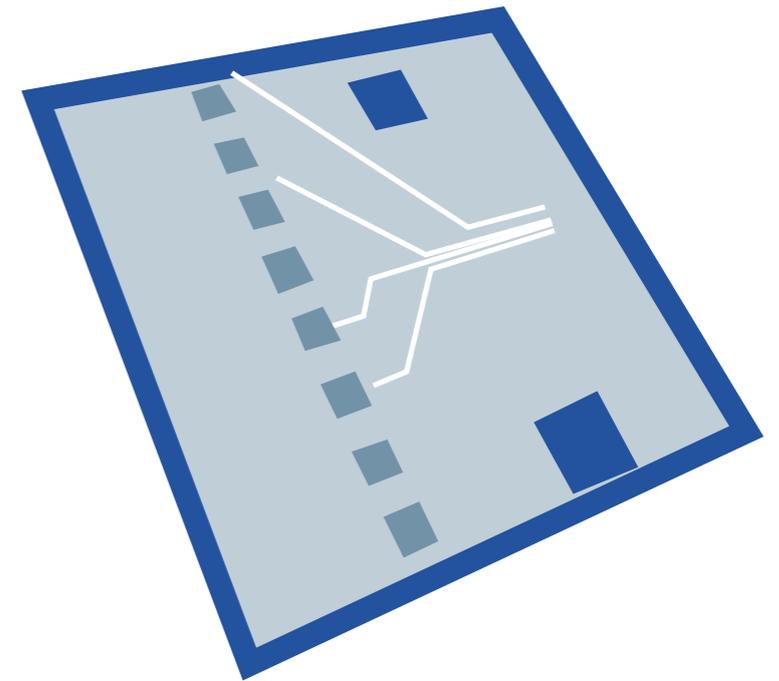
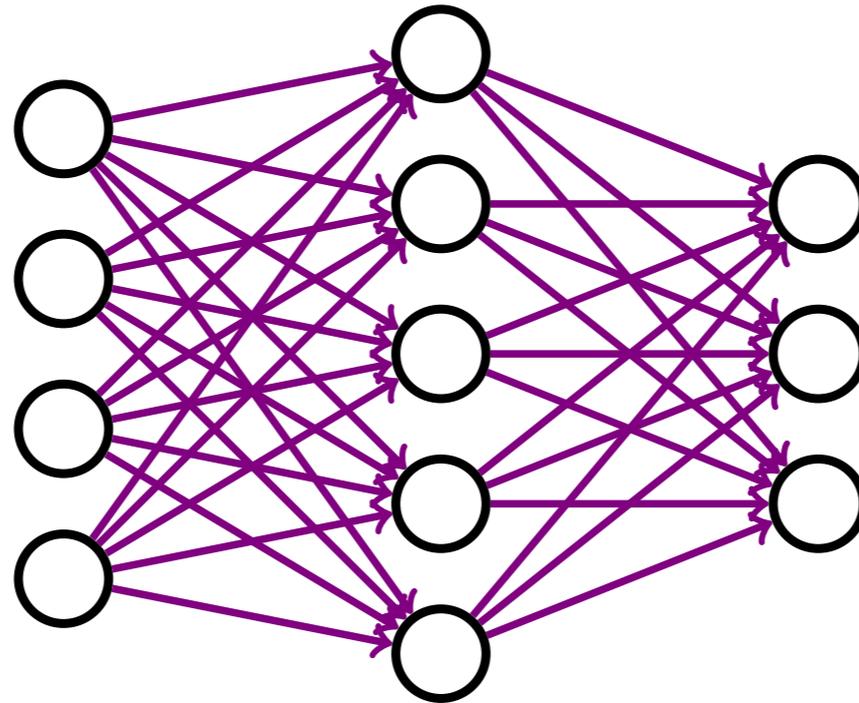
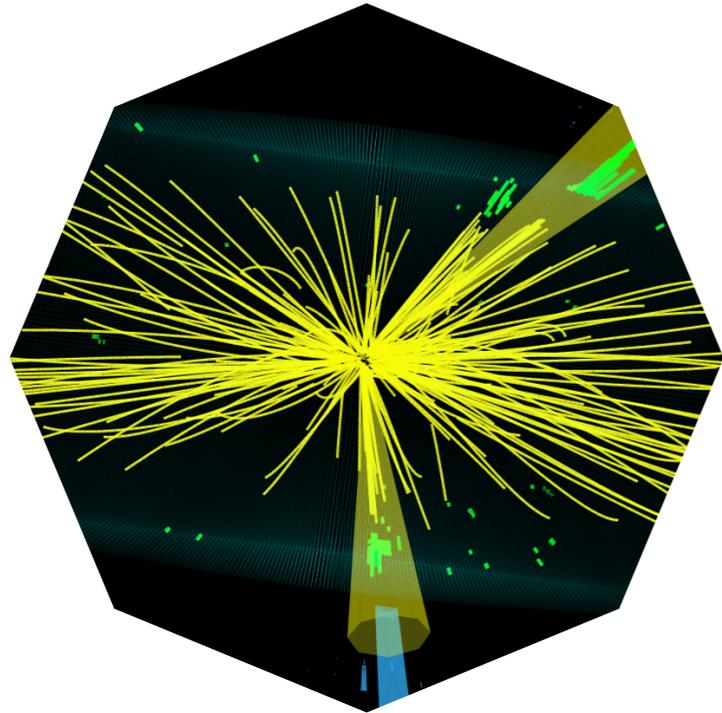


Fast Inference of Deep Neural Networks in FPGAs for Particle Physics [\(arXiv:1804.06913\)](https://arxiv.org/abs/1804.06913)



Jennifer Ngadiuba, Maurizio Pierini [CERN]

Javier Duarte, Sergo Jindariani, Ben Kreis, Ryan Rivera, Nhan Tran [Fermilab]

Edward Kreinar [Hawkeye 360]

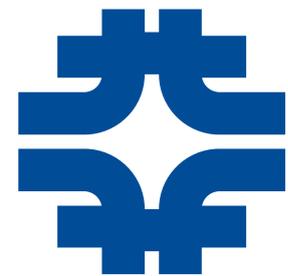
Song Han, Phil Harris [MIT]

Zhenbin Wu [University of Illinois at Chicago]

Topical Workshop for Electronics in Particle Physics (TWEPP) 2018

Antwerp, Belgium

9/19/2018



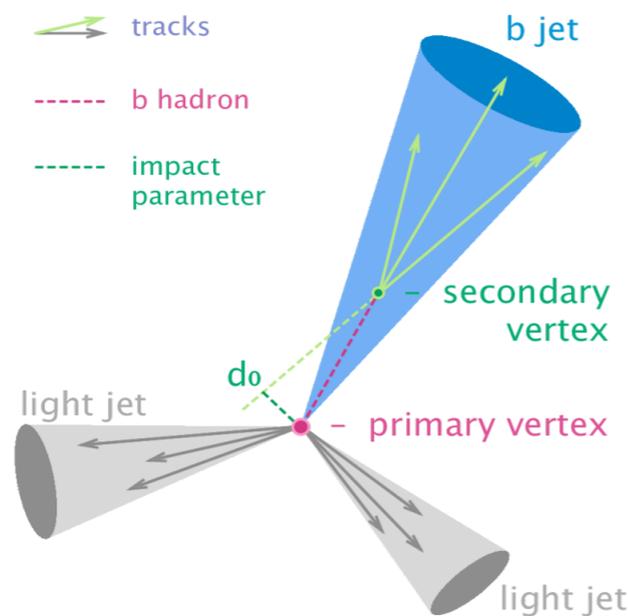
Outline

- Machine Learning in HEP
- hls4ml: HLS for Machine Learning
- Case Study and Design Exploration
- Summary and Outlook
 - New Developments and Features
 - Acceleration



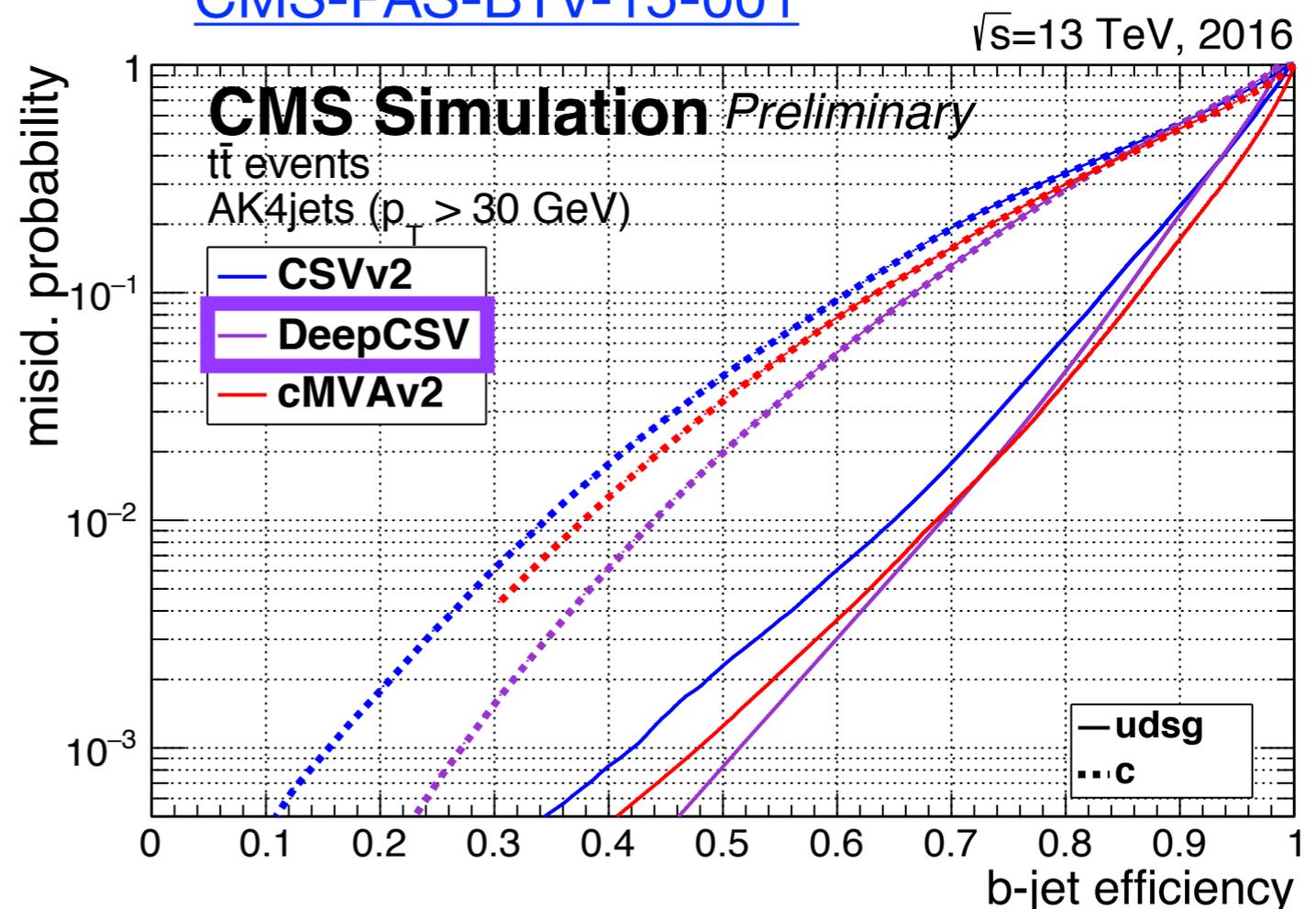
Machine Learning in HEP

- Learning optimized nonlinear functions of many inputs for performing difficult tasks from (real or simulated) data
- Many successes in HEP: **identification of b-quark jets**, Higgs candidates, particle energy regression, **analysis selection**, ...



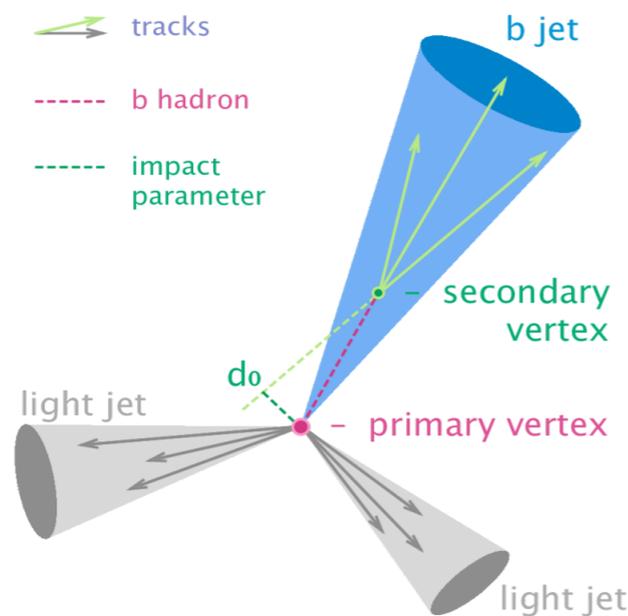
- **Neural network** based on high-level features

[CMS-PAS-BTV-15-001](#)



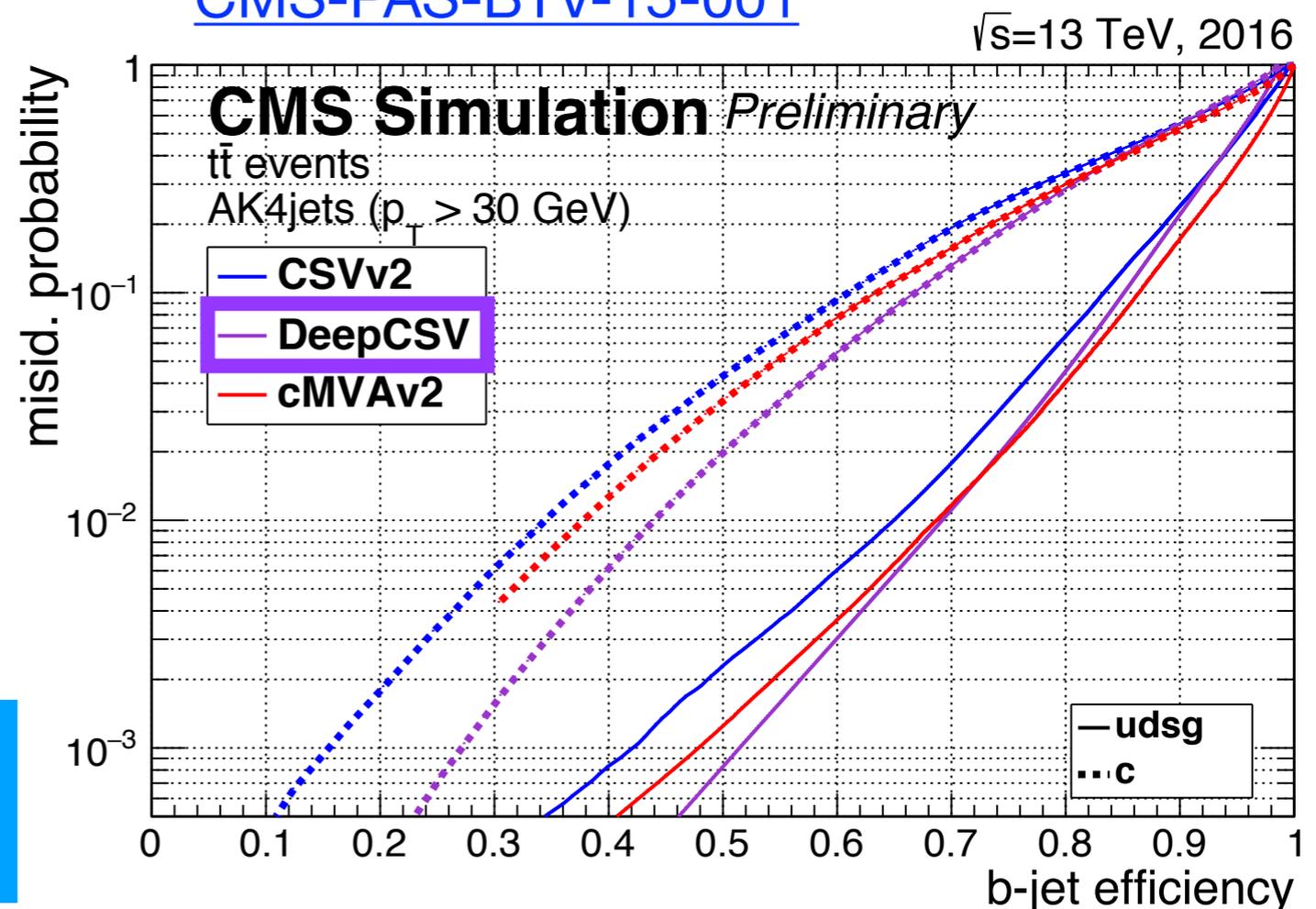
Machine Learning in HEP

- Learning optimized nonlinear functions of many inputs for performing difficult tasks from (real or simulated) data
- Many successes in HEP: **identification of b-quark jets**, Higgs candidates, particle energy regression, **analysis selection**, ...

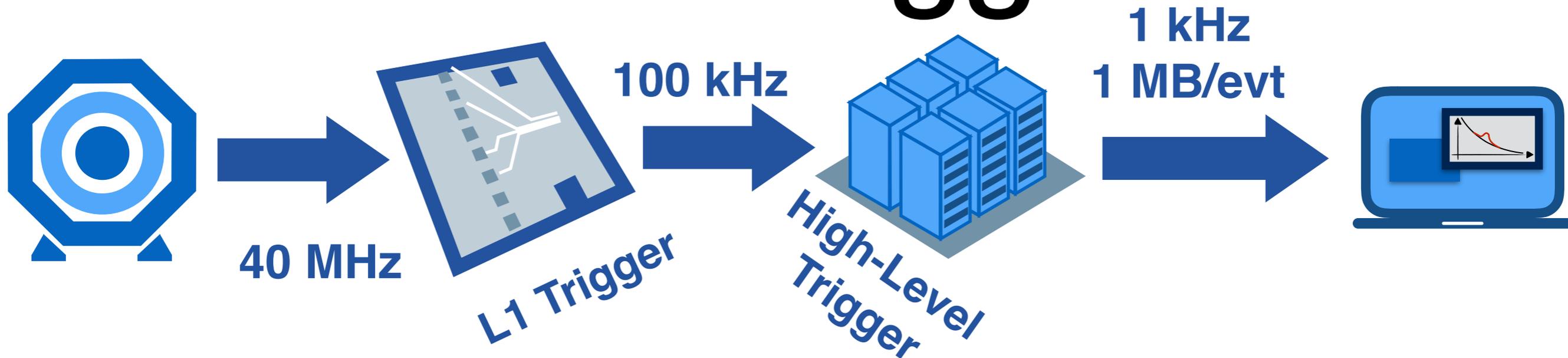


Typically applied offline,
not online (in hardware trigger)

[CMS-PAS-BTV-15-001](#)



CMS Trigger

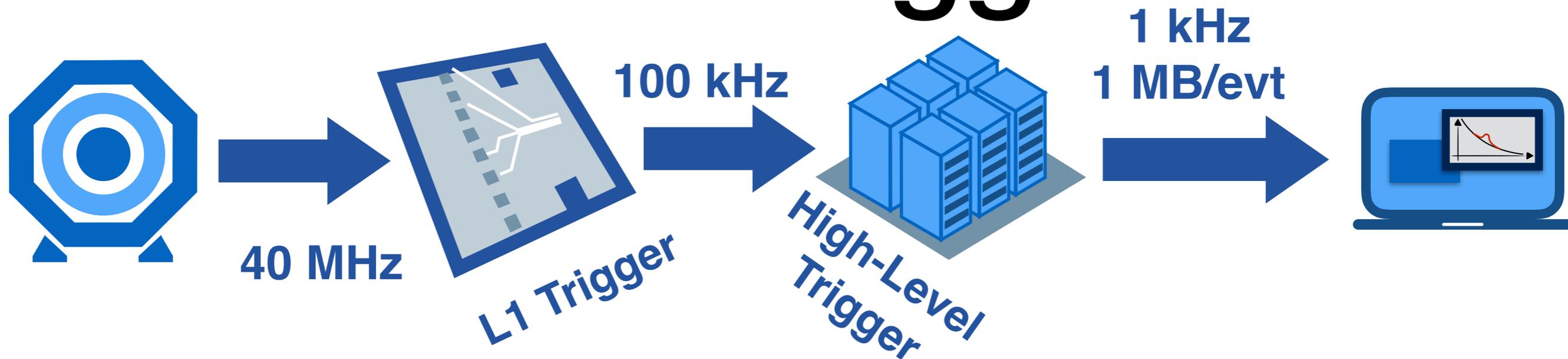


- Level-1 Trigger (hardware)
- 99.75% rejected
- decision in $\sim 4 \mu\text{s}$

- High-Level Trigger (software)
- 99% rejected
- decision in $\sim 100\text{s ms}$



CMS Trigger



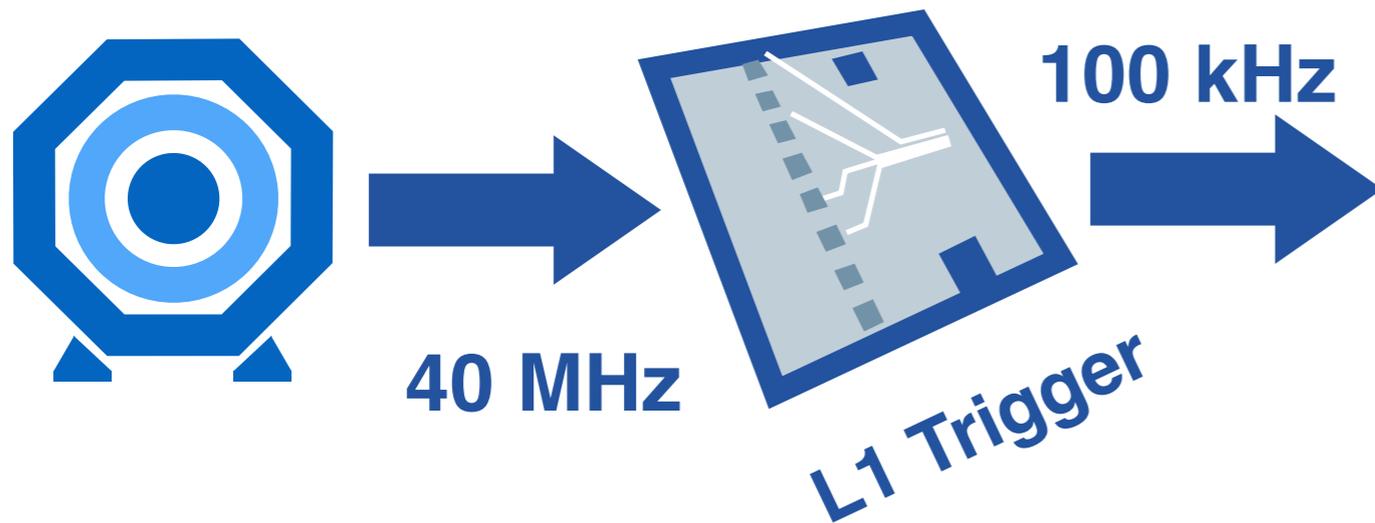
- Level-1 Trigger (hardware)
 - 99.75% rejected
 - decision in $\sim 4 \mu\text{s}$
 - High-Level Trigger (software)
 - 99% rejected
 - decision in $\sim 100\text{s ms}$
-
- After trigger, 99.99975% of events are gone forever



