

# Towards FPGA-deployable real-time boosted top jet identification

Tianjia Du, Ben Rosser, Timothy Hoffman, Ezra Santos, David Miller  
University of Chicago  
November 21, 2024



# Outline

## Introduction

- The case for machine learning on FPGAs
- Adaptive Intelligence Engines (AI Engines)
- Context for our work: the Next Generation Trigger Project

## Physics applications

- In the ATLAS trigger system
- Top tagging

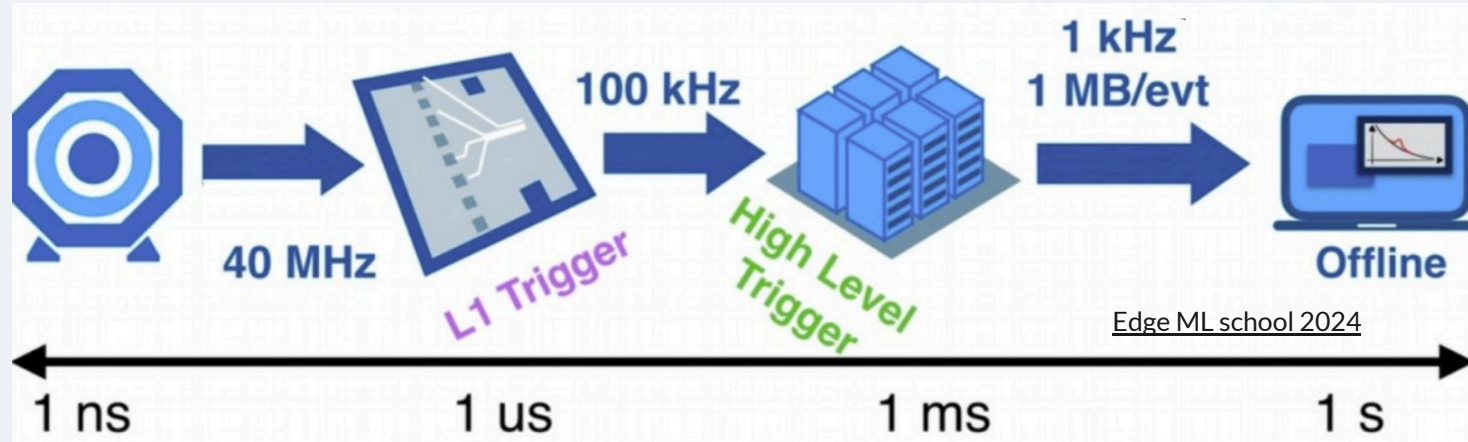
## Workflows: what supports AI Engines?

- Hls4ml
- Vitis-AI

# Intro: the case for machine learning on FPGA

Current collisions at the Large Hadron Collider (LHC) occur every 25 ns  $\Rightarrow$  40 MHz data rate

- Detectors such as ATLAS and CMS at the LHC use a “trigger” to select events for storage



The case for running trigger algorithms on FPGAs

- Tight limits on latency and computing resources
- FPGAs can be reprogrammed  $\rightarrow$  algorithms can be updated



# Intro: Adaptive Intelligence Engines

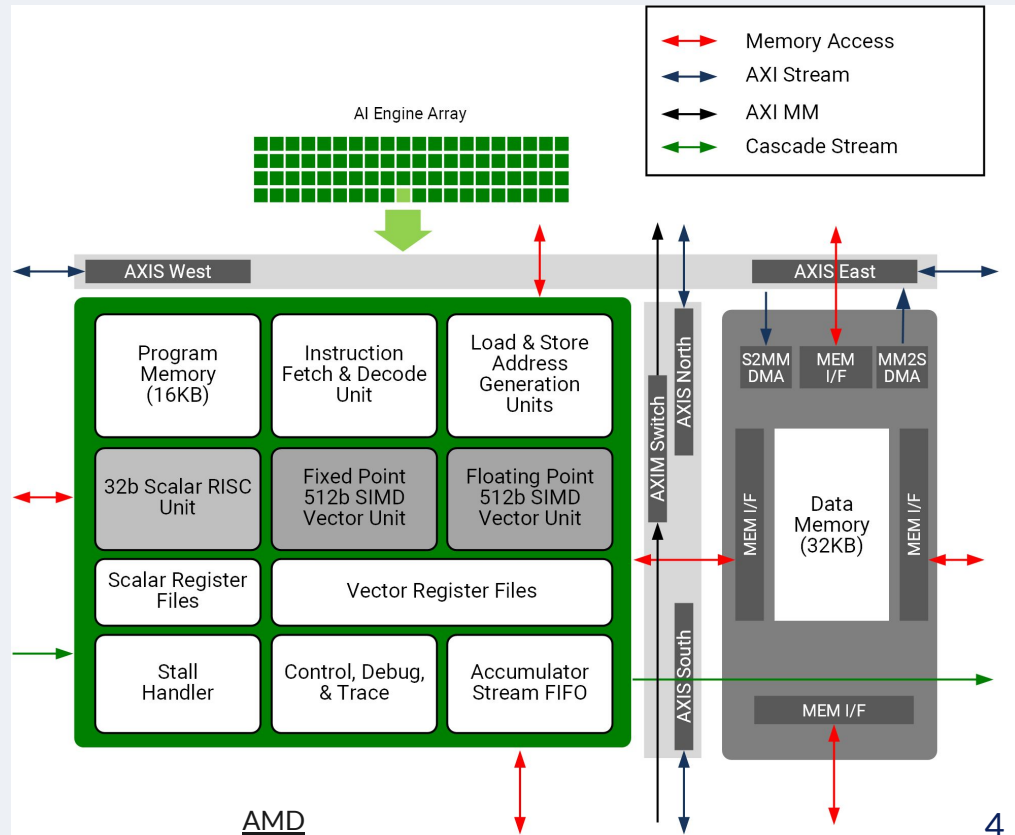
AI Engines are dedicated resources on newer Xilinx FPGAs

- A single FPGA device has an array of many AI Engines
- Each AI Engine consists of
  - Memory: program, data memory
  - Processor units
  - In/out: AXI stream

The advantage of AI engines is parallelization:

1. Instruction unit allows multiple operations to be done in the same clock cycle
2. Vector processors can compute many elements simultaneously
3. Up to 400 tiles in the FPGA device can be run simultaneously

...but they're difficult to program



# Related: the Next Generation Trigger project

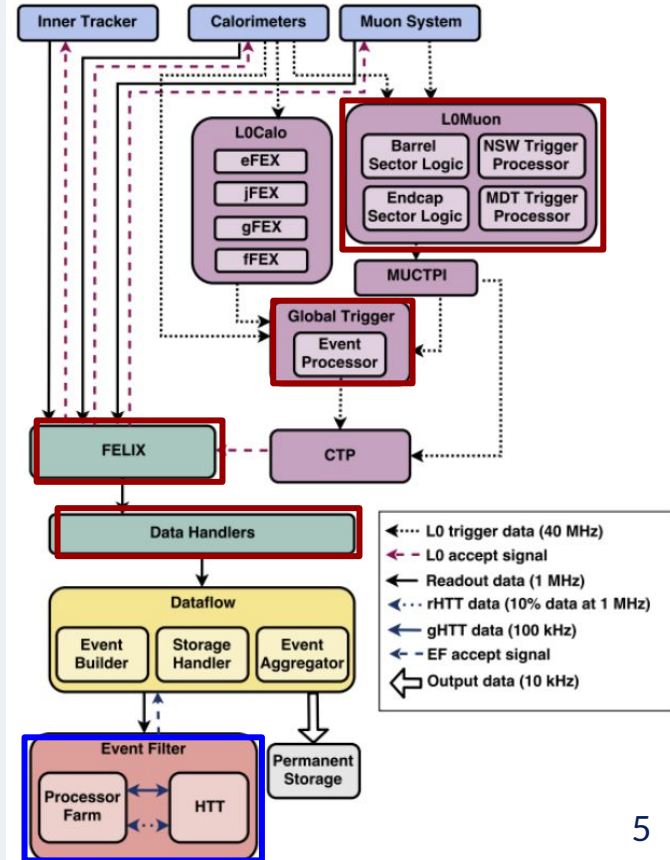


- The NextGen Trigger project is an R&D effort for developing “triggers” – data processing and event picking algorithms– for the ATLAS and CMS experiments at the LHC

