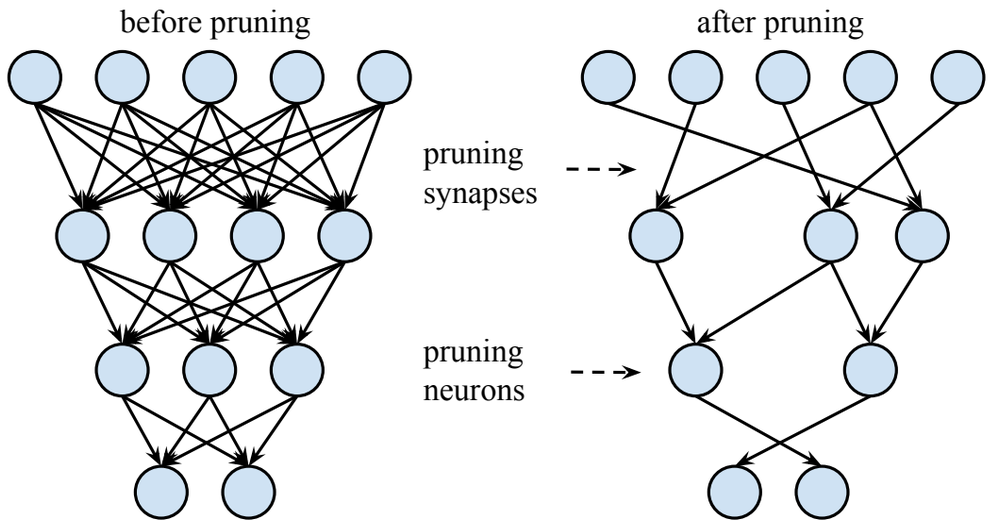
Compressing deep NN on FPGAs to ultra-low precision

---

8 October 2019, CERN

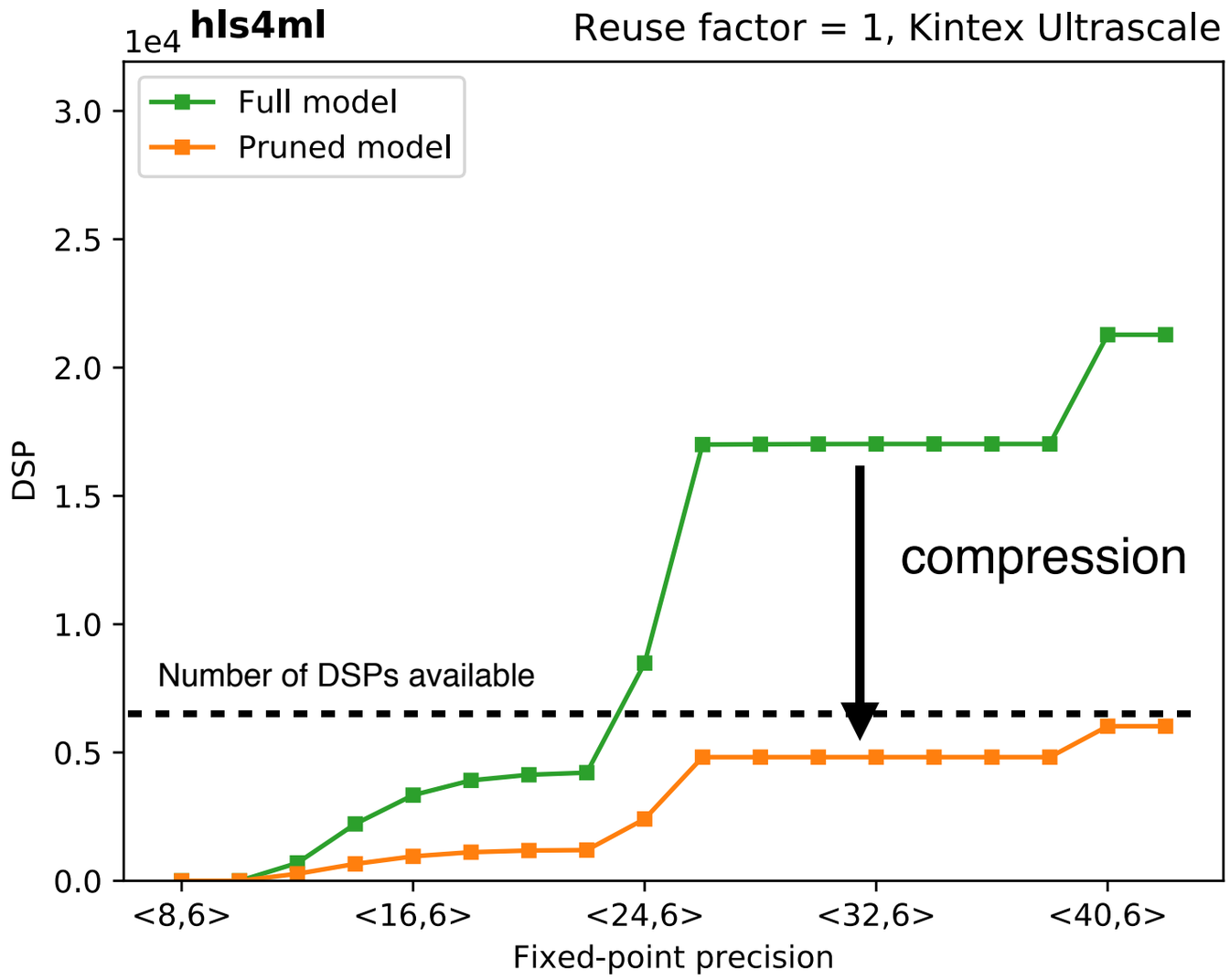
Efficient NN design



DSPs can be a limiting resource
→ how to fit my model on a FPGA?

Happy to have many zeros but not straightforward to implement sparse matrix multiplication!

FPGA can optimize those away but not in all cases.



