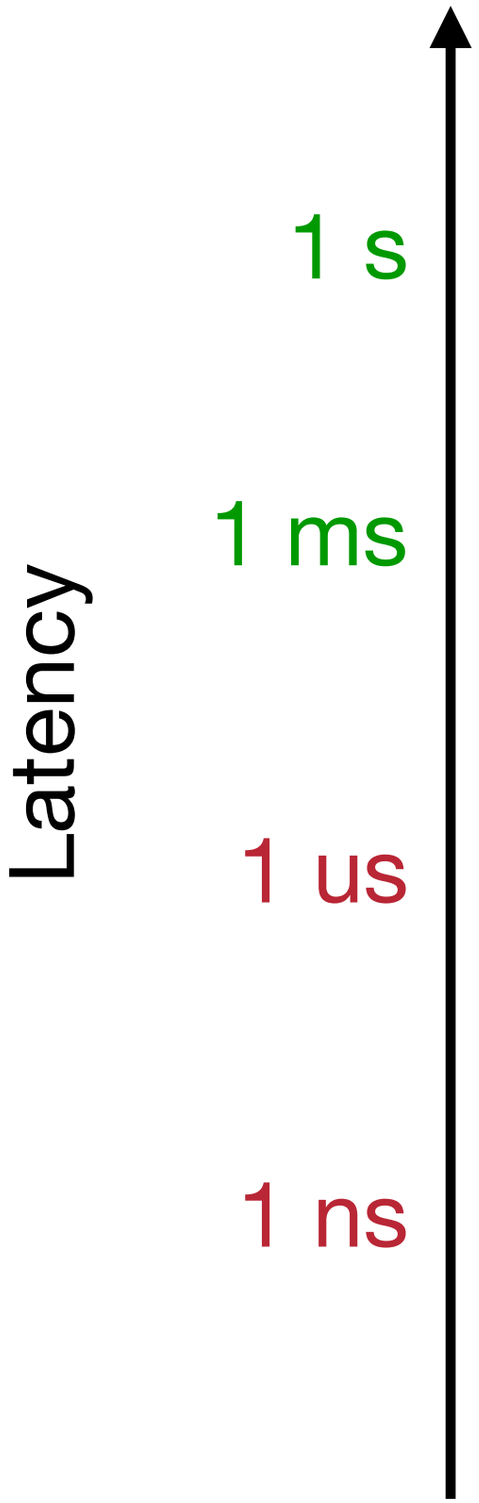
Data processing @ LHC



ML methods typically employed in offline analysis or longer latency trigger tasks

Many successes in HEP: identification of b-quark jets, Higgs candidates, particle energy regression, analysis selections, ….

Data processing @ LHC



ML methods typically employed in offline analysis or longer latency trigger tasks

Many successes in HEP: identification of b-quark jets, Higgs candidates, particle energy regression, analysis selections, ….

Exploration of ML algorithms in low-latency, real-time processing has just begun!

What can we do in < us on one FPGA?

What are FPGAs?

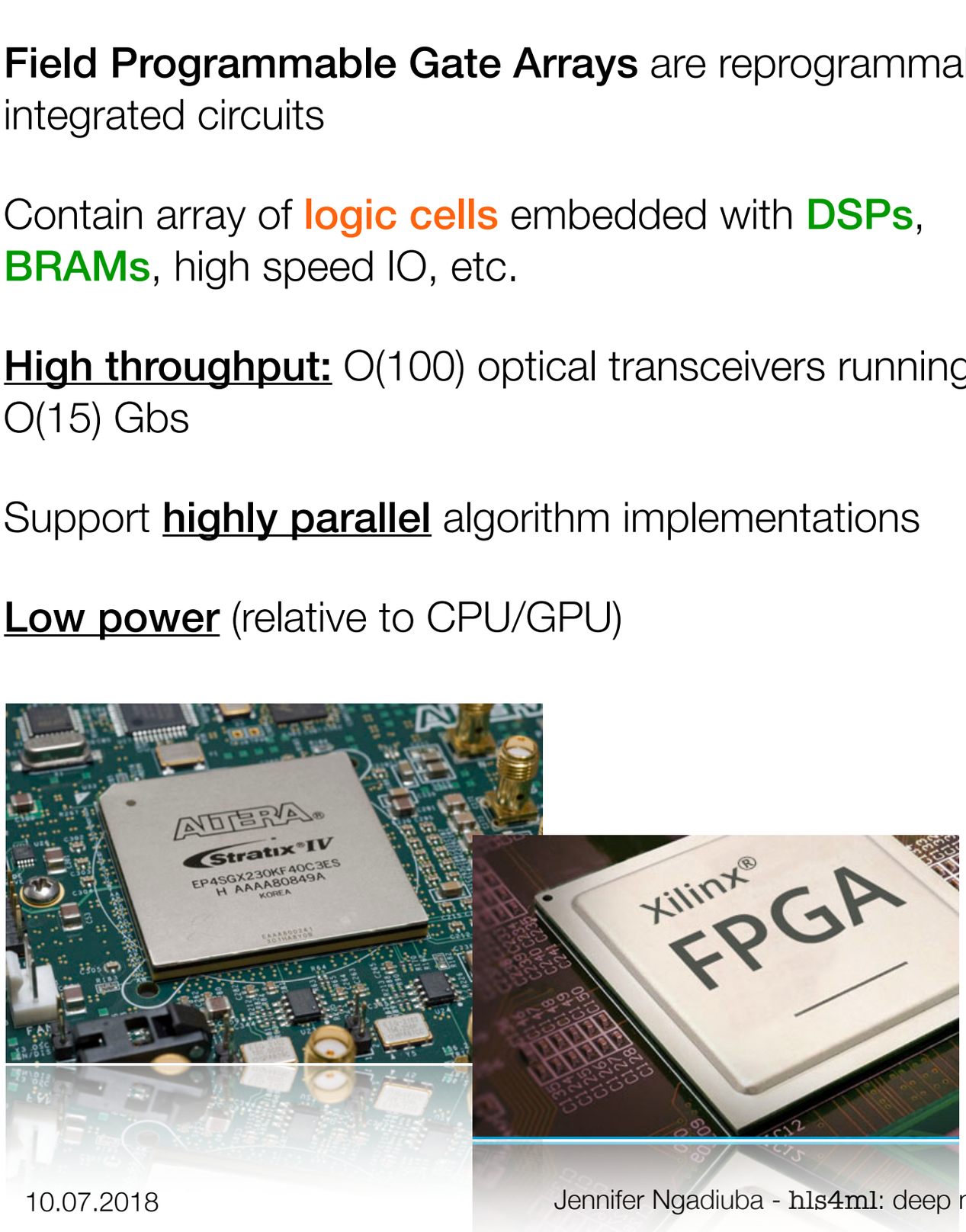
Field Programmable Gate Arrays are reprogrammable integrated circuits

Contain array of **logic cells** embedded with **DSPs**, **BRAMs**, high speed IO, etc.

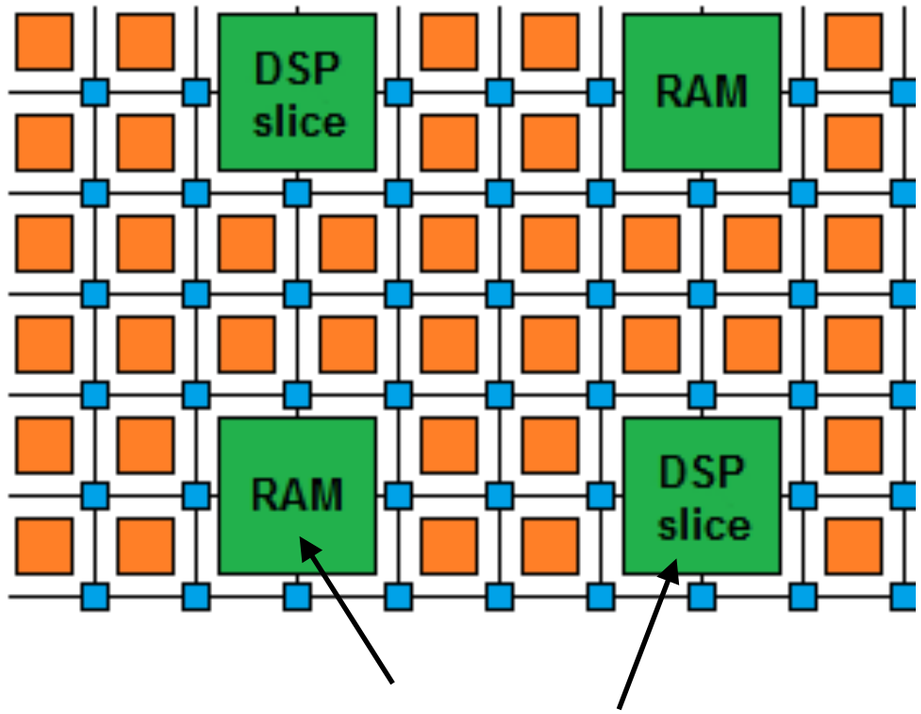
High throughput: O(100) optical transceivers running at O(15) Gbs

Support **highly parallel** algorithm implementations

Low power (relative to CPU/GPU)



FPGA diagram



Digital Signal Processors (DSPs):
logic units used for multiplications

Random-access memories (RAMs):
embedded memory elements

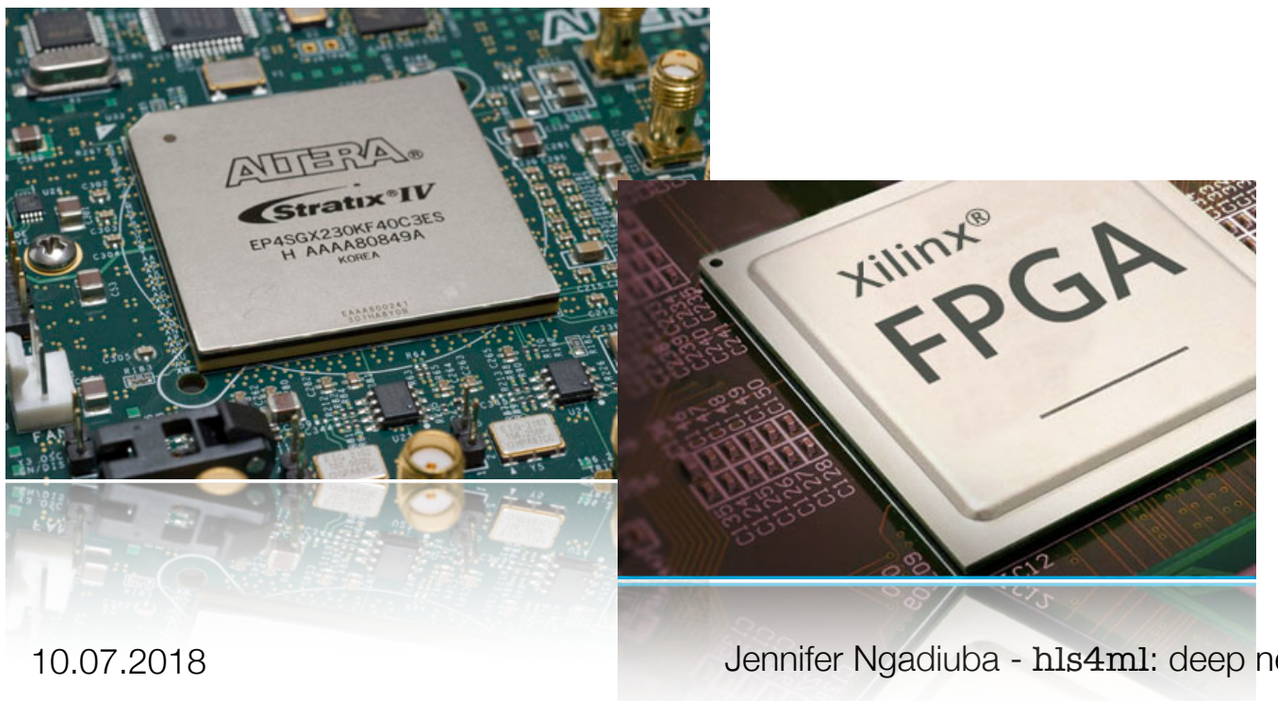
How are FPGAs programmed?

High Level Synthesis

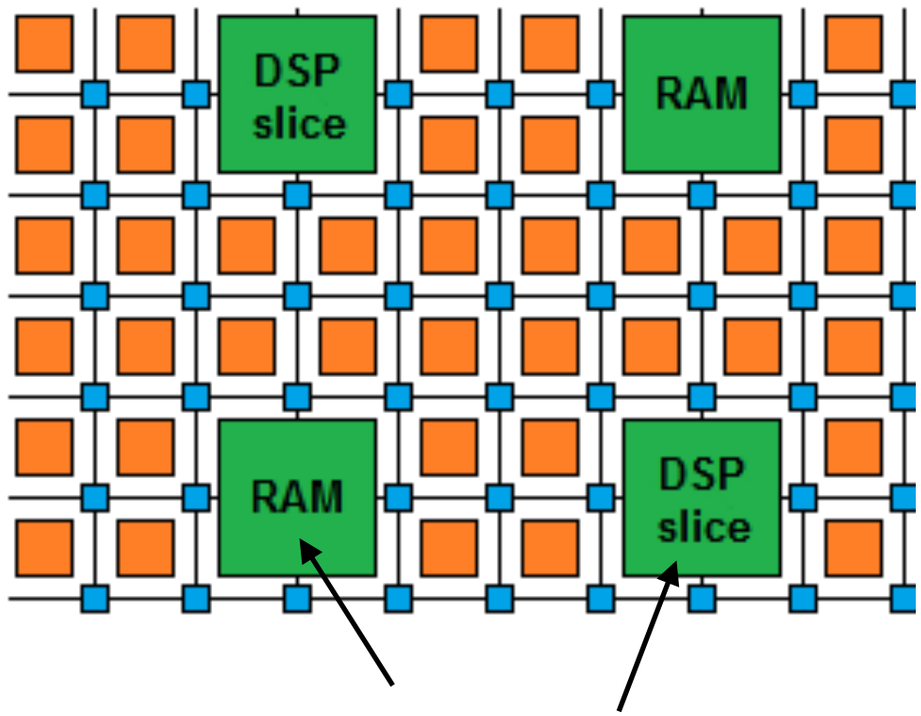
generate standard register-transfer level (RTL) code for FPGA from more common C/C++ code