HEP Latency Landscape



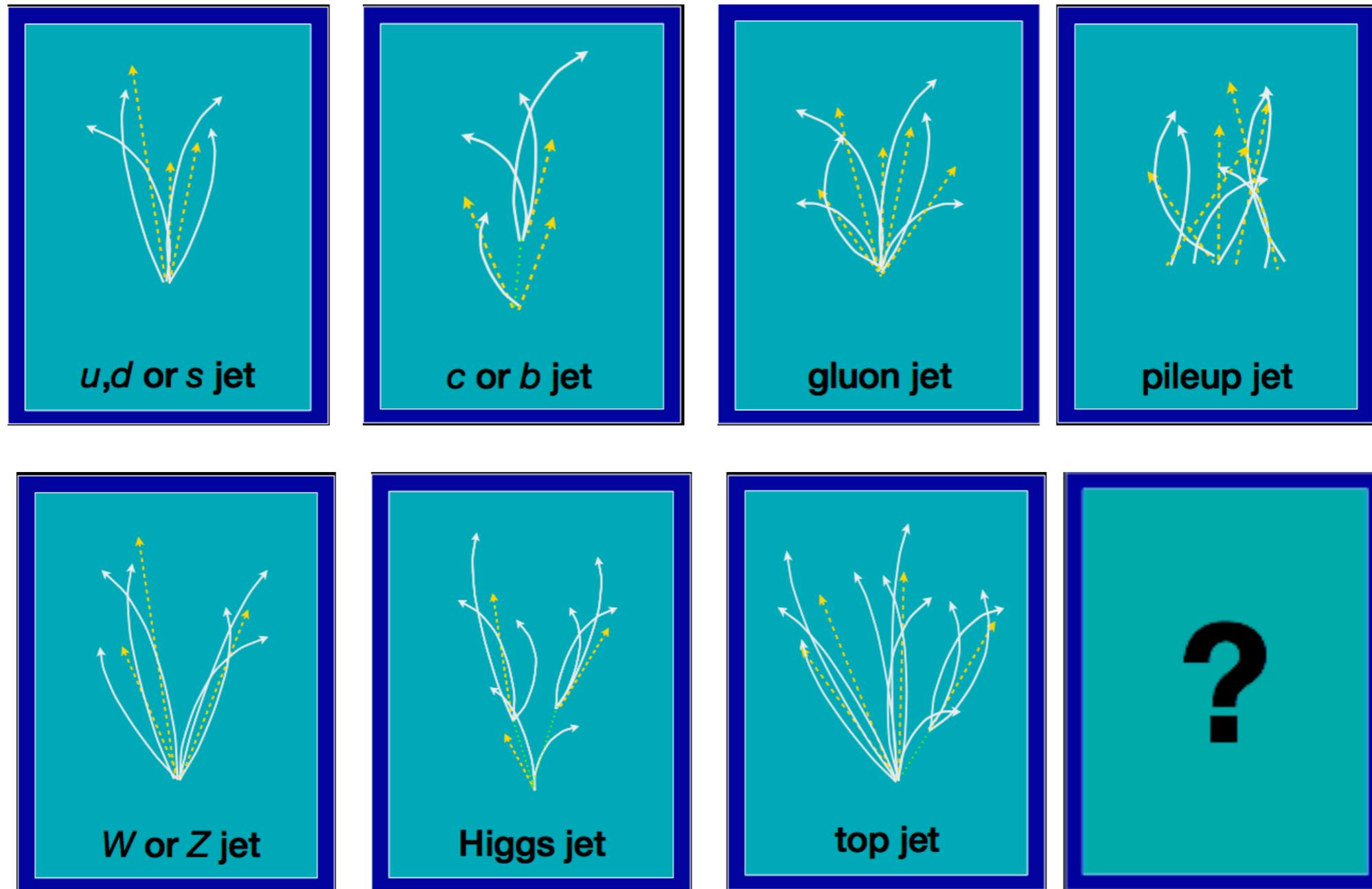
hls4ml



- hls4ml: neural network translation library for HLS
 - Support for common ML workflows and architectures
 - Tunable configuration for different use cases
- Focus on **L1 trigger** as **1st application**
 - What can we do in $< \mu\text{s}$ on one FPGA?



Case Study: Jet Substructure



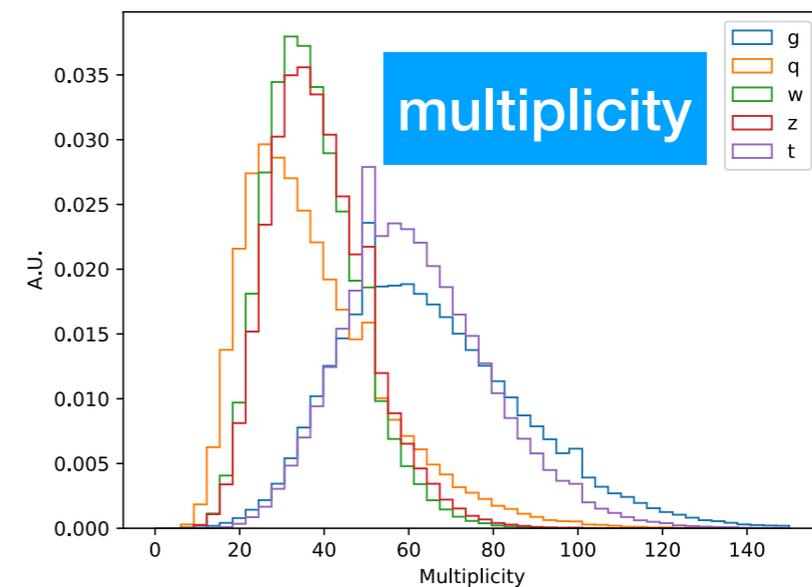
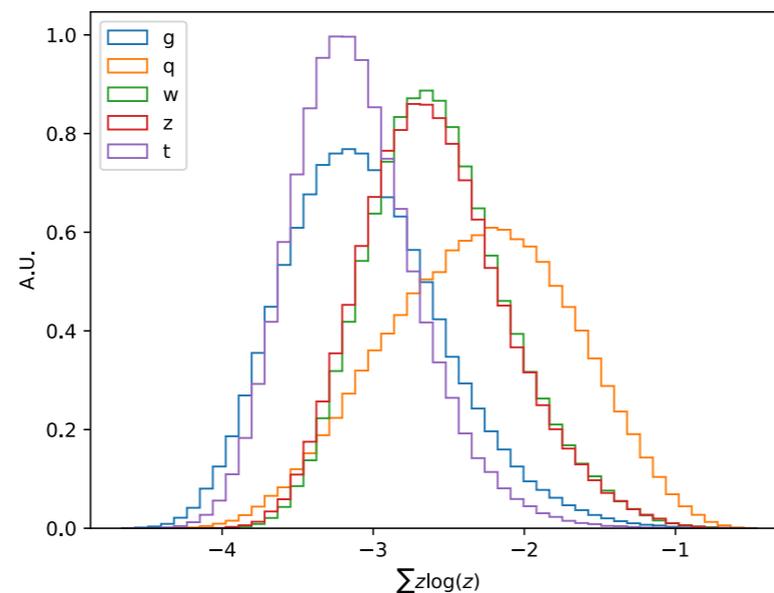
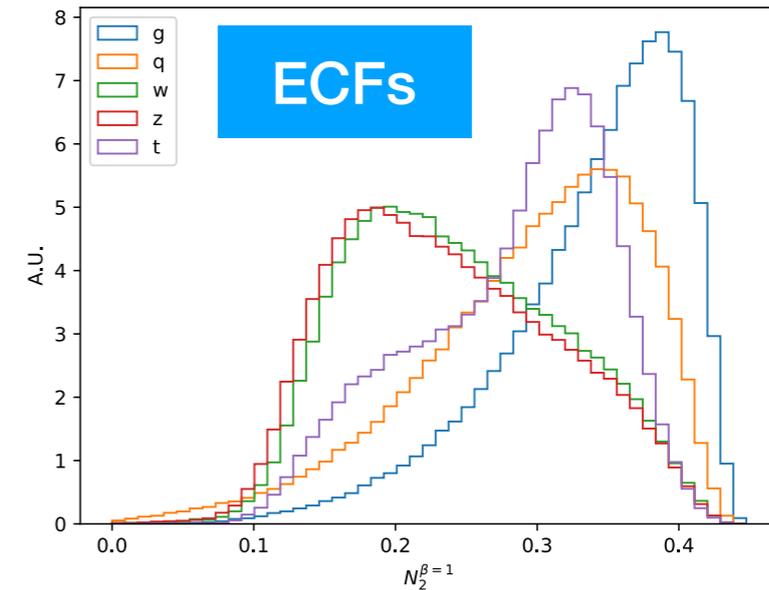
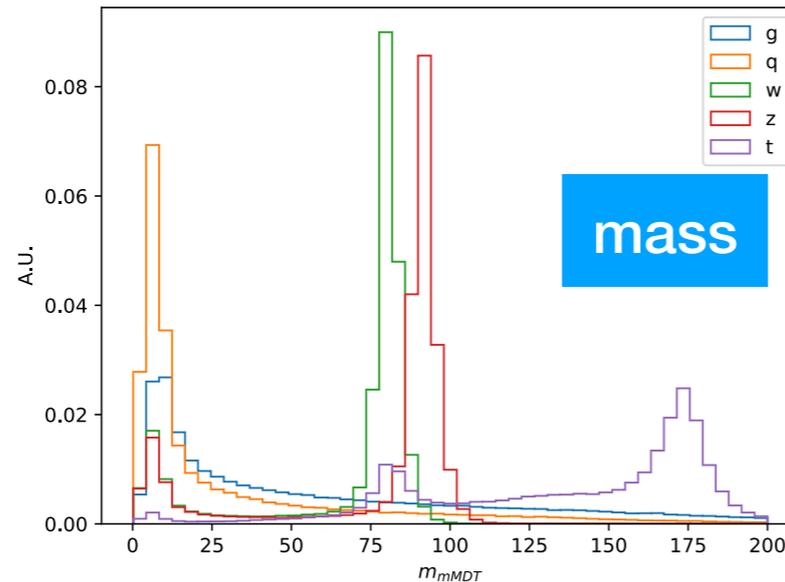
- Illustrative example: not necessarily the most realistic for L1 *today*, but lessons are generic



Jet Substructure Inputs

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

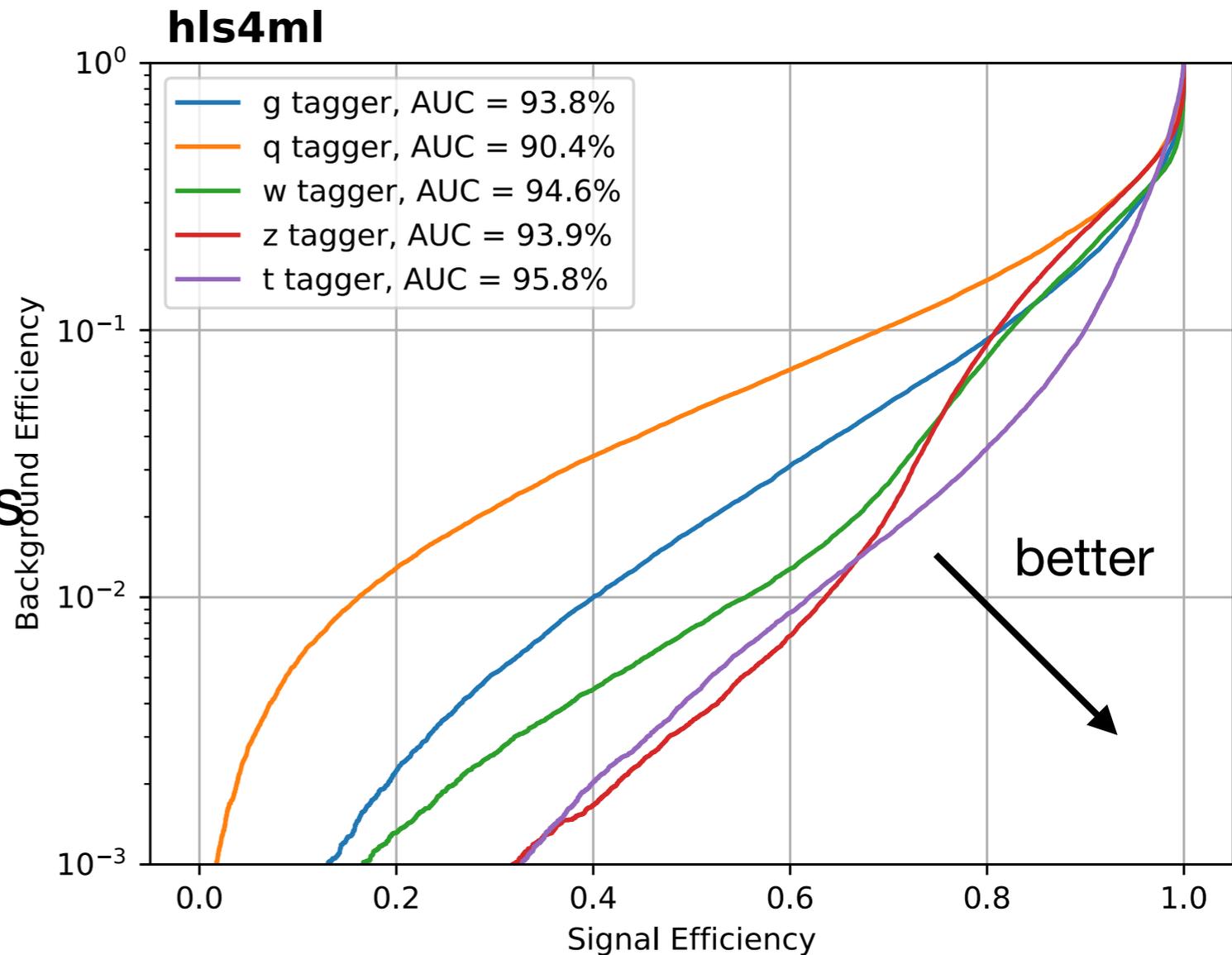
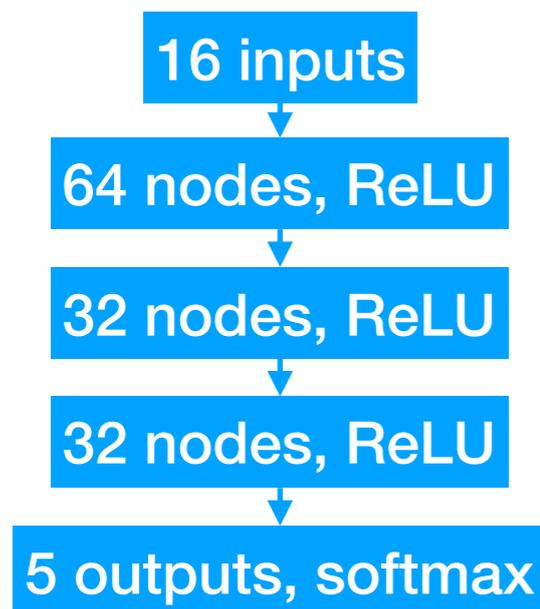


- **Groomed mass** separates top, W/Z, and quark/gluon
- top/gluon have greater **multiplicity** than W/Z/quark
- **ECF $N_2^{\beta=1}$** separates 2 and 3-prong jets (W/Z/top) from 1-prong jets (quark/gluon)



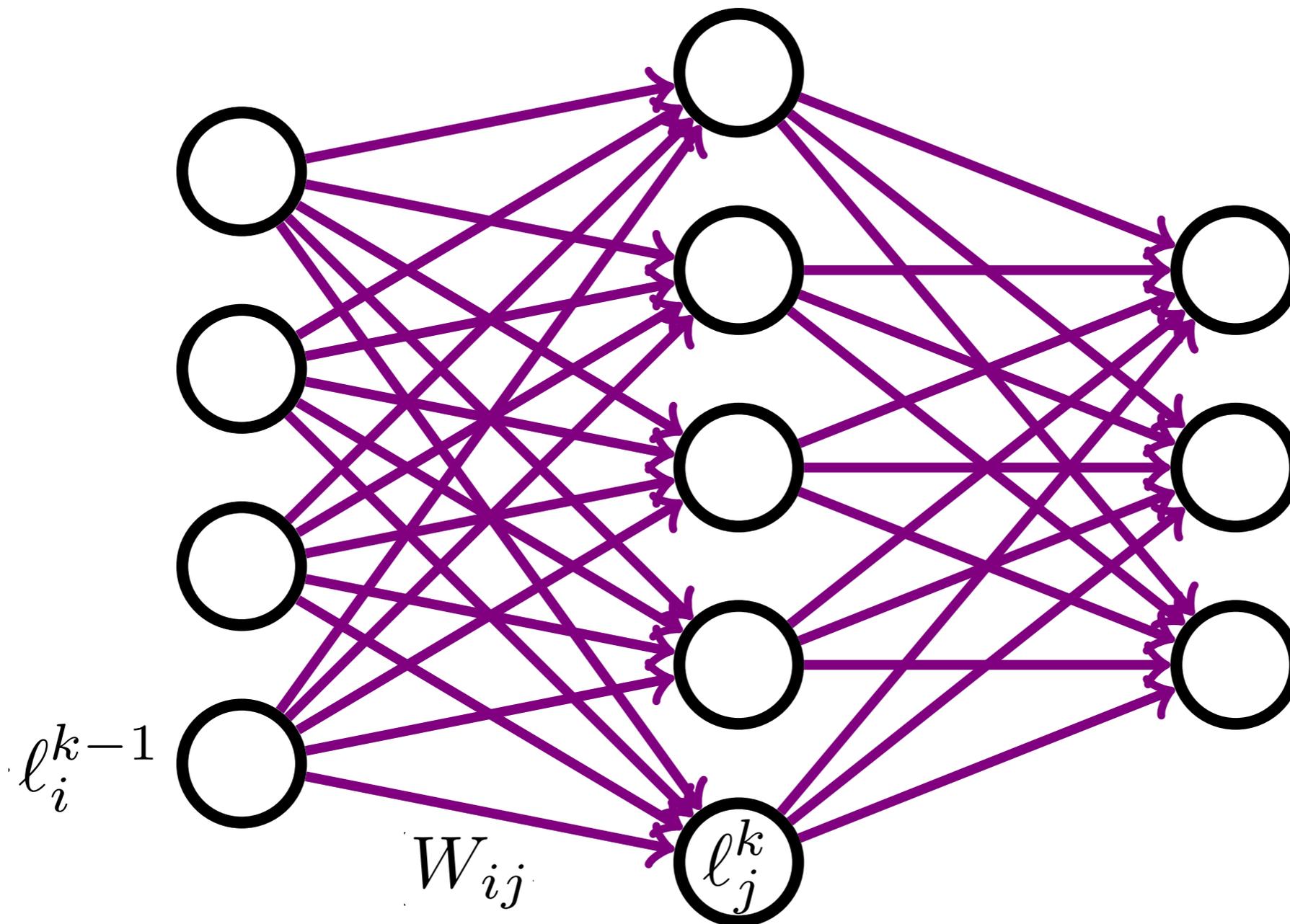
Case Study: Jet Substructure

- 5 output multi-classifier
 - Does a jet originate from a quark, gluon, W/Z boson, top quark?
- Fully connected network
- 16 expert inputs
- jet mass, multiplicity, ECFs