The ATLAS experiment trigger and data acquisition system:

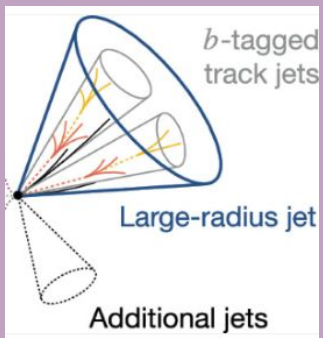
- **Hardware trigger** must reduce data rate from 40 MHz→1 MHz in the HL-LHC (10x larger than the current system)
- Events selected by the hardware trigger must then pass a **software trigger** before being saved for offline data analysis

ATLAS trigger & data acquisition upgrade for HL-LHC

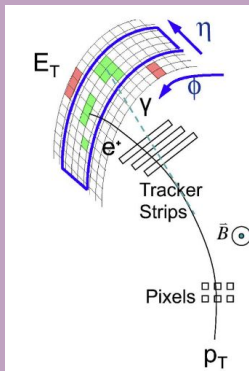


# Possible physics applications in ATLAS trigger system

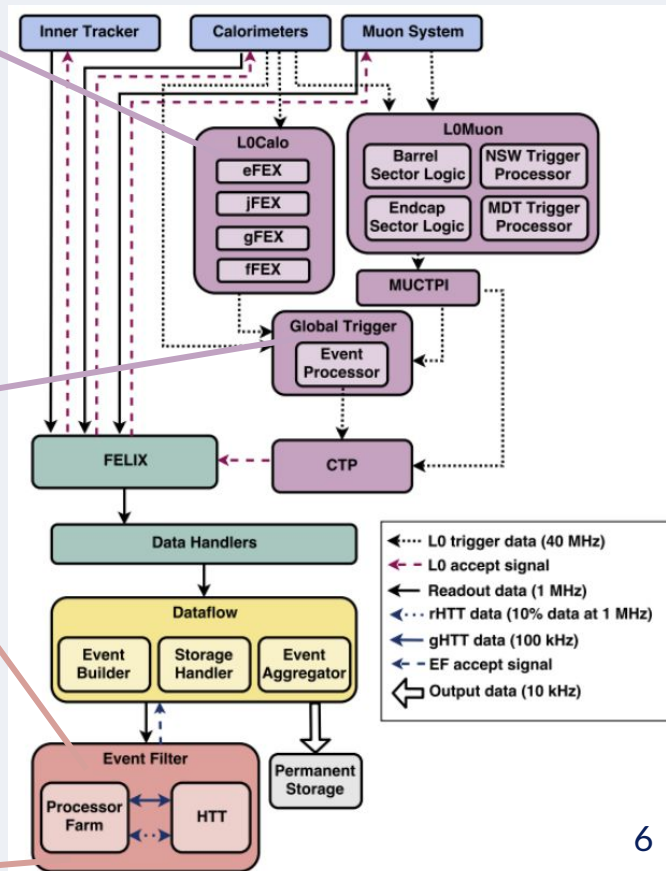
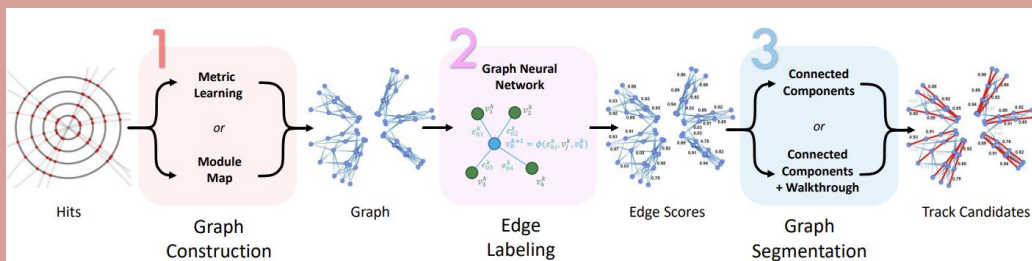
## CNN for jet finding



## BDT for e/gamma identification



## GNN for track reconstruction can be “hardware accelerated” by using FPGAs



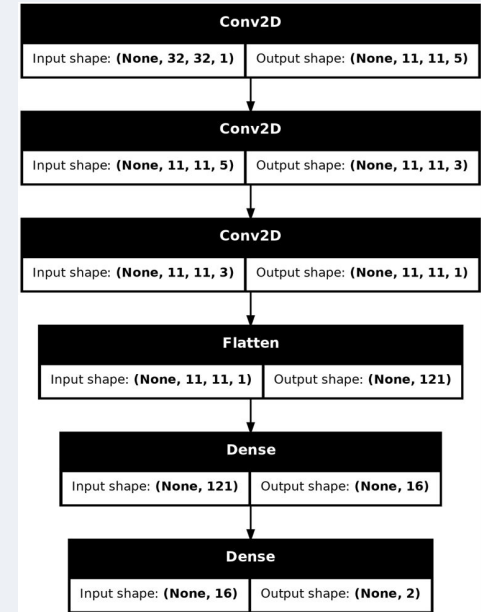
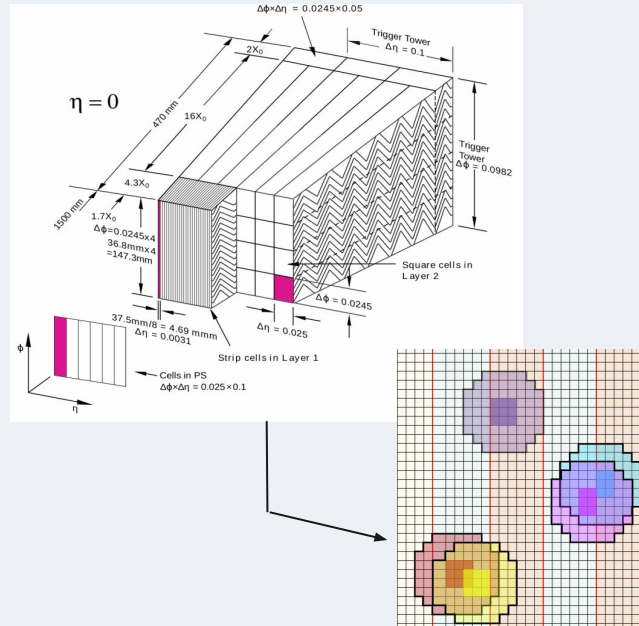
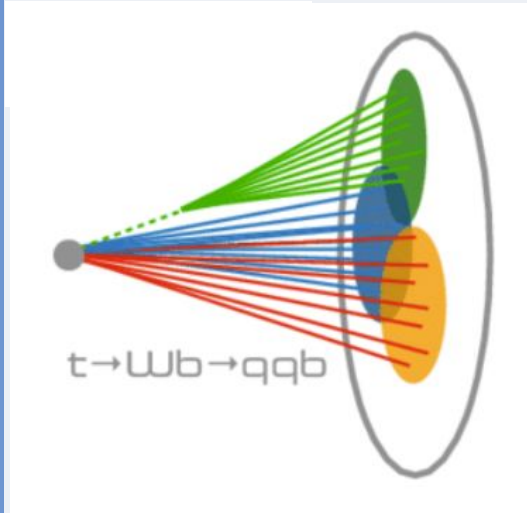
# Physics applications for top tagging