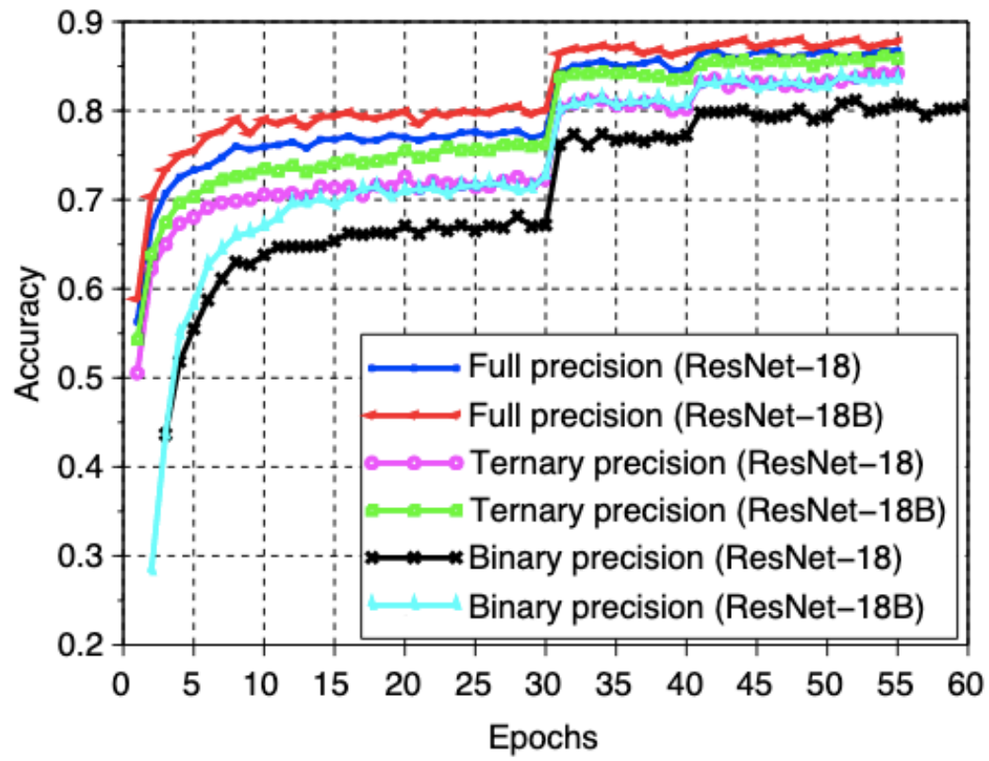
Ultra-low precision arithmetic

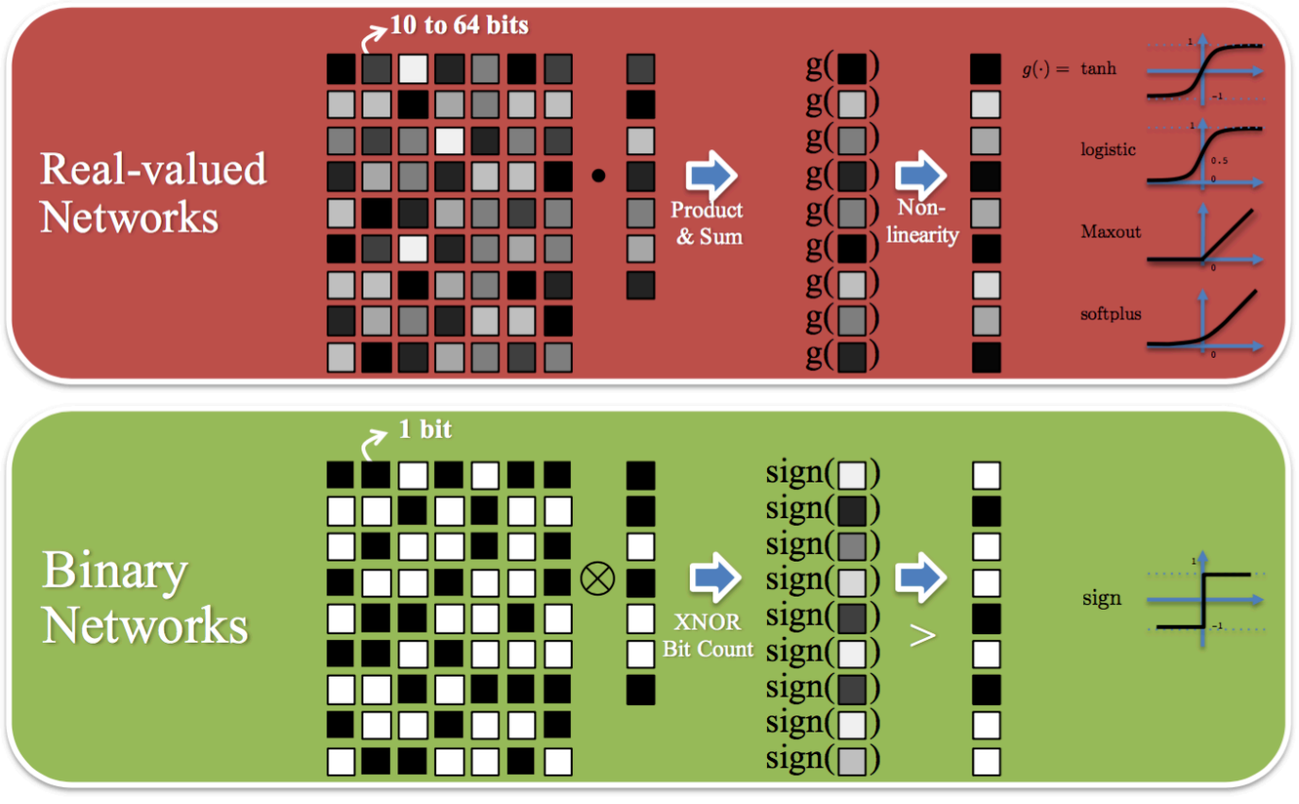
Replace 32-bit floating point multiplications with 1/2 bits arithmetics with limited loss in accuracy:

- 1-bit: binary NN (arxiv.1602.02830)
- 2-bits: ternary NN (arxiv.1605.04711)

nb, only the weights and activations are binarized and not the gradients used to update parameters during backpropagation.

Extremely attractive from a hardware perspective! **BNN/ TNN computationally efficient at low power.**





A bit of literature: Xilinx BNN

FINN: A Framework for Fast, Scalable Binarized Neural Network Inference

arxiv.1612.07119

Yaman Umuroglu*†, Nicholas J. Fraser*‡, Giulio Gambardella*, Michaela Blott*,
Philip Leong‡, Magnus Jahre† and Kees Vissers*
*Xilinx Research Labs; †Norwegian University of Science and Technology; ‡University of Sydney
yamanu@idi.ntnu.no

- Demonstrated that for by binarizing dense and Conv2D the small memory required removes the off-chip memory bottleneck by keeping parameters on-chip, even for large networks!

Table 1: Accuracy results - BNN vs NN.

Neurons/layer	Binary Err. (%)	Float Err. (%)	# Params	Ops/frame
128	6.58	2.70	134,794	268,800
256	4.17	1.78	335,114	668,672
512	2.31	1.25	932,362	1,861,632
1024	1.60	1.13	2,913,290	5,820,416
2048	1.32	0.97	10,020,874	20,029,440
4096	1.17	0.91	36,818,954	73,613,312

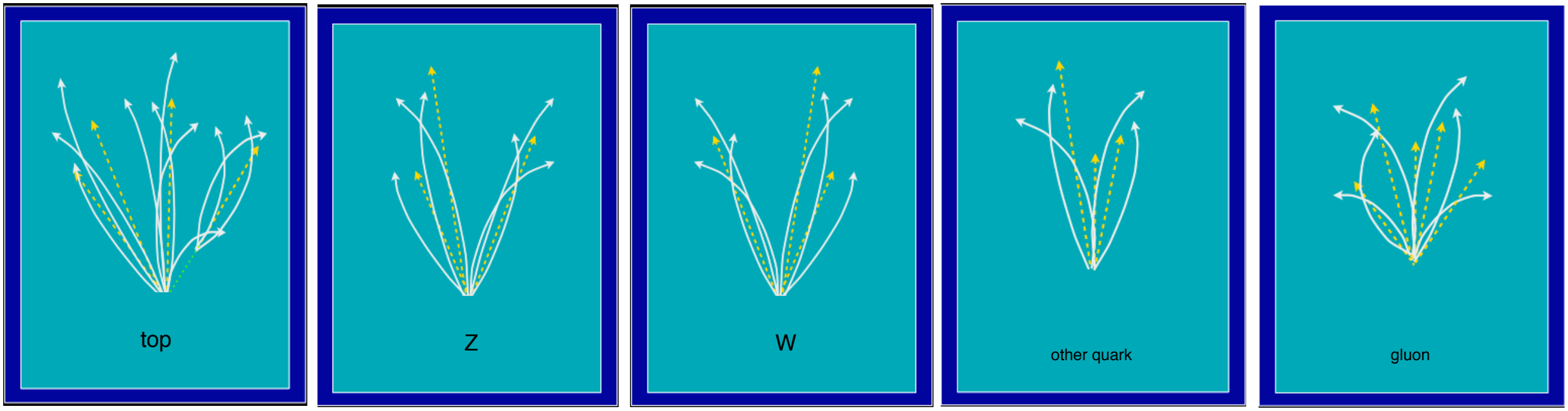
Table 3: Summary of results from FINN 200 MHz prototypes.

Name	Thr.put (FPS)	Latency (µs)	LUT	BRAM	P_{chip} (W)	P_{wall} (W)
SFC-max	12361 k	0.31	91131	4.5	7.3	21.2
LFC-max	1561 k	2.44	82988	396	8.8	22.6
CNV-max	21.9 k	283	46253	186	3.6	11.7
SFC-fix	12.2 k	240	5155	16	0.4	8.1
LFC-fix	12.2 k	282	5636	114.5	0.8	7.9
CNV-fix	11.6 k	550	29274	152.5	2.3	10

Zynq device

Benchmark models: jet tagging

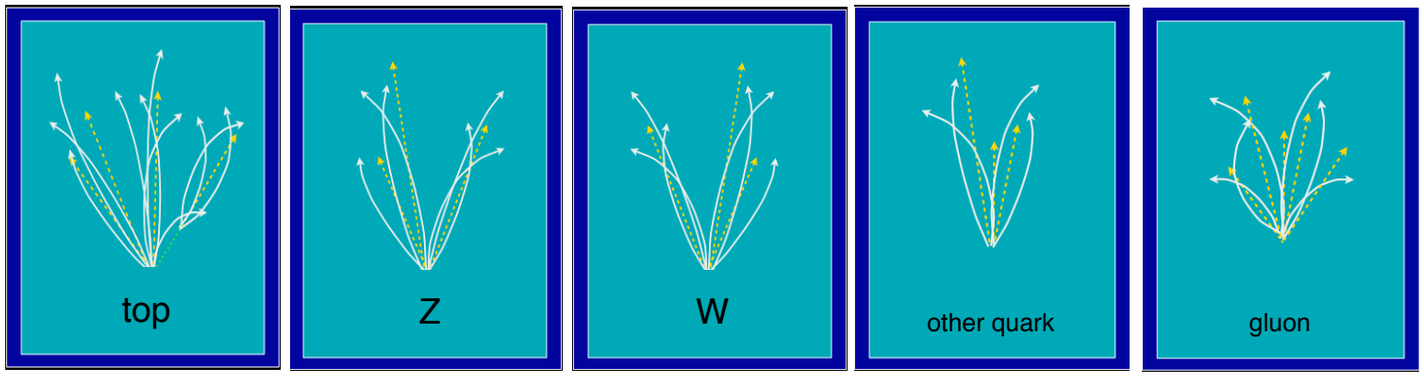
Study a multi-classification task:
discrimination between highly energetic (boosted) **q, g, W, Z, t** initiated jets