Neural Network

$$l_j^k = \phi(W_{ij}l_i^{k-1} + b_j)$$



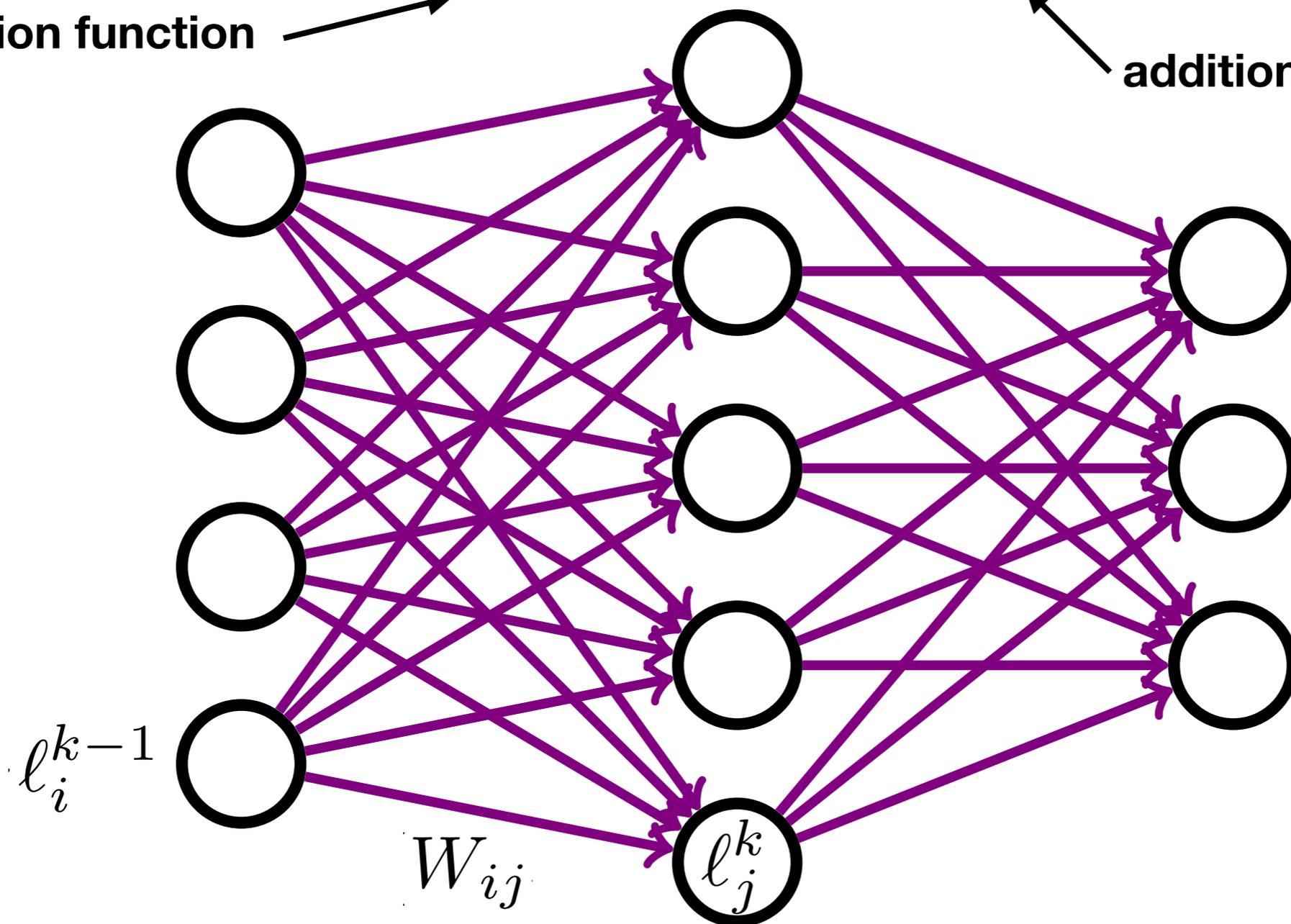
Neural Network

$$l_j^k = \phi(W_{ij}l_i^{k-1} + b_j)$$

activation function

multiplication

addition



Neural Network

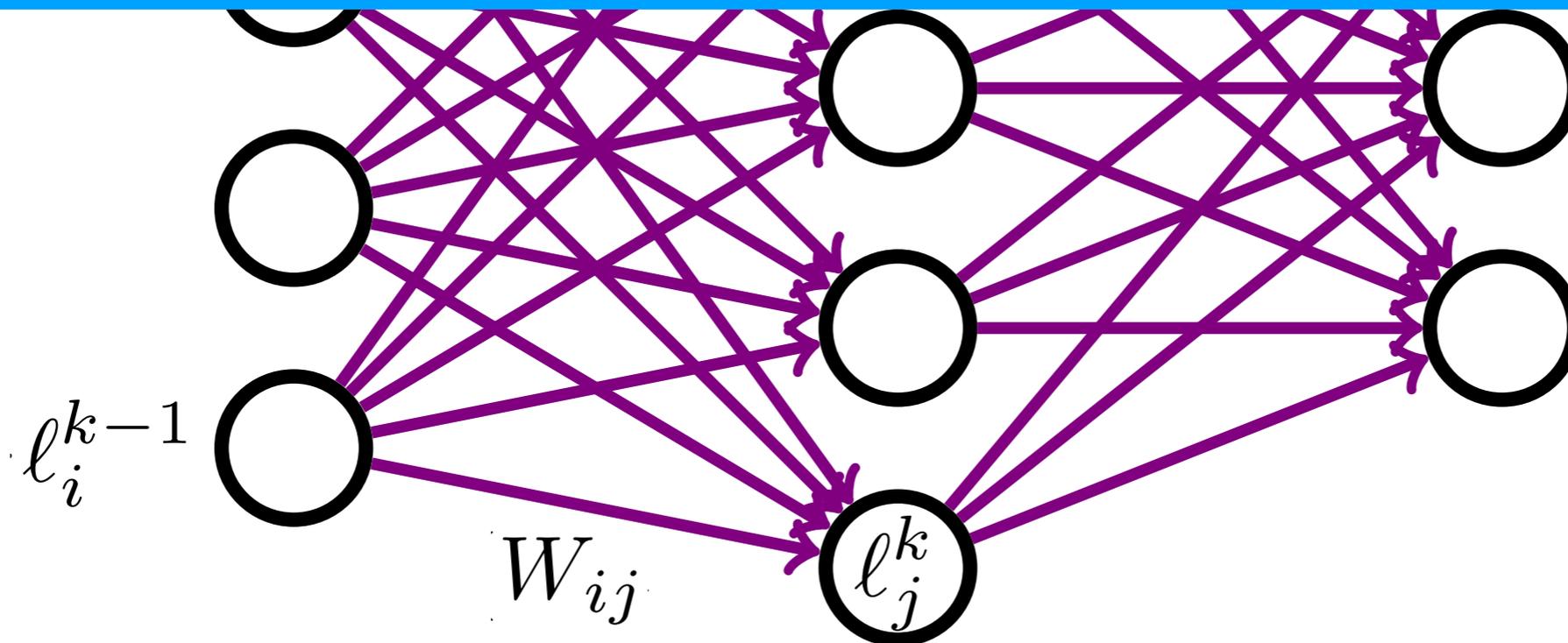
$$l_j^k = \phi(W_{ij}l_i^{k-1} + b_j)$$

multiplication

activation function

addition

NN = multiplications, additions, and pre-computed activation functions



Neural Network

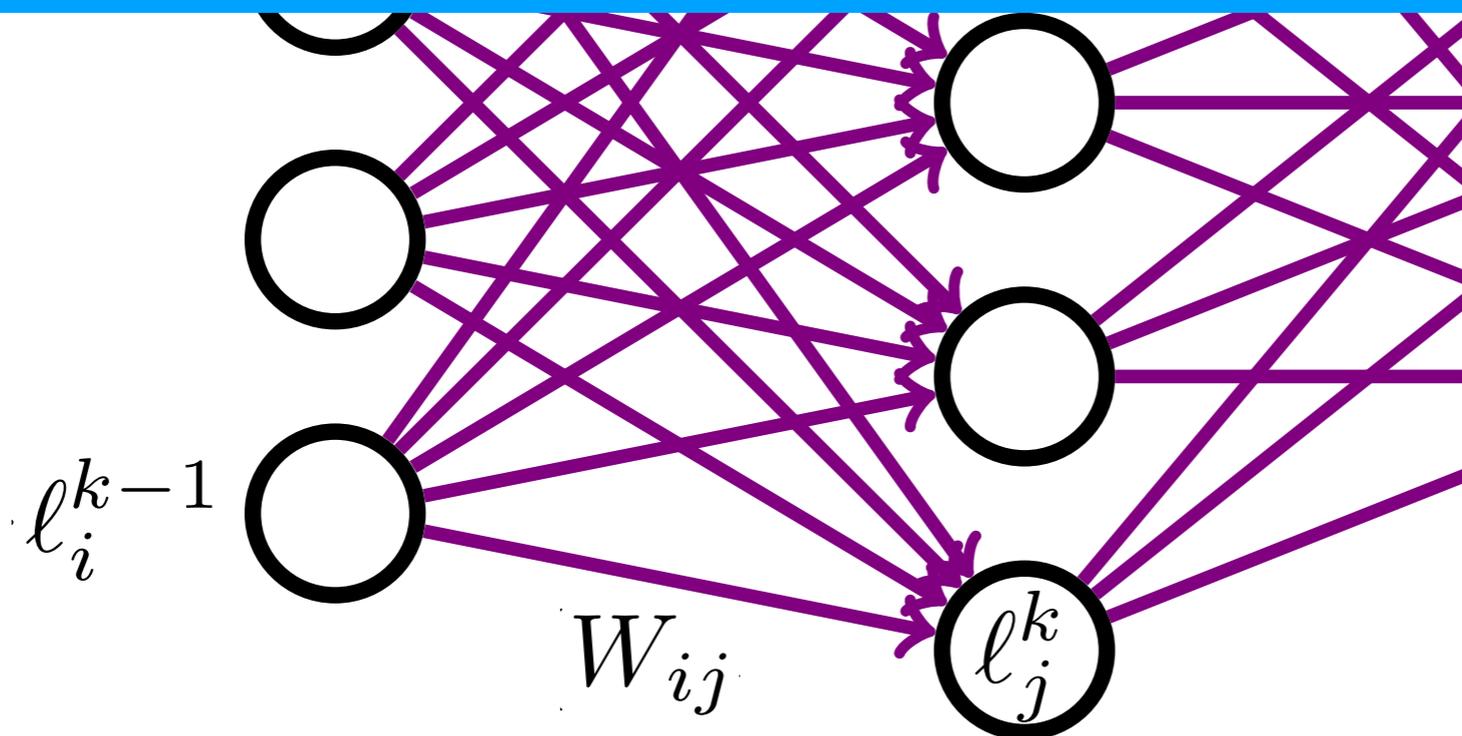
$$l_j^k = \phi(W_{ij}l_i^{k-1} + b_j)$$

activation function

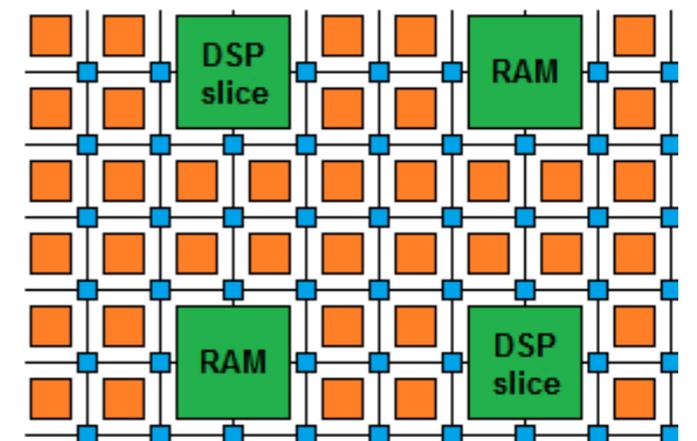
multiplication

addition

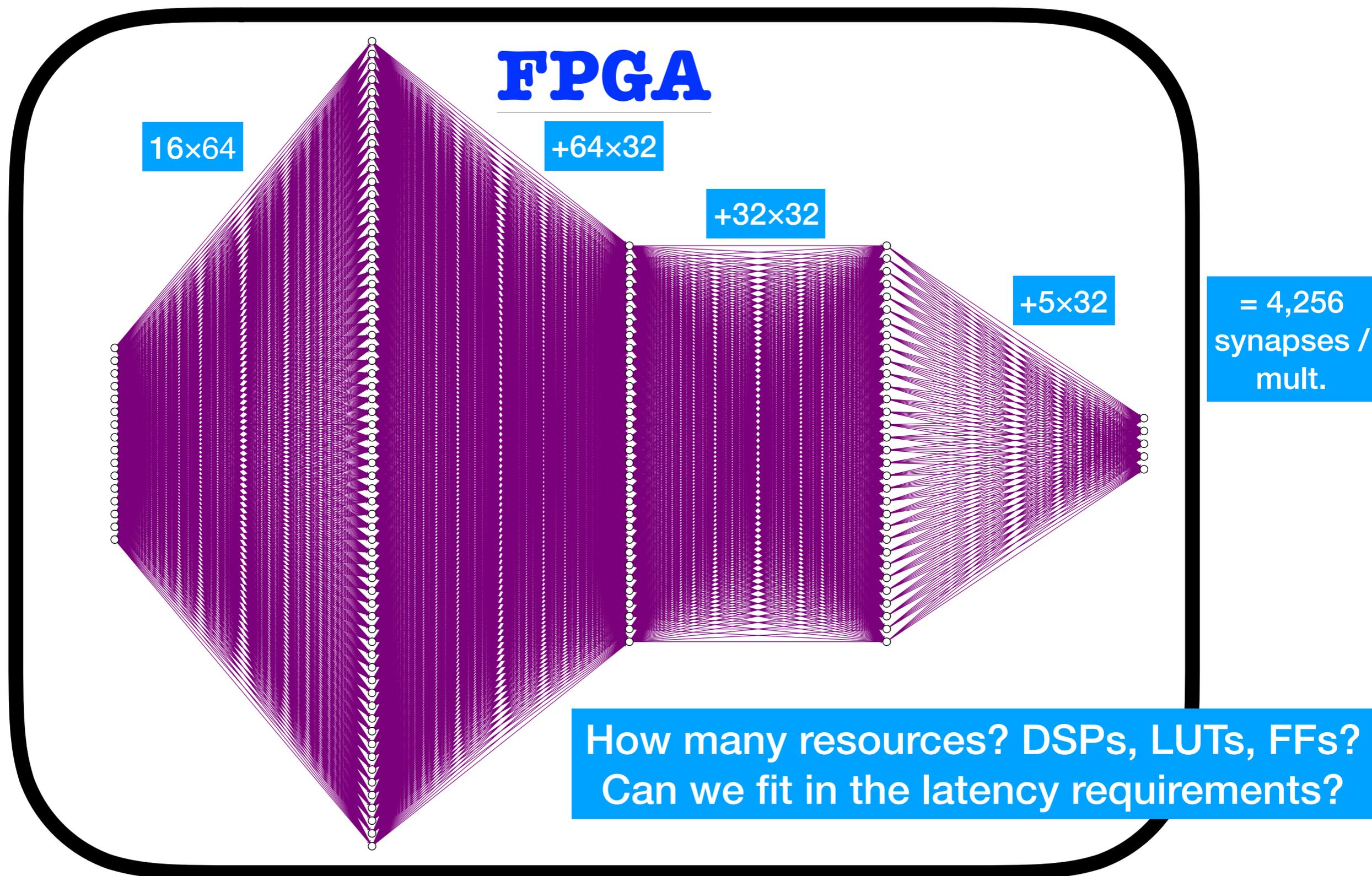
NN = multiplications, additions, and pre-computed activation functions



Maps nicely onto FPGA resources: high IO, DSPs, LUTs, etc.

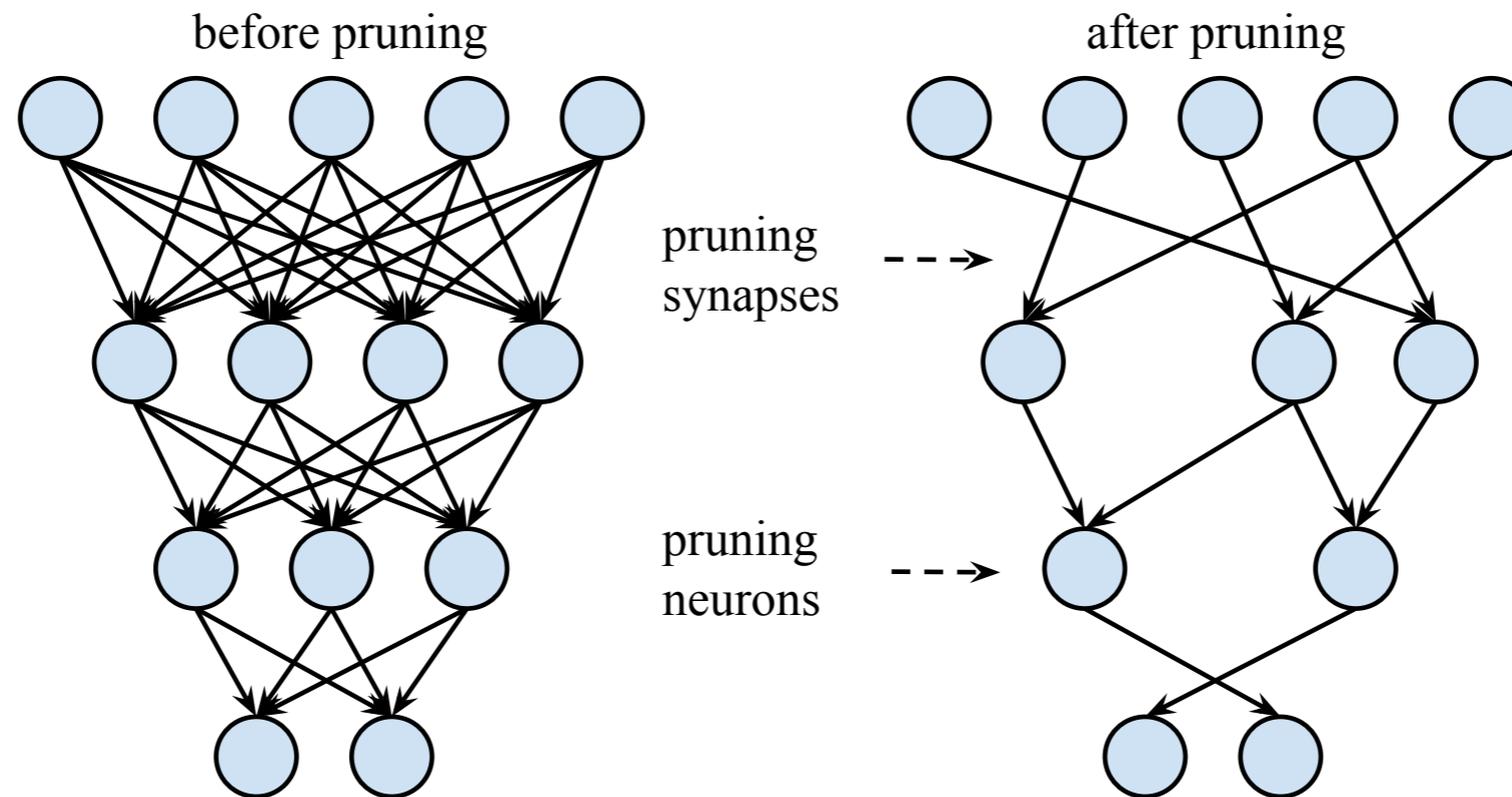


ML in FPGAs?



Efficient Neural Networks

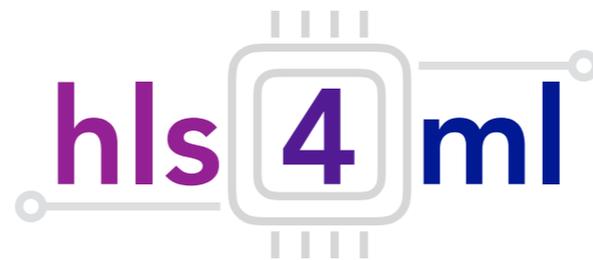
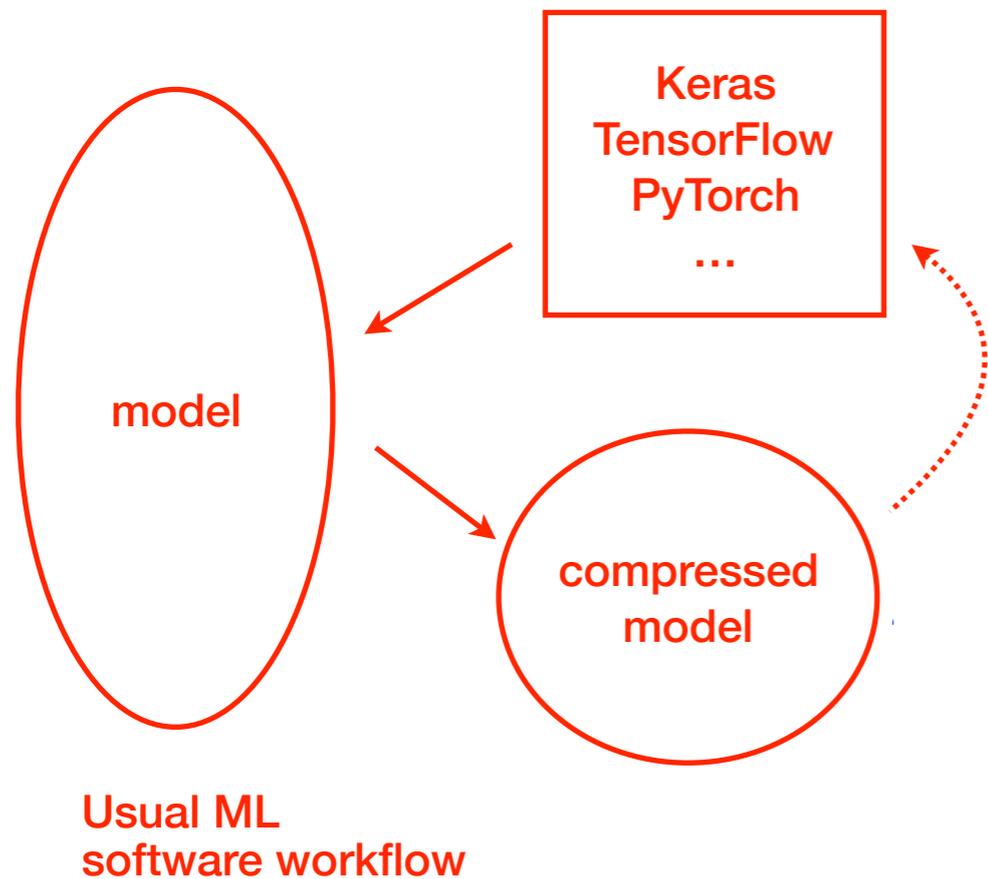
- Compression
 - Maintain same performance while removing redundant synapses and neurons
- Quantization
 - 32-bit floating point math is overkill
 - 20-bit, 18-bit, 8-bit, ...? fixed point, integers? binarized?