- Build a CNN inspired by [Pulling out all the tops](#) to identify jets from moderately boosted top quarks

1. Physics process of interest:  
 $t \rightarrow Wb \rightarrow qqb$

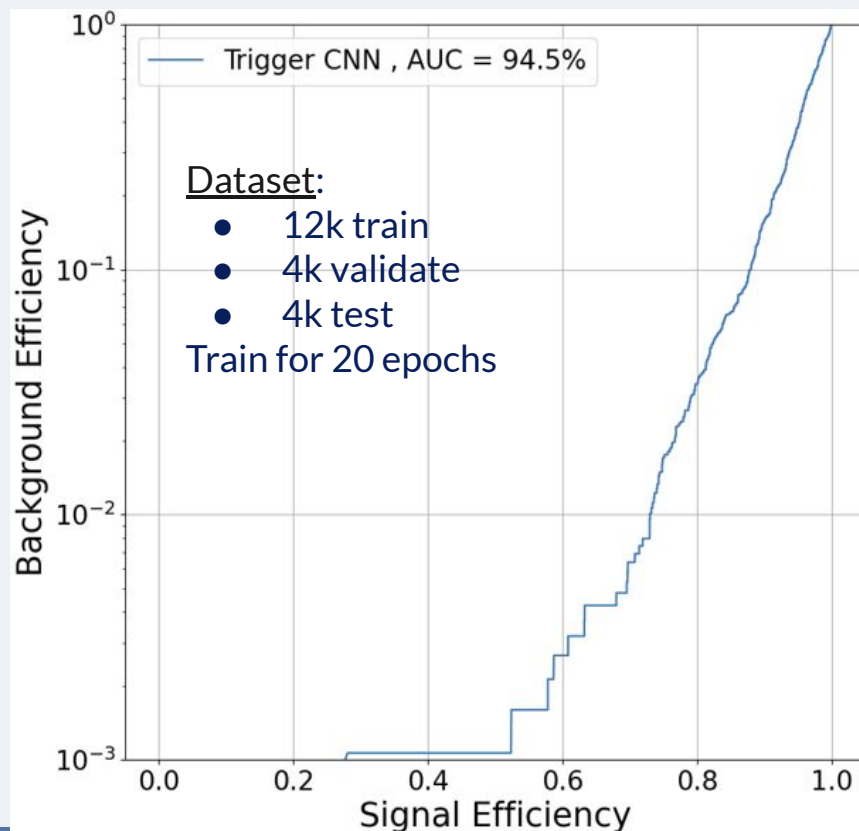
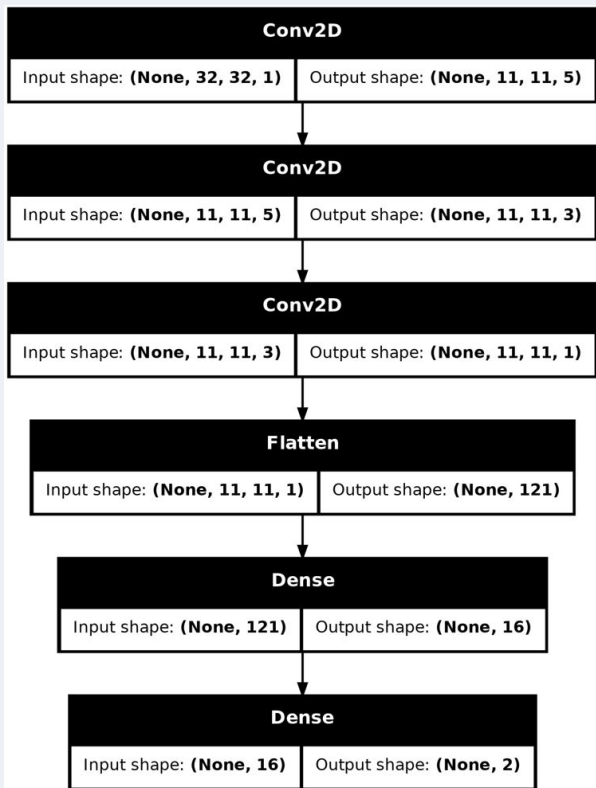
2. Energy deposited in calorimeter appear as “jet images”

3. Trigger CNN identifies jets resulting from  $t \rightarrow Wb \rightarrow qqb$



# Physics case: top tagging

- Build a CNN inspired by [Pulling out all the tops](#) to identify jets from moderately boosted top quarks



# Workflows for FPGA

We compare two workflows

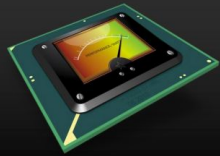
1. **hls4ml**

- no support for AI Engines at the moment, but people are interested in developing it!
- supports custom precision for weights and activations

2. **Vitis-AI (Xilinx/AMD)**

- compatible with “AI Engines”: computational units with scalar and vector processors, and local memory
- what is the resource usage and latency advantage from using AI Engines?

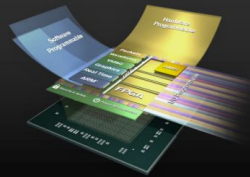
Source: [AMD](https://www.amd.com/en/press-releases/2020/04/04-2020-04-20-01)