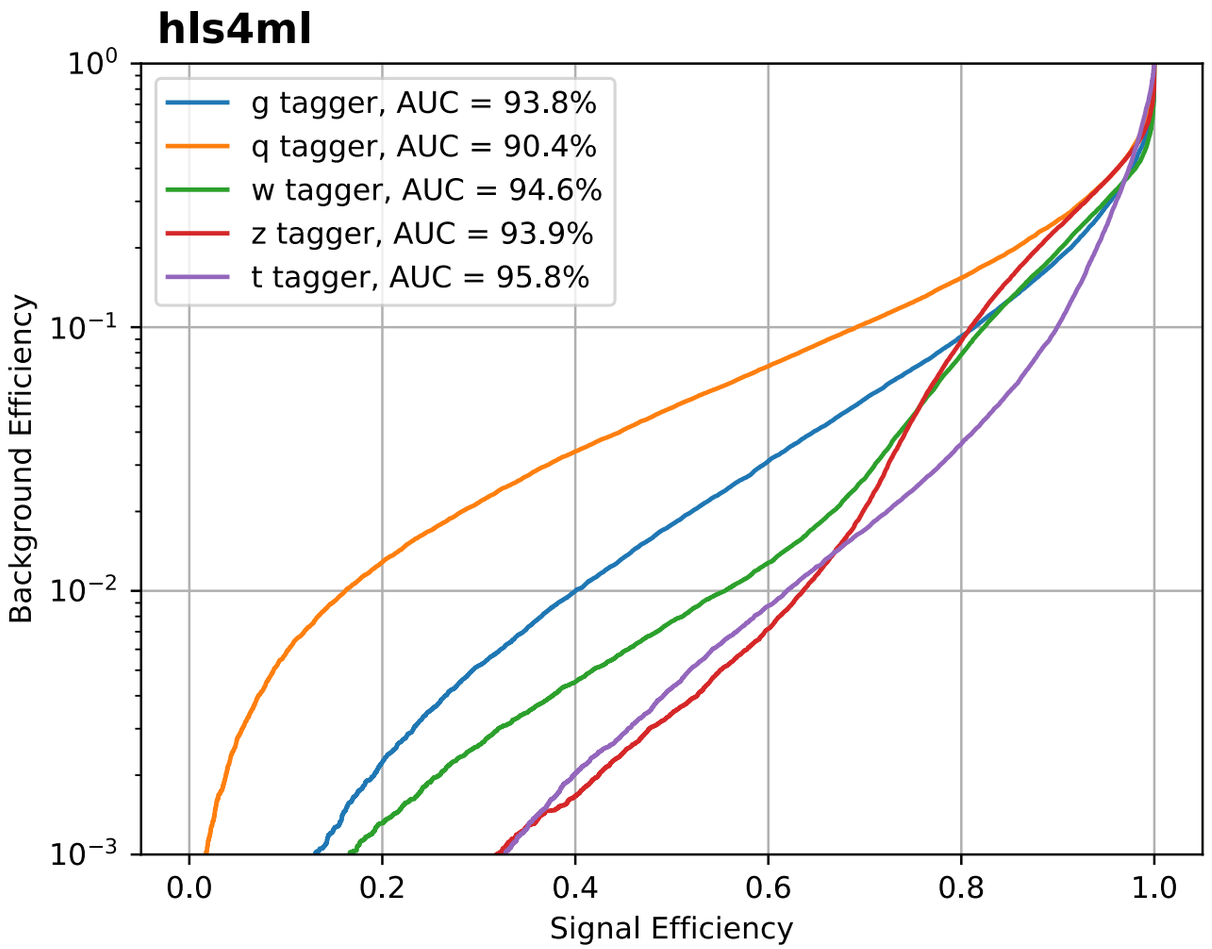


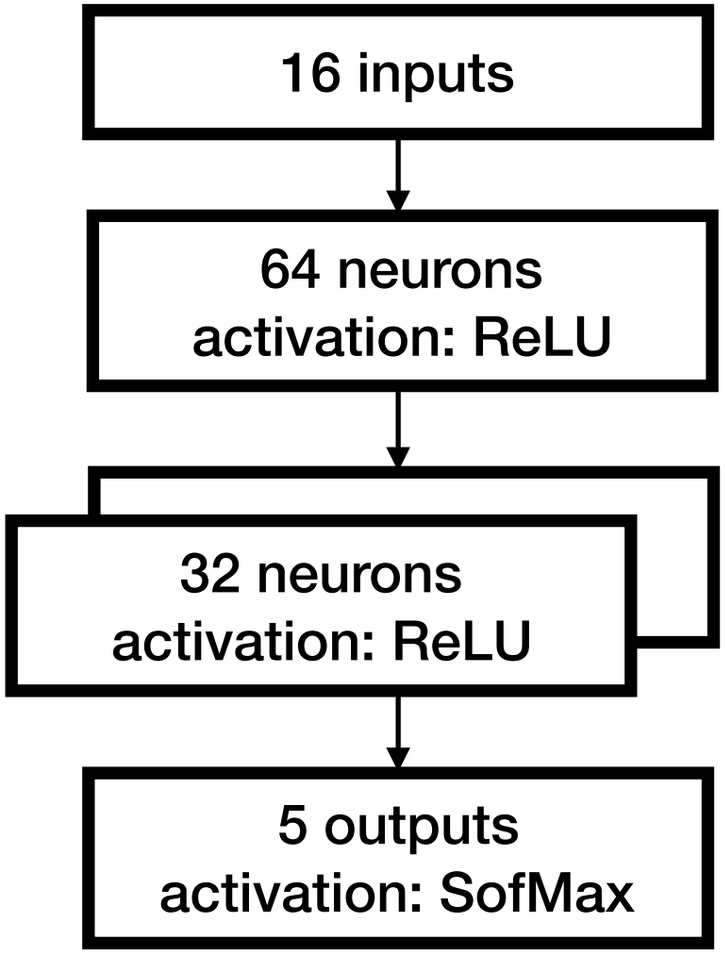
t→bW→bqq	**Z→qq**	**W→qq**	**q/g background**
3-prong jet	2-prong jet	2-prong jet	no substructure and/or mass ~ 0