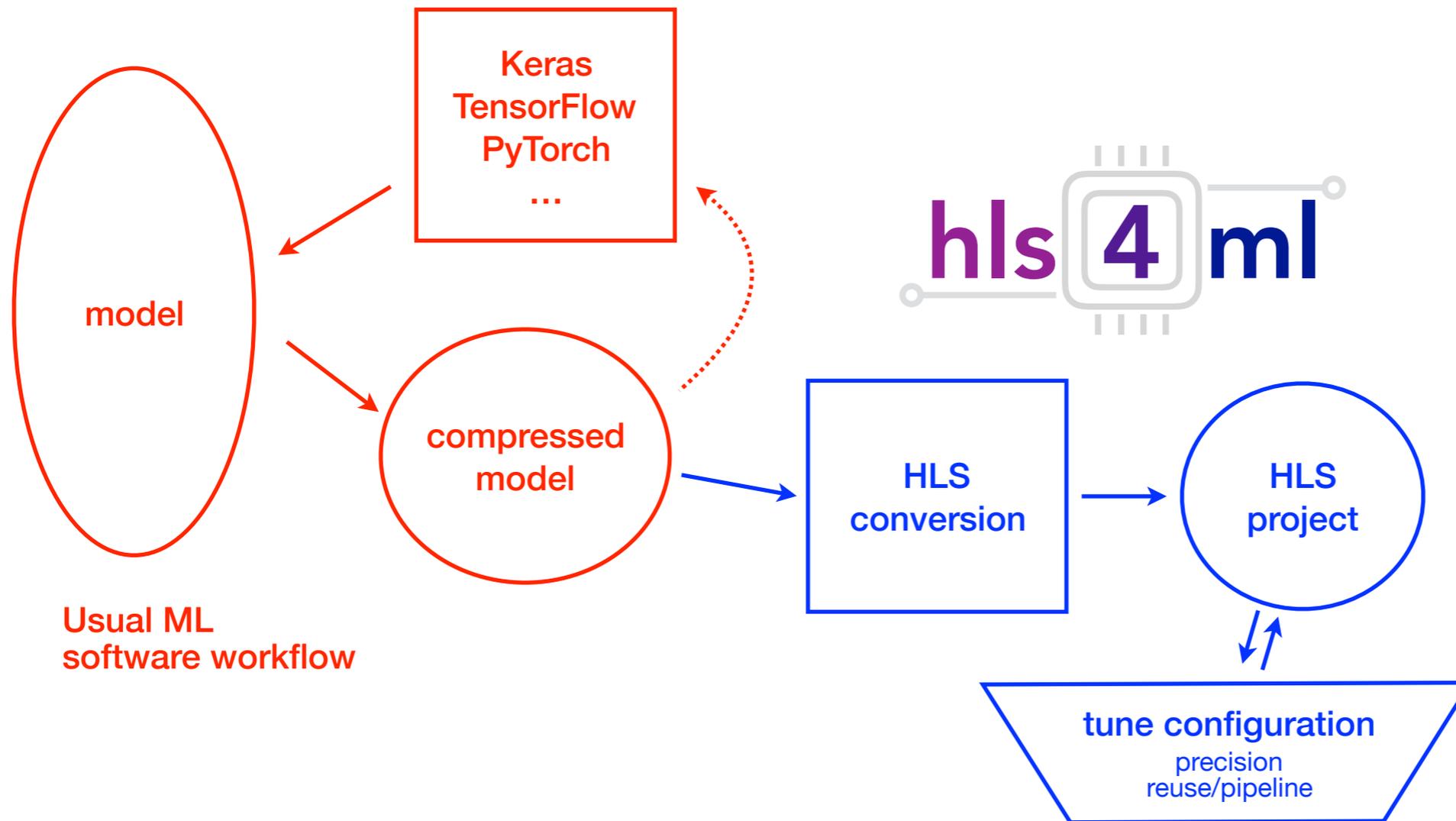
For further reading: [arXiv:1510.00149](https://arxiv.org/abs/1510.00149)