## FPGAs

From high-bandwidth connectivity to massive compute engines

AMD SPARTAN ARTIX KINTEX VIRTEx



## SoCs

Multi-processing subsystem with Arm® cores and integrated FPGA logic

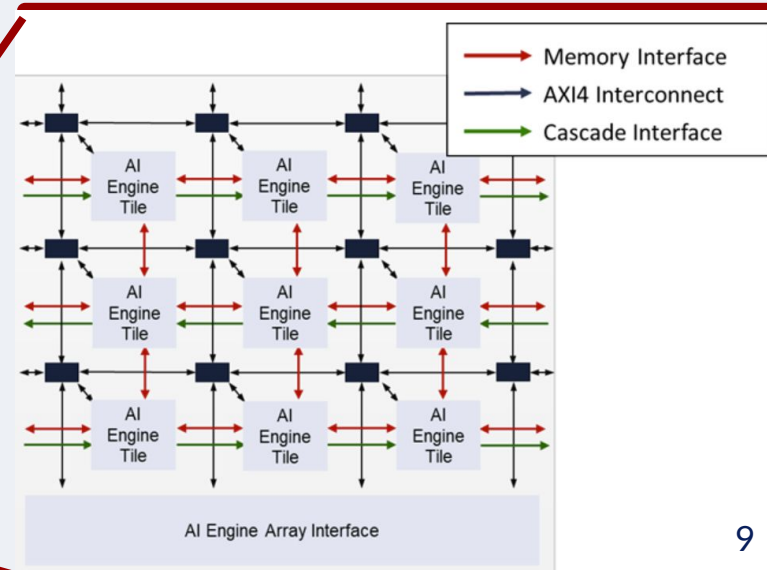
AMD ZYNQ



## Adaptive SoCs

Adaptive Compute Acceleration Platforms for any application, any developer

AMD VERSAL



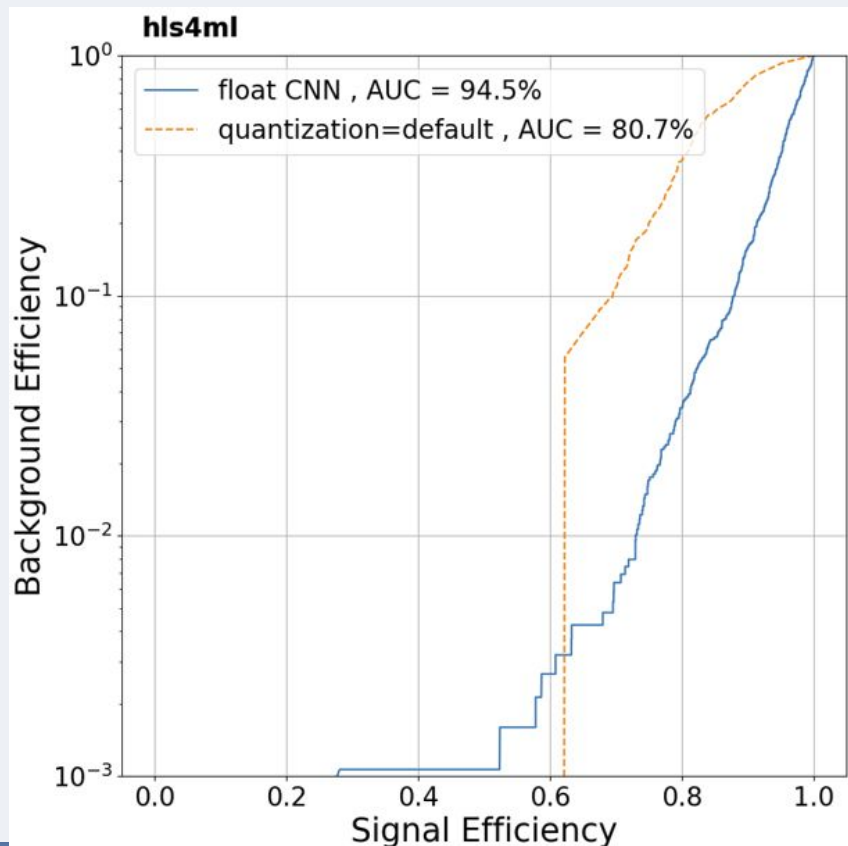
# hls4ml for the top-tagging CNN: quantization

The hls4ml workflow provides several tools for optimizing a model

- Quantization strategies:
  - Default (quantized post training)

Default quantization config snapshot

```
conv2d
  Trace:          False
  Precision
    result:       fixed<16,6>
    weight:       fixed<16,6>
    bias:         fixed<16,6>
    accum:        fixed<16,6>
  ReuseFactor:   1
  ParallelizationFactor: 1
  ConvImplementation: LineBuffer
conv2d_relu
  Trace:          False
  Precision
    result:       fixed<16,6>
    table:        fixed<18,8>
  ReuseFactor:   1
```



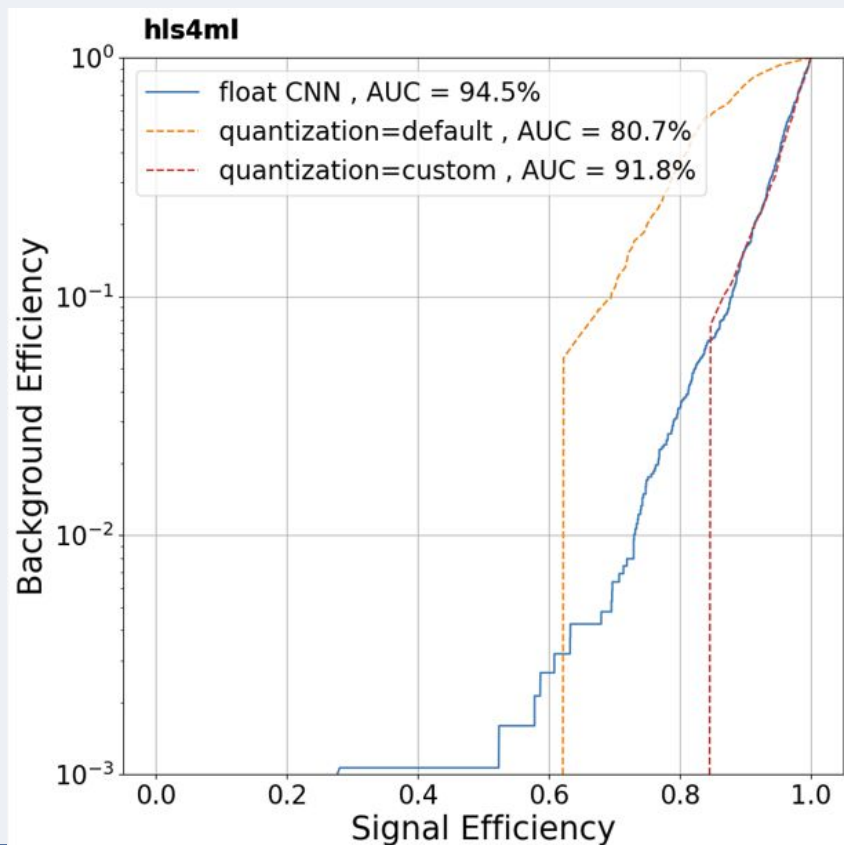
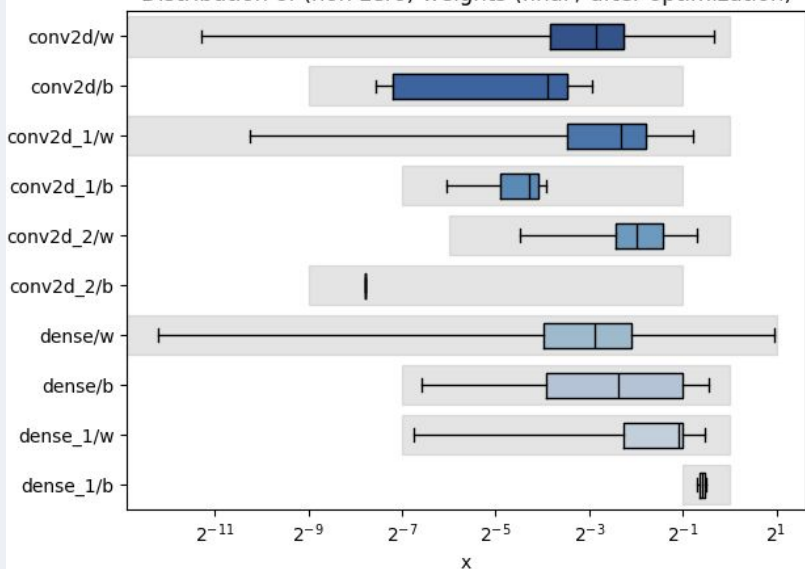
# hls4ml for the top-tagging CNN: quantization

The hls4ml workflow provides several tools for optimizing a model

- Quantization strategies:
  - Default (quantized post training)
  - Custom (quantized post training)

## Customize precision of layer's weights, biases

Distribution of (non-zero) weights (final / after optimization)