pre-processor directives and constraints used to optimize the timing

<u>drastic decrease in firmware development time!</u>



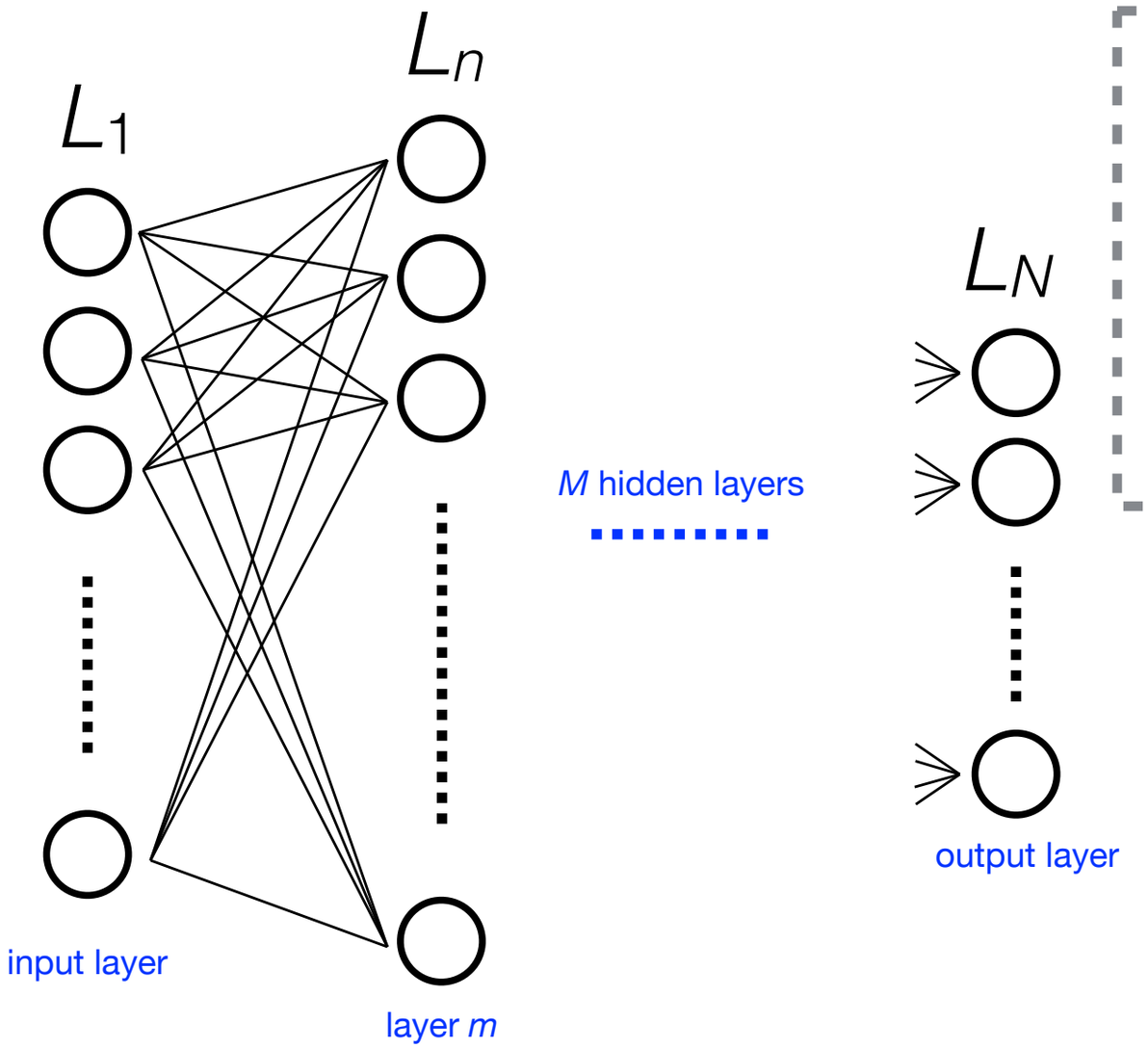
FPGA diagram



Digital Signal Processors (DSPs):
logic units used for multiplications

Random-access memories (RAMs):
embedded memory elements

Neural network inference



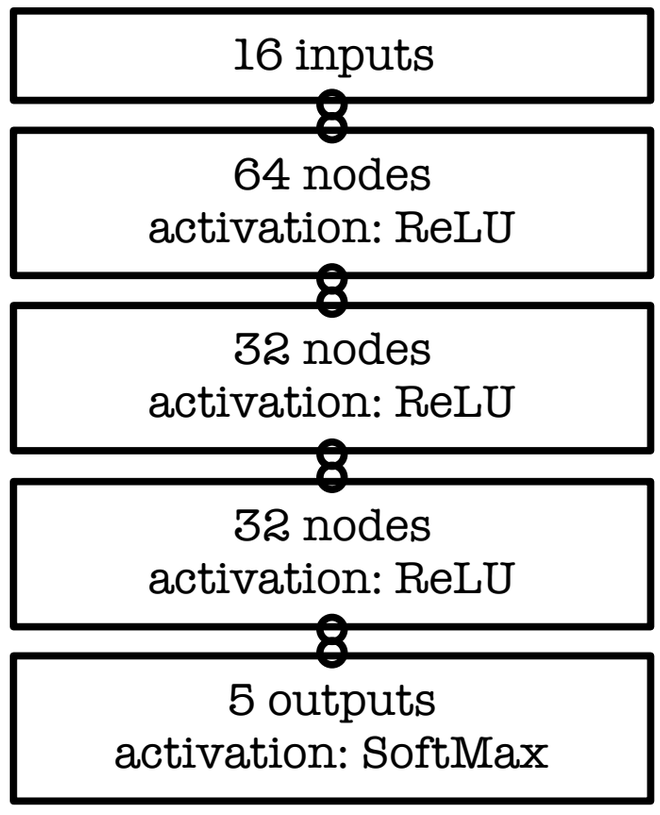
$$\mathbf{x}_n = g_n(\mathbf{W}_{n,n-1}\mathbf{x}_{n-1} + \mathbf{b}_n)$$

activation function — precomputed and stored in BRAMs

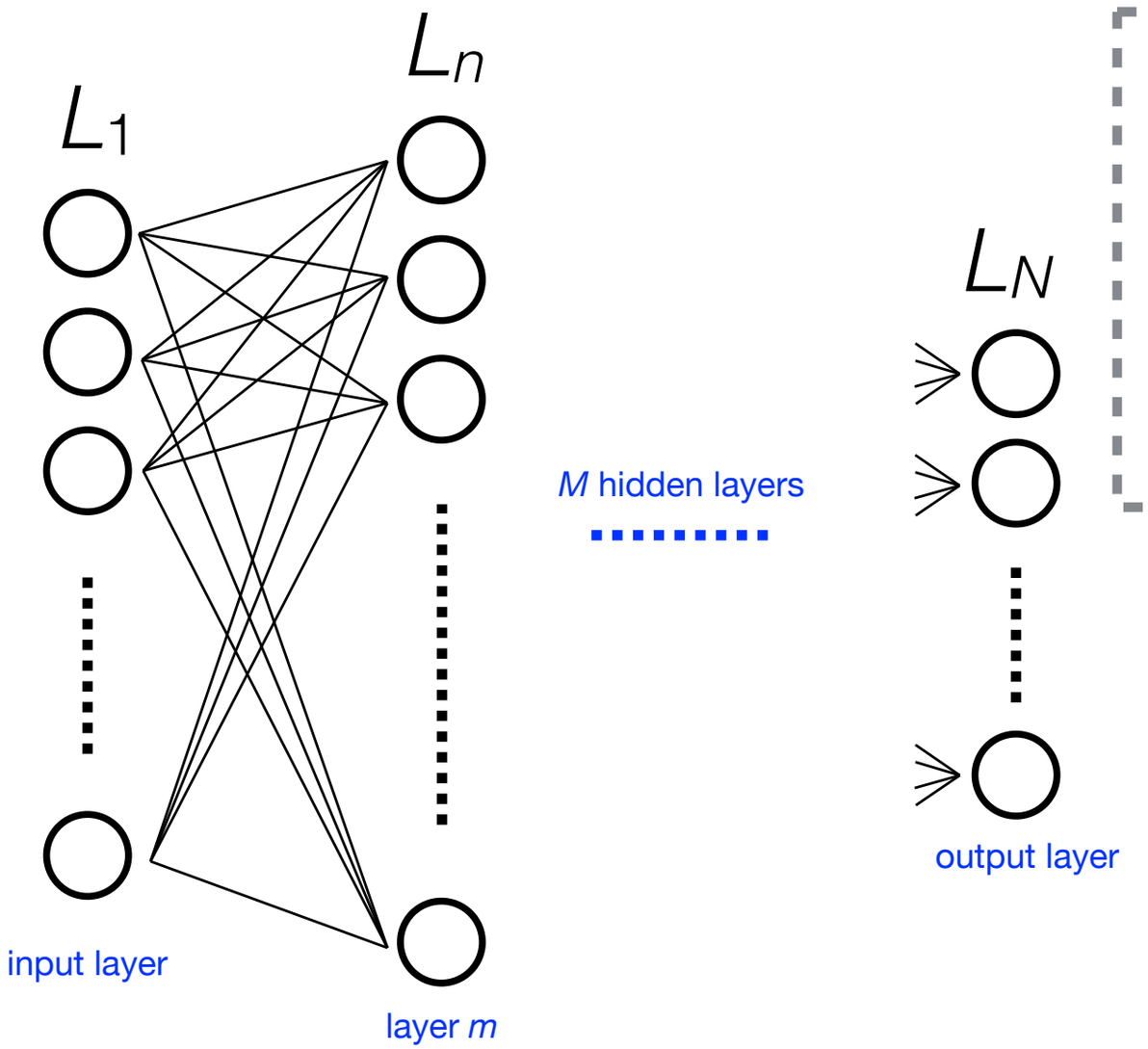
multiplication — DSPs

addition — logic cells

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



Neural network inference



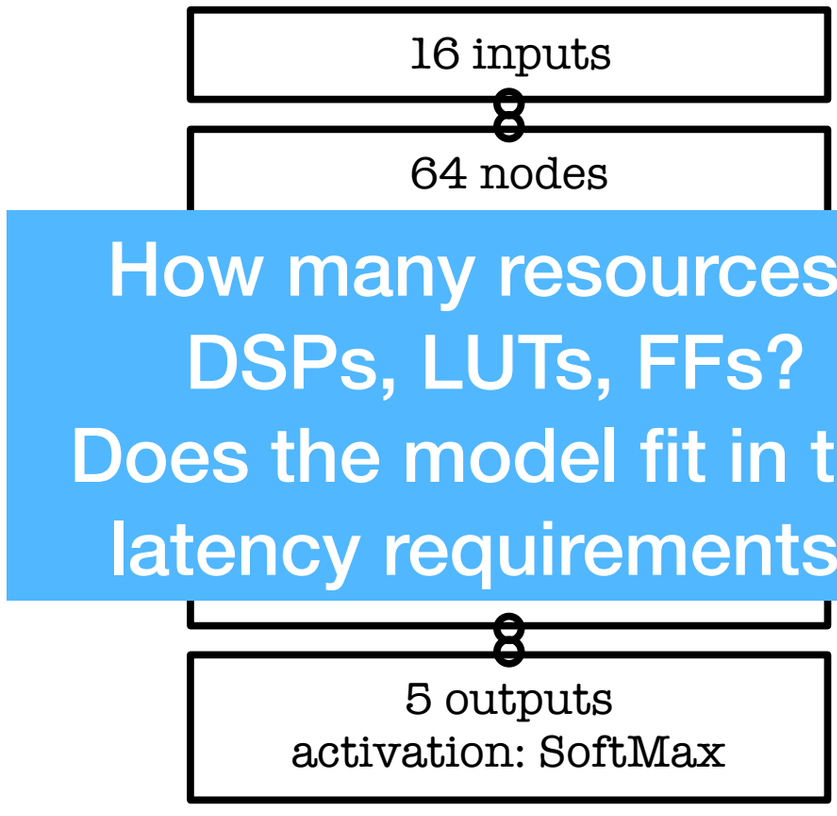
$$\mathbf{x}_n = g_n(\mathbf{W}_{n,n-1}\mathbf{x}_{n-1} + \mathbf{b}_n)$$

activation function — precomputed and stored in BRAMs

multiplication — DSPs

addition — logic cells

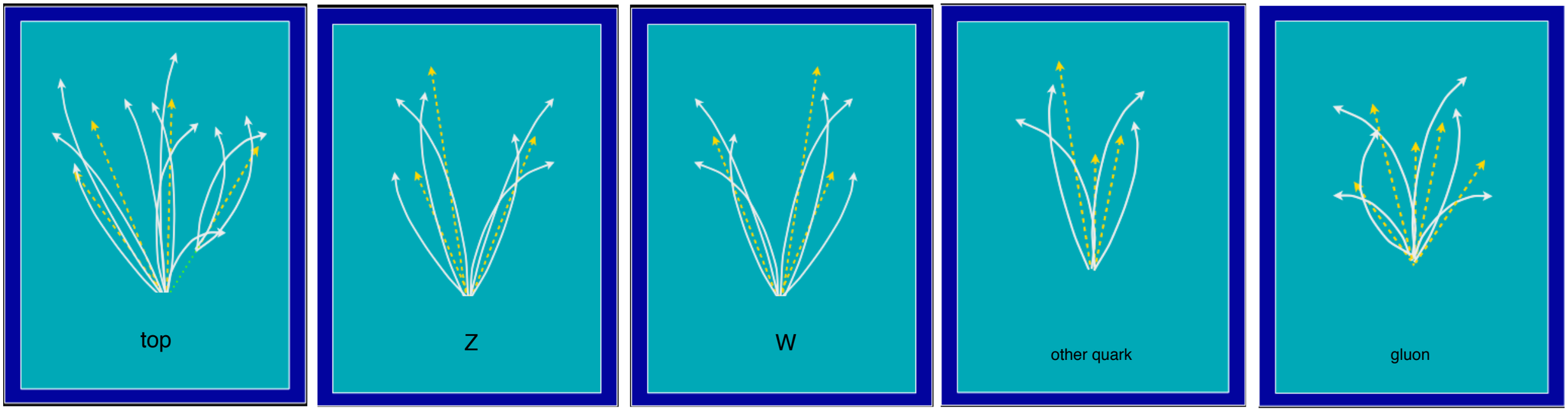
$$N_{\text{multiplications}} = \sum_{n=2}^{N} L_{n-1} \times L_n$$



How many resources?
DSPs, LUTs, FFs?
Does the model fit in the latency requirements?