Reconstructed as one massive jet with substructure

Benchmark models: jet tagging

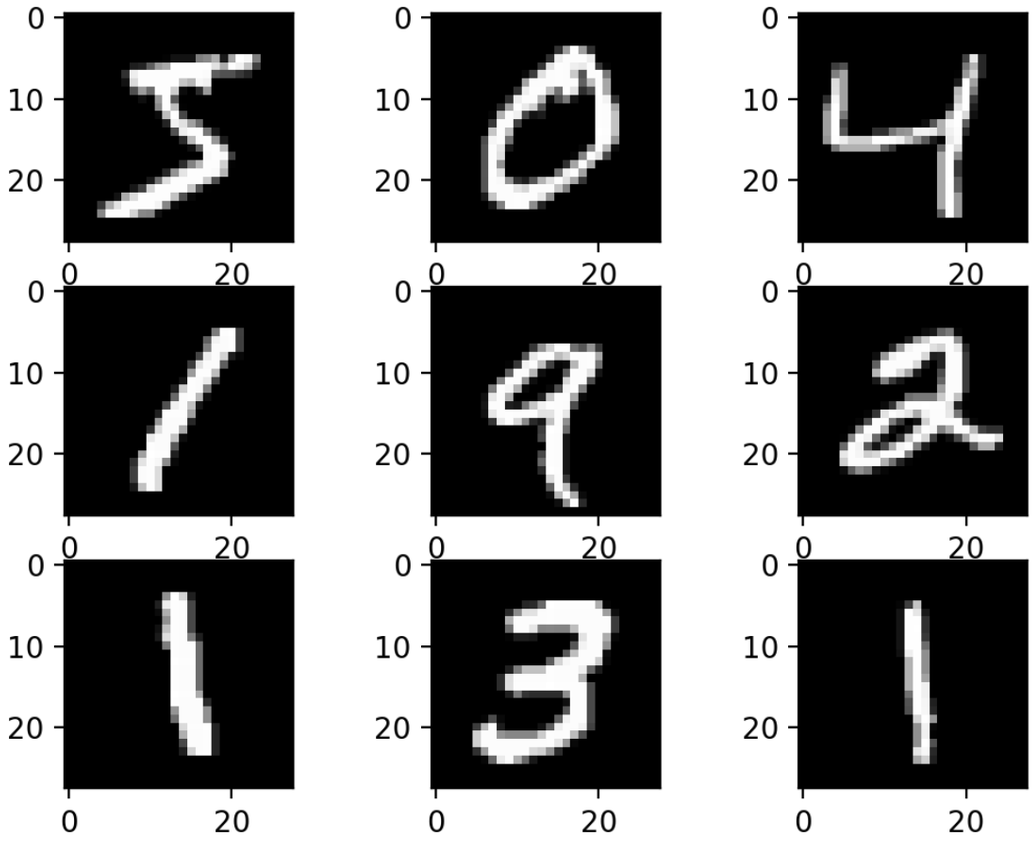


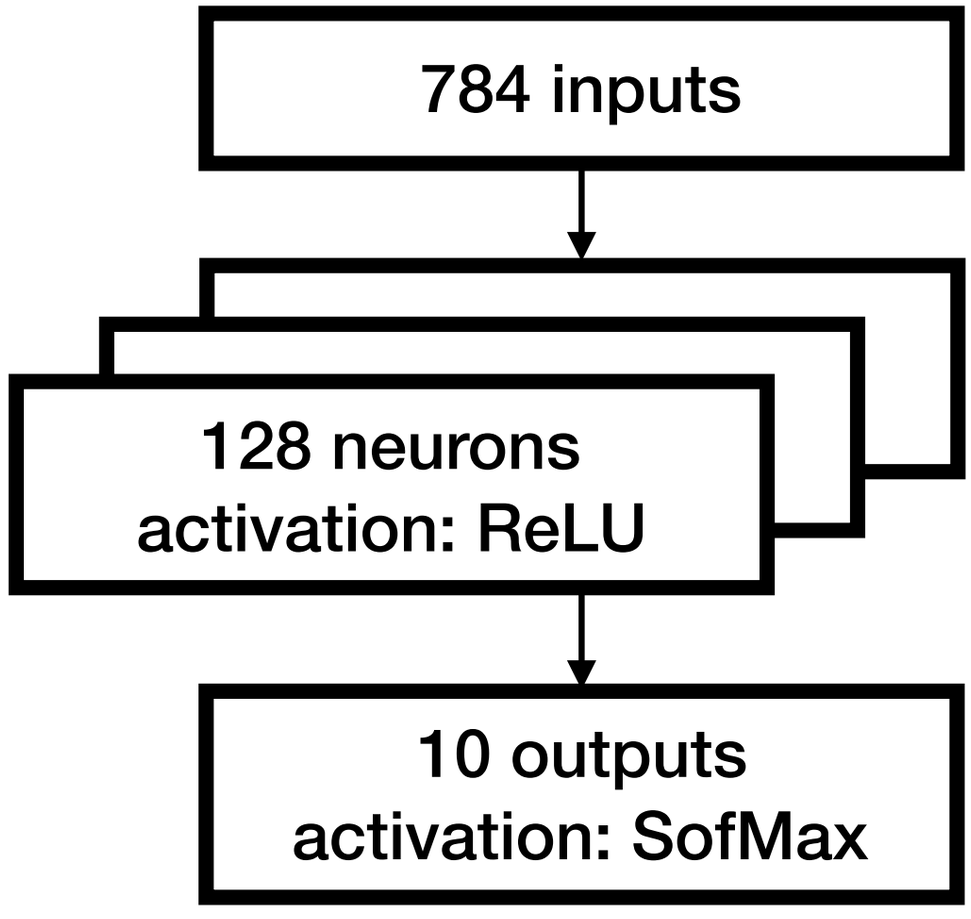




Average accuracy ~ 0.75

Benchmark models: MNIST



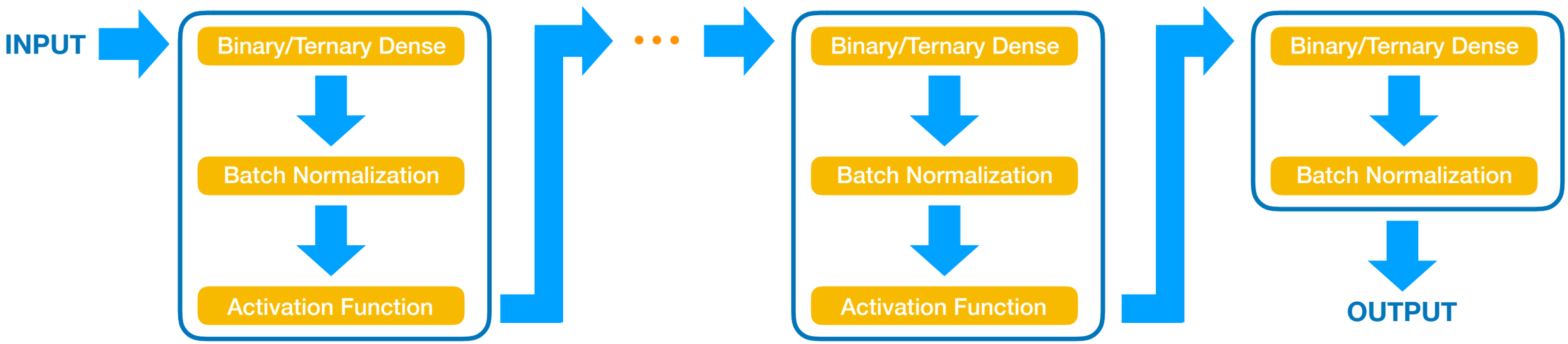


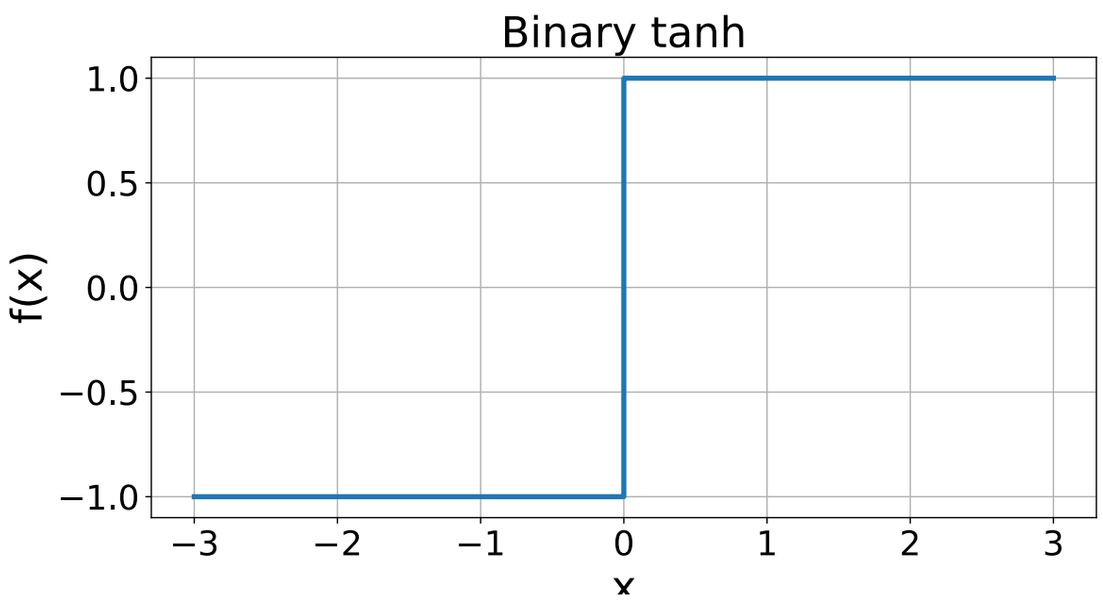
Average accuracy ~ 0.98

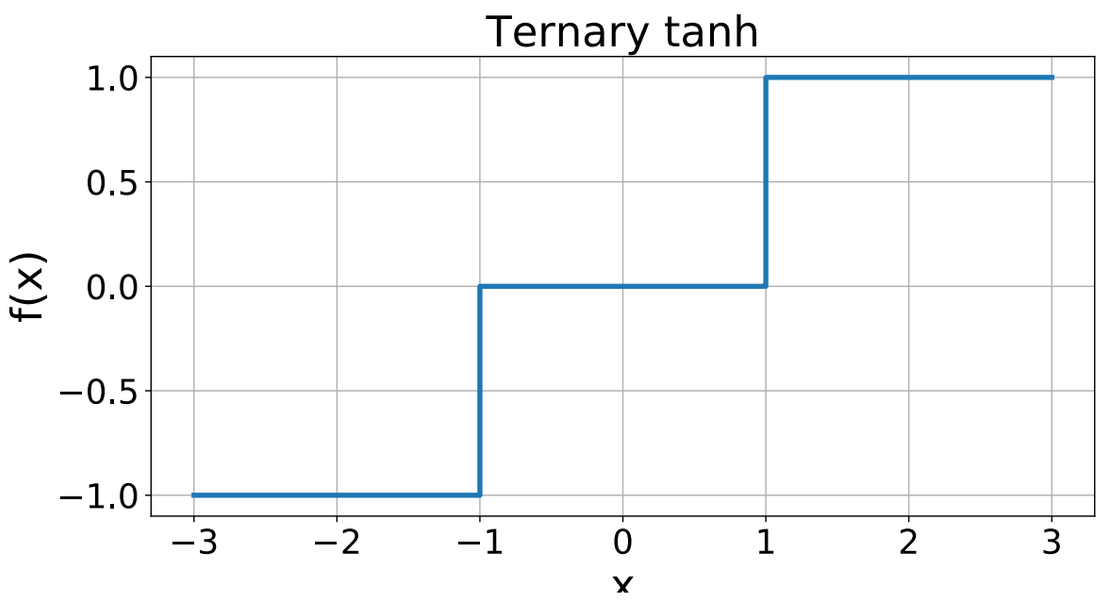
AUC per class > 99%

Binary/Ternary architectures

(First tests and implementation for MLP)







hls4ml implementation

```
template<class data_T, class weight_T, class ret_T>
inline typename std::enable_if<(not std::is_same<data_T, ap_uint<1>>::value)
        and std::is_same<weight_T, ap_uint<1>>::value, ret_T>::type
product(data_T a, ap_uint<1> w){
    // Specialisation for 1-bit weights, arbitrary data
    #pragma HLS inline off
    return w == 0 ? (data_T) -a : a;
}
```