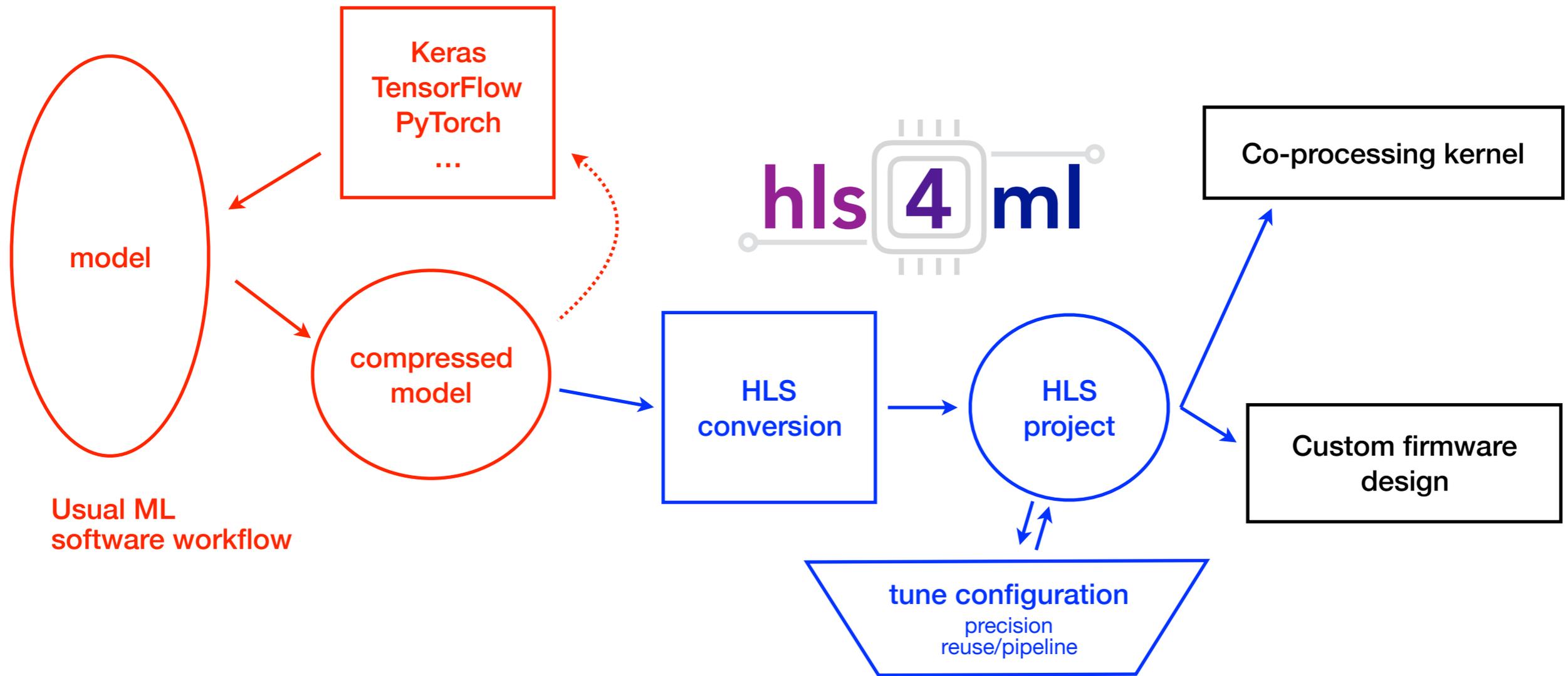
Design Exploration



Design Exploration



Design Exploration



Translation of ML models

Translation

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

Inputs



Keras



```
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

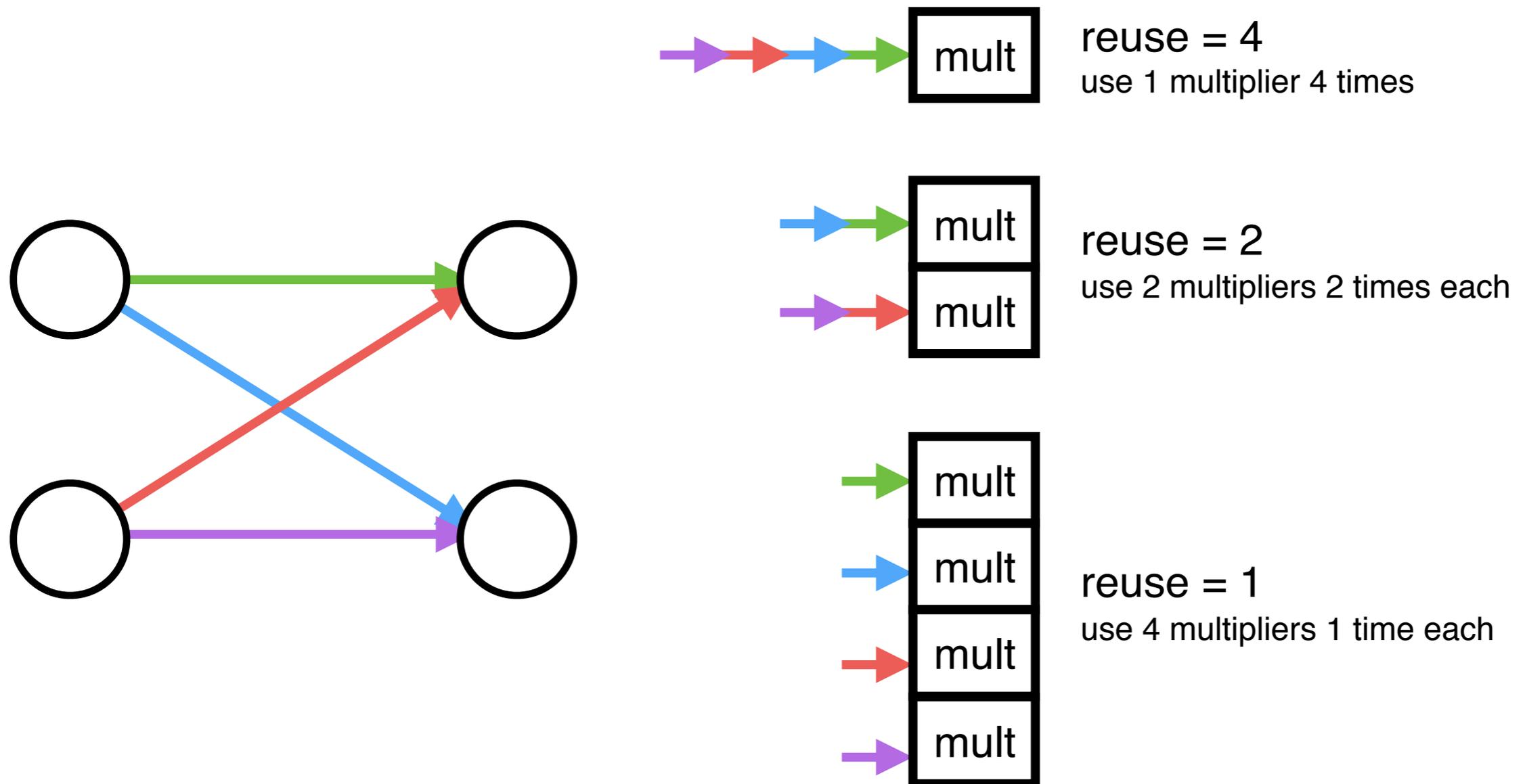
- IOType: parallelize or serialize
- ReuseFactor: how much to parallelize
- DefaultPrecision: inputs, weights, biases

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



Network Tuning: Parallelization

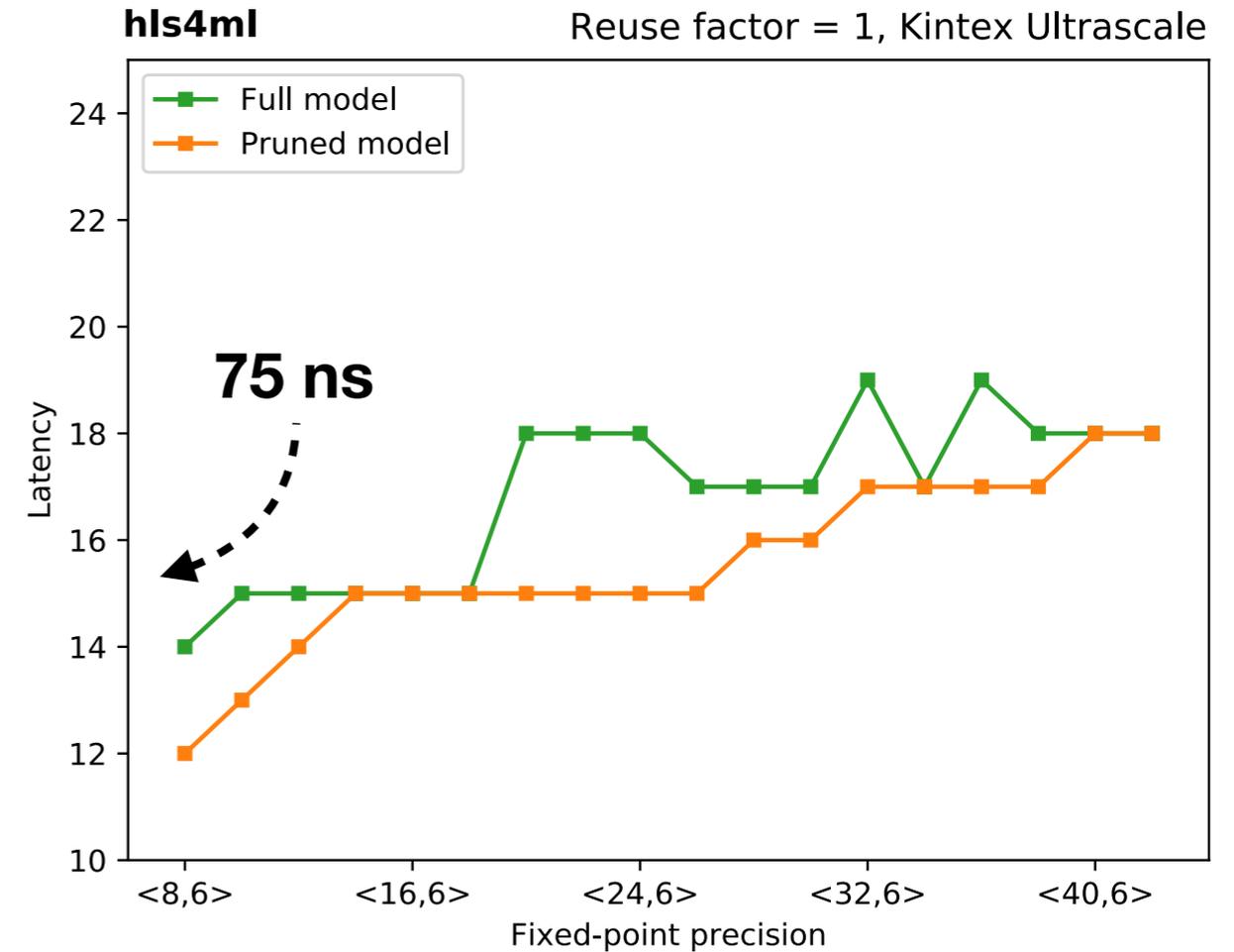
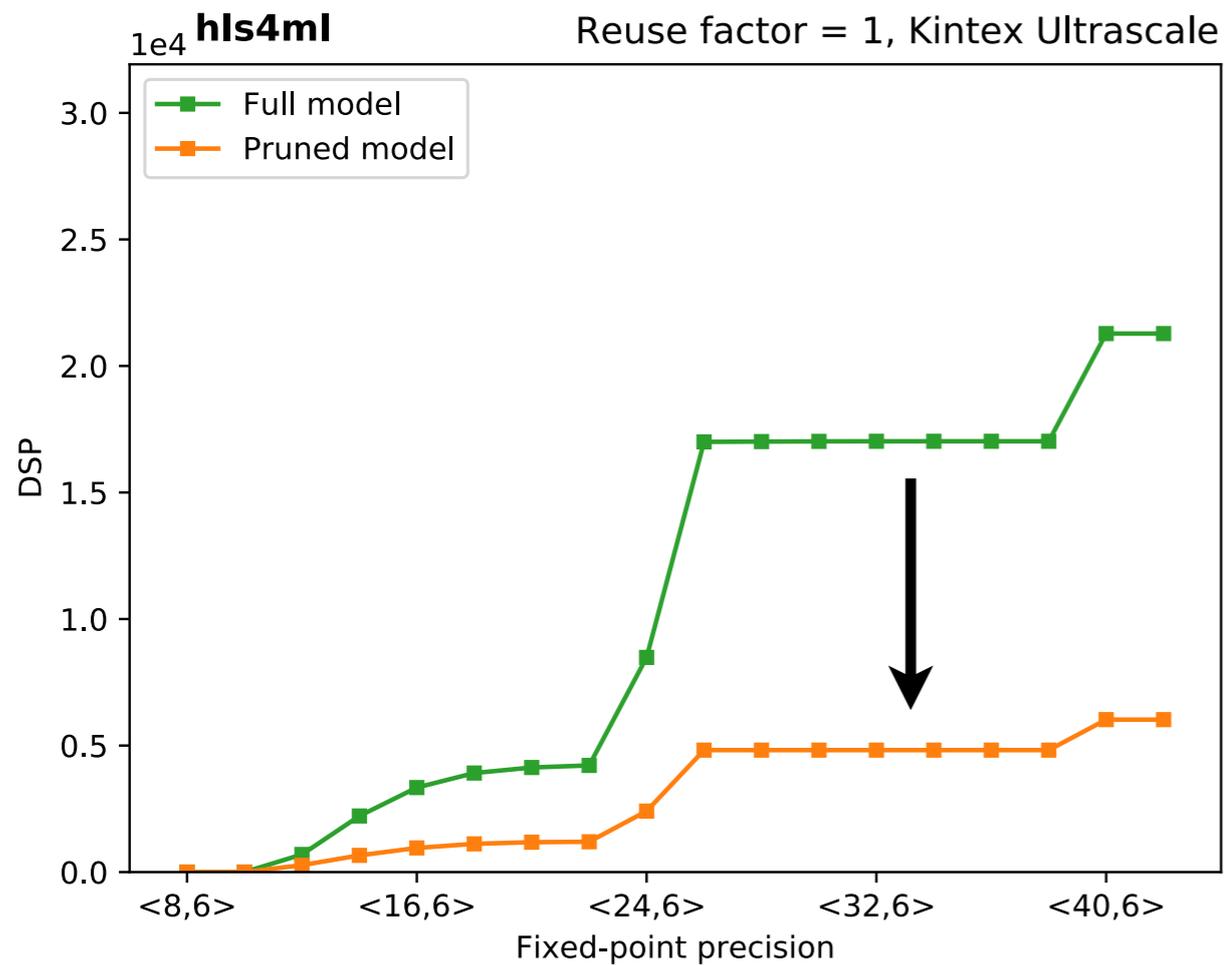
- ReuseFactor: how much to parallelize



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



Compression

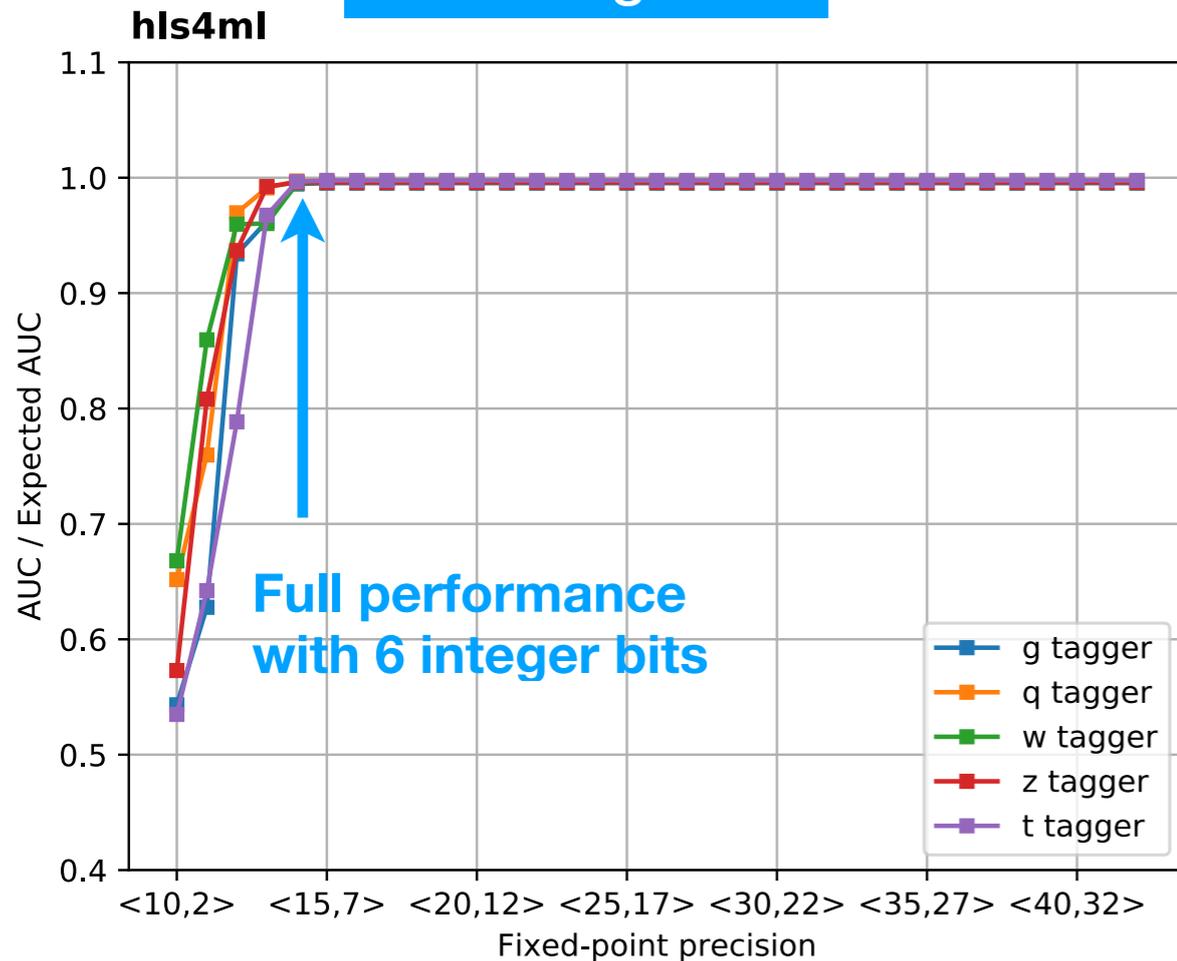


- Big reduction in DSP usage with pruned model!
- ~15 clocks @ 200 MHz = 75 ns inference

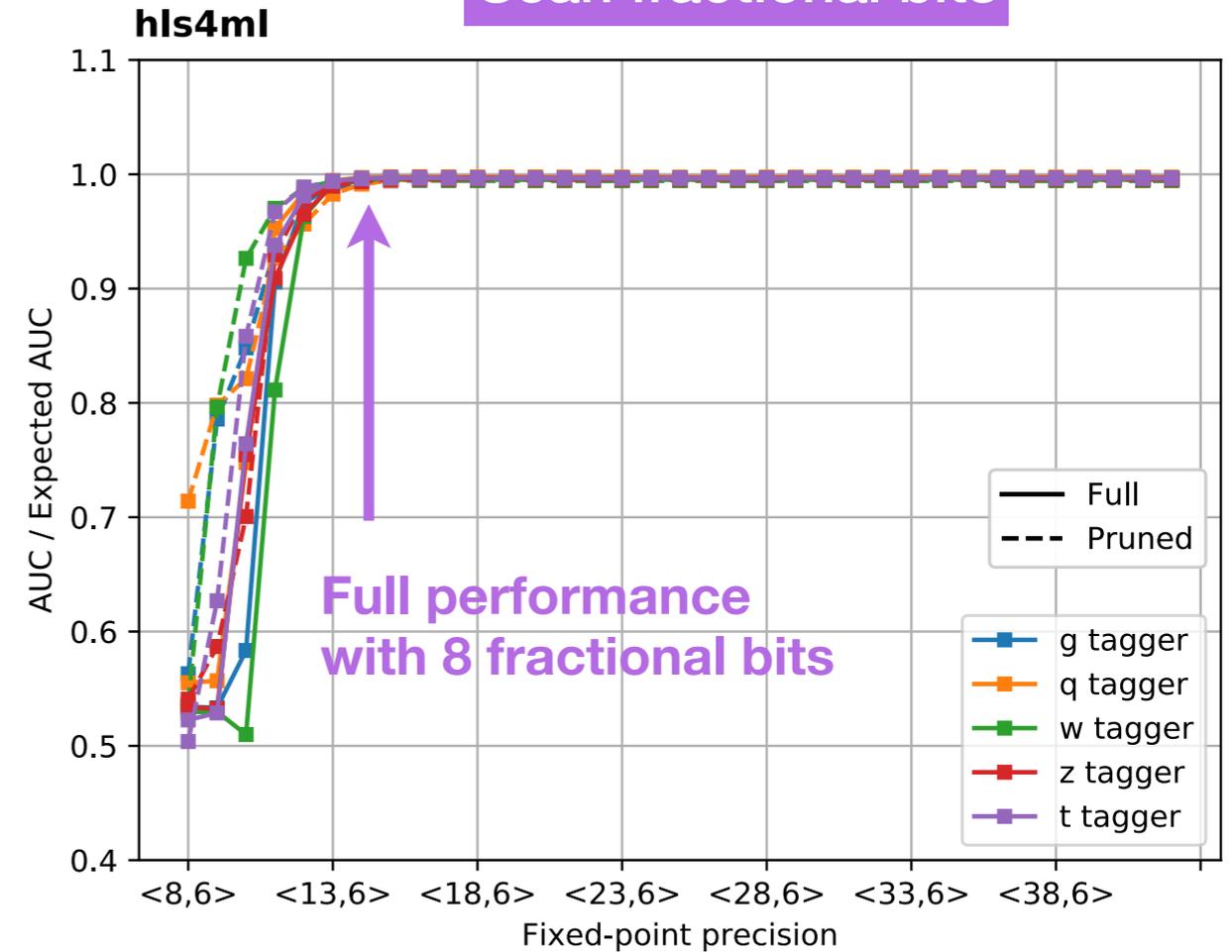


Quantization

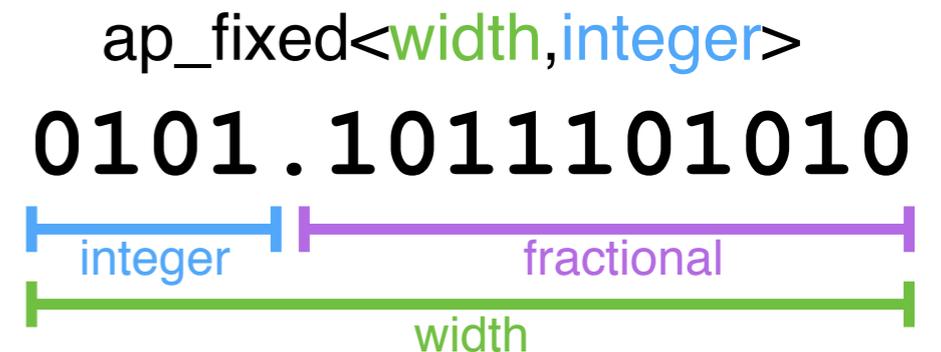
Scan integer bits



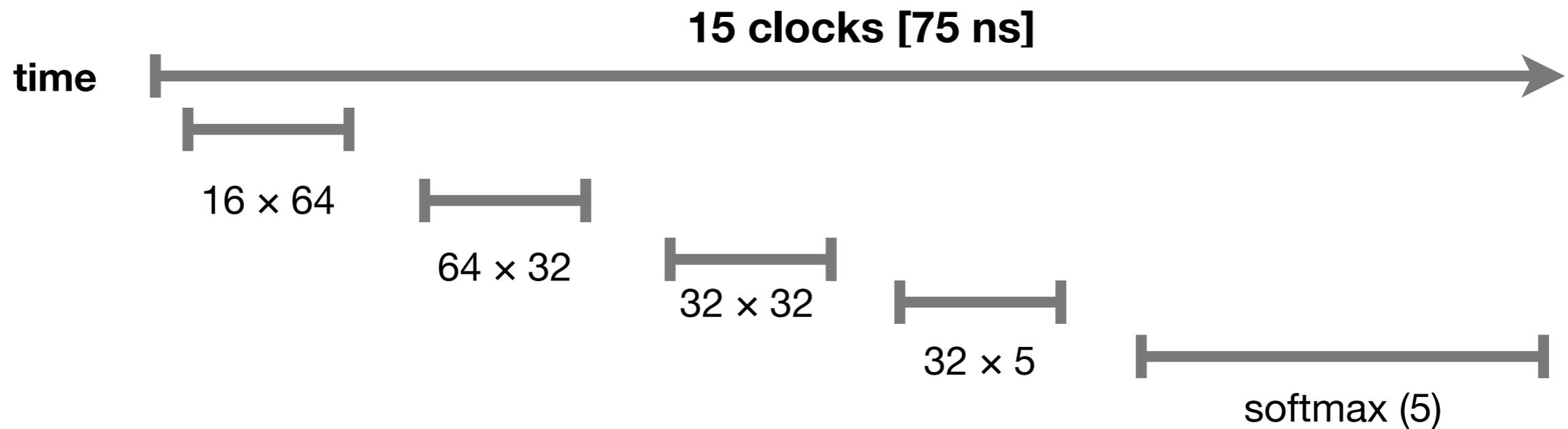
Scan fractional bits



- General strategy: avoid overflows in integer bit then scan the decimal bit until reaching optimal performance



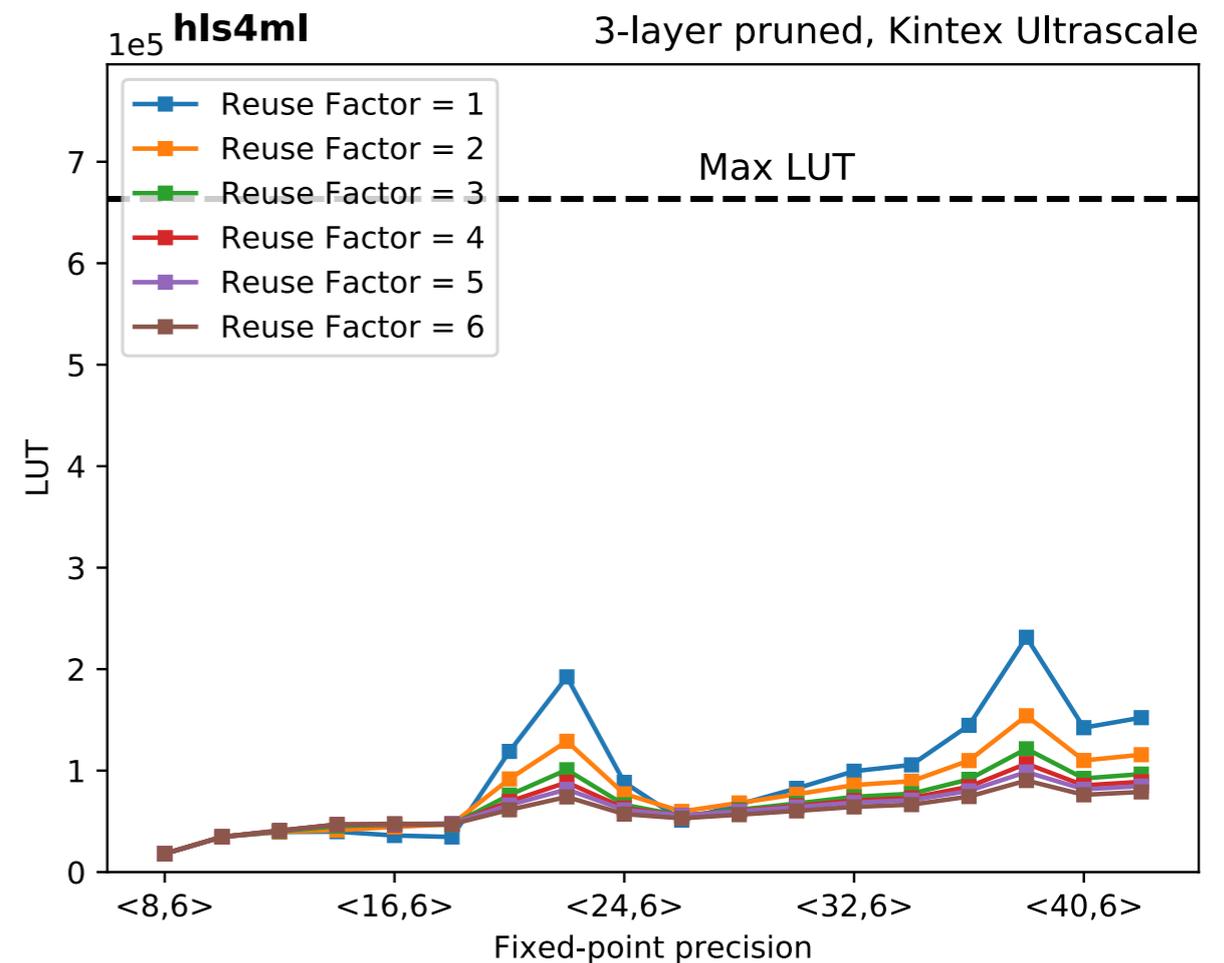
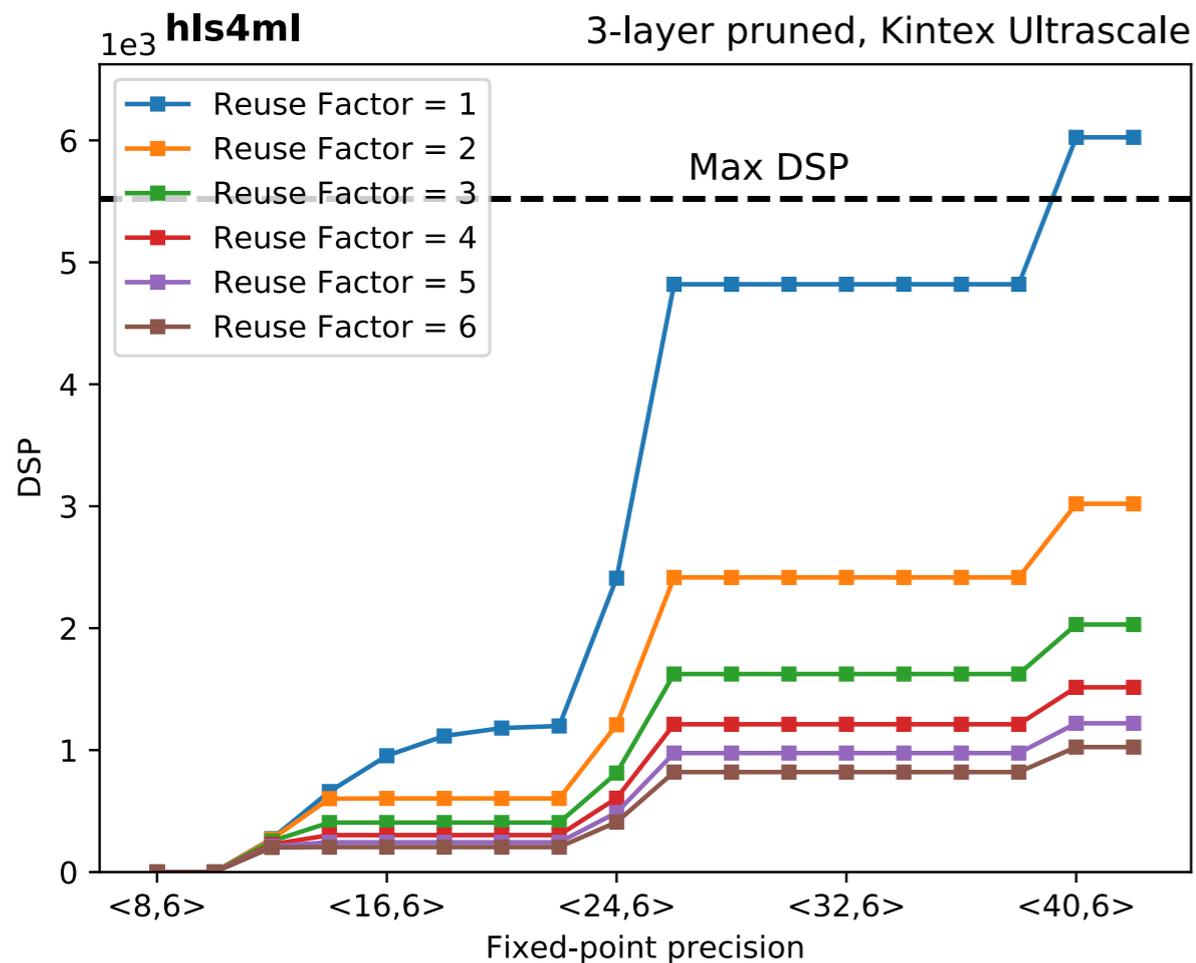
Resource Usage and Timing



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



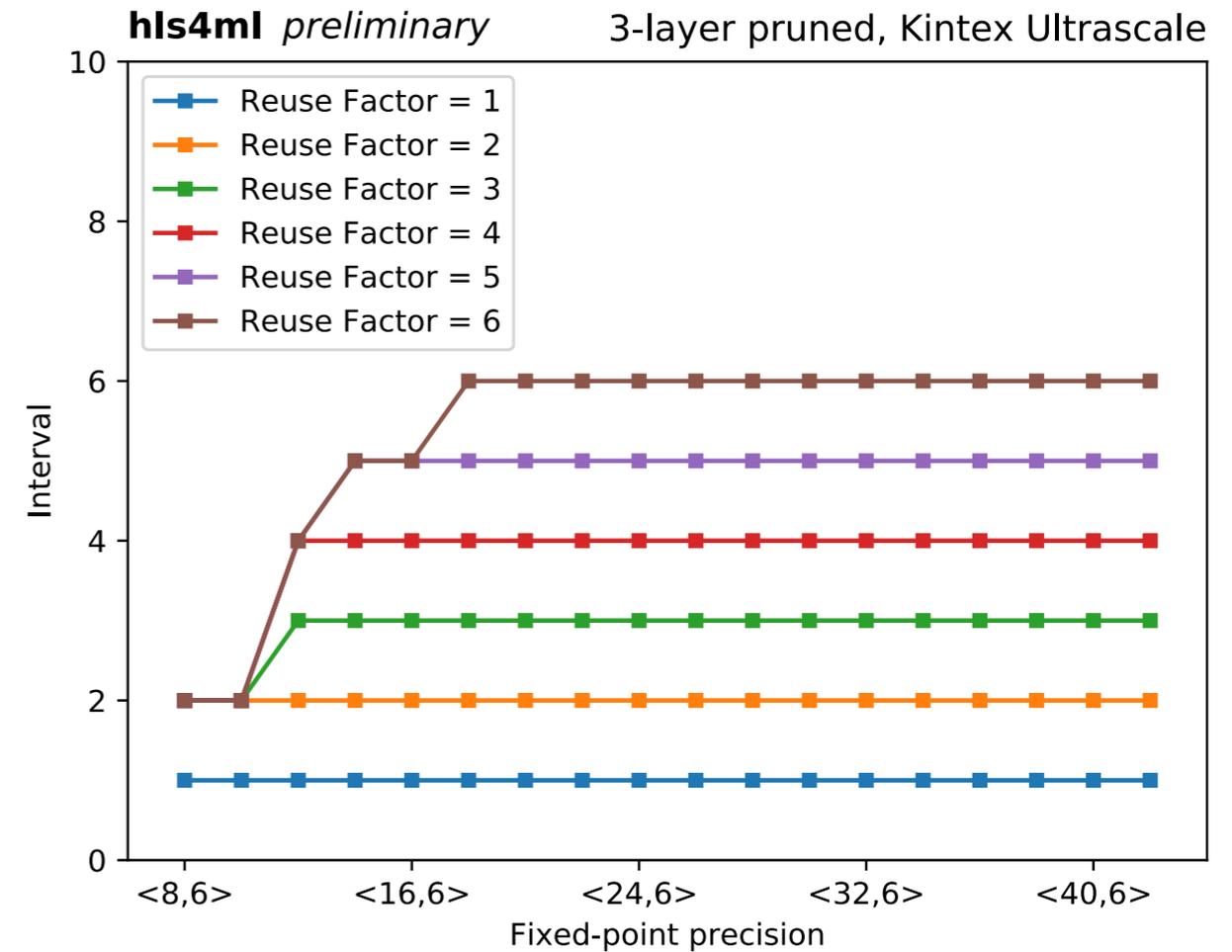
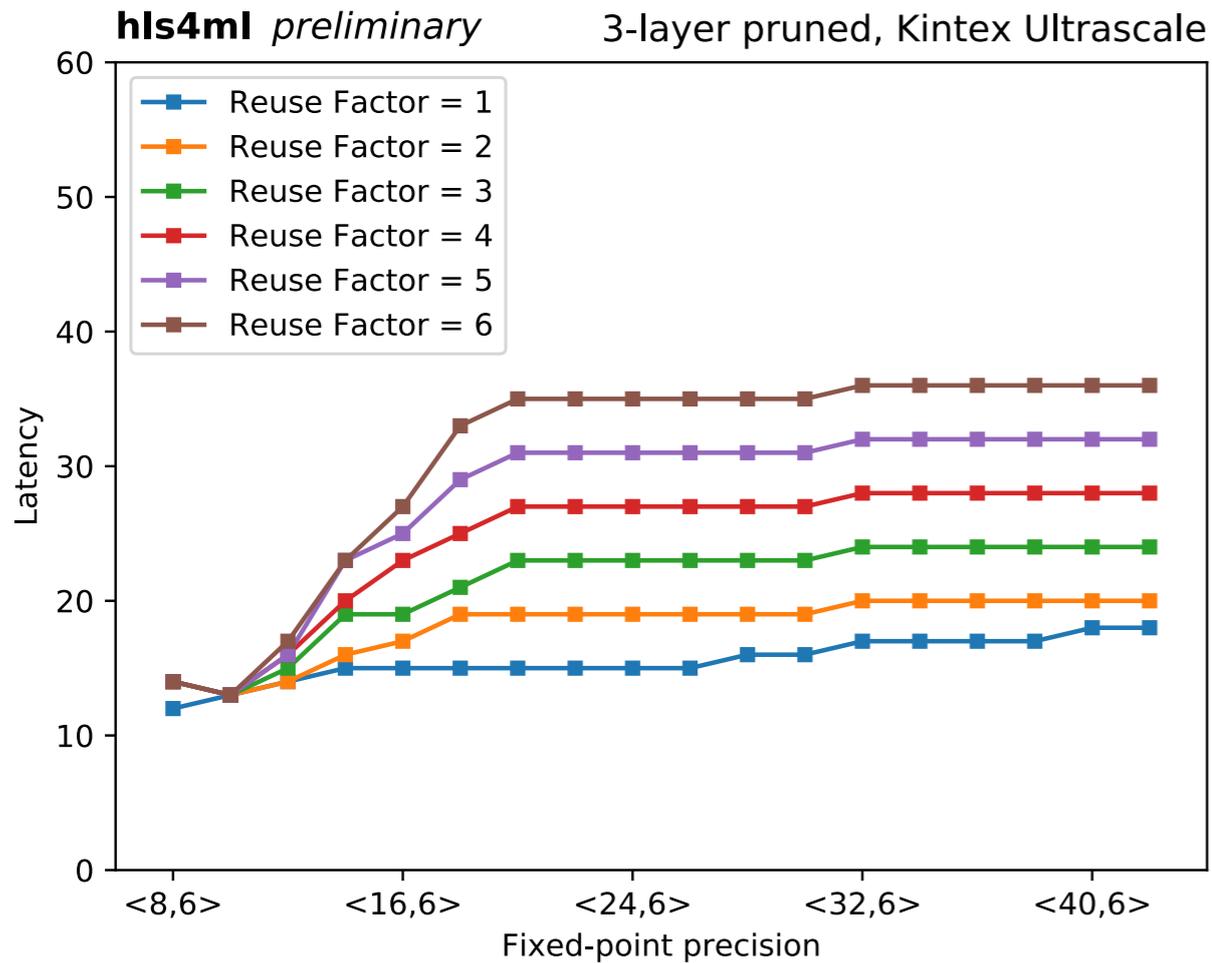
Resource Usage with Reuse



- Tuning the throughput with reuse factor reduces the DSP usage
- Steady increase of LUTs and FFs vs. bit precision
- Spikes in LUTs at the DSP precision transitions (not present in final implementation)



Timing with Reuse

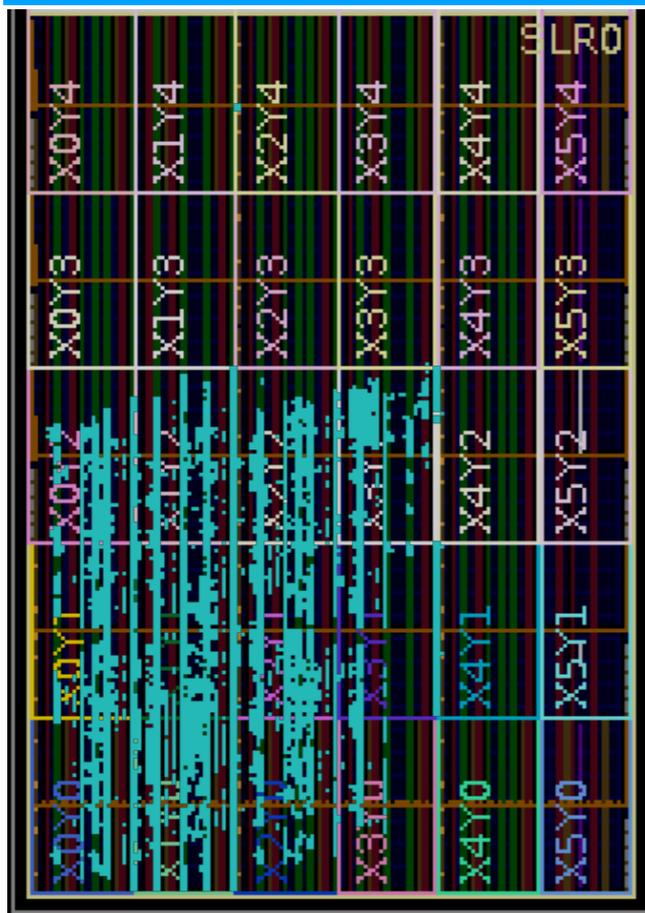


- Additional latency introduced by reusing the multipliers
- Initiation interval scales with the reuse factor



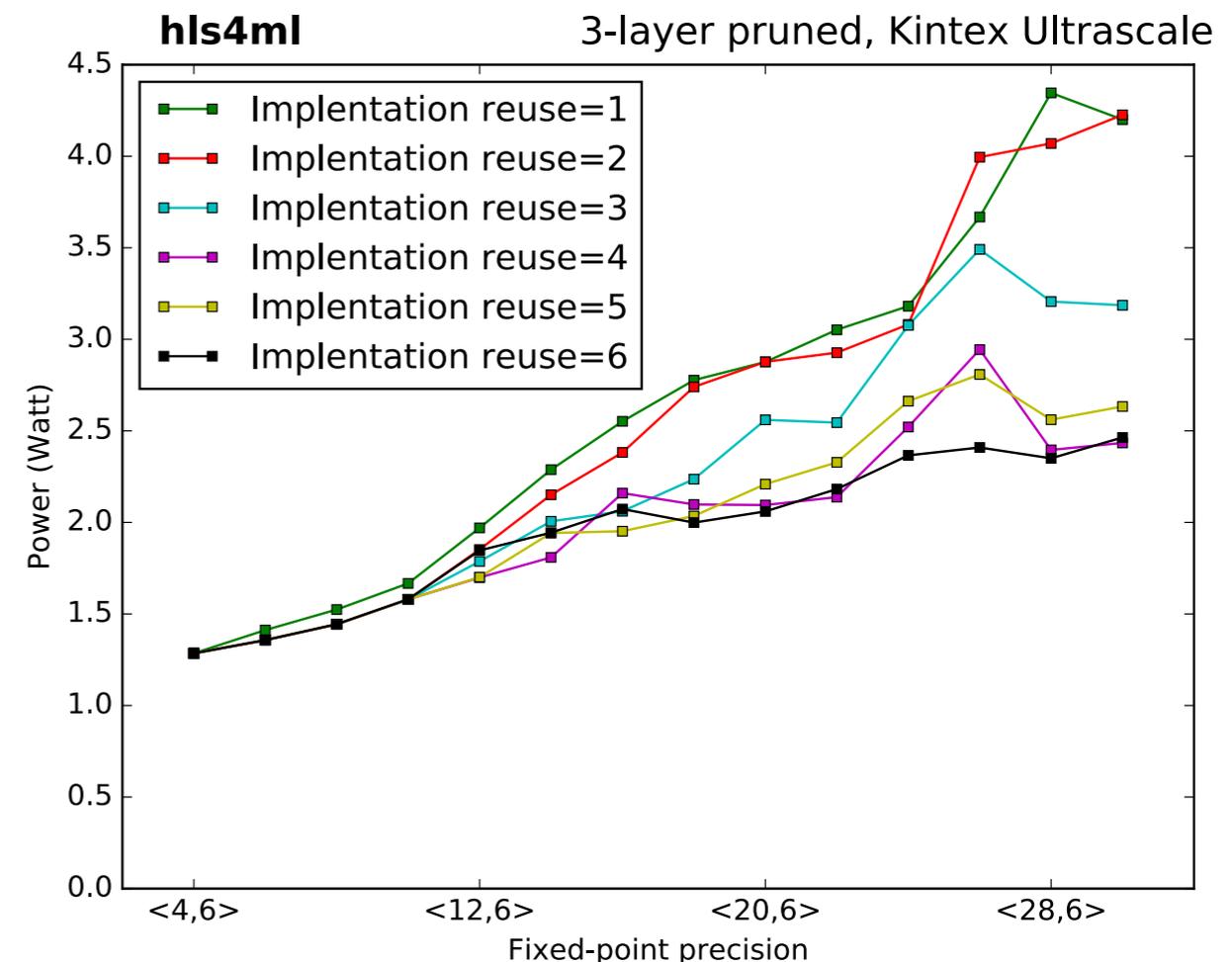
Implementing the HLS Design

“place and route”

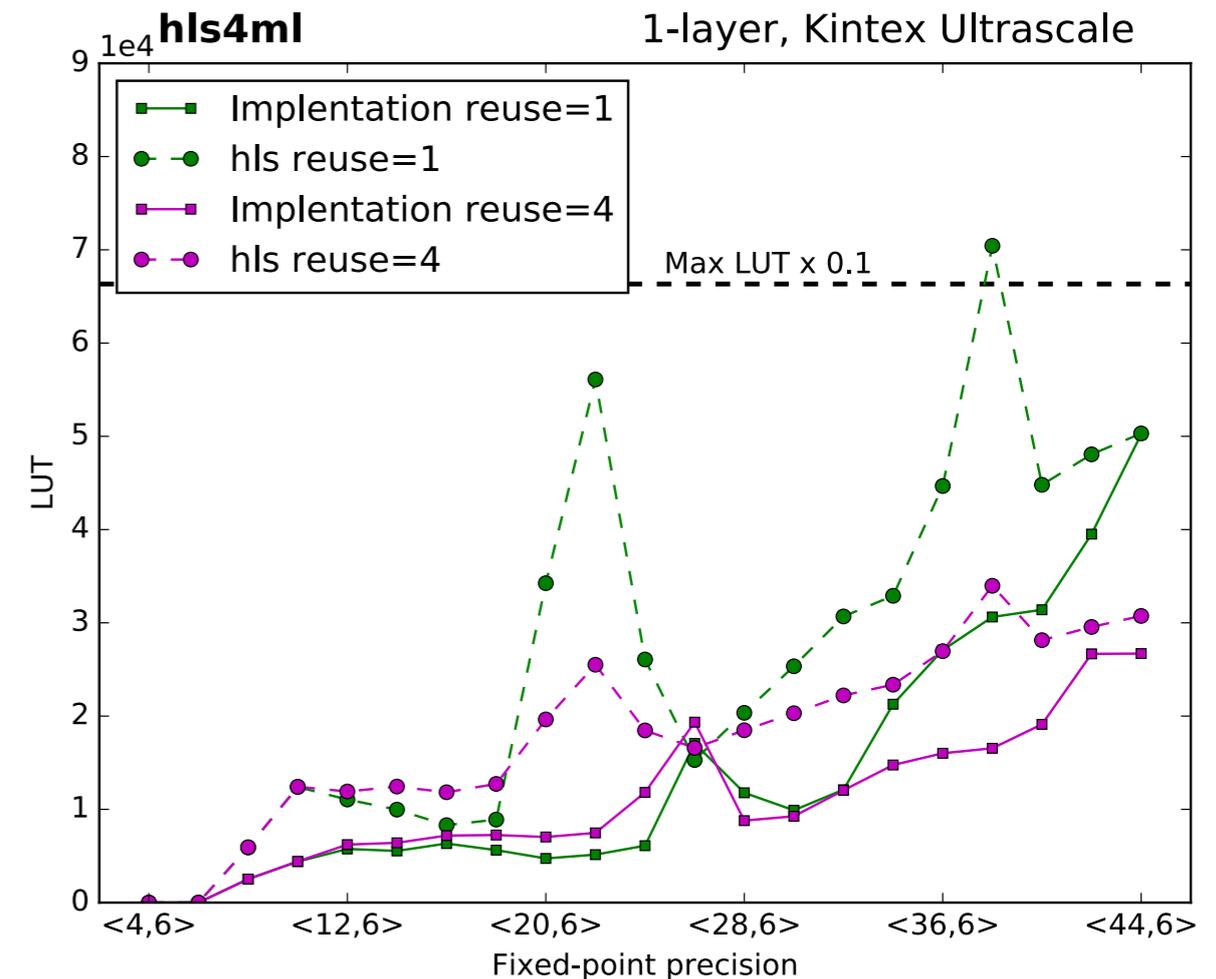
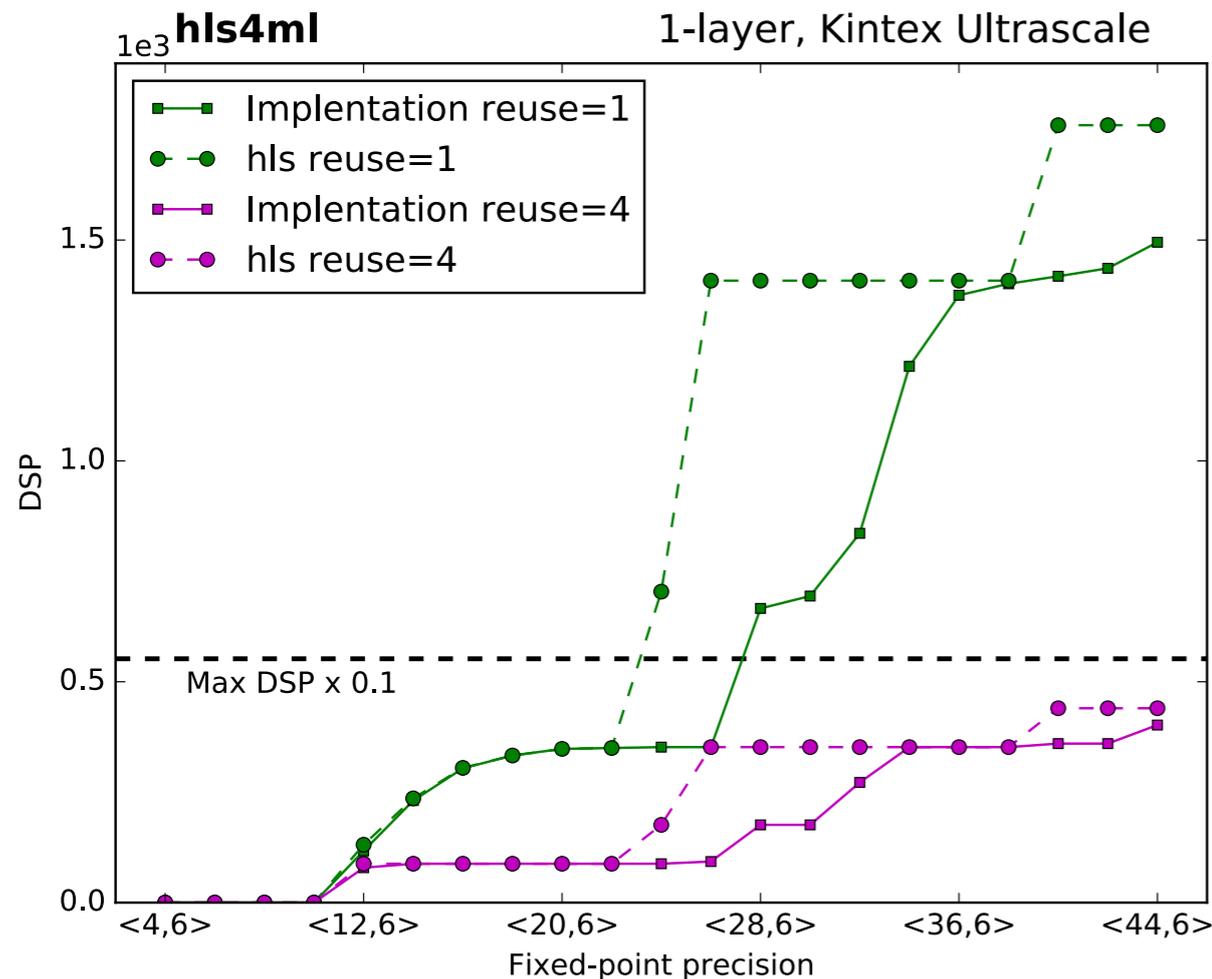


- Power decreases as throughput is decreased (by increasing reuse factor)

- How optimal is the HLS design (vs. RTL)?
 - For DSPs, HLS seems close to “back-of-the-envelope” optimal estimate
- HLS is good for quickly getting a **conservative** estimate of resources



HLS vs. Implementation



- HLS estimates are **conservative** compared to final implementation
- No spikes in LUTs at the DSP precision transitions in implementation



Summary and Outlook





New Developments

- Beta version is live! [arXiv:1804.06913](https://arxiv.org/abs/1804.06913)
- Work in progress:
 - LHC/DUNE applications
 - More network architectures:
 - Boosted Decision Tree (testing)
 - Binary Dense NN (testing)
 - **Conv1D, 2D (testing)**
 - BatchNormalization (prototyping)
 - LSTM, GRU (prototyping)
 - Graph-based NN (prototyping)

Fast inference of deep neural networks in FPGAs for particle physics