Case study: jet tagging

Study a multi-classification task: discrimination between highly energetic (boosted) ***q, g, W, Z, t*** initiated jets



t→bW→bqq	**Z→qq**	**W→qq**	**q/g background**
3-prong jet	2-prong jet	2-prong jet	no substructure and/or mass ~ 0

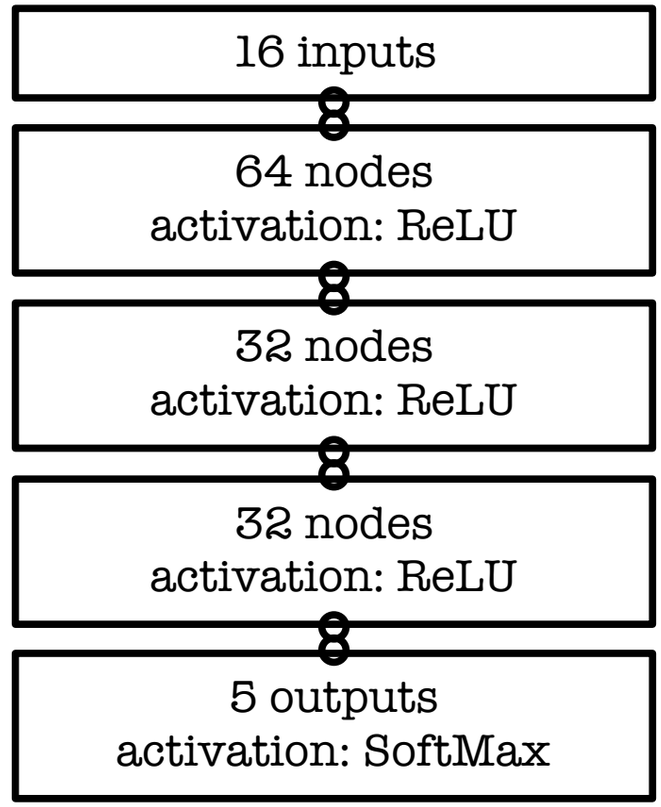
Signal: reconstructed as one massive jet with substructure

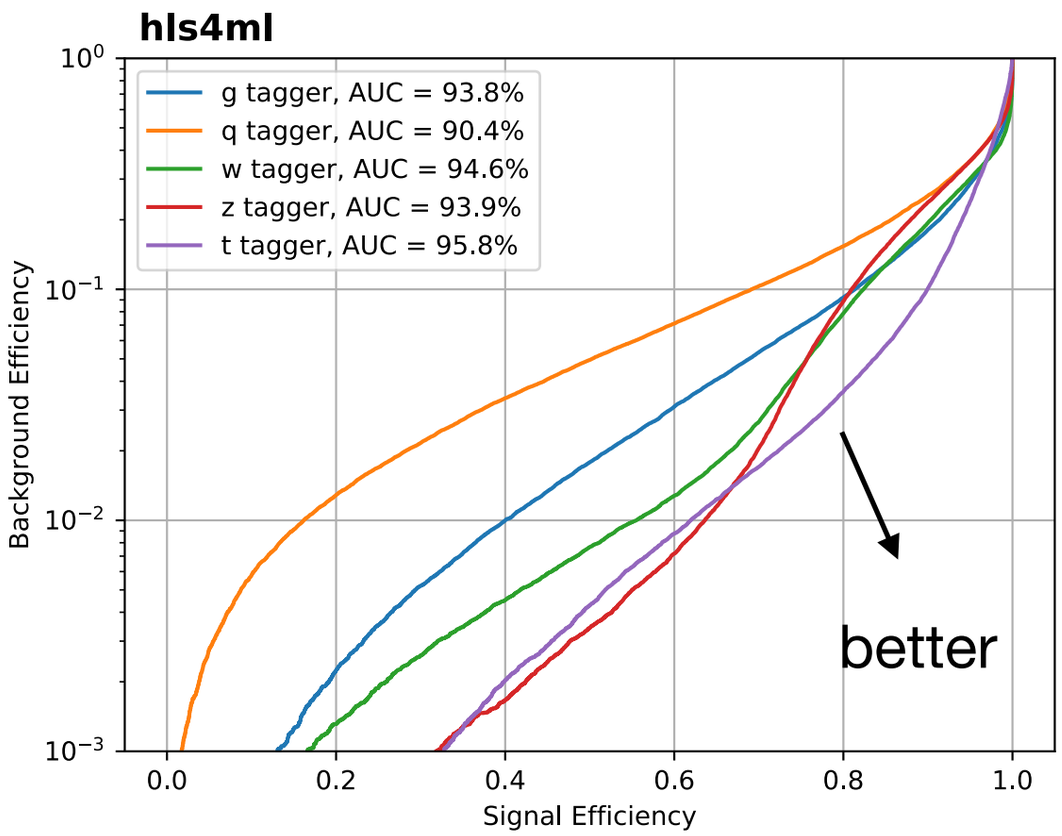
Jet substructure observables used to distinguish signal vs background [*]

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

Case study: jet tagging

- Study a 5-output multi-classification task: discrimination ***q, g, W, Z, t*** initiated jets
- Fully connected neural network with **16 inputs**:
  - **mass** (Dasgupta et al., arXiv:1307.0007), **multiplicity, energy correlation functions** (Larkoski et al., arXiv:1305.0007)
  - expert-level features not necessarily realistic for L1trigger, but the lessons here are generic





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

Efficient NN design for FPGAs

- We focus on tuning the neural network inference such that it uses the FPGA resources efficiently and meets latency constraints

- We have three handles:

  - **compression:** reduce number of synapses or neurons

  - **quantization:** reduces the precision of the calculations (inputs, weights, biases)

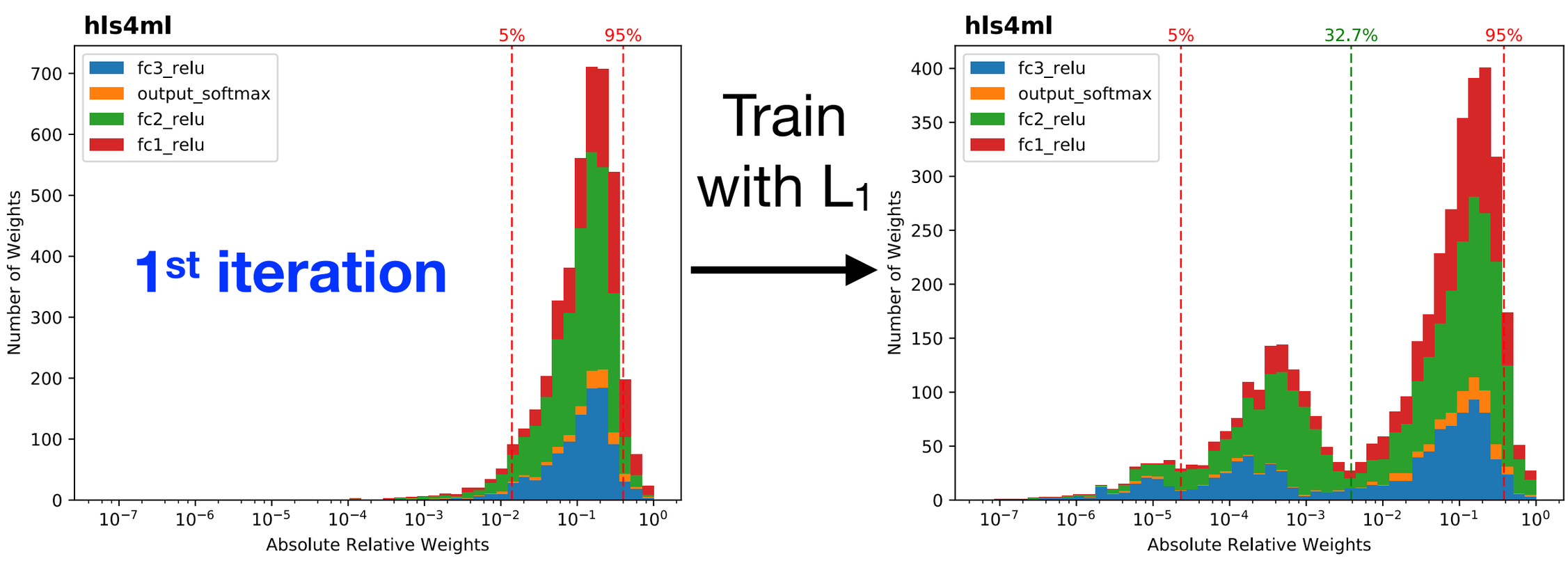
  - **parallelization**: tune how much to parallelize to make the inference faster/slower versus FPGA resources

Efficient NN design: compression

- Iterative approach:
  - train with **L1 regularization** (loss function augmented with penalty term):

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

  - sort the weights based on the value relative to the max value of the weights in that layer



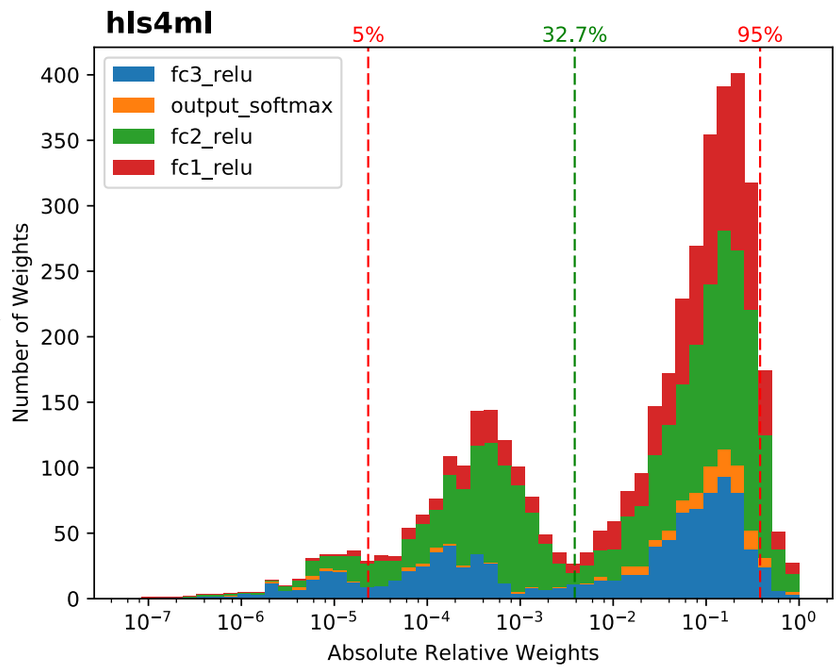
Efficient NN design: compression

- Iterative approach:
  - train with **L1 regularization** (loss function augmented with penalty term):

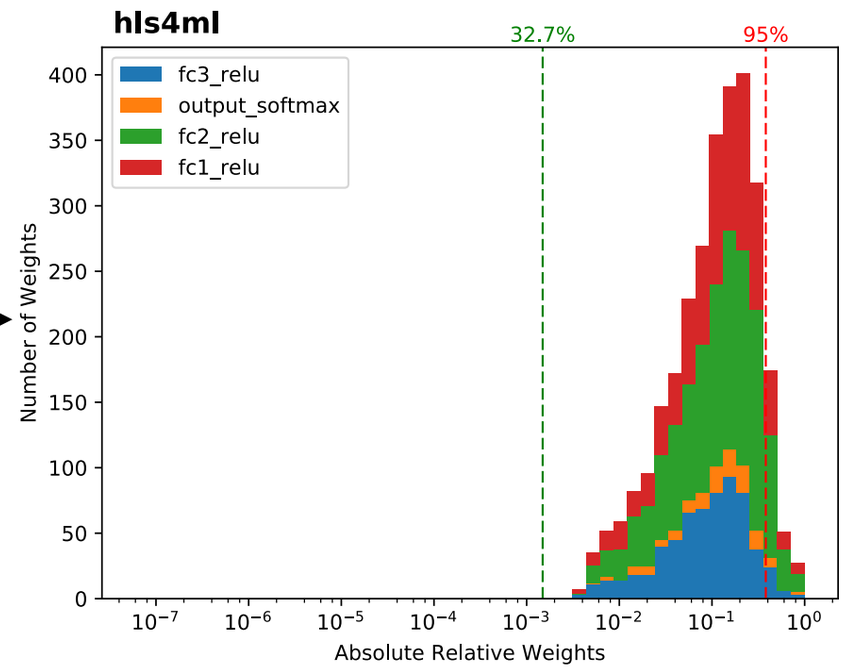
$$L_\lambda(\vec{w}) = L(\vec{w}) + \lambda||\vec{w}_1||$$

  - sort the weights based on the value relative to the max value of the weights in that layer
  - prune weights falling below a certain percentile and retrain