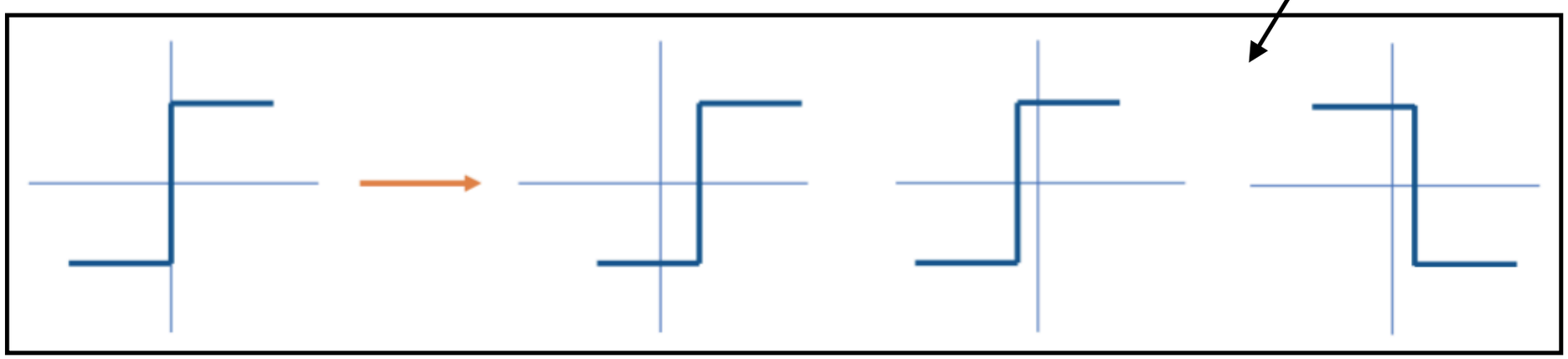
binary product

+ optimizer of layer-by-layer fixed point precision

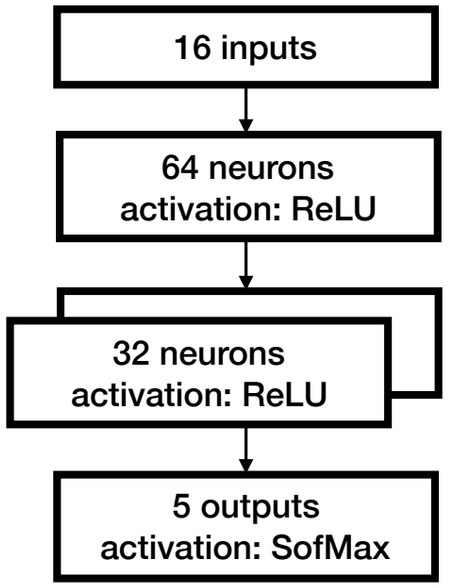
```
template<class data_T, class weight_T, class ret_T>
inline typename std::enable_if<(not std::is_same<data_T, ap_uint<2>>::value)
        and std::is_same<weight_T, ap_int<2>>::value, ret_T>::type
product(data_T a, ap_int<2> w){
    // Specialisation for 2-bit weights, arbitrary data
    #pragma HLS inline off
    if (w == 0) return (data_T) 0;
    else if(w == -1) return (data_T) -a;
    else return (data_T) a; // if(w == 1)
}
```

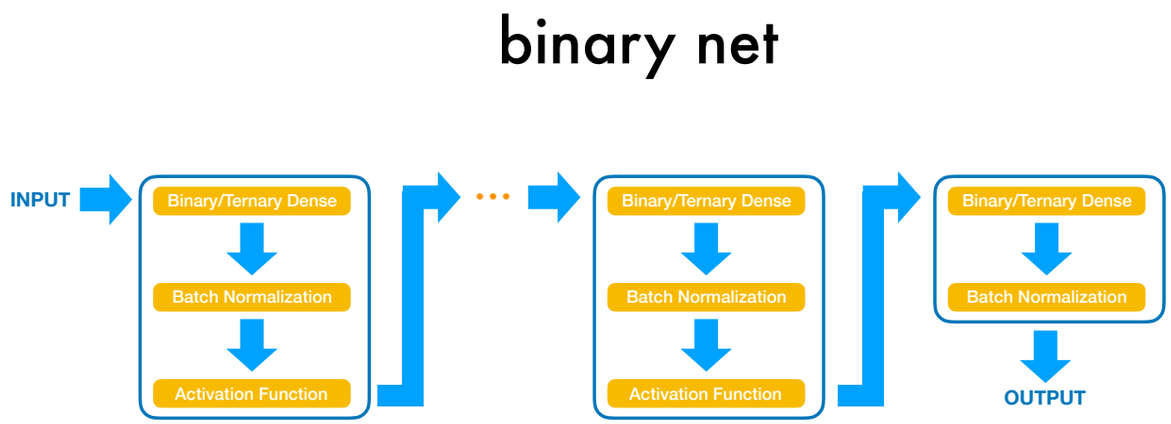
ternary product

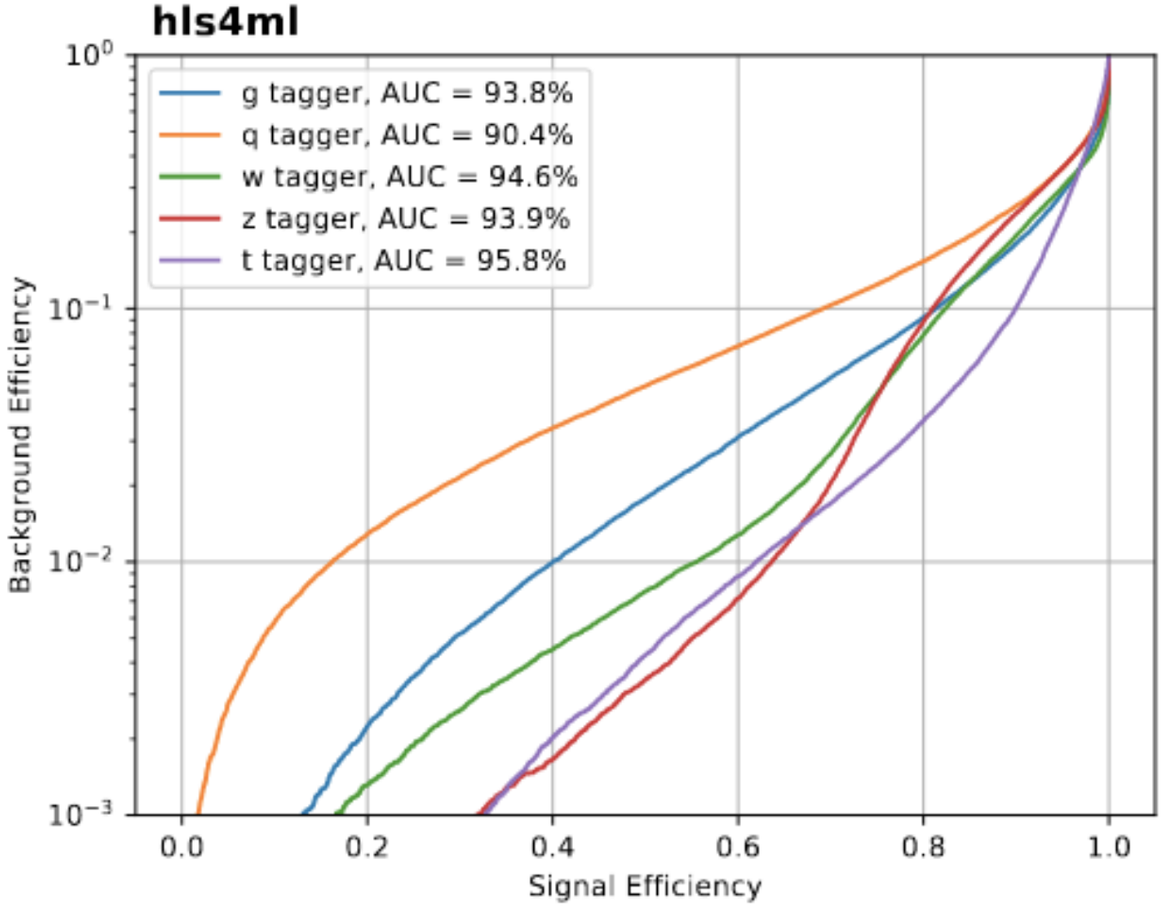
Fused batch normalization + ternary/binary tanh: compare with a threshold