Javier Duarte^a, Song Han^b, Philip Harris^b, Sergo Jindariani^a, Edward Kreinar^c, Benjamin Kreis^a, Jennifer Ngadiuba^d, Maurizio Pierini^d, Ryan Rivera^a, Nhan Tran^a, Zhenbin Wu^e

^aFermi National Accelerator Laboratory, Batavia, IL 60510, USA

^bMassachusetts Institute of Technology, Cambridge, MA 02139, USA

^cHawkEye360, Herndon, VA 20170, USA

^dCERN, CH-1211 Geneva 23, Switzerland

^eUniversity of Illinois at Chicago, Chicago, IL 60607, USA

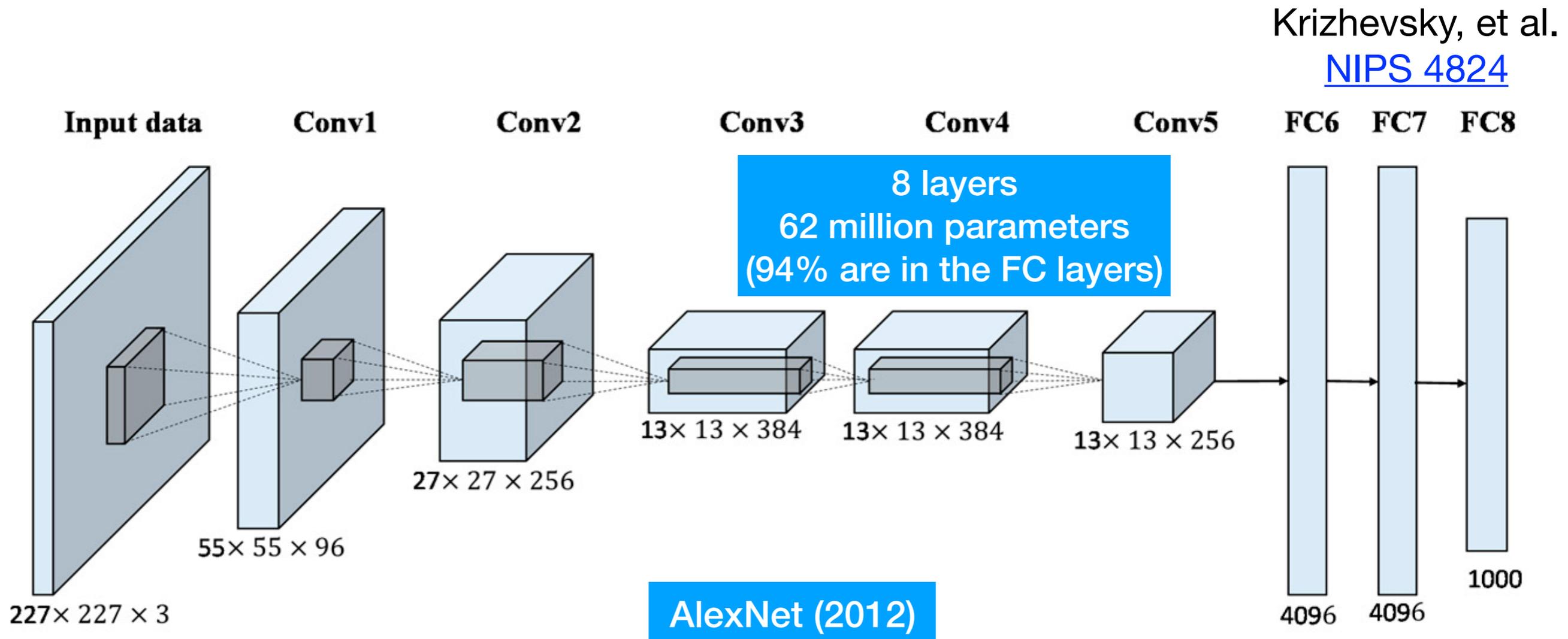
E-mail: hls4ml.help@gmail.com

[hls-fpga-machine-learning.github.io/hls4ml](https://github.com/hls4ml/hls-fpga-machine-learning)



Big Convolutional Neural Networks

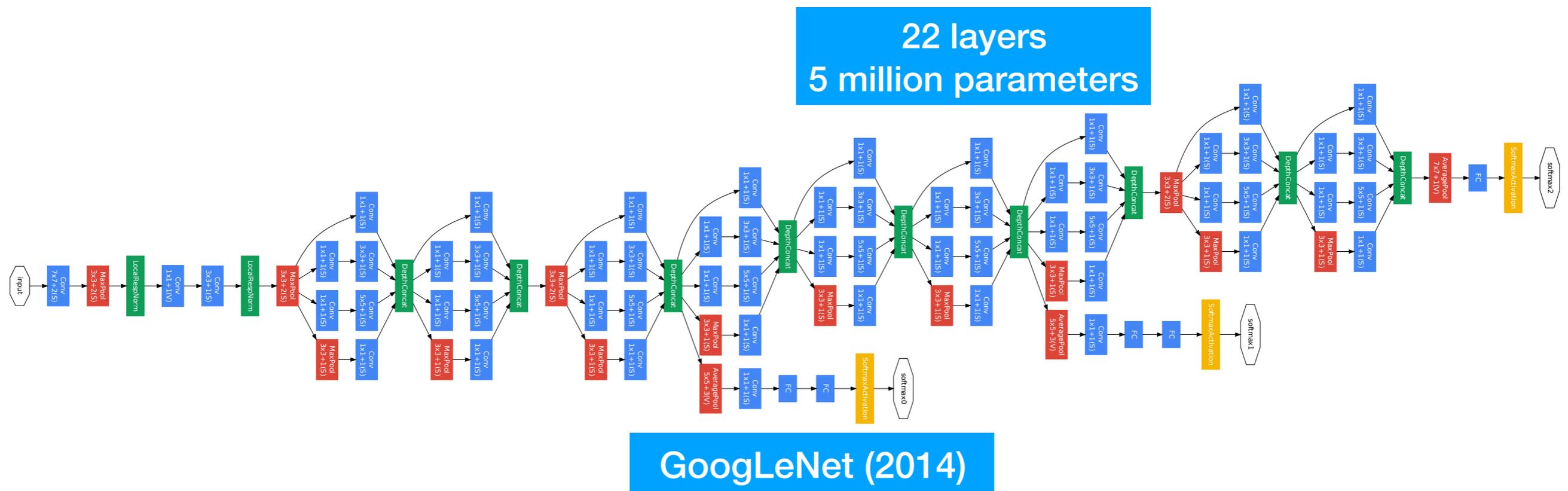
- Main task is computer vision/image recognition
- Control the number of parameters by baking in assumptions like locality and translation invariance to **share weights** within a layer



Big Convolutional Neural Networks

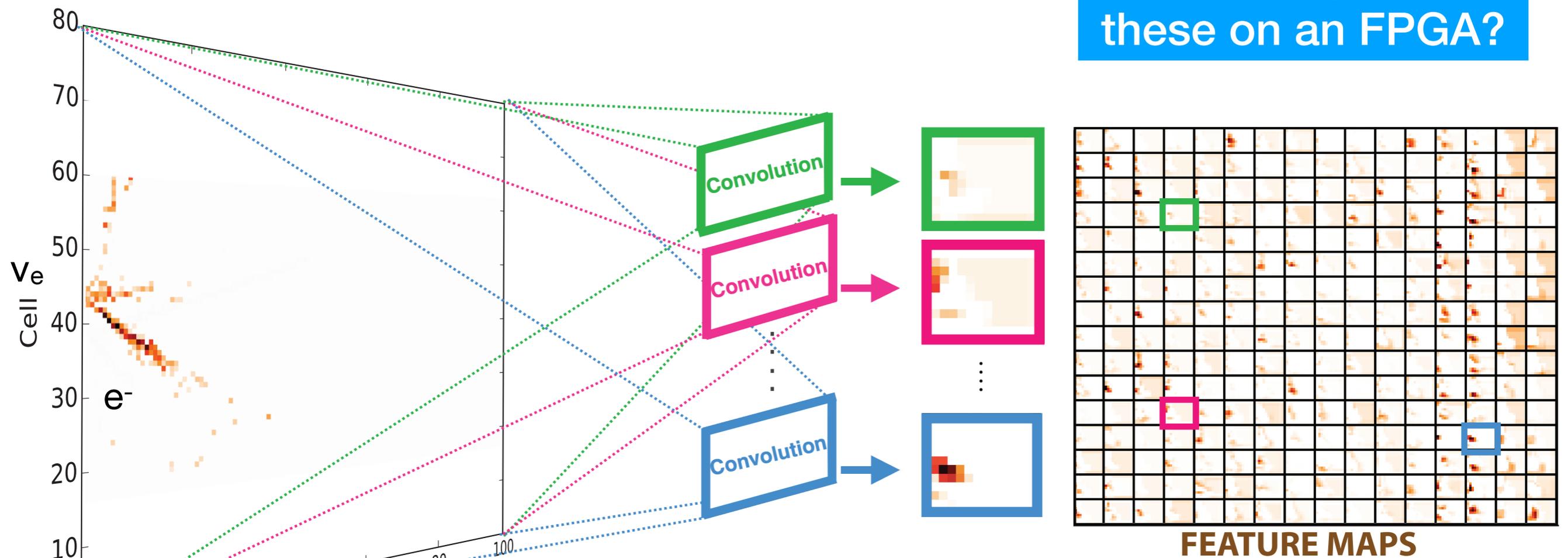
- Main task is computer vision/image recognition
- Control the number of parameters by baking in assumptions like locality and translation invariance to **share weights** within a layer

Szegedy et al.
[arXiv:1409.4842](https://arxiv.org/abs/1409.4842)



CNNs in Neutrino Experiments

- Readout detector as a (multidimensional) image
- Shown to be effective at classifying NovA neutrino events
- Adapted from GoogLeNet



Evan Niner
Deep Learning in NovA
[arXiv:1604.01444](https://arxiv.org/abs/1604.01444)



HEP Latency Landscape



Pure FPGAs



HEP Latency Landscape

Tools to infer big networks:

- “Serial” inputs
- Use BRAM to store weights
- Increase reuse factor $\times 1000$
- Additional compression: weight sharing, Huffman coding
- Multi-pumping DSPs

Leads to $O(\text{ms})$ latencies



Pure FPGAs



HEP Latency Landscape

Tools to infer big networks:

- “Serial” inputs
- Use BRAM to store weights
- Increase reuse factor $\times 1000$
- Additional compression: weight sharing, Huffman coding
- Multi-pumping DSPs

Leads to $O(\text{ms})$ latencies

Natural application:
accelerate offline workflows



Summary and Outlook

- hls4ml: an HLS-based software/firmware compiler for ultra low-latency applications
 - Case study with jet substructure for Level-1 trigger
 - Features in development:
 - More network architectures (CNN, RNN, etc.)
 - Applications planned within CMS, DUNE, and more...
- FPGAs (with support from industry/cloud computing) may help to accelerate HEP computing workflows
- **Exciting times** ahead at the intersection of machine learning, custom hardware, and high energy physics



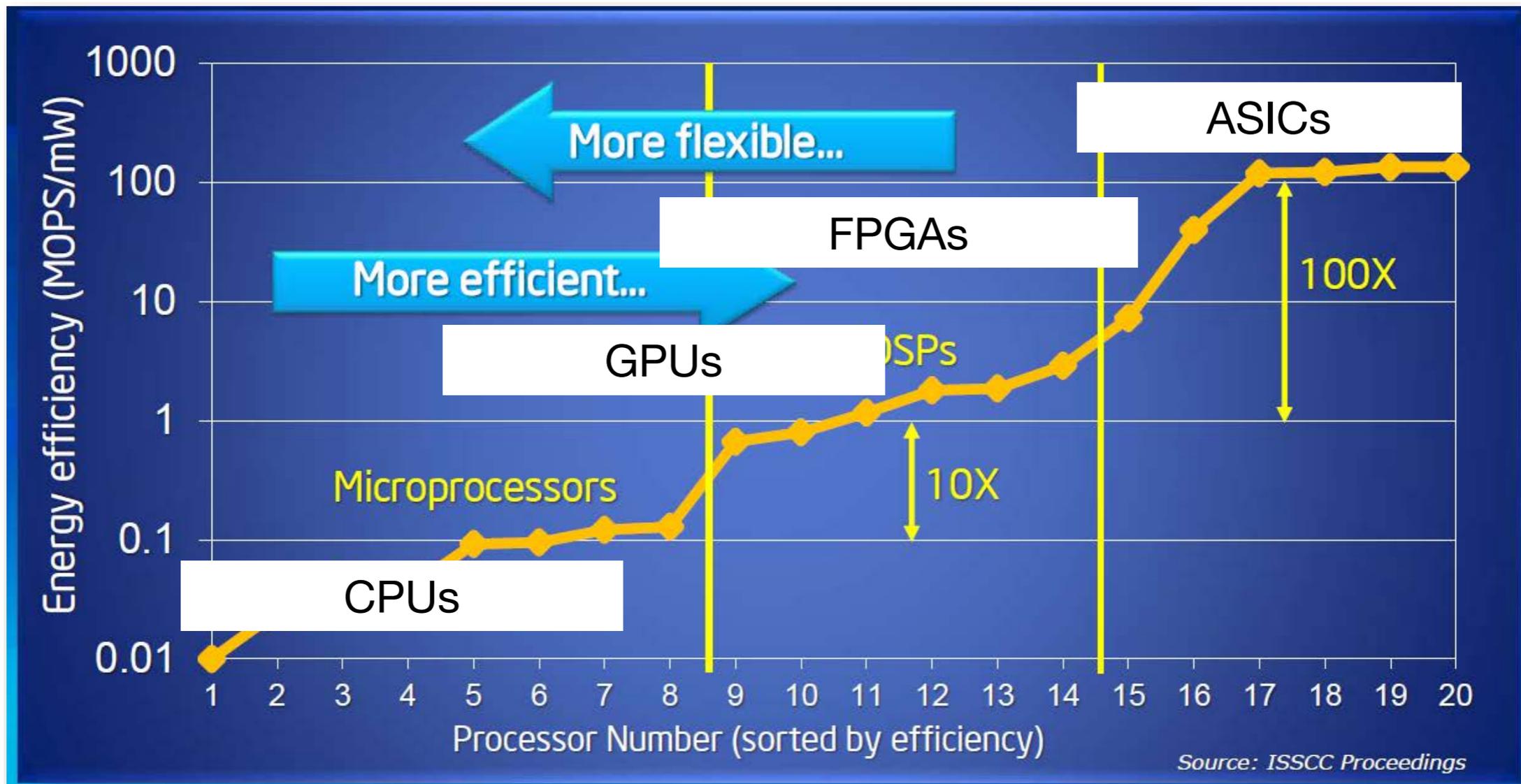
Backup



CPUs, GPUs, FPGAs, and ASICs

How are they different?

- FPGAs are the middle ground of latency, energy efficiency, and flexibility



Source: Bob Broderson, Berkeley Wireless group



High-Level Synthesis

- **FPGA** development is becoming more accessible, with tools like **High-Level Synthesis: untimed C code** with additional **directives** that is synthesized into **RTL Verilog/VHDL**