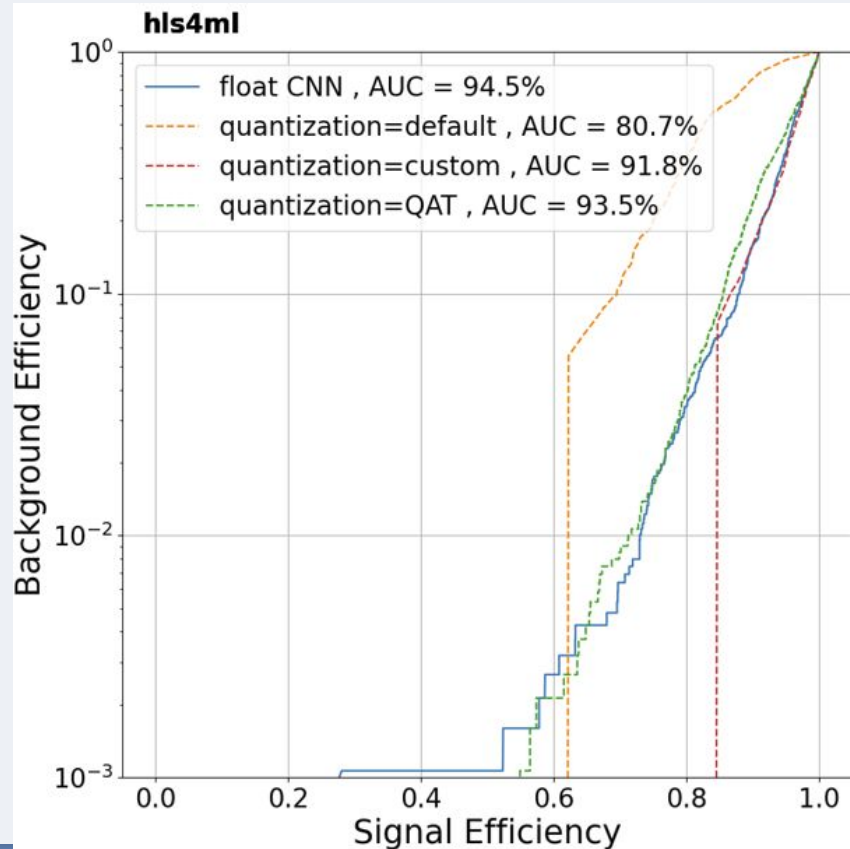
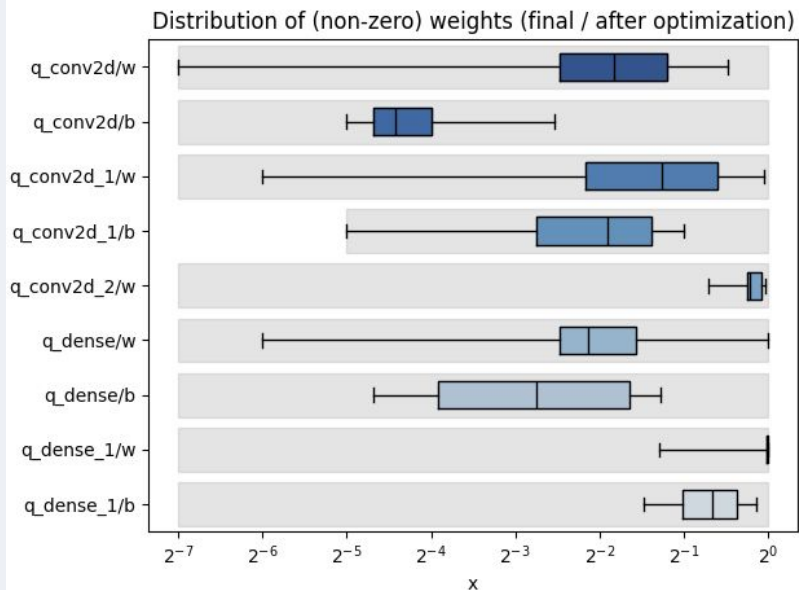


# hls4ml for the top-tagging CNN: quantization

The hls4ml workflow provides several tools for optimizing a model

- Quantization strategies:
  - Default (quantized post training)
  - Custom (quantized post training)
  - Quantization aware training

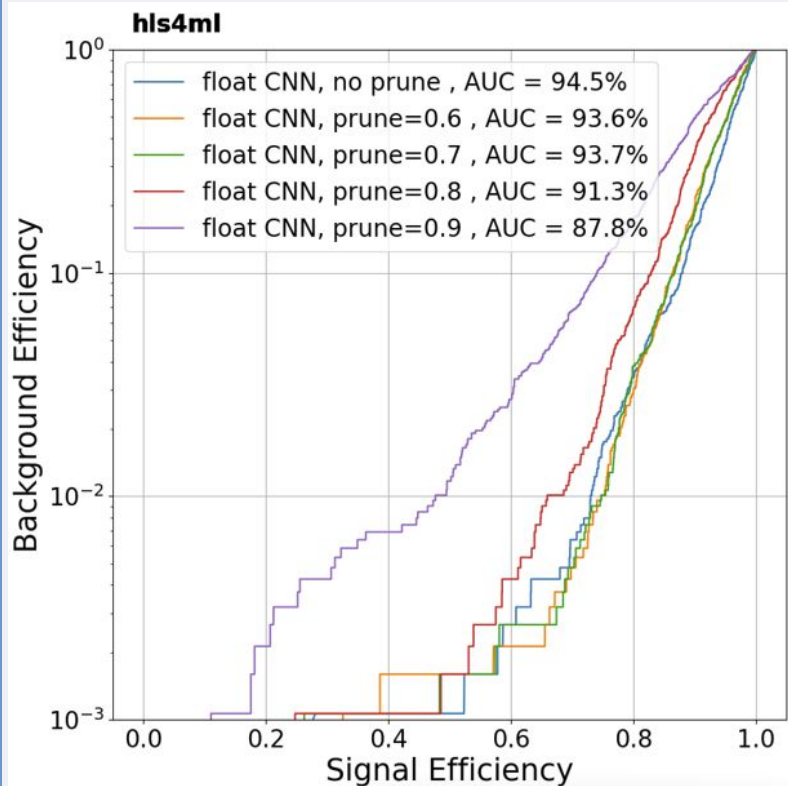
## QAT weights distributions



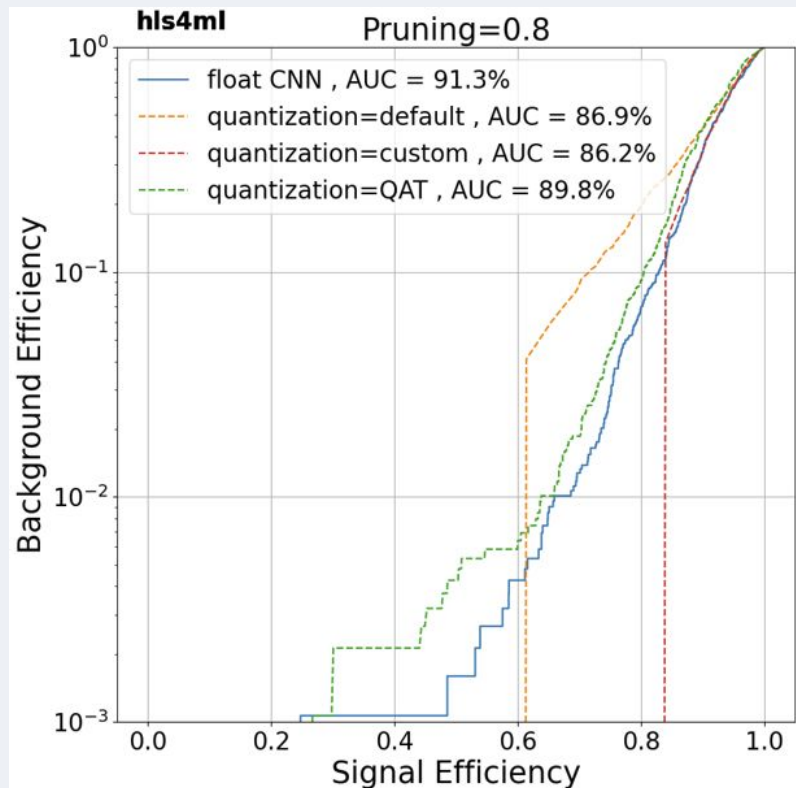
# hls4ml for the top-tagging CNN: pruning