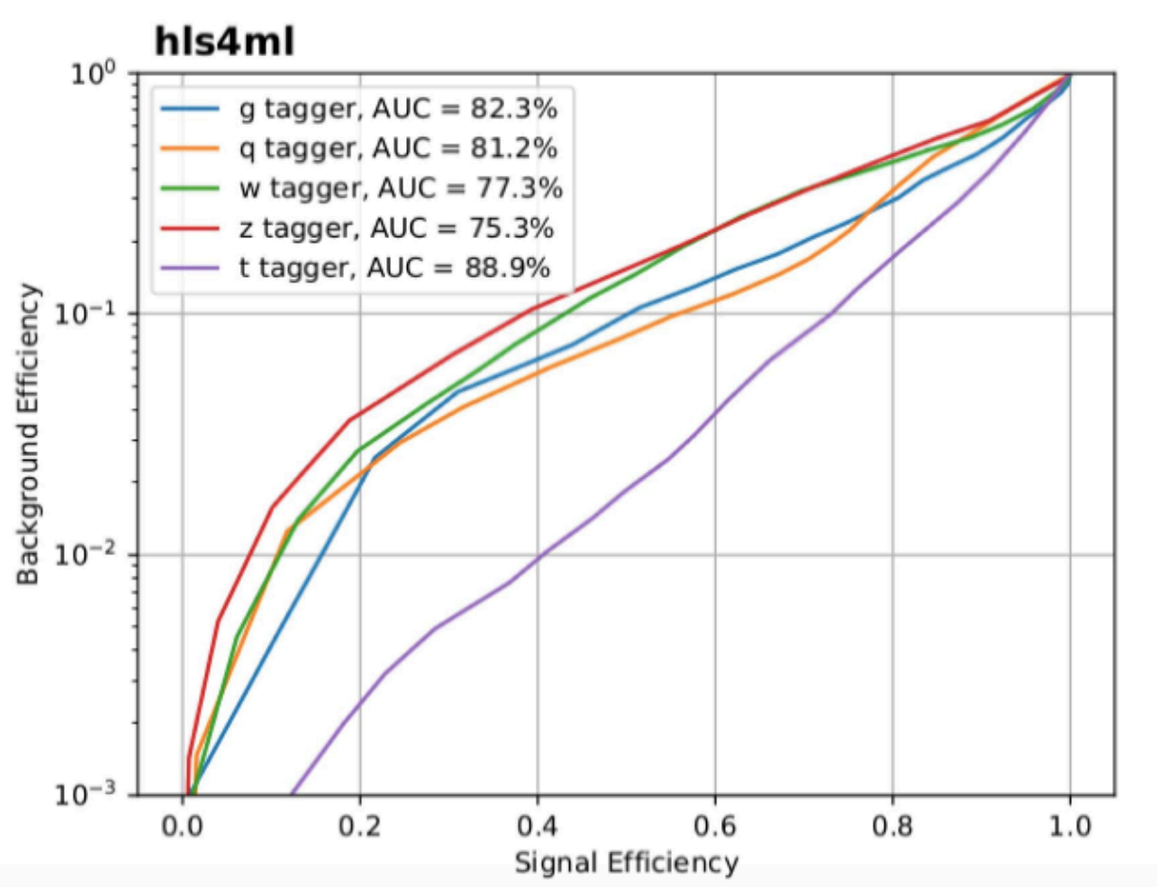


Results: jet tagging model





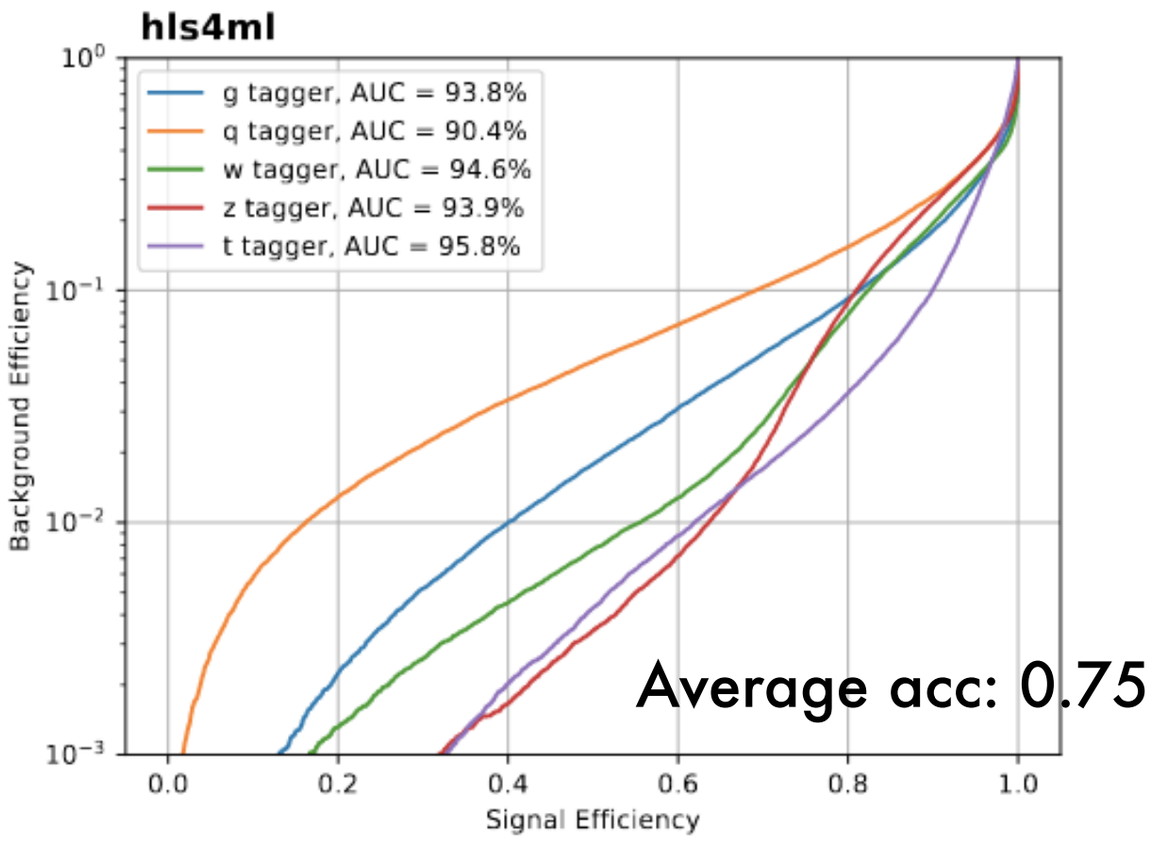




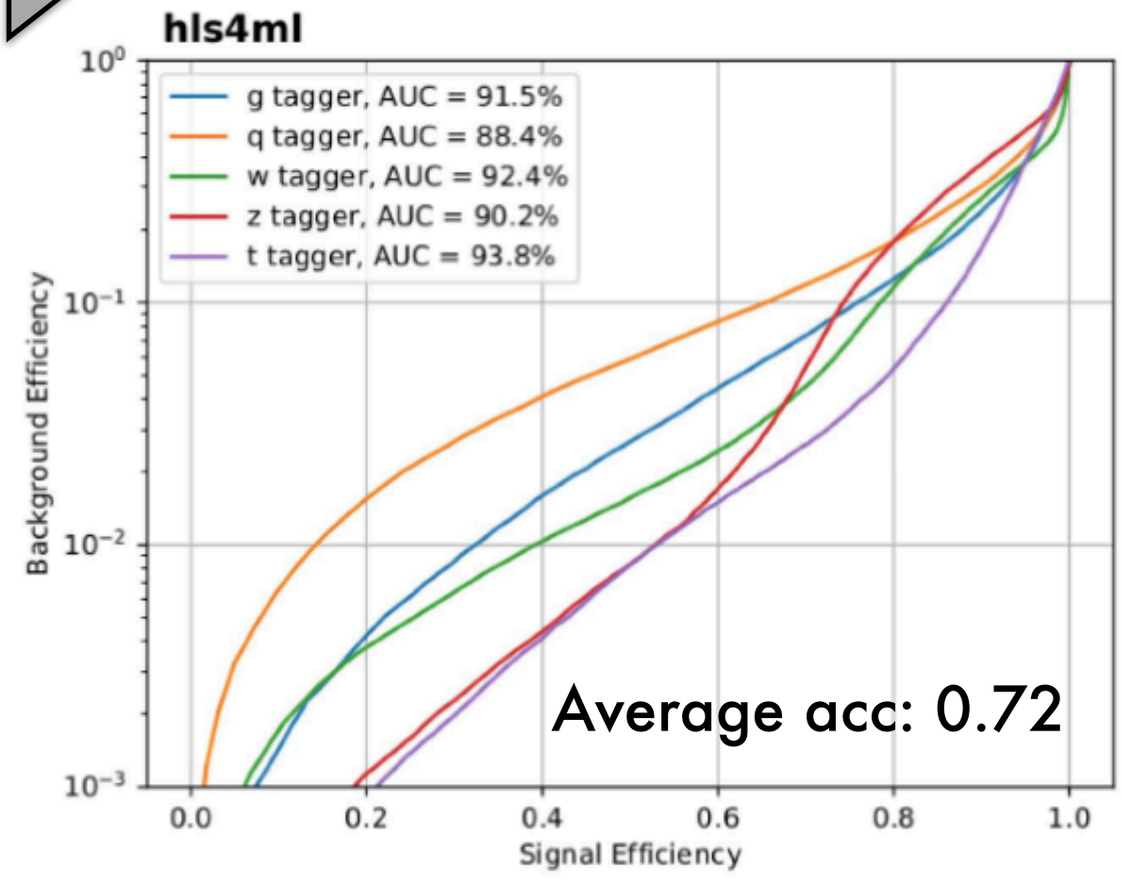
Results: jet tagging model

- Run hyper parameter bayesian optimization: neurons, layers, batch size, learning rate, different optimizers
- Recover performance with **16x448x224x224x5** model (*7 times more neurons*)