```
module dut(rst, clk, q);  
  input rst;  
  input clk;  
  output q;  
  reg [7:0] c;  
  
  always @ (posedge clk)  
  begin  
    if (rst == 1b'1) begin  
      c <= 8'b00000000;  
    end  
    else begin  
      c <= c + 1;  
    end  
  
    assign q = c;  
  endmodule
```

vs.

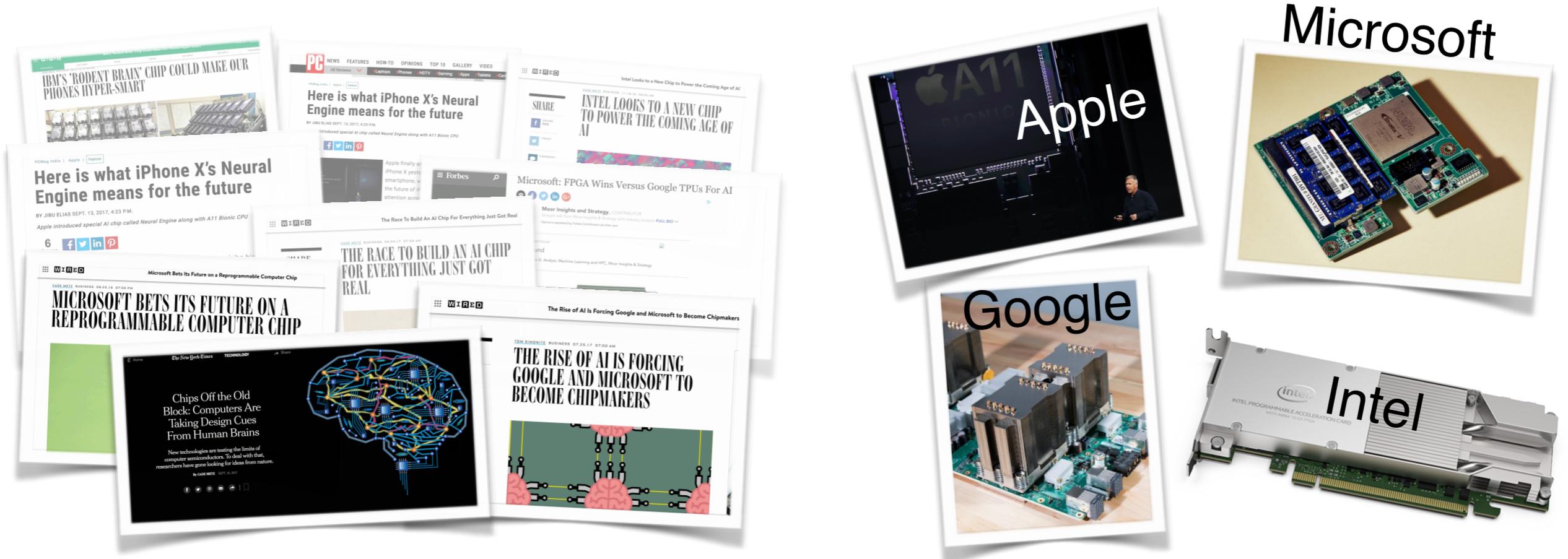
```
uint8 dut() {  
  static uint8 c;  
  c+=1;  
}
```

Untimed C code



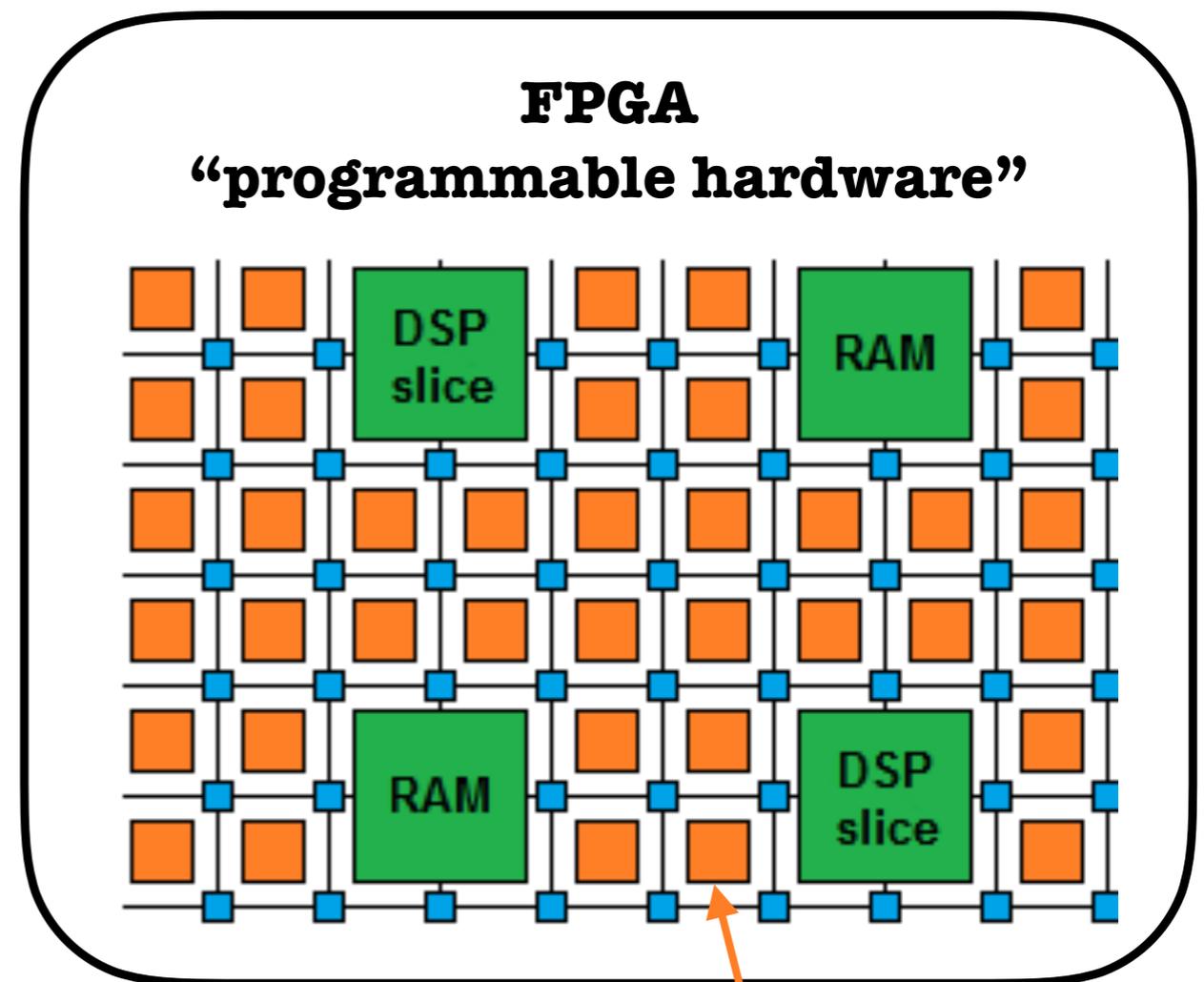
Industry Trends

- Industry is moving toward custom hardware and **FPGAs** to quickly apply ML algorithms
- Already being used as backend accelerators for **Bing** web searches, **Siri** queries, and more...

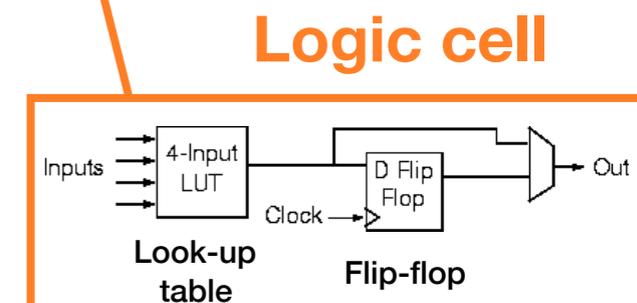


Field-Programmable Gate Array

- **Flexible:** re-programmable interconnects between **configurable logic blocks** and **embedded components**
 - LUTs (logic),
Flip-Flops (registers),
DSPs (arithmetic),
Block RAMs (memory)
- **High Throughput:** O(100) optical transceivers running at O(15) Gbs
- **Massively parallel**
- **Low power** (relative to CPU/GPU)



Typical modern FPGA
(Kintex UltraScale):
1.3 M FFs
700k LUTs
5500 DSPs
2200 BRAMs



Training

Training/Compression

Example: <https://github.com/hls-fpga-machine-learning/keras-training/>

K Keras

(Some useful stuff for compression too)

Export Model

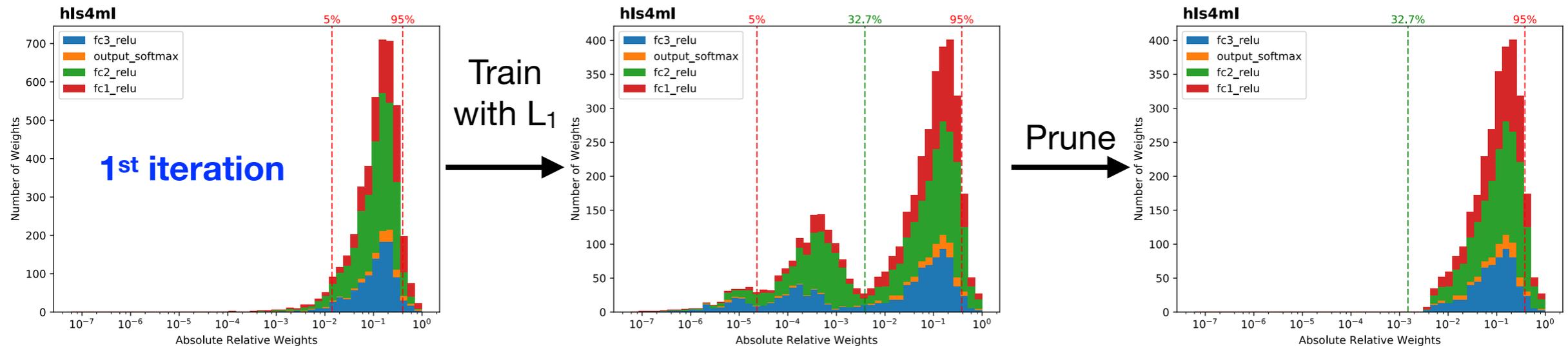
```
json_string = keras_model.to_json()
keras_model.save_weights('my_model_weights.h5')
```