Train with L_1

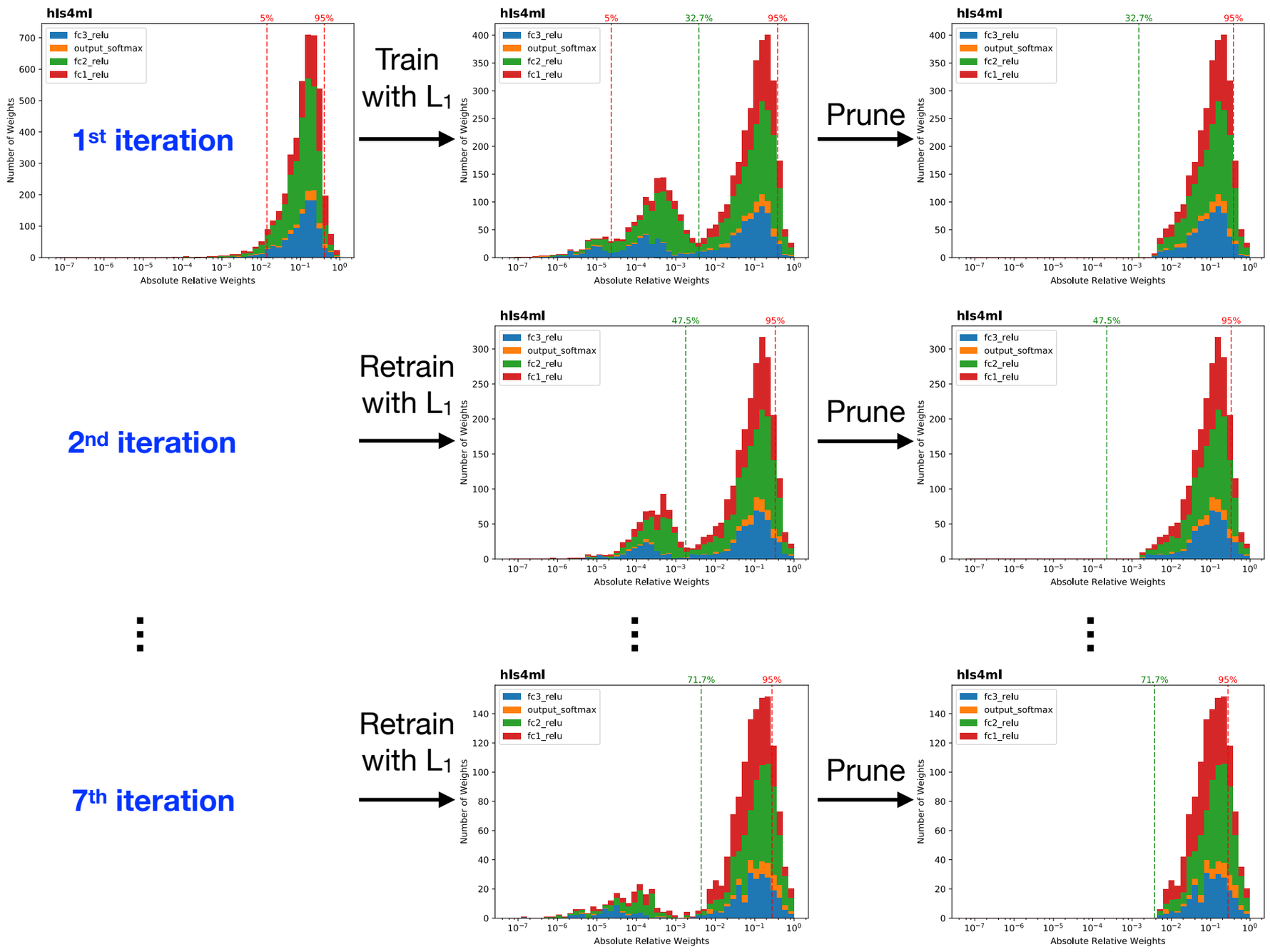


Prune



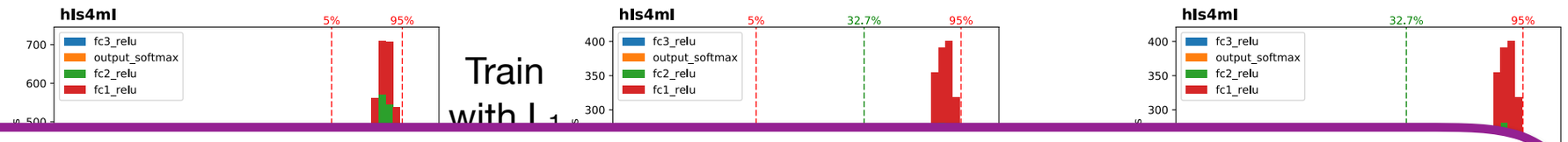
Efficient NN design: compression

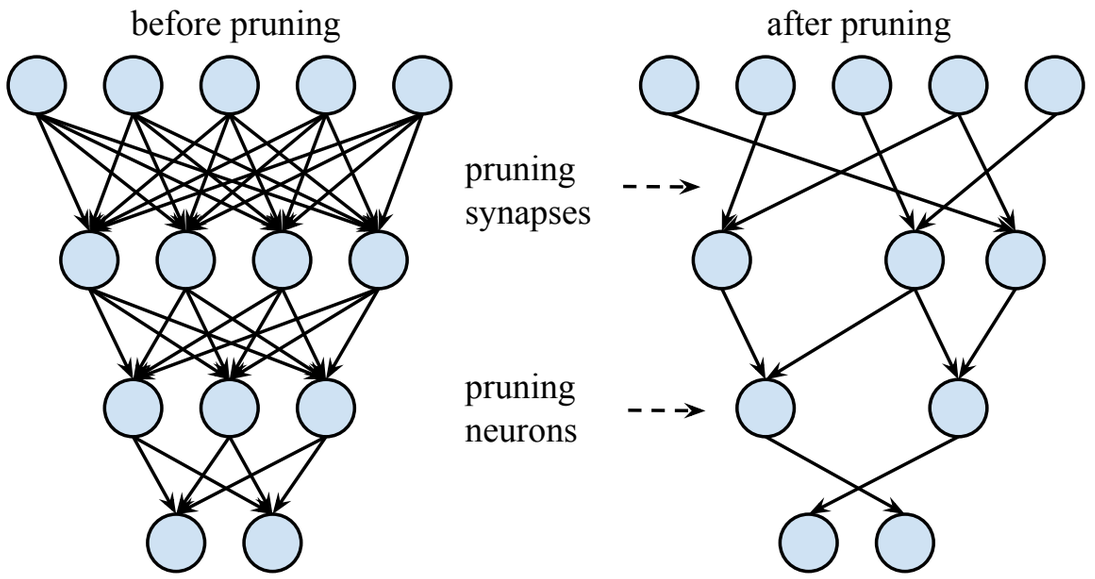
Prune and repeat the training for 7 iterations



Efficient NN design: compression

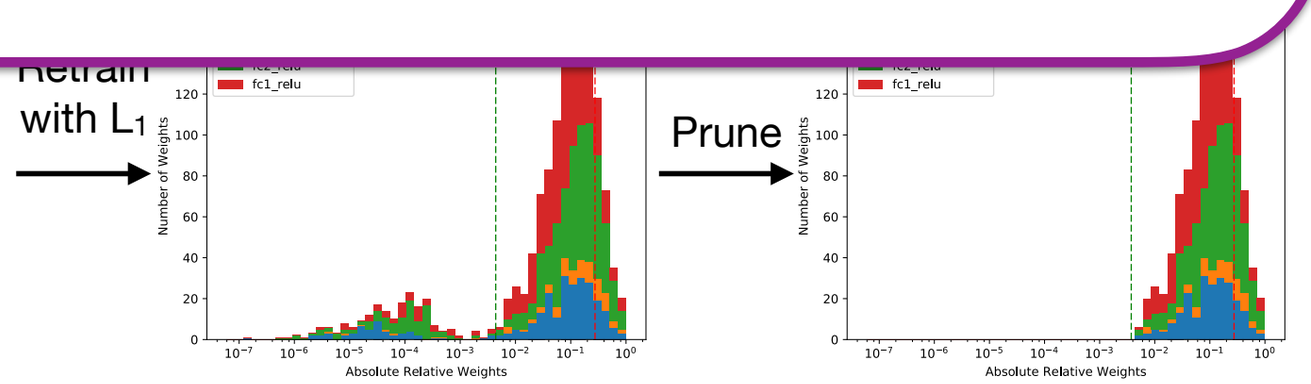
Prune and repeat the train for 7 iterations





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

7th iteration



Efficient NN design for FPGAs

- We focus on tuning the neural network inference such that it uses the FPGA resources efficiently and meets latency constraints

- We have three handles:

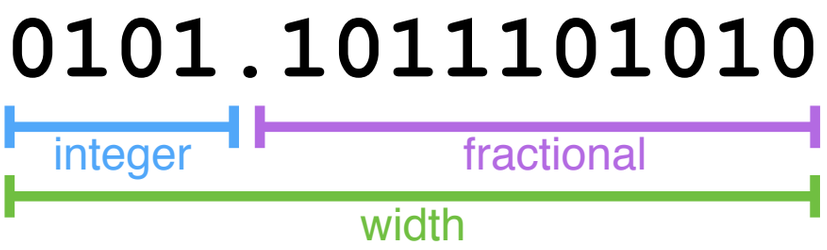
  - **compression:** reduce number of synapses or neurons

  - **quantization:** reduces the precision of the calculations (inputs, weights, biases)

  - **parallelization**: tune how much to parallelize to make the inference faster/slower versus FPGA resources

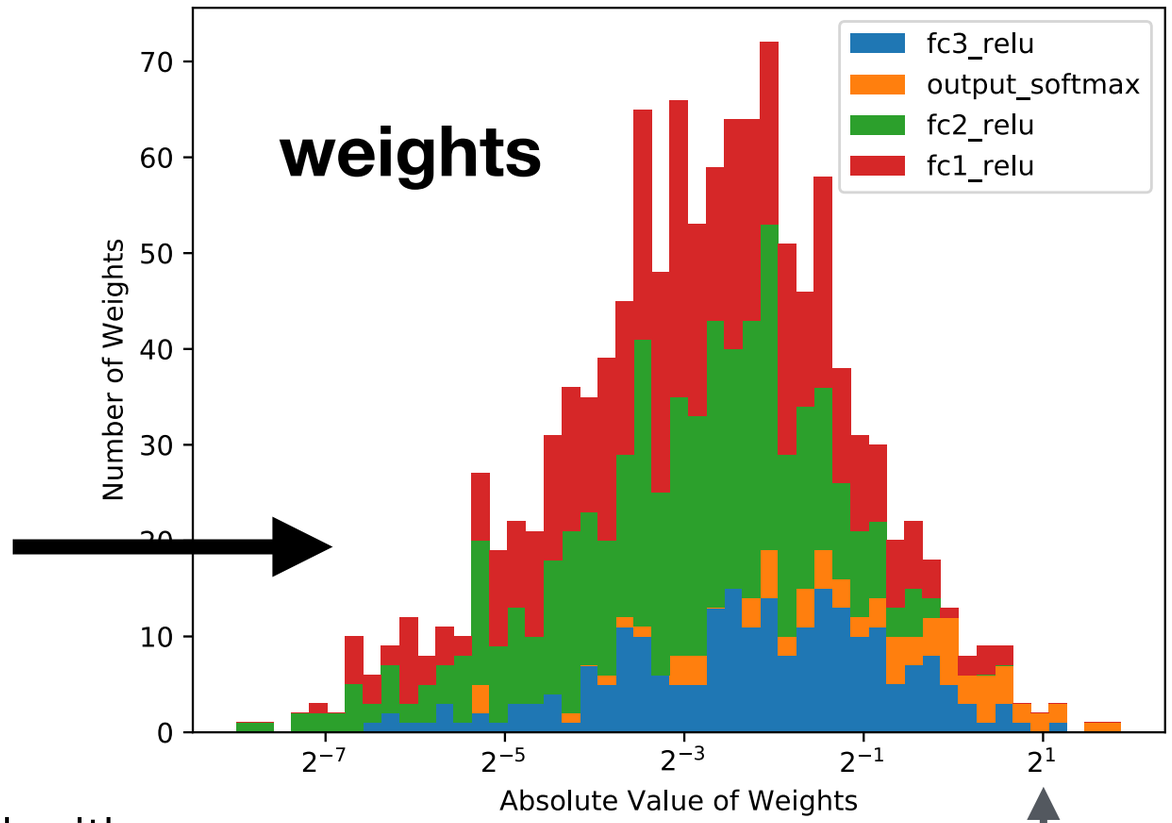
Efficient NN design: quantization

- In FPGAs use **fixed point data types** → less resources and latency than 32-bit floating point
- NN inputs, weights, biases, outputs represented as ap_fixed<width,integer>




```
ap_fixed<14,4>
```

- To avoid overflow/underflow of weights at least 3 bits needed



- But need more bits for neurons as computed with multiplications and sums → we perform a **scan of physics performance versus bit precision**

integer bits = 2 + 1 for sign
(need more for neurons)

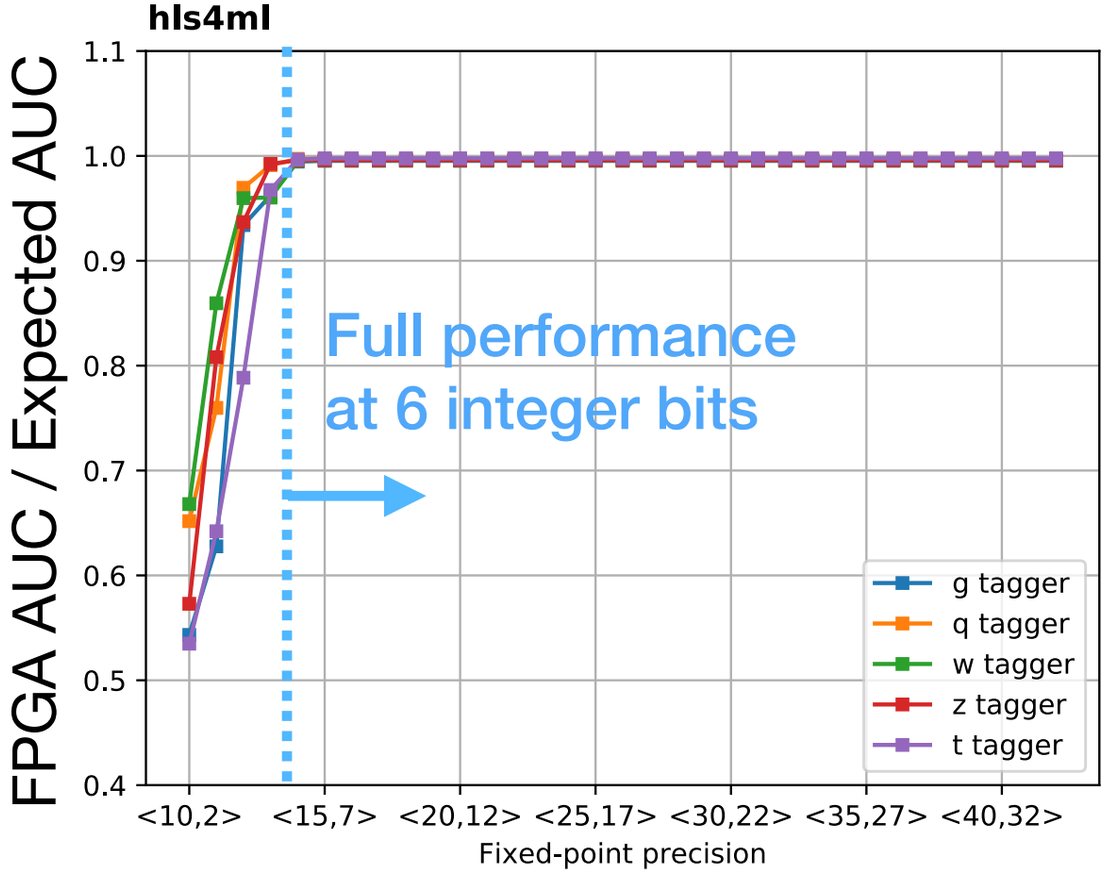
Efficient NN design: quantization

ap_fixed<width,integer>

- Quantify the performance of the classifier with the AUC
- Expected AUC = AUC achieved by 32-bit floating point inference of the neural network

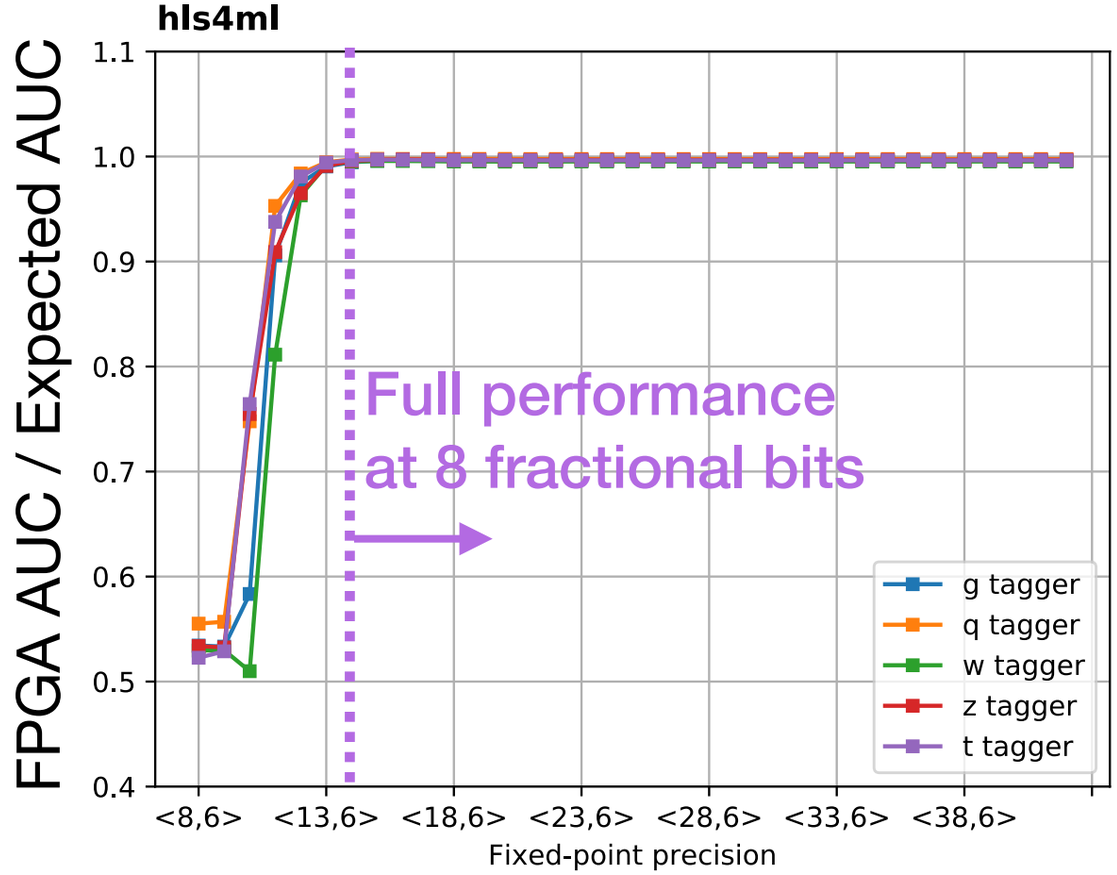
Scan integer bits

Fractional bits fixed to 8



Scan fractional bits

Integer bits fixed to 6



Efficient NN design for FPGAs

- We focus on tuning the neural network inference such that it uses the FPGA resources efficiently and meets latency constraints