full precision



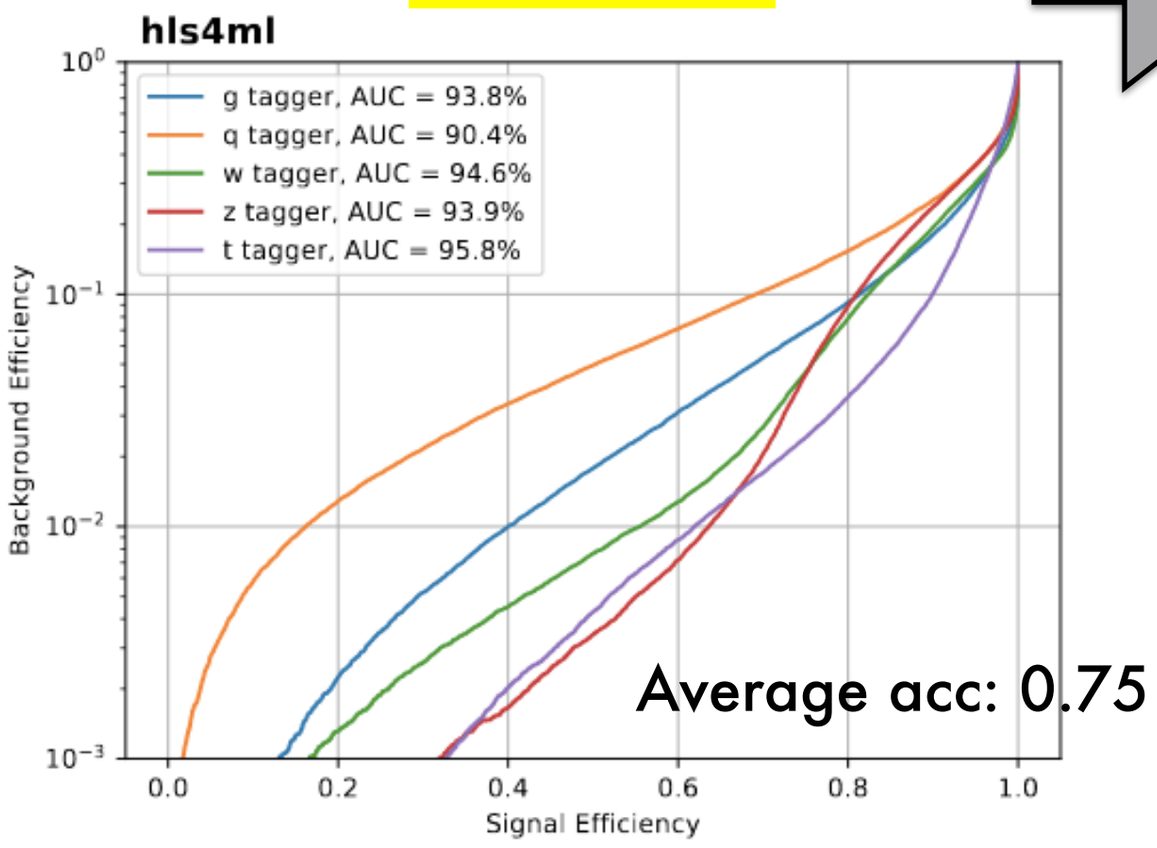
Optimized binary net



Results: jet tagging model

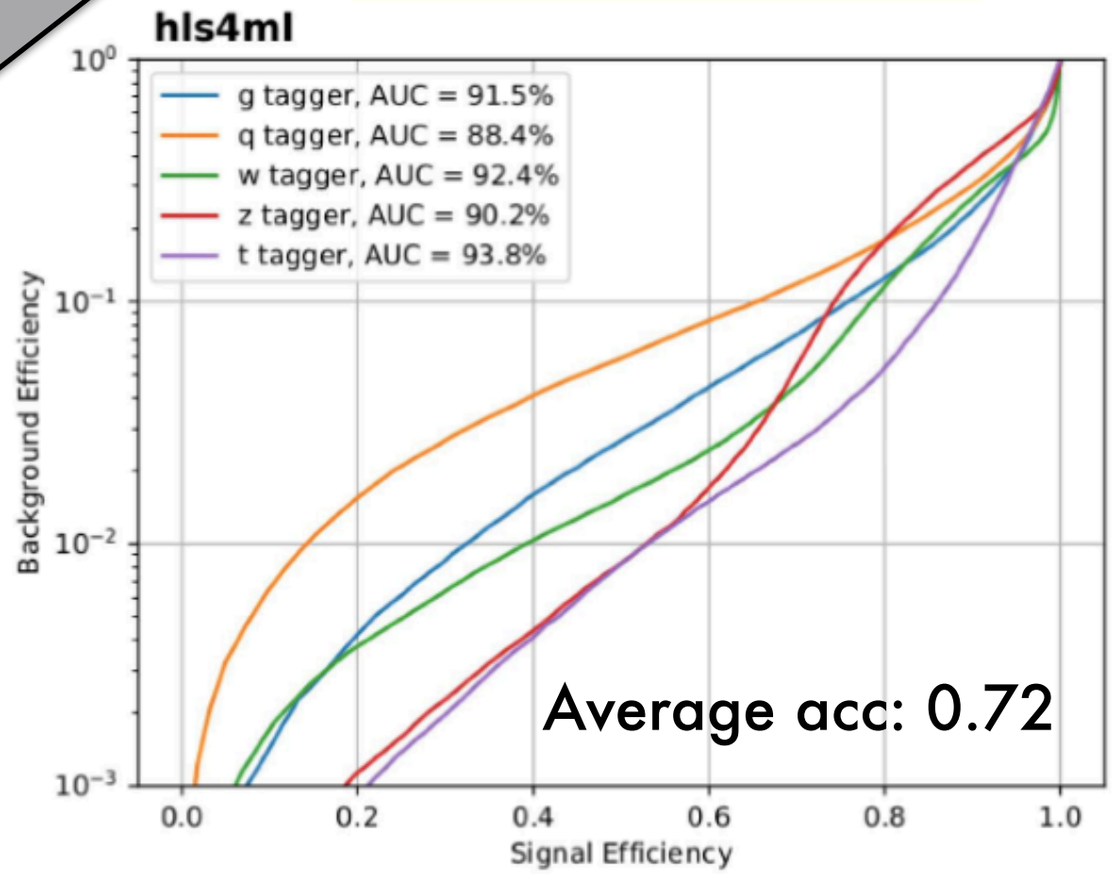
- Run hyper parameter bayesian optimization: neurons, layers, batch size, learning rate, different optimizers
- Recover performance with **16x448x224x224x5** model (7 times more neurons)
- For **Ternary Net** converge to smaller model: **16x128x64x64x5** (2 times more neurons + one more layer)

full precision



Average acc: 0.75

Optimized ternary net



Average acc: 0.72

Results: jet tagging model

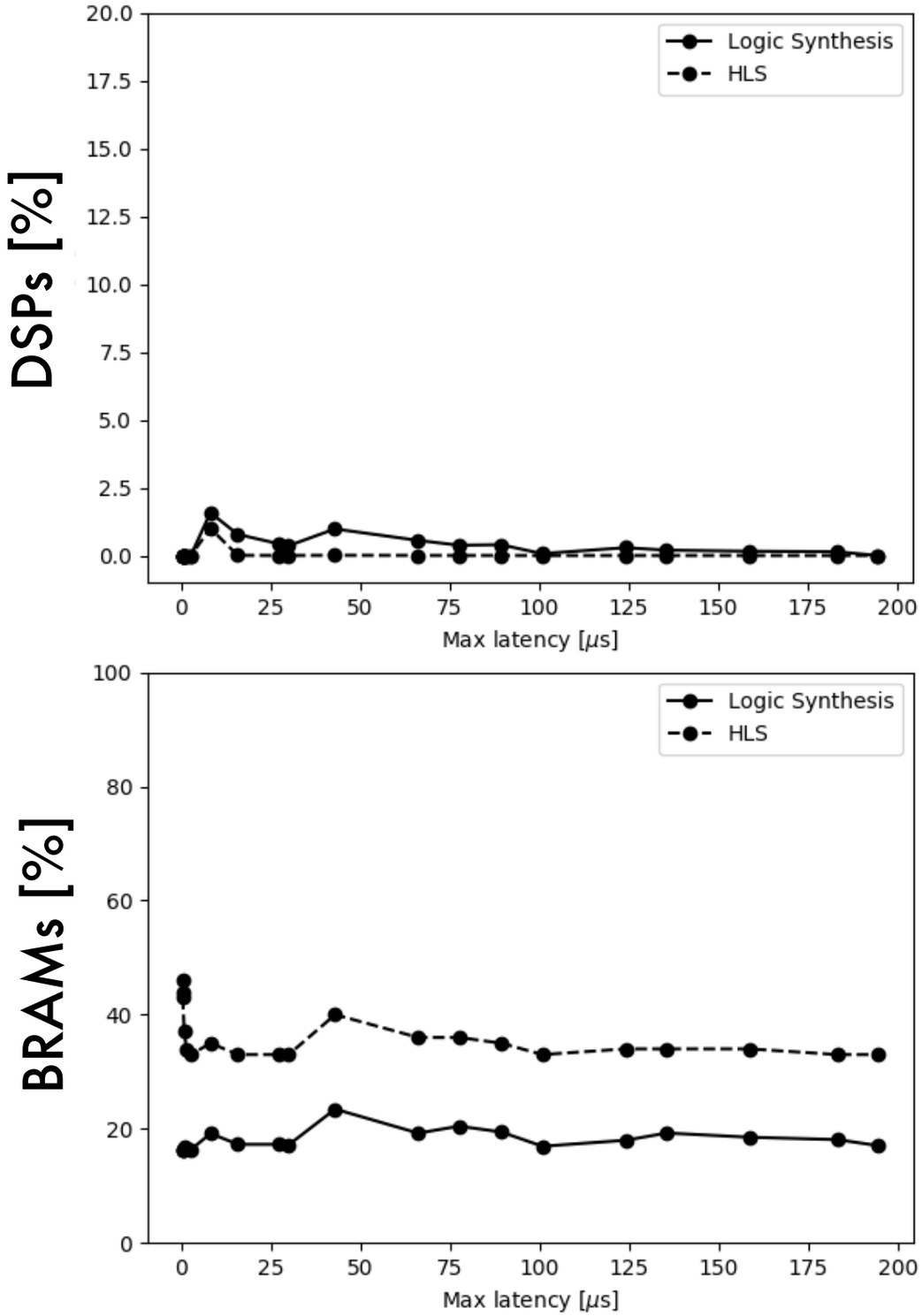
Architecture	AUCs [%]	Average accuracy	Minimum latency [μs]	DSPs [%]	LUTs [%]	FFs [%]	BRAMs [%]
float model (16x64x32x32x5)	90 - 96	0.75	0.060	**60**	7	1	0
float model (compressed)	91 - 96	**0.75**	0.090	**15**	1.7	0.7	0.3
Small BNN (16x64x32x32x5)	75 - 89	0.62	0.040	**0**	0.8	0.1	0
Optimized BNN (16x448x224x224x5)	88 - 94	**0.72**	**0.210**	**0**	**15**	7	0
BNN + ReLu (16x128x64x64x5)	88 - 93	**0.70**	**0.140**	**4**	6	1	0
Optimized TNN (16x128x64x64x64x5)	88 - 94	**0.72**	**0.110**	**0**	6	1	0
TNN + ReLu (16x64x32x32x5)	88 - 92	**0.68**	**0.060**	**2**	**2**	0.2	0