The hls4ml workflow provides several tools for optimizing a model

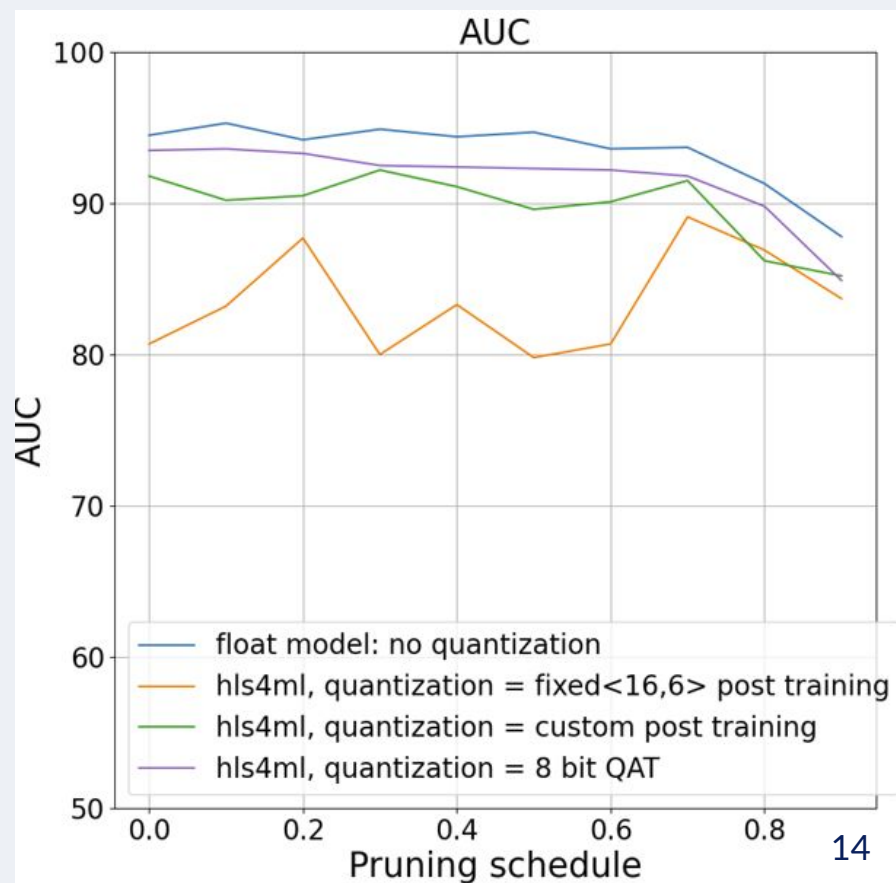
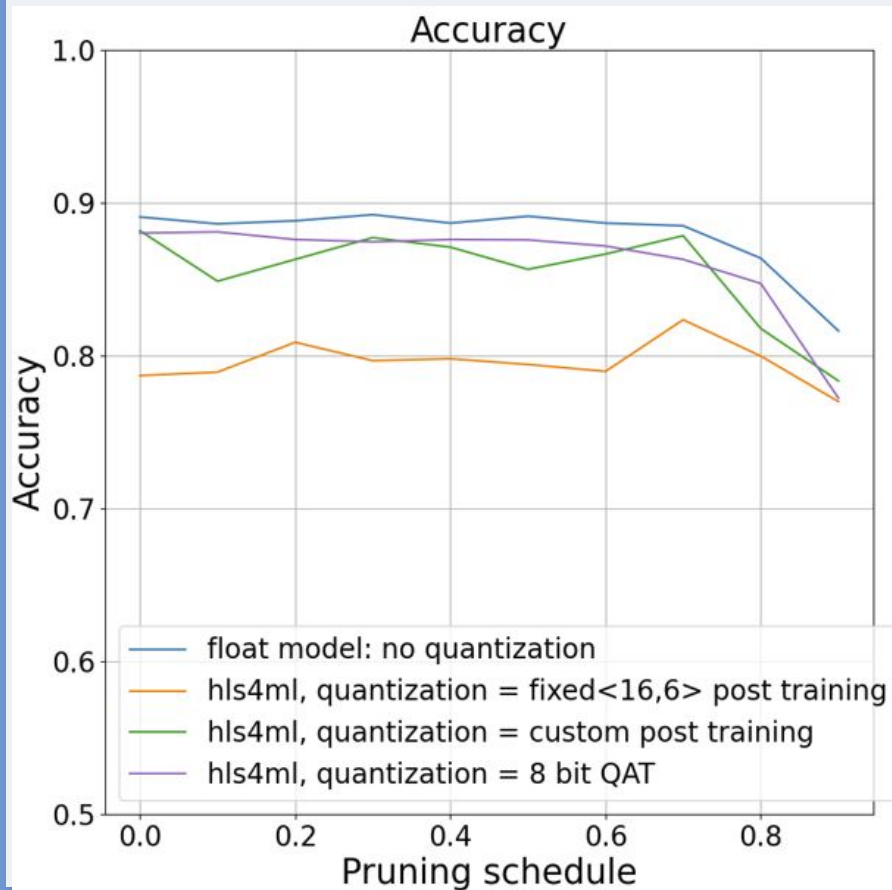
- Pruning is done in-training, and can be done quite aggressively!



... then  
quantize for  
each pruning  
schedule



# hls4ml for the top-tagging CNN: model optimization



# hls4ml for the top-tagging CNN: resource estimates

Key: AI engines

Resource estimates: default quantization, no pruning

FPGA	BRAM_18K (usage %)	DSP (usage %)	FF (usage %)	LUT (usage %)	URAM (usage %)
VU9P	0	27	9	29	0
VP1802	0	13	3	10	0
VP2802	0	13	3	10	0
VC1902	0	95	12	40	0
ZU9EG	0	74	57	157	0

FPGA	Latency min (clock cycles)	Latency max (cycles)	Latency min (us)	Latency max (us)
VU9P	386	389	1.93	1.945
VP1802	387	390	1.935	1.95
VP2802	387	390	1.935	1.95
VC1902	387	390	1.935	1.95
ZU9EG	387	390	1.935	1.95

- Versal premium (VP1802, VP2802) have identical resource utilization and latency estimates!
  - VP2802 has AI engines, but hls4ml currently has no AI engine implementation
- CNN implementation can't fit on the ZU9EG board due to LUT usage

# hls4ml for the top-tagging CNN: resource estimates

Resource usage %: default quantization, prune=0

FPGA	BRAM_18K	DSP	FF	LUT	URAM
VU9P	0	27	9	29	0
VP1802	0	13	3	10	0
VP2802	0	13	3	10	0
VC1902	0	95	12	40	0
ZU9EG	0	74	57	157	0

Resource usage %: default quantization, **prune=0.75**

FPGA	BRAM_18K	DSP	FF	LUT	URAM
VU9P	0	5	7	20	0
VP1802	0	2	2	7	0
VP2802	0	2	2	7	0
VC1902	0	18	11	27	0
ZU9EG	0	14	61	122	0



Resource usage:

- Pruning reduces DSP usage more than any other resource
- Quantization has a smaller impact on resource utilization than pruning