- We have three handles:

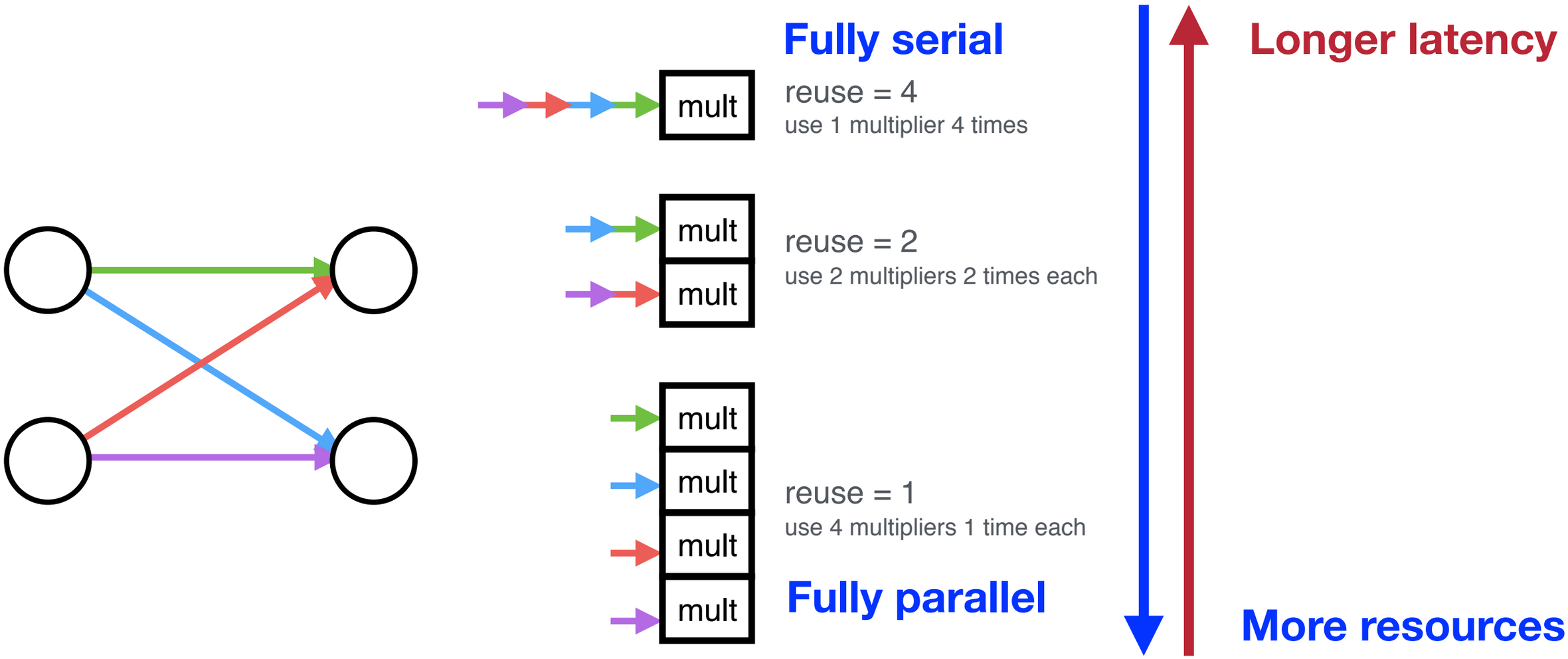
  - **compression:** reduce number of synapses or neurons

  - **quantization:** reduces the precision of the calculations (inputs, weights, biases)

  - **parallelization**: tune how much to parallelize to make the inference faster/slower versus FPGA resources

Efficient NN design: parallelization

- Trade-off between latency and FPGA resource usage determined by the parallelization of the calculations in each layer
- Configure the **“reuse factor”** = number of times a multiplier is used to do a computation



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

Compression, Quantization, and Parallelization made easy in

high level synthesis for machine learning

PYTORCH

Keras
TensorFlow
PyTorch
...

model

compressed model

Usual ML software workflow

Vivado™ HLS

hls4ml

Co-processing kernel

HLS conversion

HLS project

Custom firmware design

tune configuration
precision
reuse/pipeline

K + TensorFlow

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

Study details

GOAL

Map out FPGA performance, resource usage and latency versus compression, quantization, and parallelization hyperparameters

SETUP

Xilinx Vivado 2017.2

HLS target clock frequency: 200 MHz (5 clocks/BX)

Kintex Ultrascale, xcku115-flvb2104-2-i

- 1.4M logic cells, 5,520 DSPs, 1.3M FFs, 700k LUTs, 2200 BRAMs

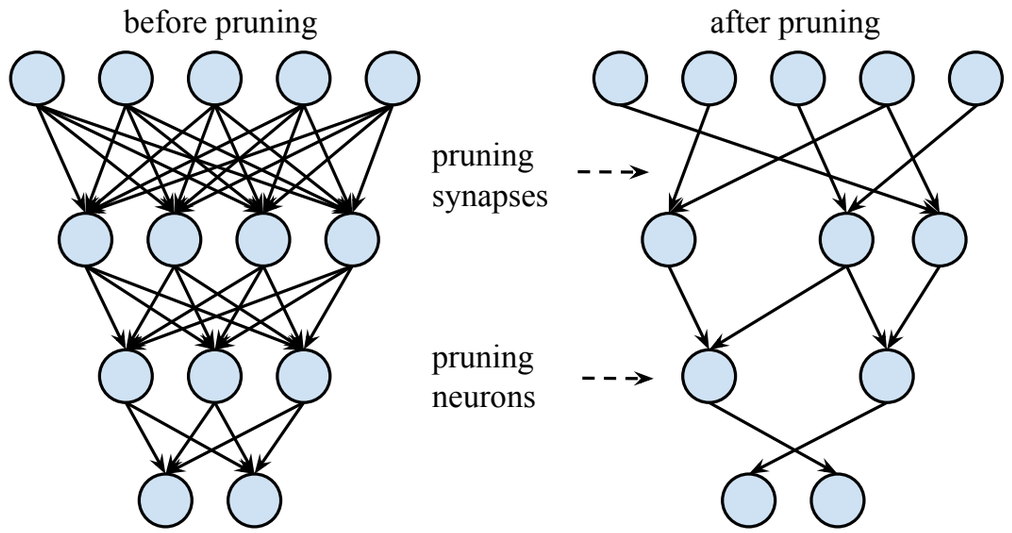
RESULTS

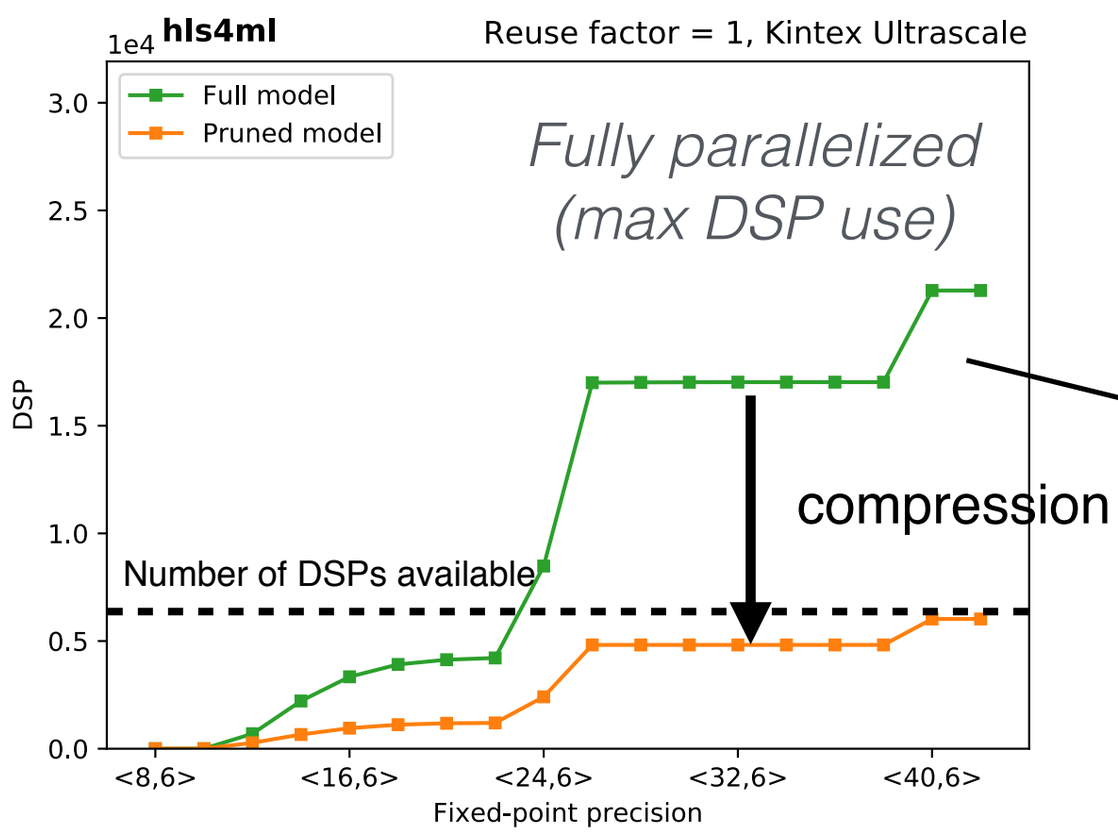
First examine resource usage coming from HLS estimate

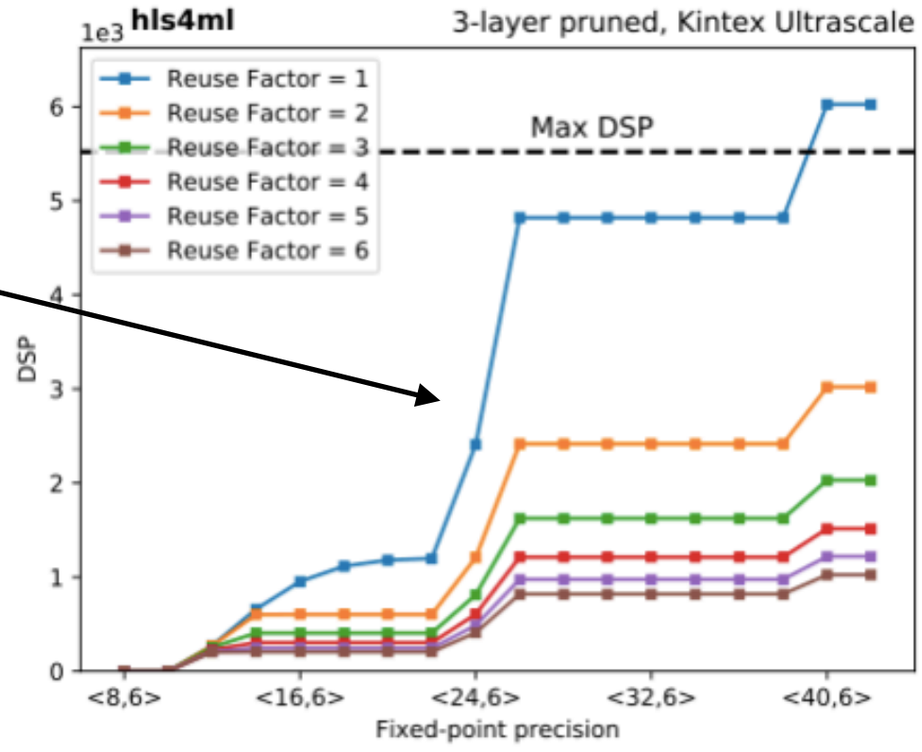
Then discuss the exact resources given by the final implementation

Resource usage: DSPs

- DSPs (used for multiplication) are often limiting resource
  - maximum use when fully parallelized
  - DSPs have a max size for input (e.g. 27x18 bits), so number of DSPs per multiplication changes with precision