```
{
  "backend": "tensorflow",
  "class_name": "Model",
  "config": {
    "input_layers": [
      [
        "input_1",
        0,
        0
      ]
    ],
    "layers": [
      {
        "class_name": "InputLayer",
        "config": {
          "batch_input_shape": [
            null,
            16
          ],
          "dtype": "float32",
          "name": "input_1",
          "sparse": false
        },
        "inbound_nodes": [],
        "name": "input_1"
      },
      {
        "class_name": "Dense",
        "config": {
          "activation": "relu",
          "activity_regularizer": null,
          "bias_constraint": null,
```

json



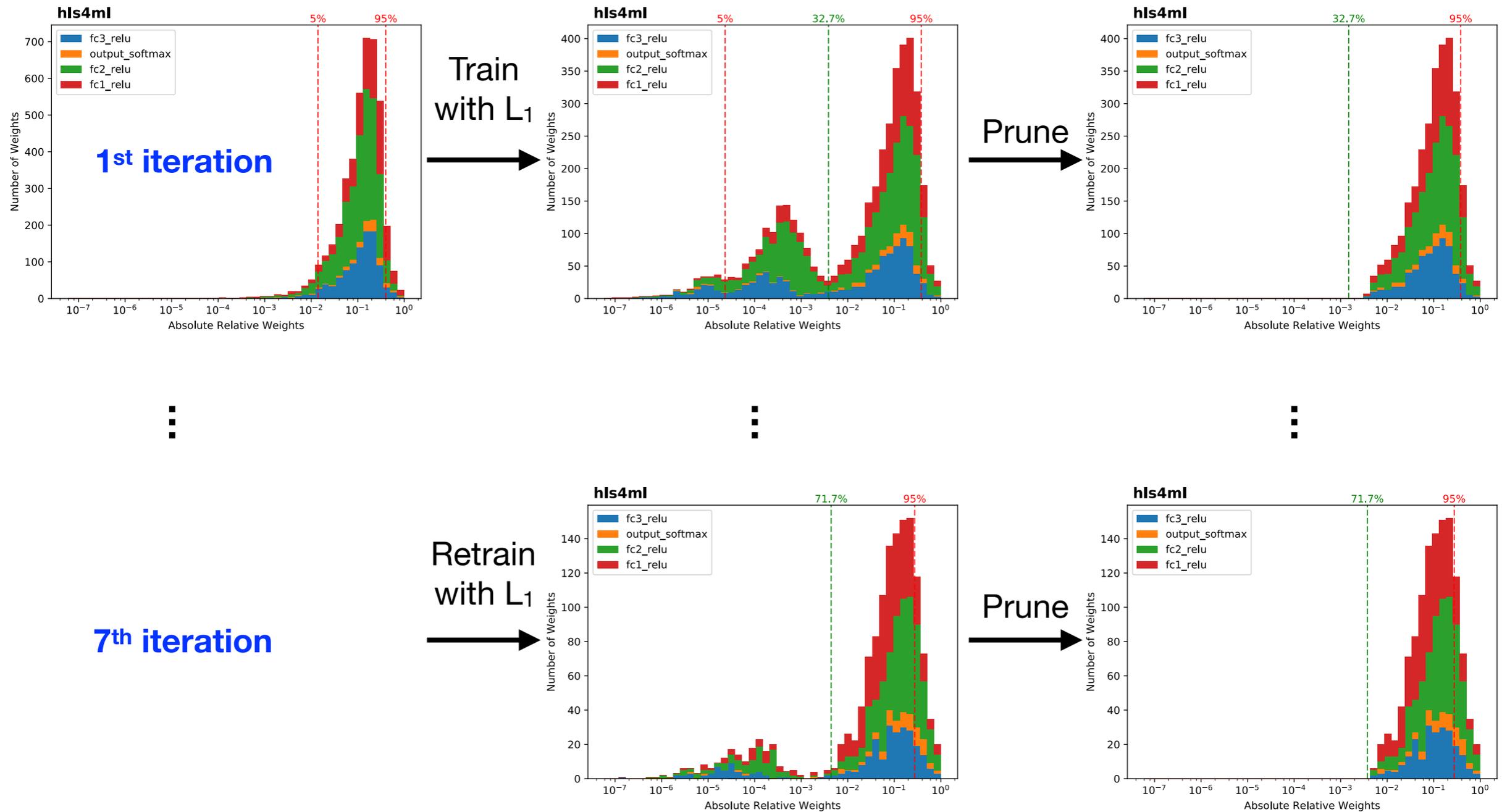
Training for Compression



- Many possible schemes for compression
- Simple, iterative version:
 - Train with L₁ regularization (down-weights unimportant synapses)
$$L_{\lambda}(\mathbf{w}) = L(\mathbf{w}) + \lambda \|\mathbf{w}\|_1 \quad \|\mathbf{w}\|_1 = \sum_i |w_i|$$
 - Remove X% of weights and retrain
 - Repeat



Training for Compression



70% reduction with no loss in performance



Other Compression Schemes

Louizos et al. 2017

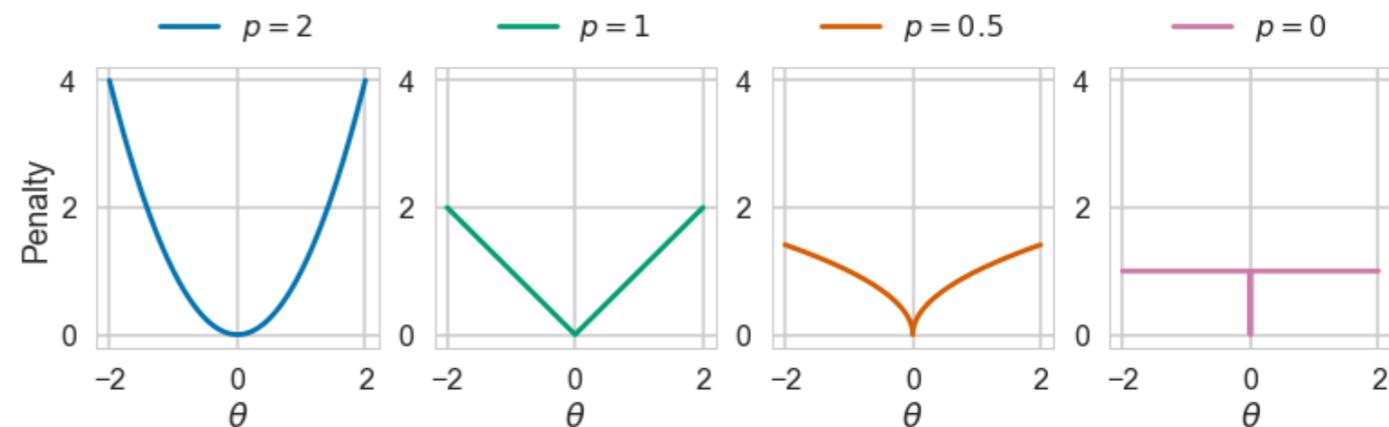
[arXiv:1712.01312](https://arxiv.org/abs/1712.01312)

- Train with L_p ($0 \leq p < 1$) regularization to promote sparsity (though difficult to optimize)

$$\|w\|_p = \left(\sum_i |w_i|^p \right)^{1/p}$$

- as $p \rightarrow 0$, $L_p \rightarrow L_0$

- “Optimal brain damage:” use second derivatives of loss function to rank **parameter saliency** (rather than using parameter magnitude)



LeCun et al. 1989

[NIPS 250](https://paperswithcode.com/sota/weight-sharing-on-natural-image-classification/1989)

- Weight-sharing using k-means clustering to identify weights to share
- Huffman coding (optimal prefix)

Han et al. 2015

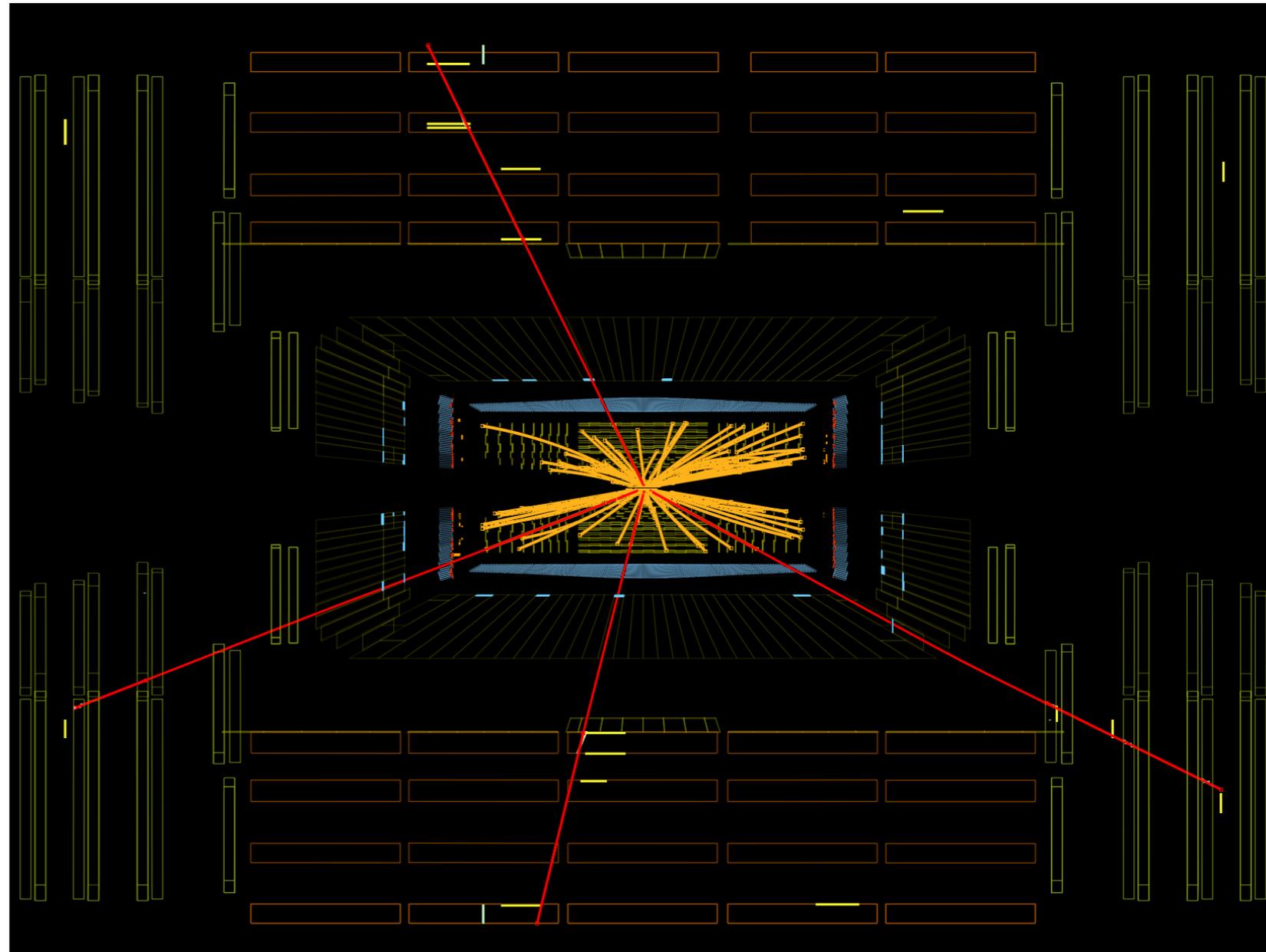
[arXiv:1510.00149](https://arxiv.org/abs/1510.00149)



CMS L1 Muon Trigger Application

[ACAT 2017 BDT L1T](#)

- Current **BDT** for CMS Level-1 muon p_T assignment based on deflection angles ($\Delta\Phi$, $\Delta\theta$) and other variables
 - Implemented using a 1 GB pre-computed LUT that stores the BDT output for every the possible input (compressed to 30-bits)
- CMS studying NN implemented with `hls4ml`, which allows additional inputs (e.g. 80 variables with 18 bits each = 1440 bits!) and improved p_T resolution



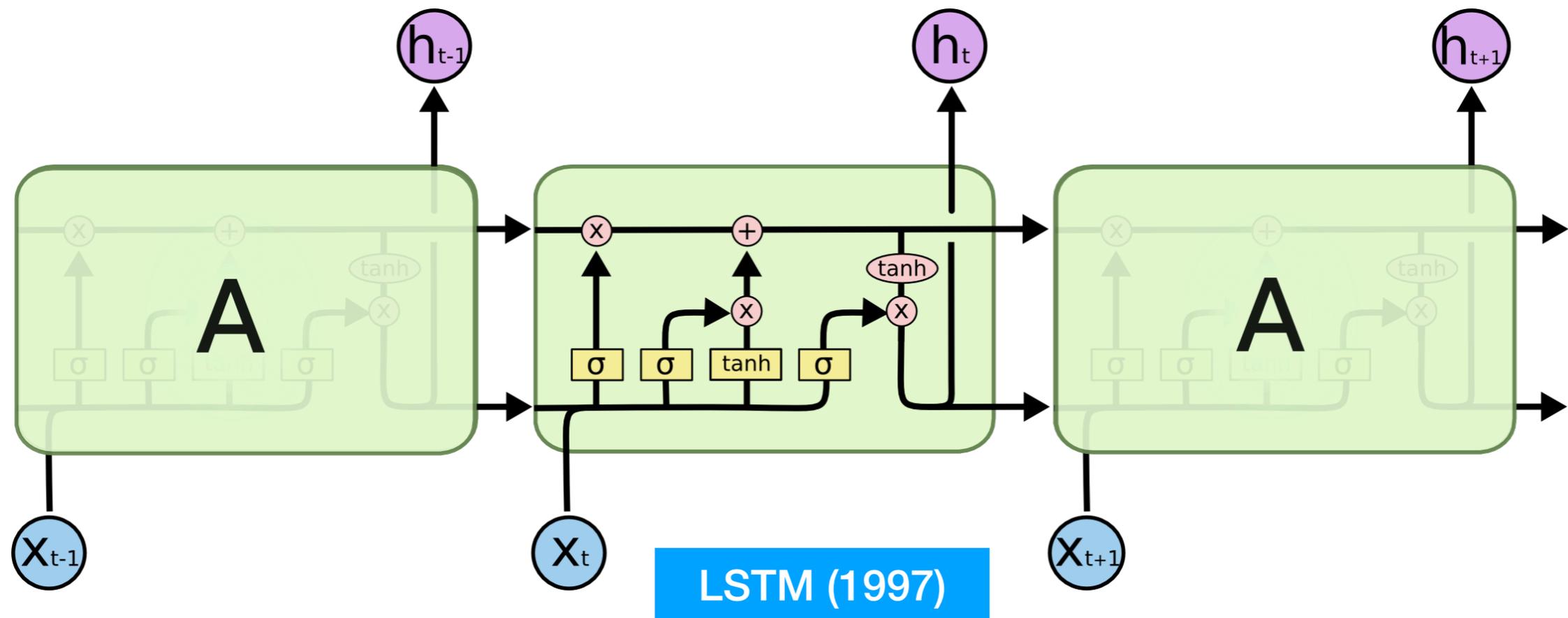
CNNs on FPGAs

- NIPS 2017 Demo: https://docs.google.com/presentation/d/1mTqsm5TronnB8MFD6yyq3CSPCV4bfck_aQ-ericM9cQ/edit#slide=id.p3
- Snowflake: [arXiv:1708.02579](https://arxiv.org/abs/1708.02579)
- DNNWeaver: <http://act-lab.org/artifacts/dnnweaver/>
- fpgaConvNet: <http://cas.ee.ic.ac.uk/people/sv1310/fpgaConvNet.html>
- Caffeine



Recurrent Neural Network

- Main task is language processing, time sequence prediction
- **LSTM layers** allow learned information to persist; network can learn long-term dependences in sequences



HLS Study Details

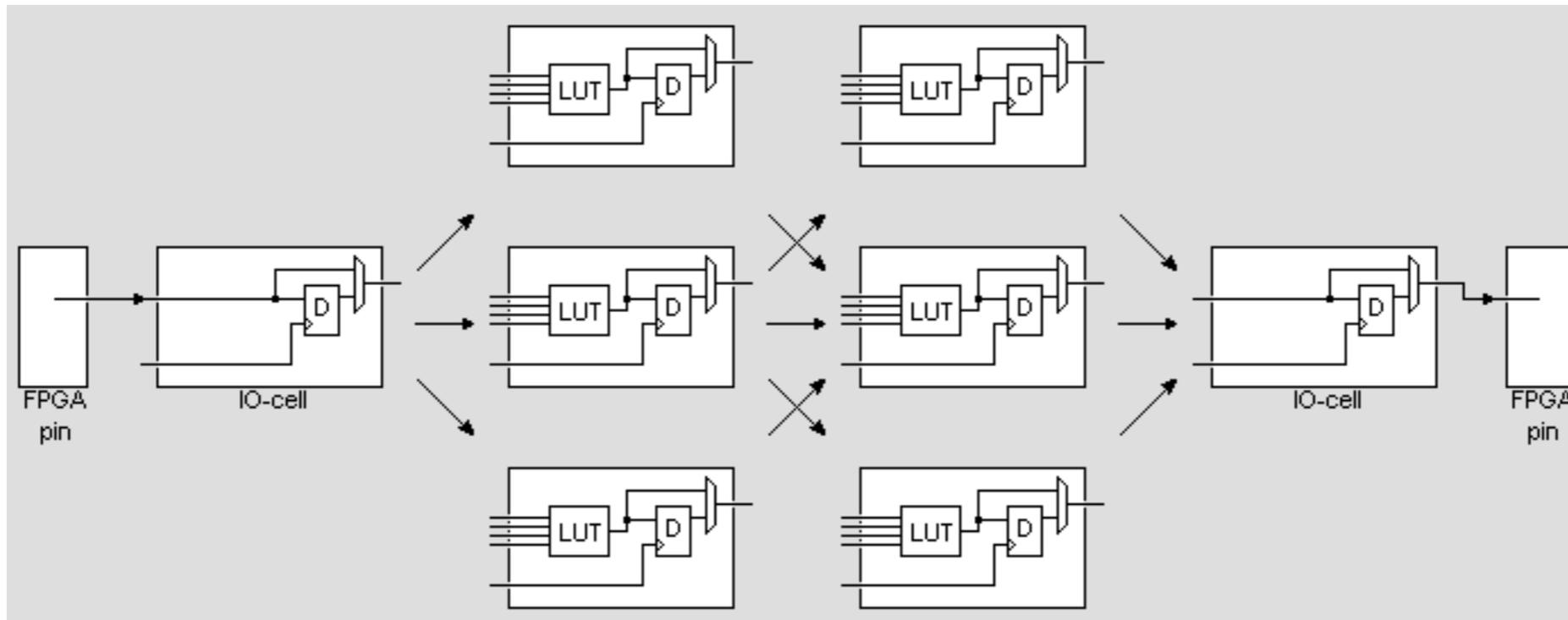
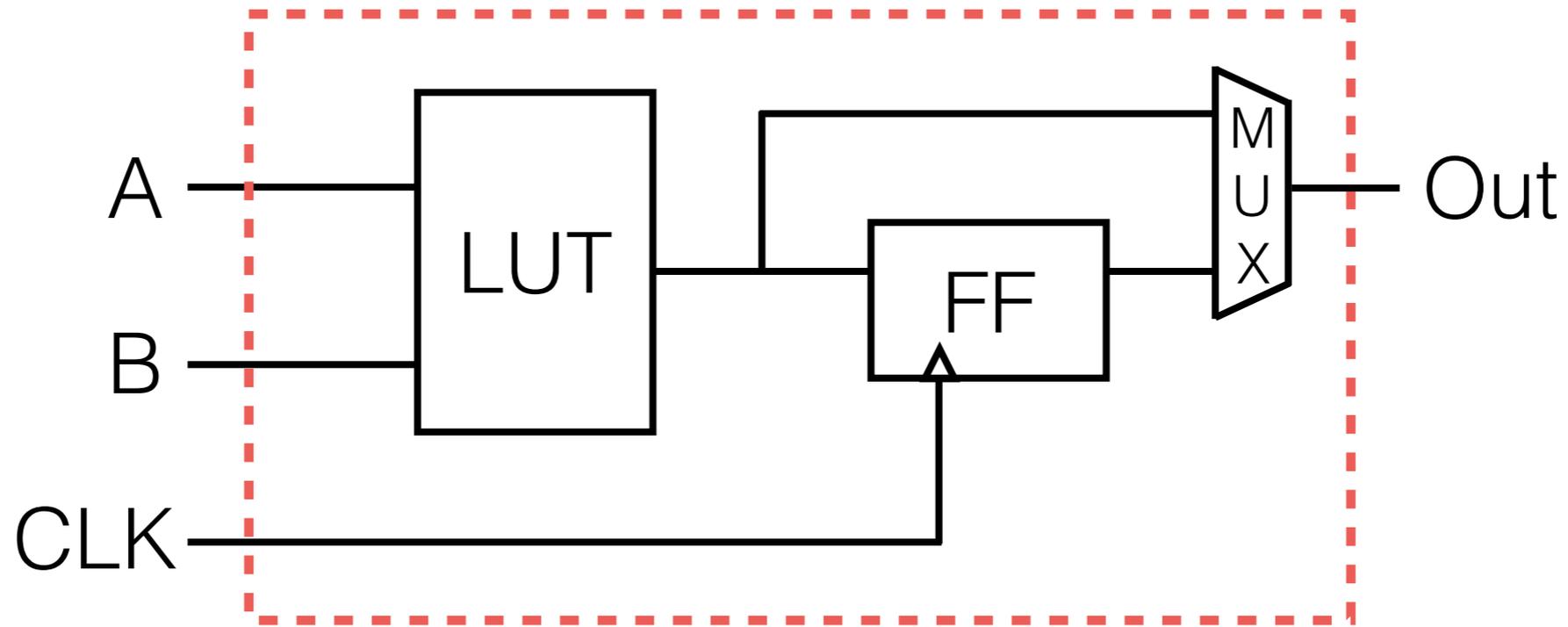
- Xilinx Vivado 2017.2
- Clock frequency: 200 MHz
- FPGA: Xilinx Kintex Ultrascale (XCKU115-FLVB2104)

- A note on inputs:
 - We assume network inputs have already been computed
 - Not a good assumption in the jet substructure case with “expert” features
 - Convolutional and recurrent networks which more naturally operate on “raw” features are in development

- Note: resource usage comes from HLS estimates
 - Discussion on differences w.r.t. implementation later



Logic Cell



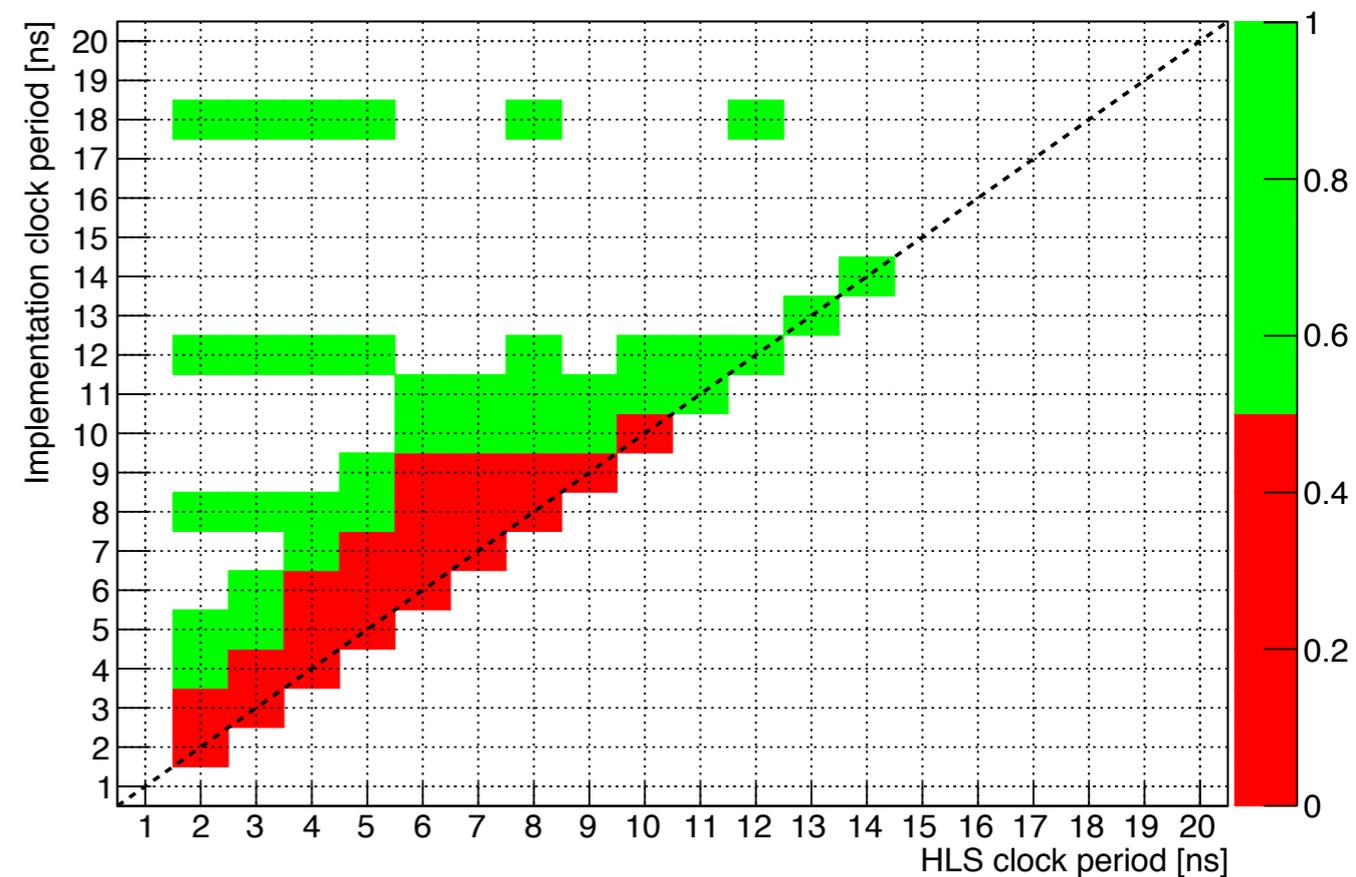
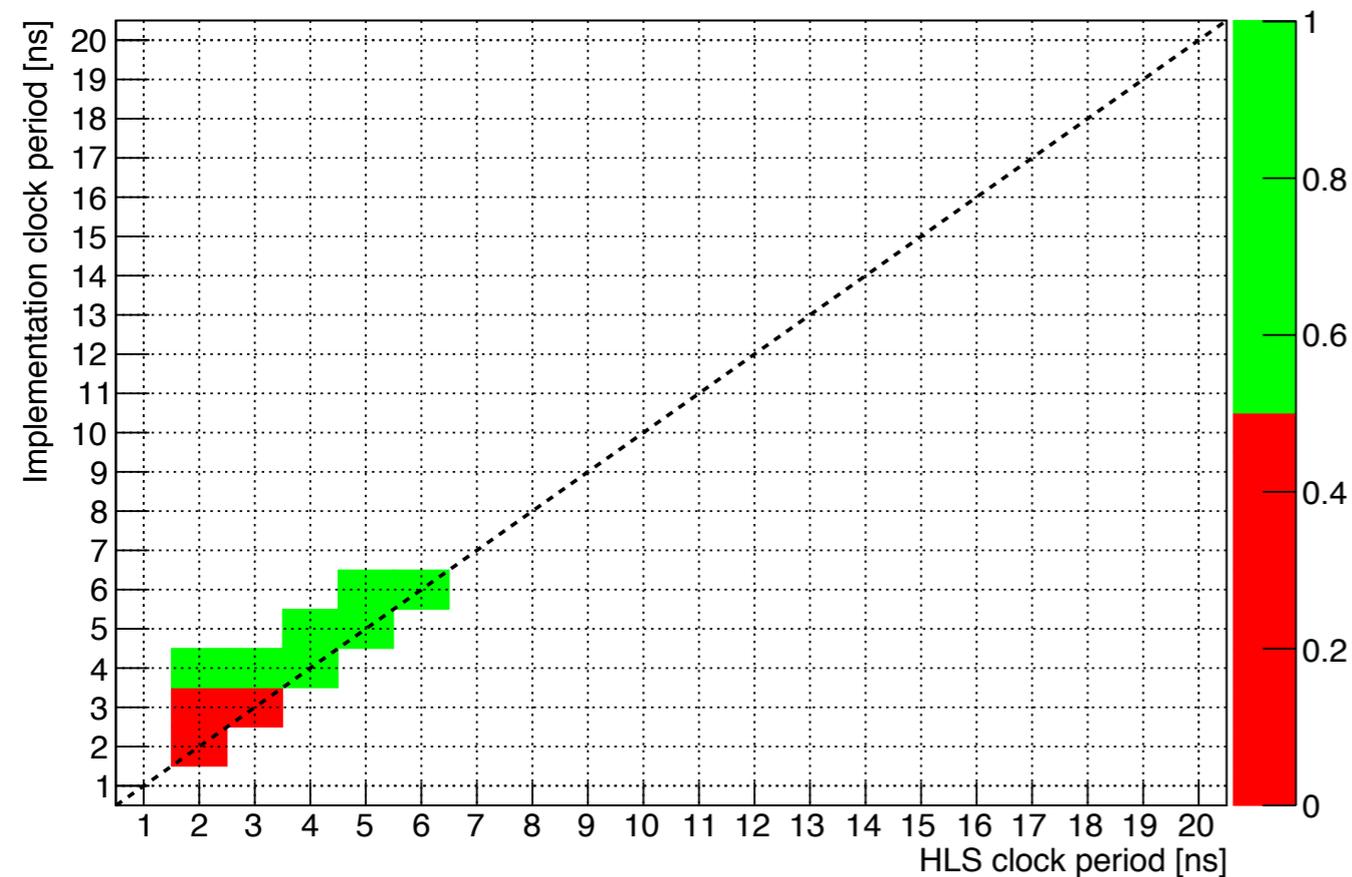
Meeting HLS Target Timing

v2016.4

v2017.2

Timing result, 3 hidden pruned, <16,8>, reuse=2

Timing result, 3 hidden pruned, <16,8>, reuse=2



- Implementation timing meets HLS target in v2017.2 for clock periods ≥ 11 ns
- Implementation timing meets HLS target in v2016.4 for clock periods ≥ 4 ns



Kintex® UltraScale™ FPGAs

	Device Name	KU025 ⁽¹⁾	KU035	KU040	KU060	KU085	KU095	KU115
Logic Resources	System Logic Cells (K)	318	444	530	726	1,088	1,176	1,451
	CLB Flip-Flops	290,880	406,256	484,800	663,360	995,040	1,075,200	1,326,720
	CLB LUTs	145,440	203,128	242,400	331,680	497,520	537,600	663,360
Memory Resources	Maximum Distributed RAM (Kb)	4,230	5,908	7,050	9,180	13,770	4,800	18,360
	Block RAM/FIFO w/ECC (36Kb each)	360	540	600	1,080	1,620	1,680	2,160
	Block RAM/FIFO (18Kb each)	720	1,080	1,200	2,160	3,240	3,360	4,320
	Total Block RAM (Mb)	12.7	19.0	21.1	38.0	56.9	59.1	75.9
Clock Resources	CMT (1 MMCM, 2 PLLs)	6	10	10	12	22	16	24
	I/O DLL	24	40	40	48	56	64	64
I/O Resources	Maximum Single-Ended HP I/Os	208	416	416	520	572	650	676
	Maximum Differential HP I/O Pairs	96	192	192	240	264	288	312
	Maximum Single-Ended HR I/Os	104	104	104	104	104	52	156
	Maximum Differential HR I/O Pairs	48	48	48	48	56	24	72
Integrated IP Resources	DSP Slices	1,152	1,700	1,920	2,760	4,100	768	5,520
	System Monitor	1	1	1	1	2	1	2
	PCIe® Gen1/2/3	1	2	3	3	4	4	6
	Interlaken	0	0	0	0	0	2	0
	100G Ethernet	0	0	0	0	0	2	0
	16.3Gb/s Transceivers (GTH/GTY)	12	16	20	32	56	64 ⁽²⁾	64
Speed Grades	Commercial	-1	-1	-1	-1	-1	-1	-1
	Extended	-2	-2 -3	-2 -3	-2 -3	-2 -3	-2	-2 -3
	Industrial	-1 -2	-1 -1L -2	-1 -1L -2	-1 -1L -2	-1 -1L -2	-1 -2	-1 -1L -2
Footprint Compatible with Virtex® UltraScale Devices	Package Footprint ^(3, 4, 5, 6)	Package Dimensions (mm)	HR I/O, HP I/O, GTH/GTY					
	A784 ⁽⁷⁾	23x23 ⁽⁸⁾	104, 364, 8	104, 364, 8				
	A676 ⁽⁷⁾	27x27	104, 208, 16	104, 208, 16				
	A900 ⁽⁷⁾	31x31	104, 364, 16	104, 364, 16				
	A1156	35x35	104, 208, 12	104, 416, 16	104, 416, 20	104, 416, 28	52, 468, 28	
	A1517	40x40				104, 520, 32	104, 520, 48	104, 520, 48
	C1517	40x40					52, 468, 40	
	D1517	40x40						104, 234, 64
	B1760	42.5x42.5					104, 572, 44	52, 650, 48
	A2104	47.5x47.5						156, 676, 52
	B2104	47.5x47.5						52, 650, 64
	D1924	45x45						156, 676, 52
	F1924	45x45					104, 520, 56	104, 624, 64

Notes:

1. Certain advanced configuration features are not supported in the KU025. Refer to the Configuring FPGAs section in DS890, *UltraScale Architecture and Product Overview*.
2. GTY transceivers in KU095 devices support data rates up to 16.3Gb/s.
3. Packages with the same package footprint designator, e.g., A2104, are footprint compatible with all other UltraScale devices with the same sequence. See the [migration table](#) for details on inter-family migration.
4. Maximum achievable performance is device and package dependent; consult the associated data sheet for details.
5. For full part number details, see the Ordering Information section in DS890, *UltraScale Architecture and Product Overview*.
6. See UG575, *UltraScale Architecture Packaging and Pinouts User Guide* for more information.
7. GTH transceivers in A784, A676, and A900 packages support data rates up to 12.5Gb/s.
8. 0.8mm ball pitch. All other packages listed 1mm ball pitch.