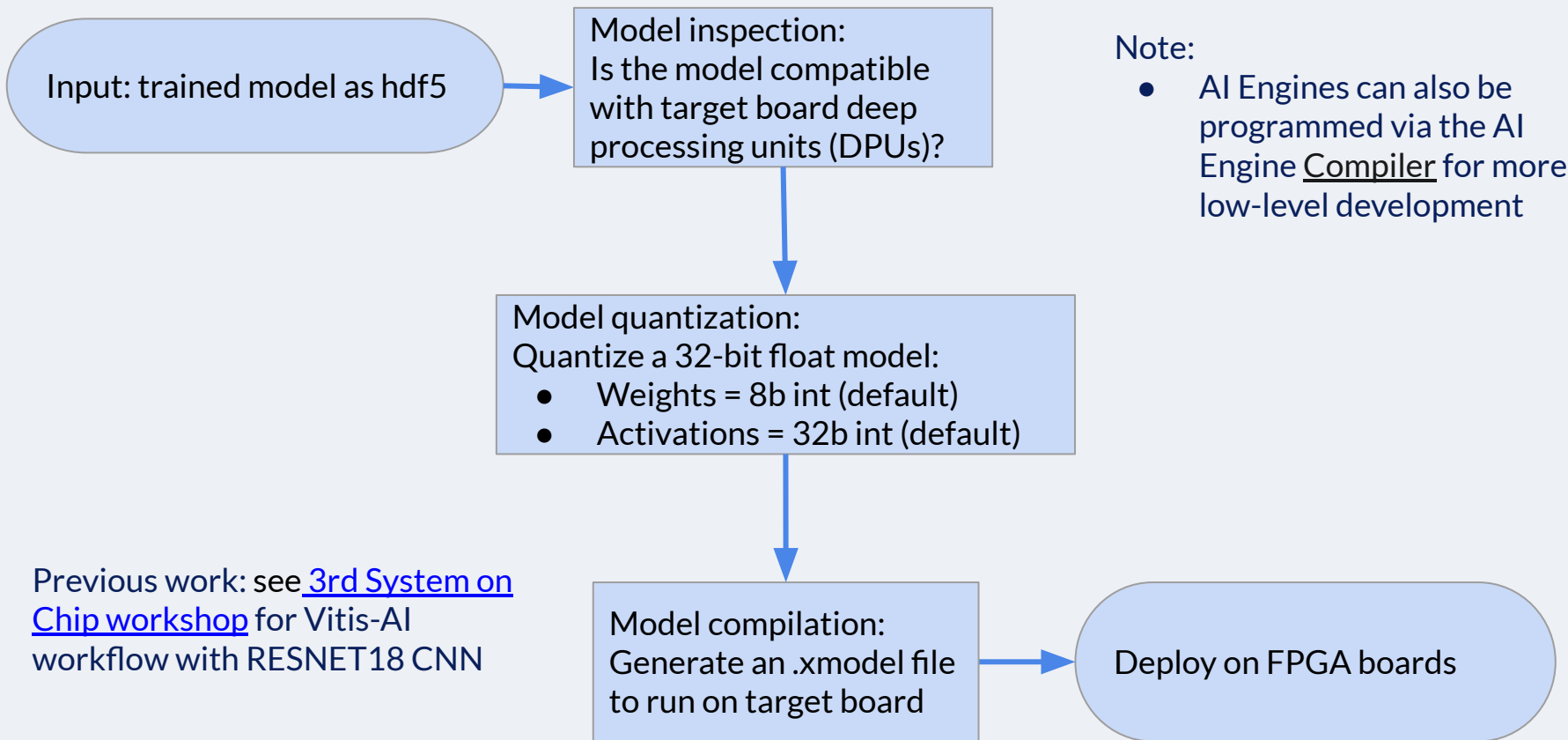
Latency estimates: (see backup for tables)

- Pruning and quantization have a similar impact on latency estimates

Resource usage %: **custom quantization**, prune=0.75

FPGA	BRAM_18K	DSP	FF	LUT	URAM
VU9P	0	4	7	19	0
VP1802	0	2	2	7	0
VP2802	0	2	2	7	0
VC1902	0	17	10	26	0
ZU9EG	0	13	60	121	0

# Vitis-AI for the top-tagging CNN: the Vitis-AI stack



Previous work: see [3rd System on Chip workshop](#) for Vitis-AI workflow with RESNET18 CNN

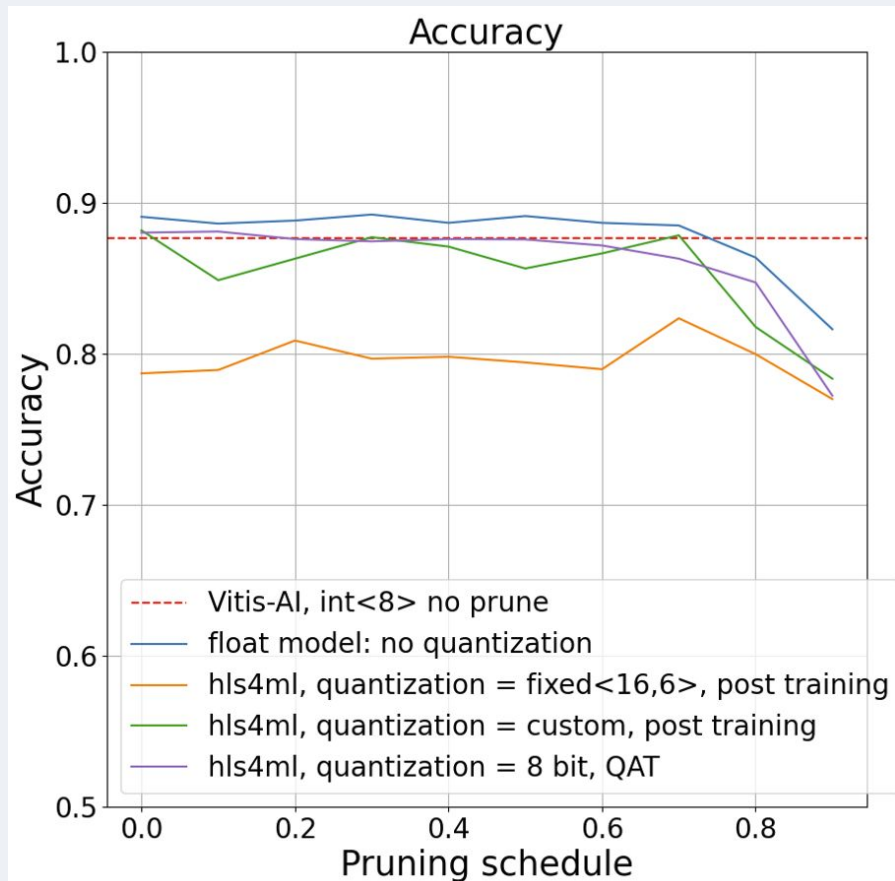
# Vitis-AI for the top-tagging CNN: quantization

Vitis-AI default quantization:

- weights= int<8>
- activations= int<32>

Next steps:

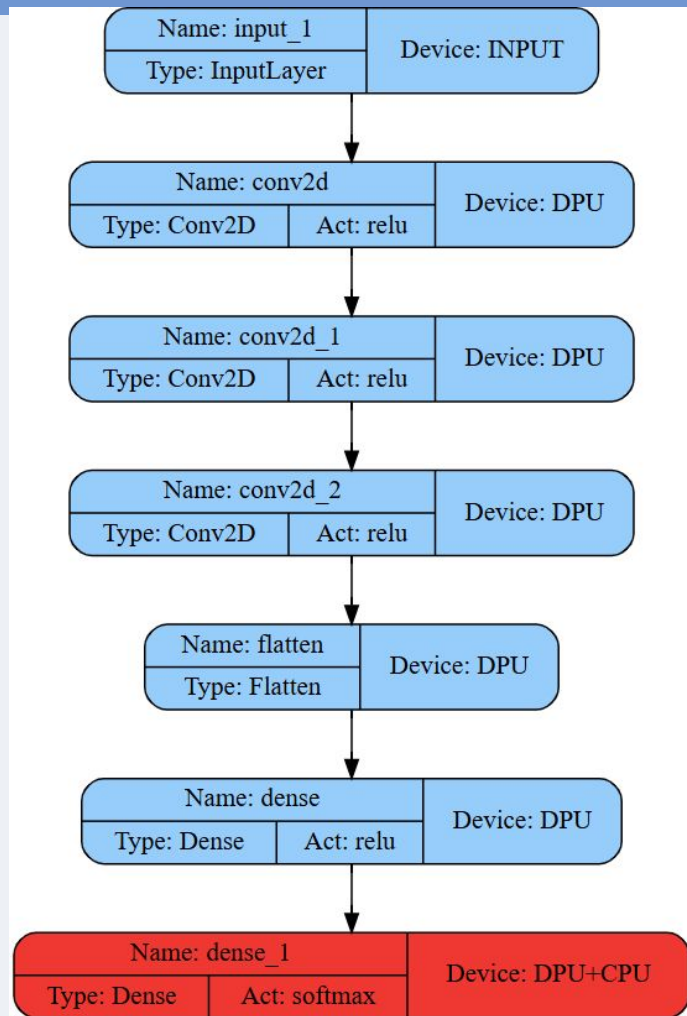
- Explore use of Vitis-AI optimizer (iterative pruning)



# Vitis-AI for the top-tagging CNN

Next steps to deploy on FPGA

- Model inspector results show all layers of top-tagging CNN (except **dense\_1**) are supported for acceleration on the AI Engines
- Softmax activation has to be done on the SoC (arm CPU)



# Conclusions + next steps

## Conclusions

- Prototyped a top-tagging CNN to compare the workflows and resource usage across candidate technologies for next generation triggers.