Fully parallel
Each mult. used 1x

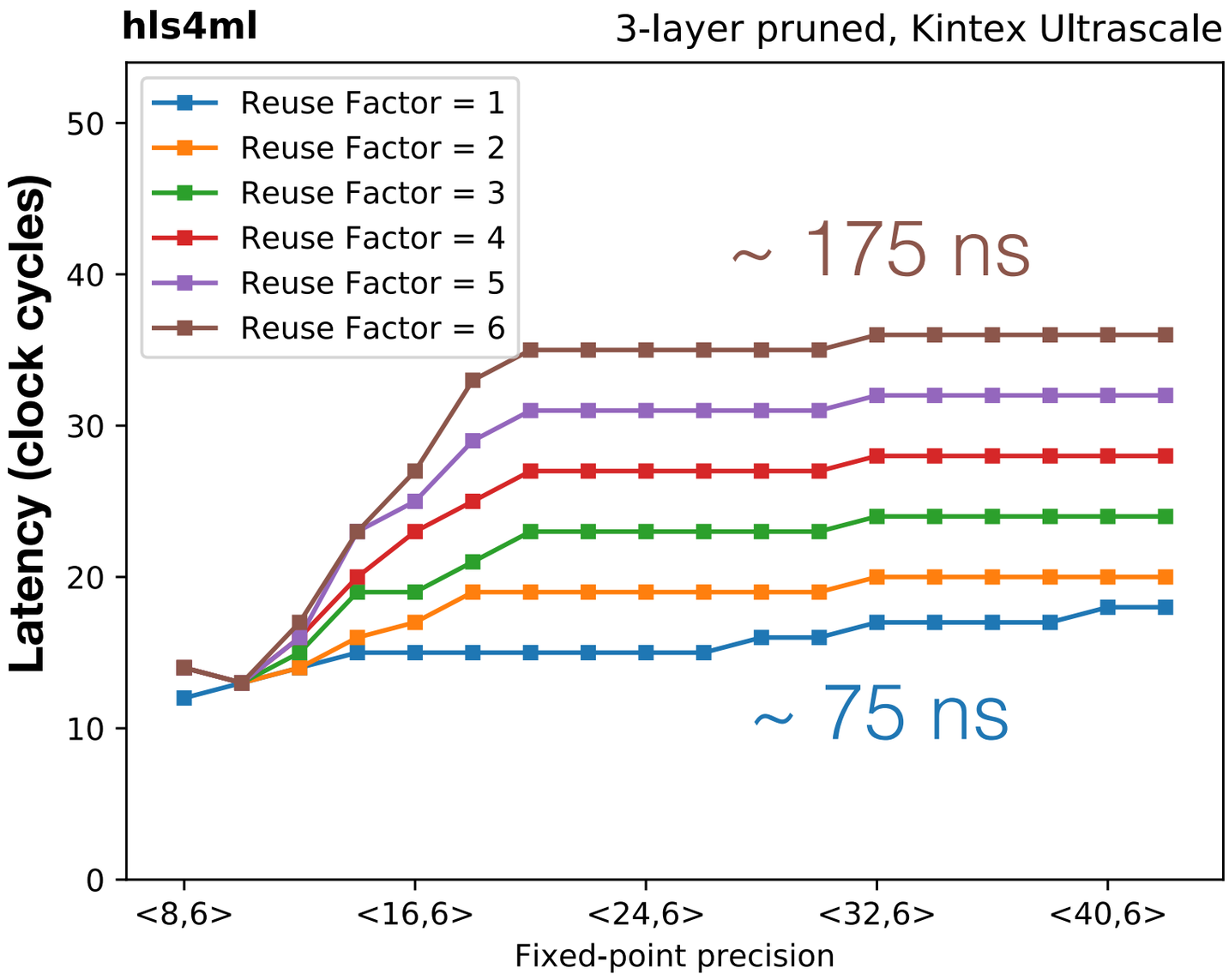
Each mult. used 2x

Each mult. used 3x

⋮

70% compression ~ 70% fewer DSPs

Parallelization: Timing



Longer latency

Each mult. used 6x

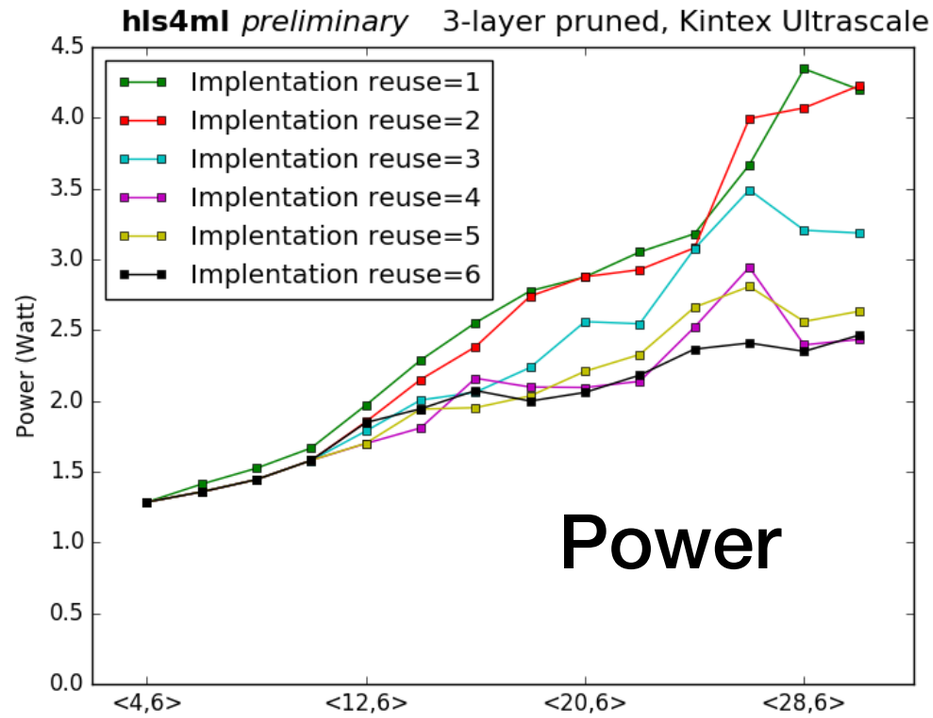
Each mult. used 3x

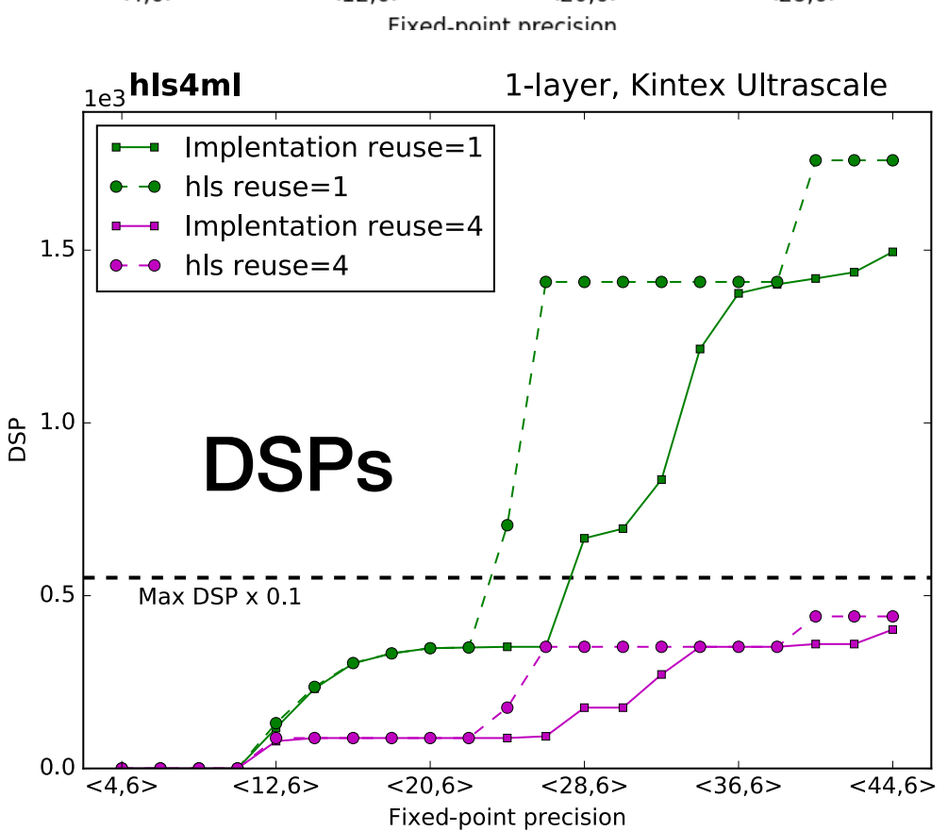
Fully parallel
Each mult. used 1x

More resources

Firmware implementation

- Final implementation gives actual resource usage and timing estimate
  - **how optimal is the HLS design?**
- Power usage increases with precision, it goes down for less throughput (higher reuse factor)
- Implement a 1-layer NN, simply routing all firmware block's inputs and outputs to FPGA available pins
- Timing target not always met (depends on the Vivado HLS version)
- HLS estimate on resource usage are conservative
  - DSPs usage agree well below DSP precision transition (27 bit), implementation does further optimization
  - FFs and LUTs overestimated by a factor 2-4

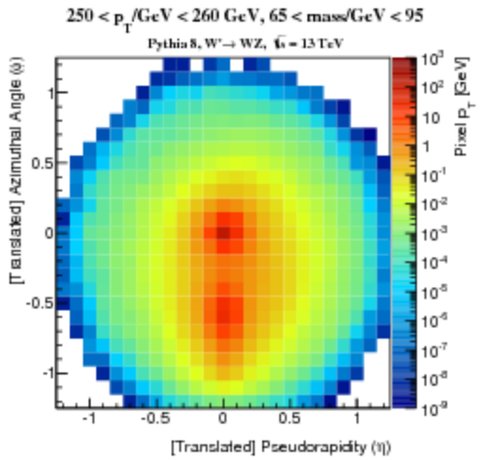




Status and outlook

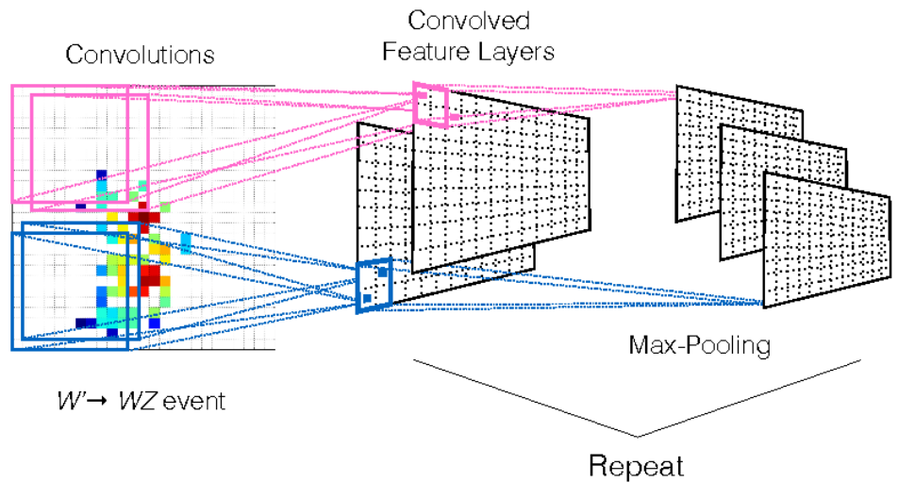
Other NN architectures

- **Convolutional Neural Networks**
  - active implementation of small Conv1D and Conv2D with hls4ml
  - resources reuse and compression supported
  - work is ongoing to ensure large scale networks

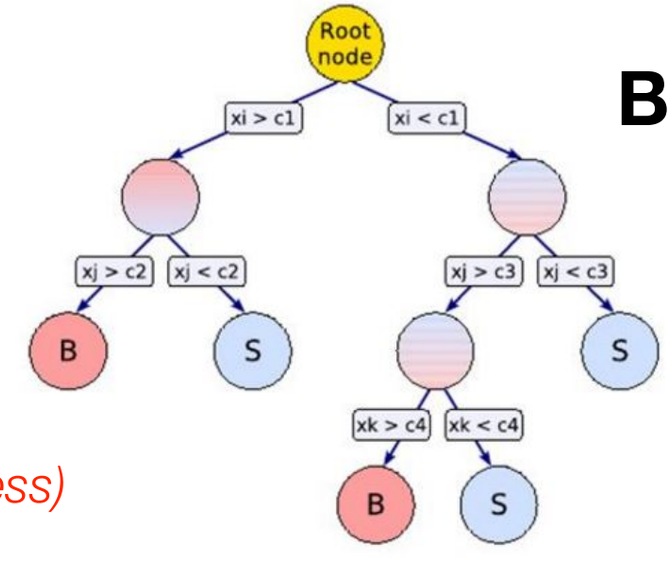


Use case of CNN @ LHC for jet image

JHEP 07 (2016) 069



- **Boosted Decision Tree** *(work in progress)*
  - each node in decision tree compares element against a threshold → boolean logic, thresholds in LUT, suitable for FPGA
  - each tree is independent → high parallelization



BDT

- **Binary/Ternary Neural Networks** *(work in progress)*
  - weights are binary/ternary in the inference = ±1,0
  - ternary NN does not need pruning/compression
  - similar performance and latency with 0% DSPs used

- **Recursive NN and LSTM** under testing *(work in progress)*

goes to the cloud

For long latency tasks
→ more time means

More resource reuse (x1000)
Bigger networks

ex: HLT or offline reconstruction @ LHC

Offload a CPU from the computational heavy parts to a FPGA “accelerator”

Increased computational speed of 10x-100x
Reduced system size of 10x
Reduced power consumption of 10x-100x

- **Amazon Web Service** provides cloud based system consisting of CPU/GPU/FPGAs
  - AWS F1 instances include up to 8 Xilinx Virtex Ultrascale+
- Used hls4ml through SDAccel to create a firmware implementation of 1D CNN
  - 5 layers with 10 four-channel inputs, latency of 116 ns
  - successfully run on an AWS F1 instance

Summary

We introduced a new software/firmware package **hls4ml**

Automated translation of everyday machine learning inference into firmware in ~ minutes

Tunable configuration for optimization of your use case

First application is single FPGA, <1 us latency for L1 trigger or DAQ

Explore also applications for acceleration with CPU-FPGA co-processors for long latency trigger tasks

For more info

- https://hls-fpga-machine-learning.github.io/hls4ml/
- https://arxiv.org/abs/1804.06913

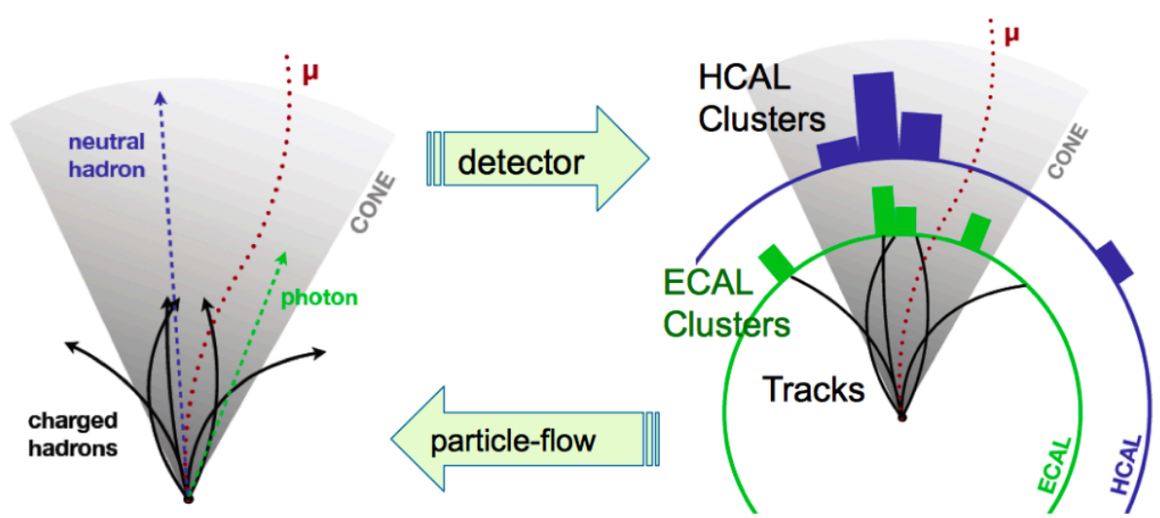
Backup

New trigger algorithms

The detector and its trigger system

Output: trigger primitives
(calo energy clusters, muons, tracks)

Particle-flow algorithm @ L1



Output: particle candidates

Trigger decision

CMS-TDR-017, JINST 12 (2017) P10003

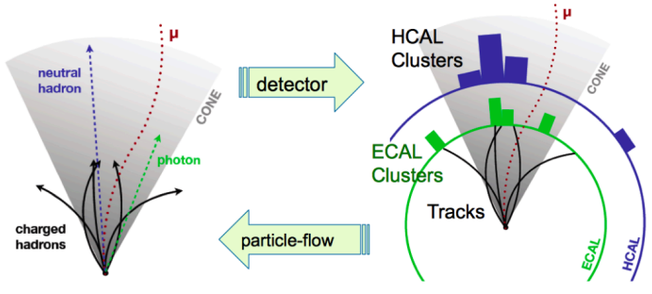
New trigger algorithms

The detector and its trigger system

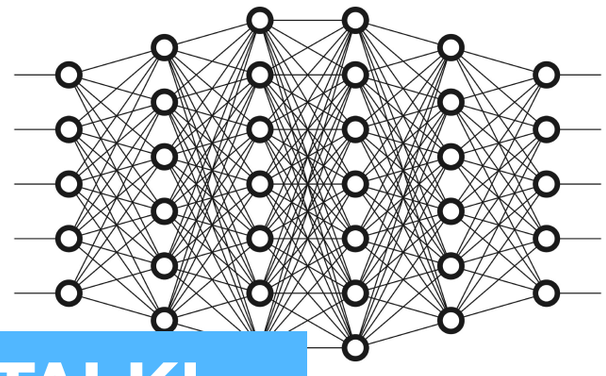
Output: trigger primitives
(calo energy clusters, muons, tracks)

Particle-flow algorithm @ L1

Output: particle candidates



Machine learning



THIS TALK!