Results: MNIST

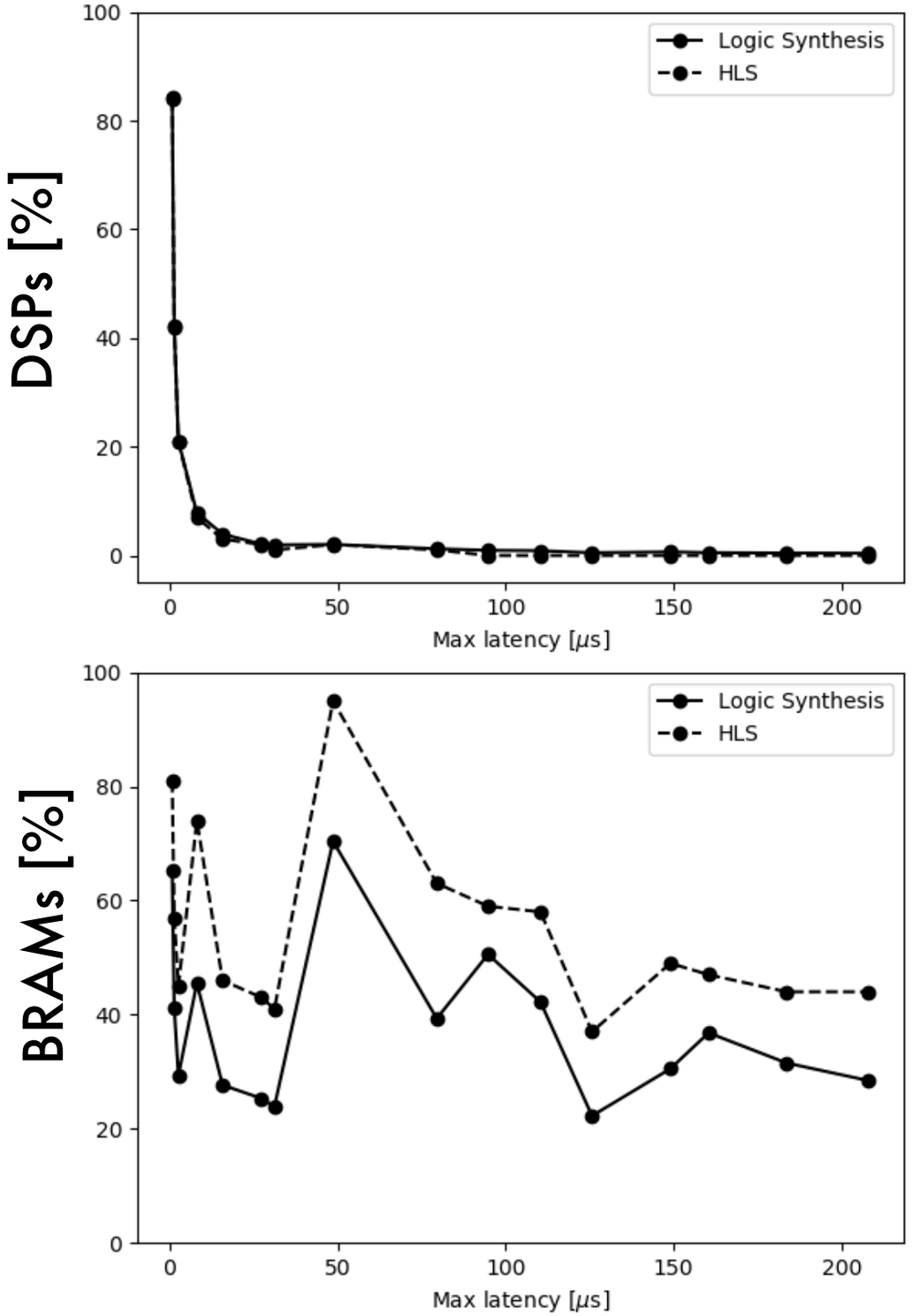
Architecture 784x128x128x128x10	Average accuracy	Minimum latency [µs]	DSPs [%]	LUTs [%]	FFs [%]	BRAMs [%]
float model	0.98	0.56	100	134	23	54
Binary model (binary tanh)	0.93	0.21	0	34	11	16
Ternary model (ternary tanh)	0.95	0.21	0	34	11	16

Results: MNIST

full precision

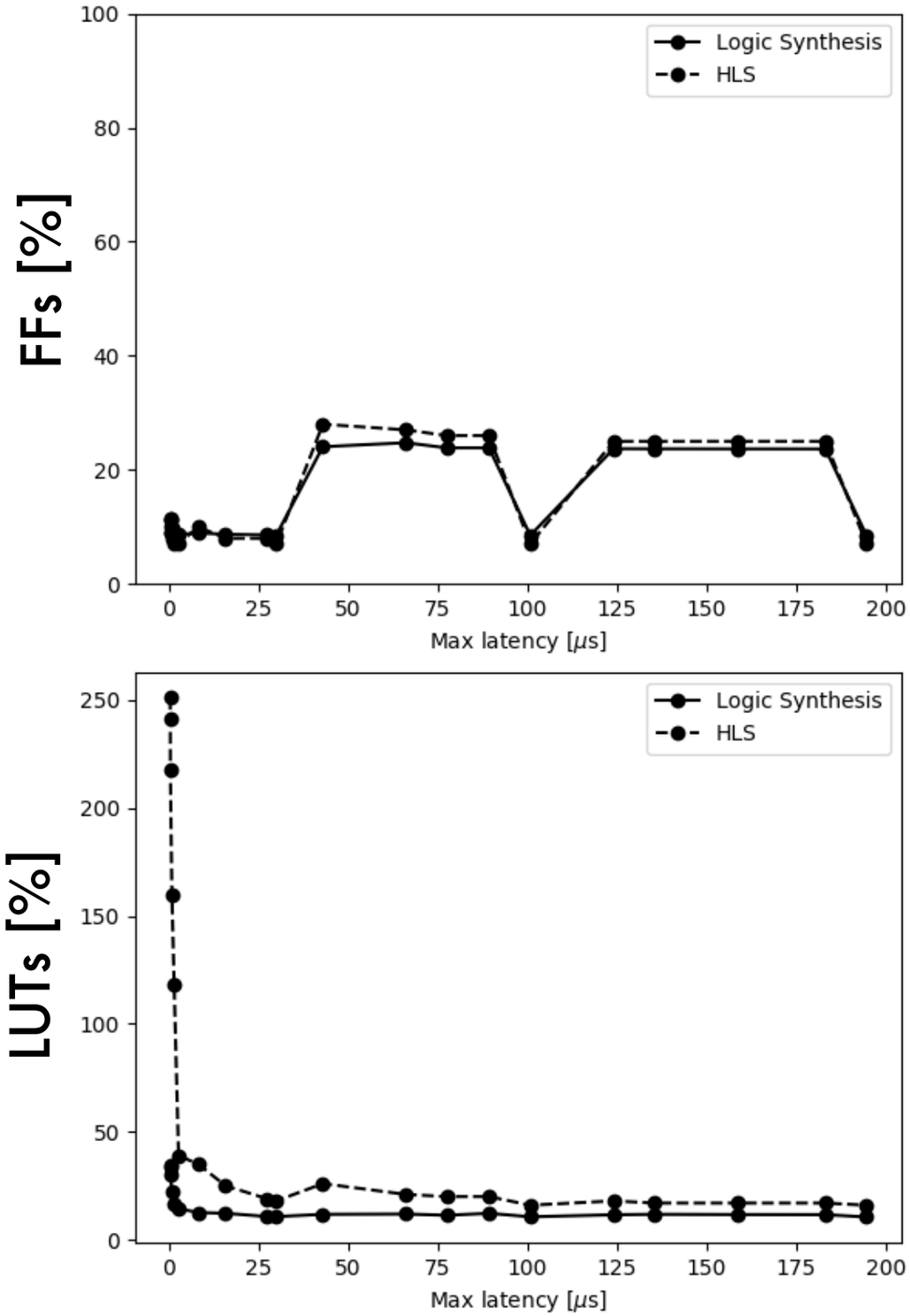


binary net



Results: MNIST

full precision



binary net