## Future directions

- Possible expansion to novel architecture designs (nanoPELICAN: see backup or [3rd System on Chip workshop](#) for further details)

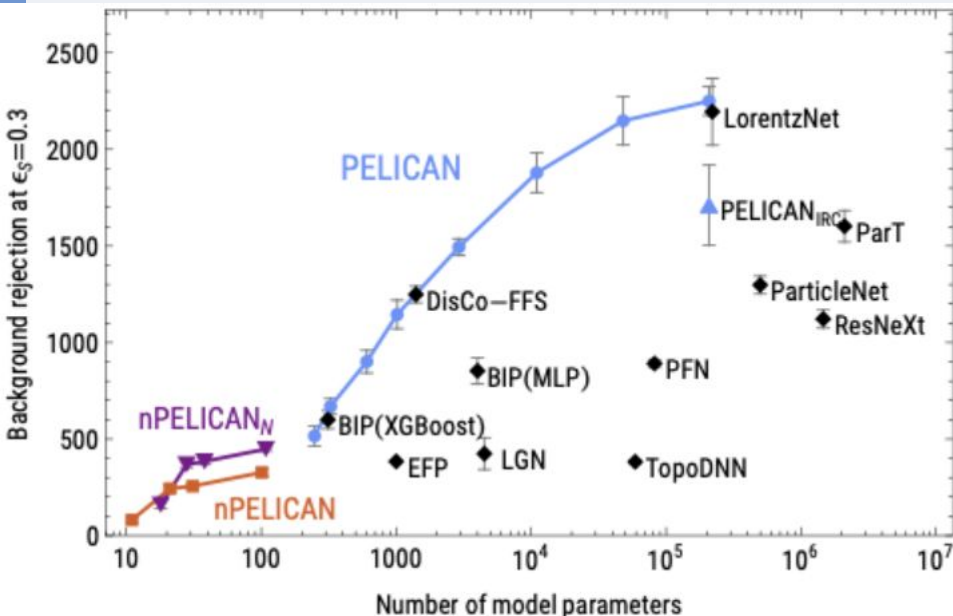
# Backup

**CREDITS:** This presentation template was created by [Slidesgo](#), and includes icons by [Flaticon](#), and infographics & images by [Freepik](#)

# Future directions: exploring complex architectures

nanoPELICAN: A low parameterization permutation and Lorentz equivariant network

## Preliminary nanoPELICAN resource utilization



```
=====
== Utilization Estimates
=====
```

\* Summary:

Name	BRAM_18K	DSP	FF	LUT	URAM
[DSP	-	1856	-	-	-
[Expression	-	-	0	1310748	-
[FIFO	-	-	-	-	-
[Instance	-	1846	0	37438	-
[Memory	-	-	-	-	-
[Multiplexer	-	-	-	36	-
[Register	-	-	451157	62912	-
[Total	0	3702	451157	1411134	0

# Dataset for the top-tagging CNN

Source: [top quark tagging reference dataset](#)

In total 1.2M training events, 400k validation events and 400k test events. Use “train” for training, “val” for validation during the training and “test” for final testing and reporting results.

## Description

- 14 TeV, hadronic tops for signal, qcd dijets background, Delphes ATLAS detector card with Pythia8
- No MPI/pile-up included
- Clustering of particle-flow entries (produced by Delphes E-flow) into anti-kT 0.8 jets in the pT range [550,650] GeV
- All top jets are matched to a parton-level top within  $\Delta R = 0.8$ , and to all top decay partons within 0.8
- Jets are required to have  $|\eta| < 2$
- The leading 200 jet constituent four-momenta are stored, with zero-padding for jets with fewer than 200
- Constituents are sorted by pT, with the highest pT one first
- The truth top four-momentum is stored as truth\_px etc.
- A flag (1 for top, 0 for QCD) is kept for each jet. It is called is\_signal\_new
- The variable "ttv" (= test/train/validation) is kept for each jet. It indicates to which dataset the jet belongs. It is redundant as the different sets are already distributed as different files.

# hls4ml for the top-tagging CNN: latency estimates

Latency estimates: default quantization, prune=0.75

FPGA	Latency min (clock cycles)	Latency max (cycles)	Latency min (us)	Latency max (us)
VU9P	384	387	1.92	1.935
VP1802	385	388	1.925	1.94
VP2802	385	388	1.925	1.94
VC1902	385	388	1.925	1.94
ZU9EG	387	390	1.935	1.95

Latency estimates: custom quantization, prune=0.75

FPGA	Latency min (clock cycles)	Latency max (cycles)	Latency min (us)	Latency max (us)
VU9P	383	386	1.915	1.93
VP1802	383	386	1.915	1.93
VP2802	383	386	1.915	1.93
VC1902	383	386	1.915	1.93
ZU9EG	385	388	1.925	1.94

Estimated maximum clock frequency in firmware:

- 299 MHz
  - VU9P, VP1802, VP2802, VC1902
- 232 MHz
  - ZU9EG

# FPGA resources

FPGA	Type	BRAM_18K (x 10 <sup>3</sup> )	DSP (x 10 <sup>3</sup> )	FF (x 10 <sup>6</sup> )	LUT (x 10 <sup>6</sup> )	URAM (x 10 <sup>3</sup> )
VU9P	Virtex Ultrascale+	4.32	6.84	2.36	1.18	0.96
VP1802	Versal premium	9.88	14.35	6.72	3.36	2.55
VP2802	Versal premium	9.88	14.3	6.7	3.35	2.55
VC1902	Versal AI core	1.93	1.97	1.8	0.9	0.46
ZU9EG	Zynq Ultrascale+	1.82	2.2	0.548	0.274	0

FPGA	High speed transceiver types	# high speed transceivers	Product information
VU9P	GTY/GTM Transceivers (32.75/58 Gb/s)	120/0	<u><a href="#">Virtex Ultrascale+</a></u>
VP1802	GTYP Transceivers (32.75 Gb/s)	28 <sup>1</sup>	<u><a href="#">Versal Premium</a></u>
	GTM Transceivers (58G (112G))	140 (70)	
VP2802	GTYP Transceivers (32.75 Gb/s)	28 <sup>1</sup>	<u><a href="#">Versal Premium</a></u>
	GTM Transceivers (58G (112G))	140 (70)	
VC1902	GTYP Transceivers	44	<u><a href="#">Versal AI Core</a></u>
ZU9EG	GTH Transceivers (16.3Gb/s)	16	<u><a href="#">Zynq Ultrascale+</a></u>

# Vitis-AI workflow with RESNET CNN

The [Vitis-AI tutorial](#) provides a workflow for deploying the RESNET18 CNN on FPGAs

Model	Float model prediction	Quantized model (ap_fixed<8>)
RESNET CNN1	test accuracy = 0.73690 train accuracy = 0.82702	test accuracy = 0.73360 train accuracy = 0.81515
RESNET CNN2	test accuracy = 0.84000 train accuracy = 0.95135	test accuracy = 0.83700 train accuracy = 0.94703

ML model is deployed on 2 types of FPGA boards

- The vck190 has AI Engines: faster performance, but larger bandwidth requirement

Board	SW Runtime (ms)	HW Runtime (ms)	Efficiency (%)	Avg Bandwidth (MB/s)
zcu102	1.583	1.420	4.3	7516.129
vck190	0.499	0.399	1.9	26876.923

See [3rd System on Chip workshop](#) for further details