Trigger decision

[CMS-TDR-017](), [JINST 12 (2017) P10003]()

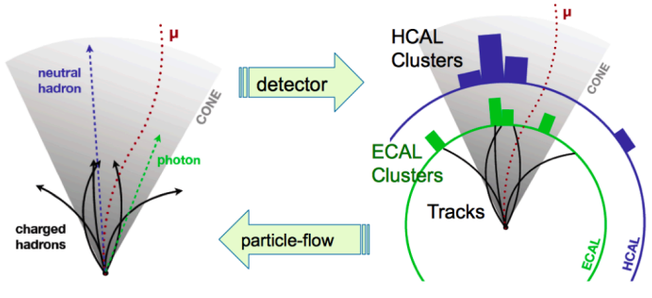
New trigger algorithms

The detector and its trigger system

Output: trigger primitives
(calo energy clusters, muons, tracks)

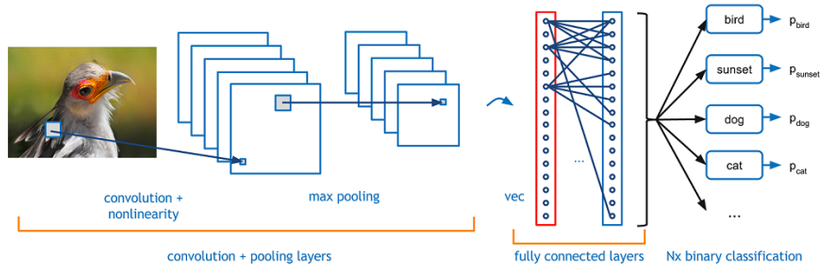
Particle-flow algorithm @ L1

Output: particle candidates



[CMS-TDR-017](), [JINST 12 (2017) P10003]()

Machine learning



THIS TALK!

Trigger decision

Muon reconstruction @ L1

First implementation of a ML algo for CMS L1 trigger on FPGAs [*]

A BDT is used to improve the momentum of muons in the forward region of the detector

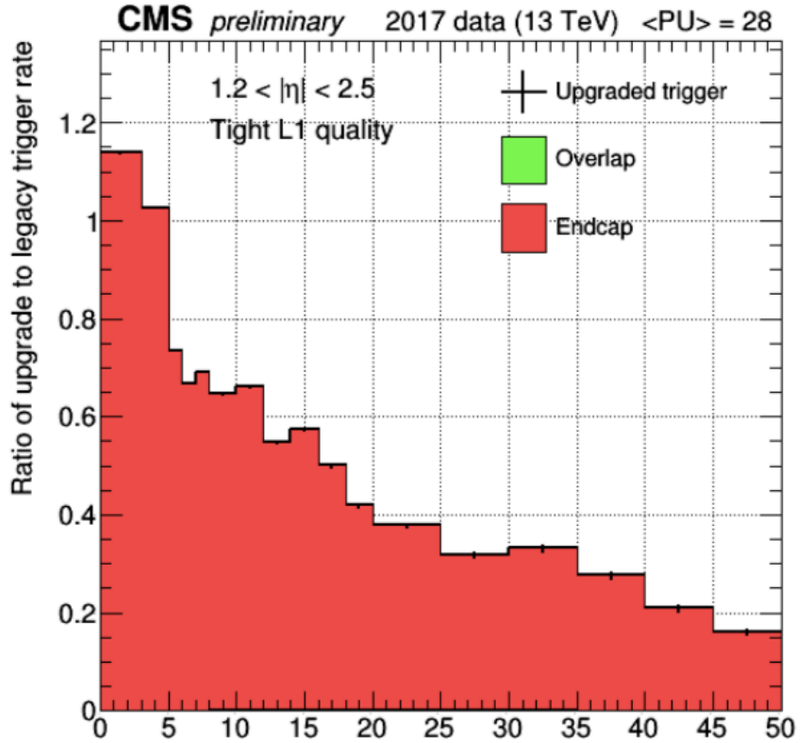
based on curvature angles in the magnetic fields ($\Delta\phi$,$\Delta\theta$) and few other variables

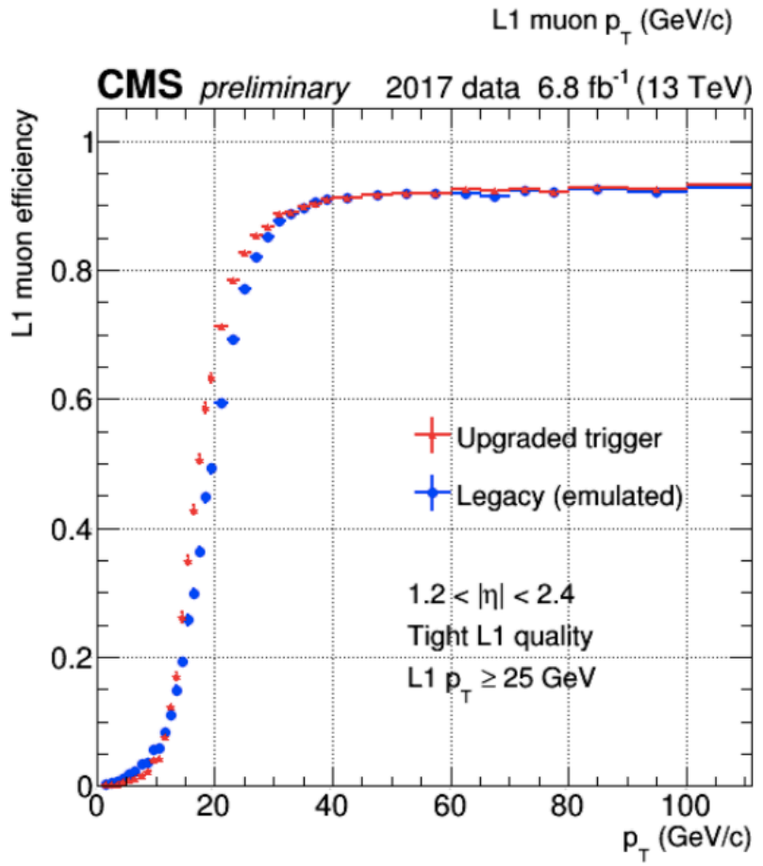
Prediction of BDT for every possible input stored into pre-computed 1.2 GB Look-Up Table (LUT) on FPGA

Achieved reduction of background rates by factor 3 w/o efficiency losses

Usage of LUTs does not scale nicely with ML algo complexity → quickly use all resources

Can we improve this approach?





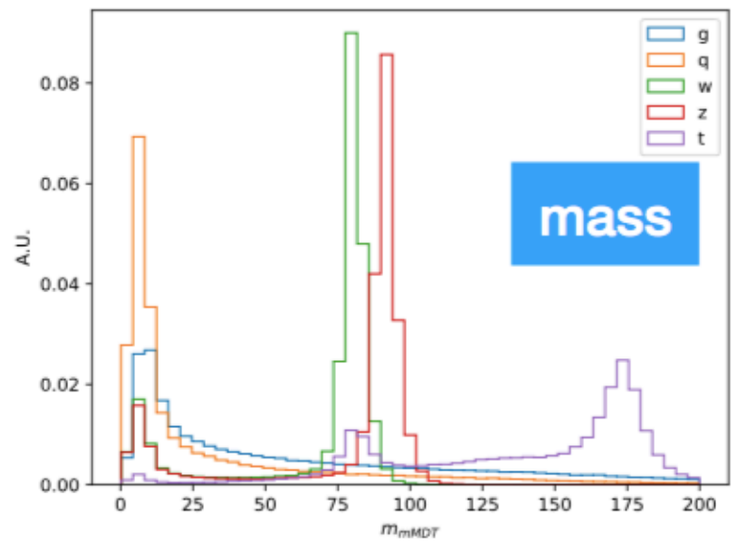
[*] http://cds.cern.ch/record/2290188/files/CR2017_357.pdf?version=1

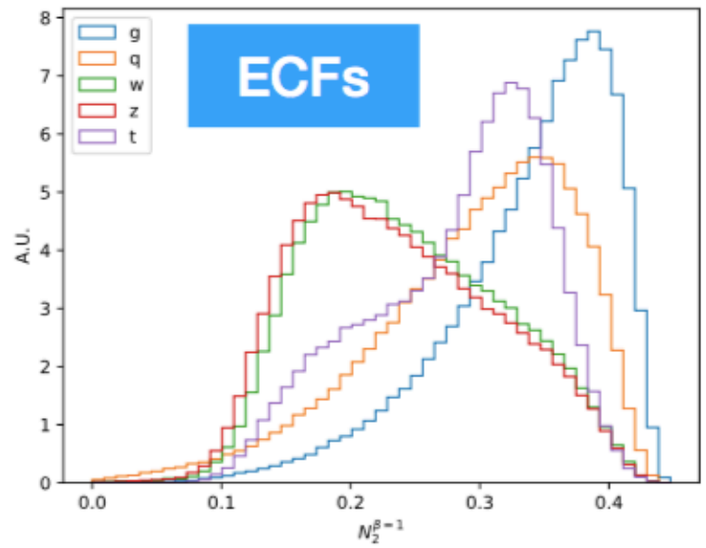
Jet substructure features

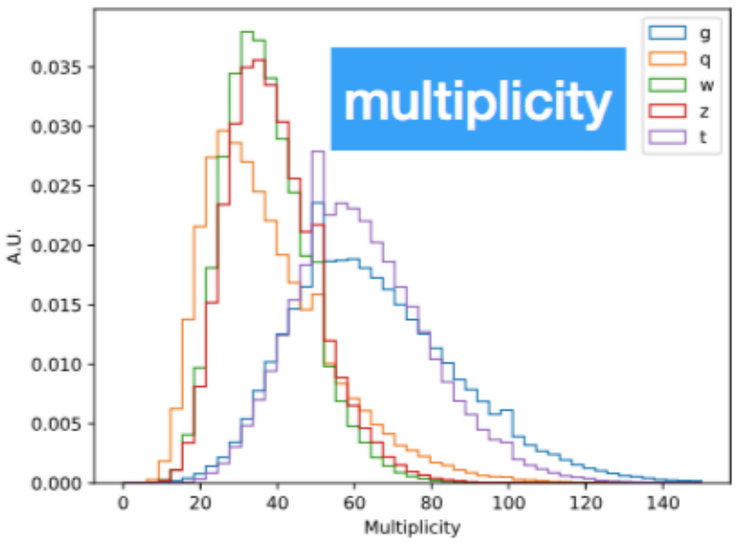
Jet substructure observables provide large discrimination power between these types of jets

mass, multipliticity, energy correlation functions, …
(computed with FastJet [*])

[*] E. Coleman et al. JINST13(2018) T01003,
M. Cacciari et al, Eur. Phys. J.C72(2012)1896







These are expert-level features

Not necessarily realistic for L1 trigger
"Raw" particle candidates more suitable *(to be studied next)*
But lessons here are generic

One more case: H→bb discrimination vs W/Z→qq requires more "raw" inputs for b-tagging information

The hls4ml package

Translation →

```
python keras-to-hls.py -c keras-config.yml
```

Inputs →

Keras

HDF5

```
KerasJson: example-keras-model-files/KERAS_1layer.json
KerasH5:   example-keras-model-files/KERAS_1layer_weights.h5
OutputDir: my-hls-test
ProjectName: myproject
XilinxPart: xc7vx690tffg1927-2
ClockPeriod: 5

IOType: io_parallel # options: io_serial/io_parallel
ReuseFactor: 1
DefaultPrecision: ap_fixed<18,8>
```

Config

- `IOType`: parallelize or serialize
- `ReuseFactor`: how much to parallelize
- `DefaultPrecision`: inputs, weights, biases

```
my-hls-test/:
build_prj.tcl
firmware
myproject_test.cpp
```

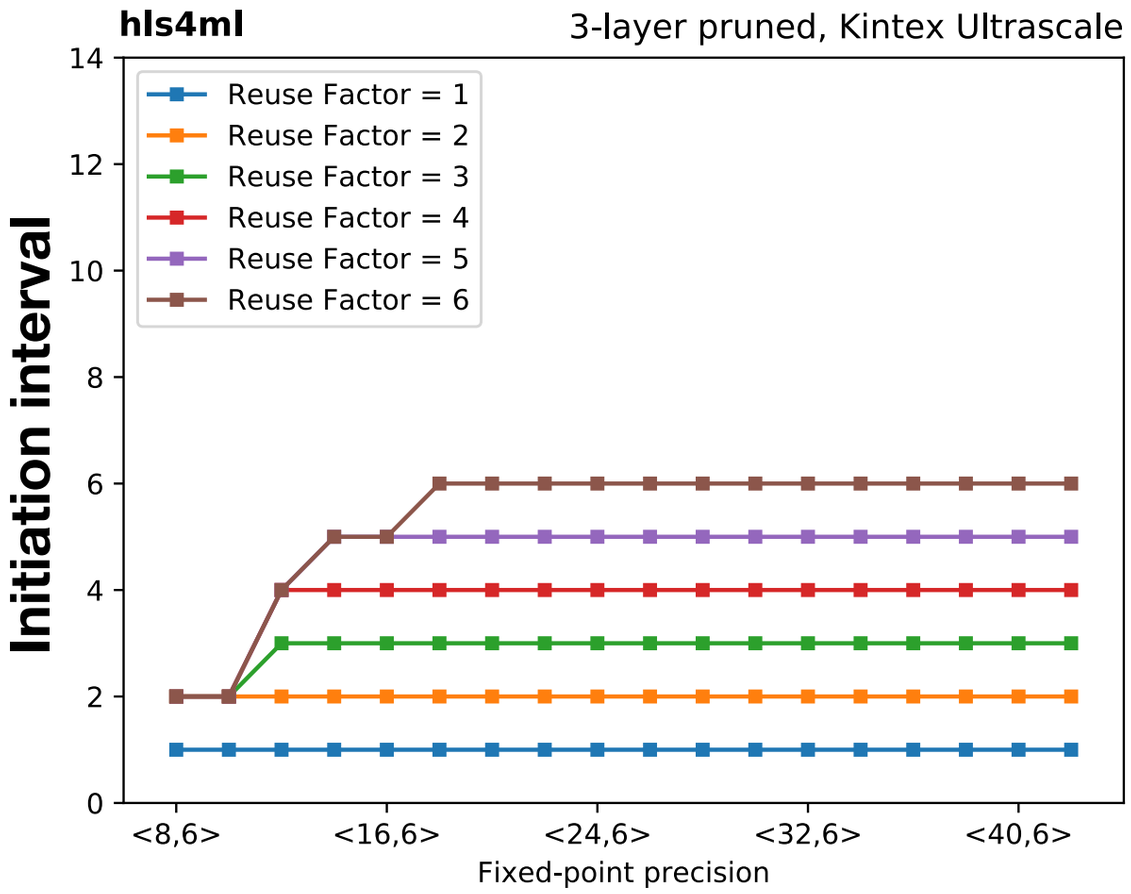
The hls4ml package

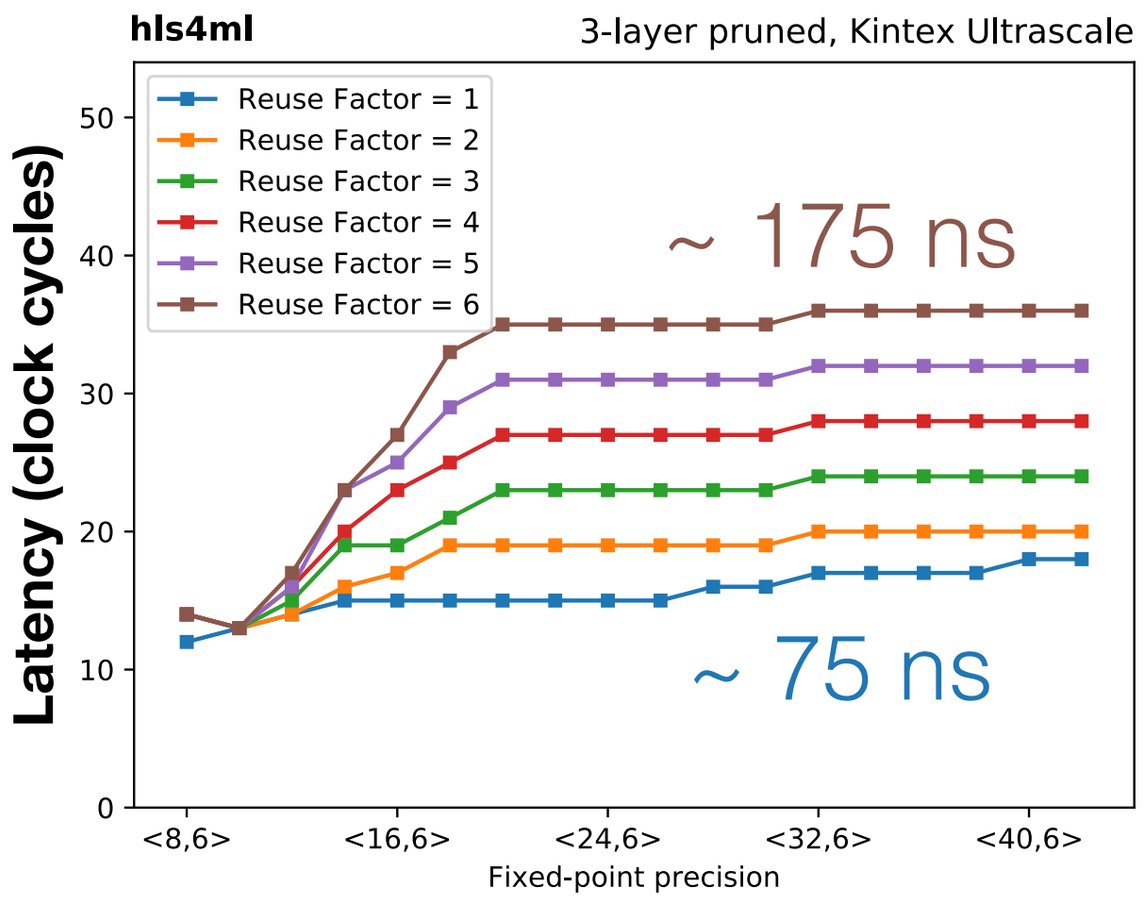
Build HLS project ⟶ `vivado_hls -f build_prj.tcl`


```
##################
#    HLS4ML      #
##################
open_project -reset myproject_prj
set_top myproject
add_files firmware/myproject.cpp -cflags "-I[file normalize ../../nnet_utils]"
add_files -tb myproject_test.cpp -cflags "-I[file normalize ../../nnet_utils]"
add_files -tb firmware/weights
open_solution -reset "solution1"
set_part {xcku115-flvf1924-2-i}
create_clock -period 5 -name default
csim_design
csynth_design
cosim_design -trace_level all
export_design -format ip_catalog
exit
```

Produce a firmware block in ~ minutes!

Parallelization: Timing





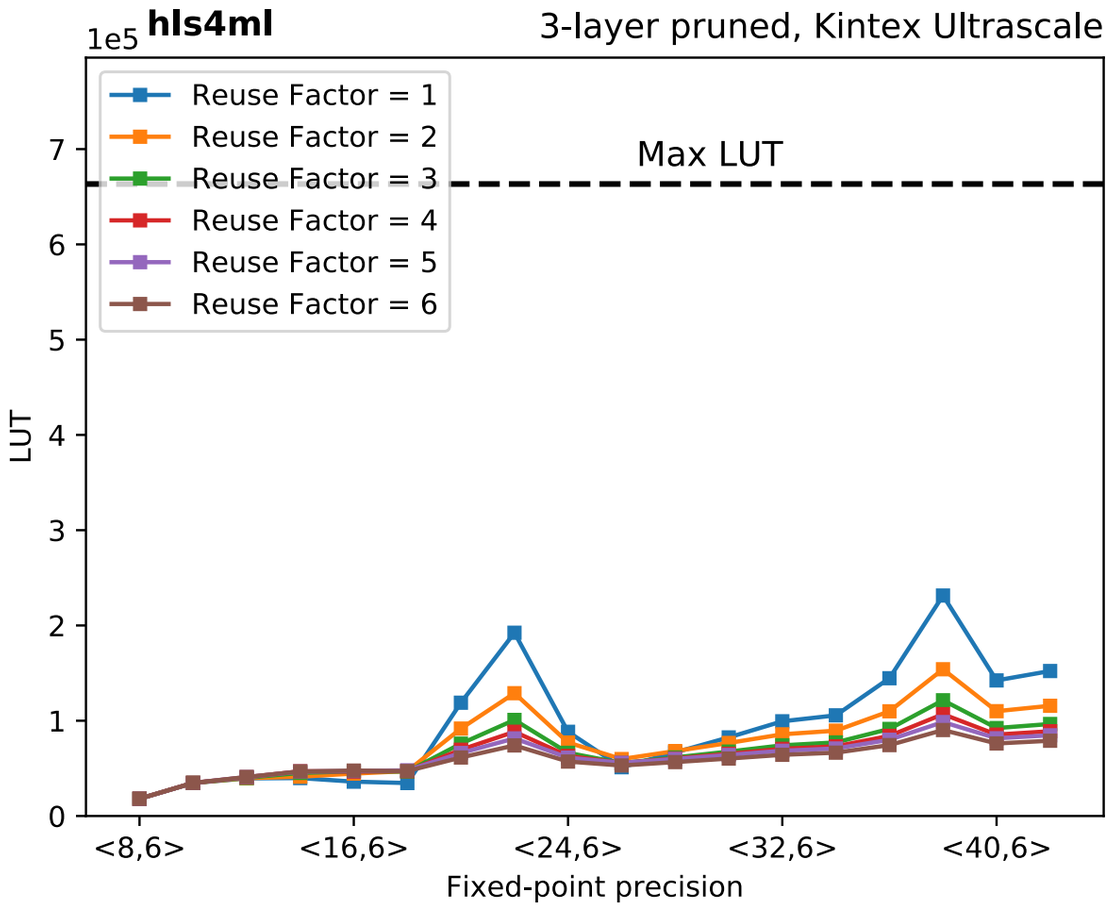
- **Initiation interval:** number of clocks before accepting new inference inputs
  - scales with reuse factor
  - for very low data precisions multiplications implemented through FFs and LUTs

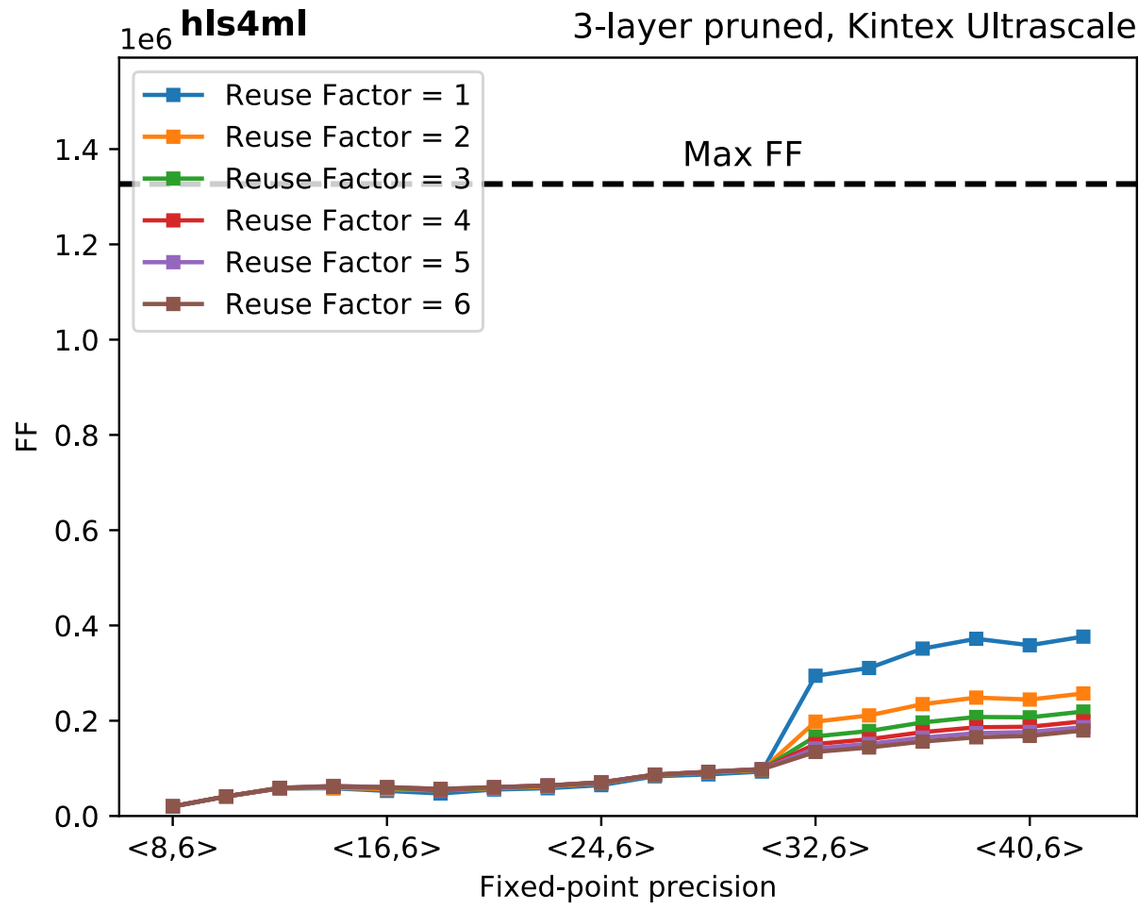
- Additional **latency** introduced by reusing the multiplier

Latency of layer m

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

Other resources





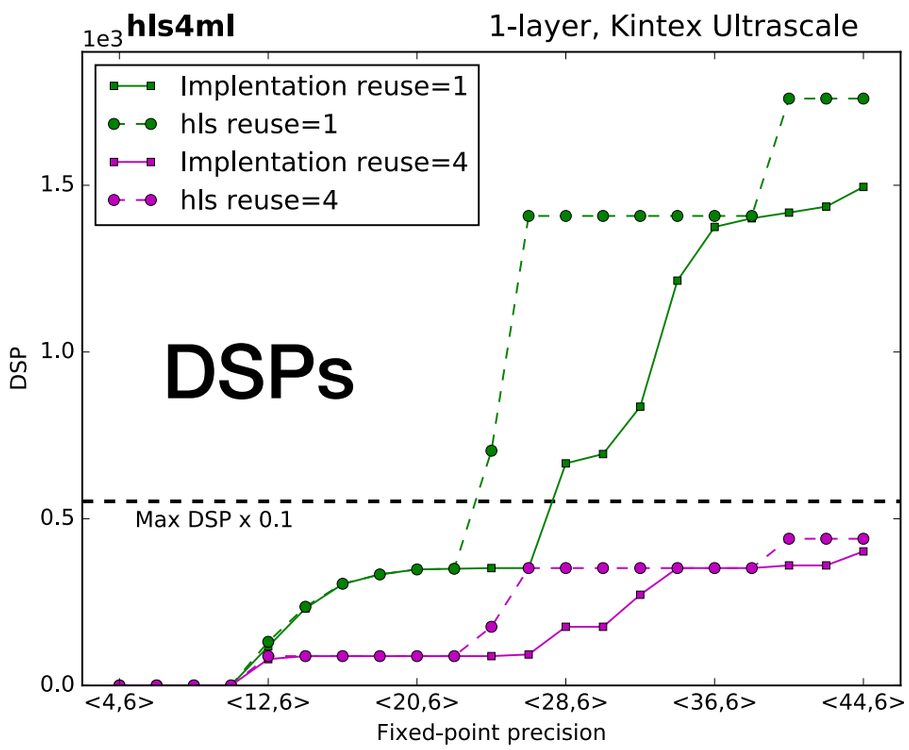
Fairly linear increase with precision

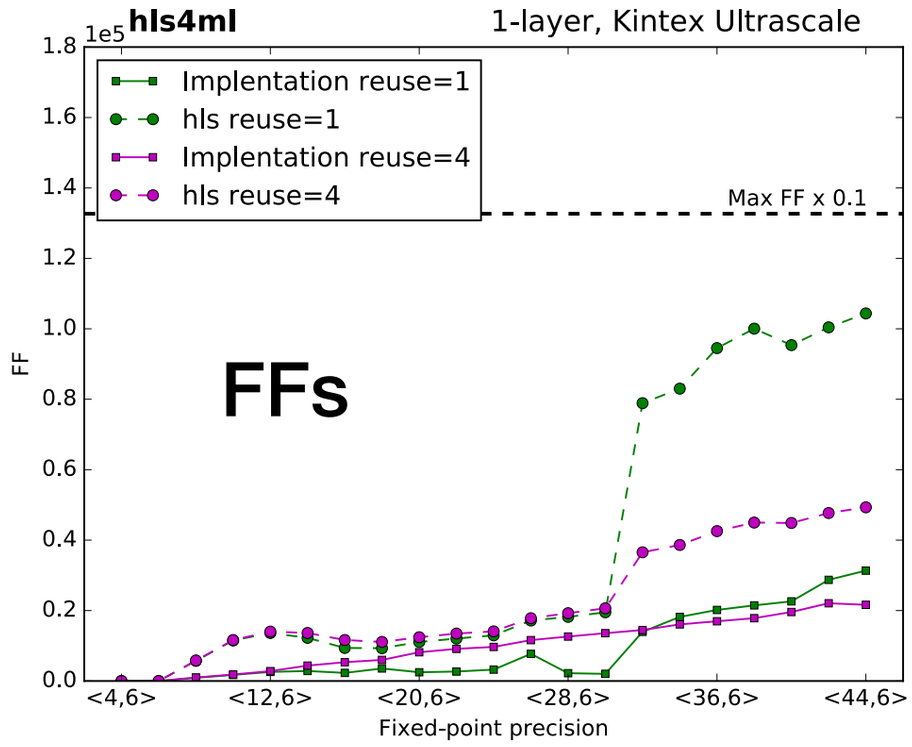
Small percentage of total available

Spikes present at steep transitions in LUTs usage as artifacts of HLS synthesis

Not observed in implementation
Found also dependence on Vivado HLS version

Firmware implementation





- HLS synthesis estimate on resource usage are usually conservative
  - DSPs usage agree well below DSP precision transition (27 bit), deviations above due to further Vivado optimizations
  - FFs and LUTs overestimated by a factor 2-4