Neural Network L1 Trigger / GRL Upgrade

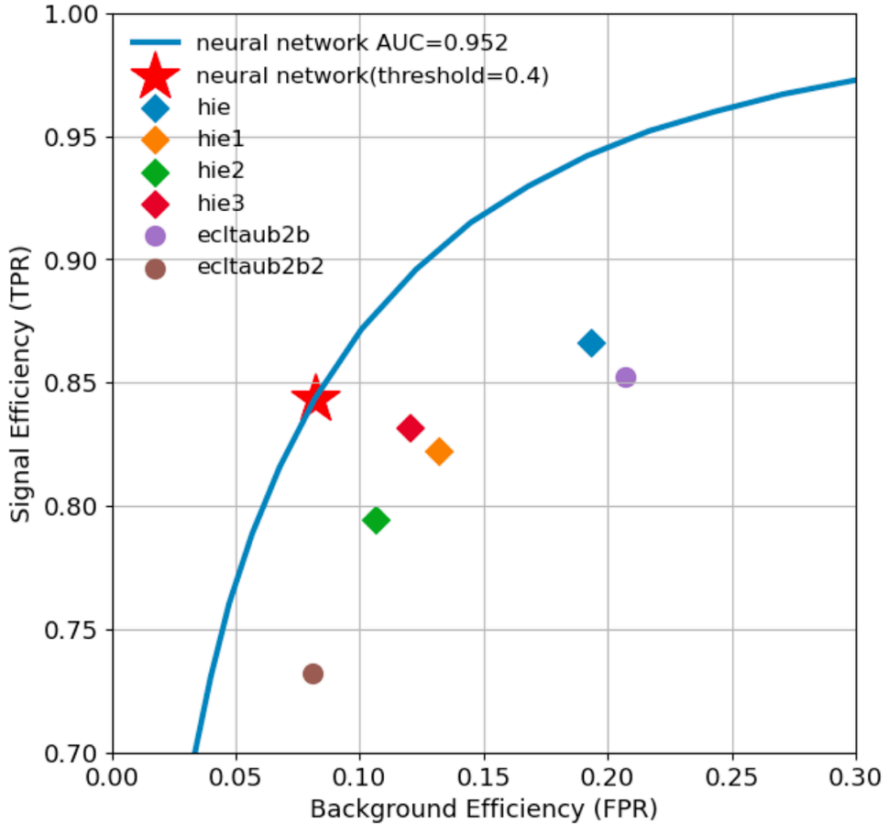
Deven Misra

Kavli IPMU | UTokyo

Introduction

Motivation

- The existing τ and dark sector lines for the L1 trigger will require higher efficiency to limit overall trigger rate as SuperKEKB luminosity increases in the future.
- Previous studies show promising results for increasing efficiency (relative to current cut-based algorithms) using a neural-network based trigger algorithm for τ event selection. This approach also has the potential to include other trigger lines if not limited by FPGA resources.



Comparing neural network trigger with existing algorithms (Nomaru 2023)

Previous Work (2023)

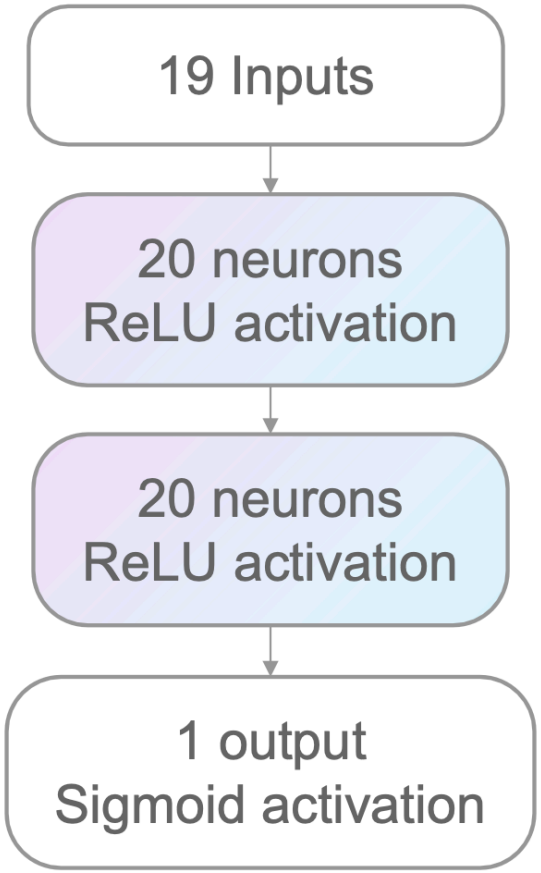
- Inputs: ECLTRG Clusters

$$\{E, \theta, \phi\} \times \#\ \text{of Clusters}$$

- Output:

$$P(\tau) \in [0, 1]$$

- Previous study included limited optimization, thus leaves significant room for improvement on these results.



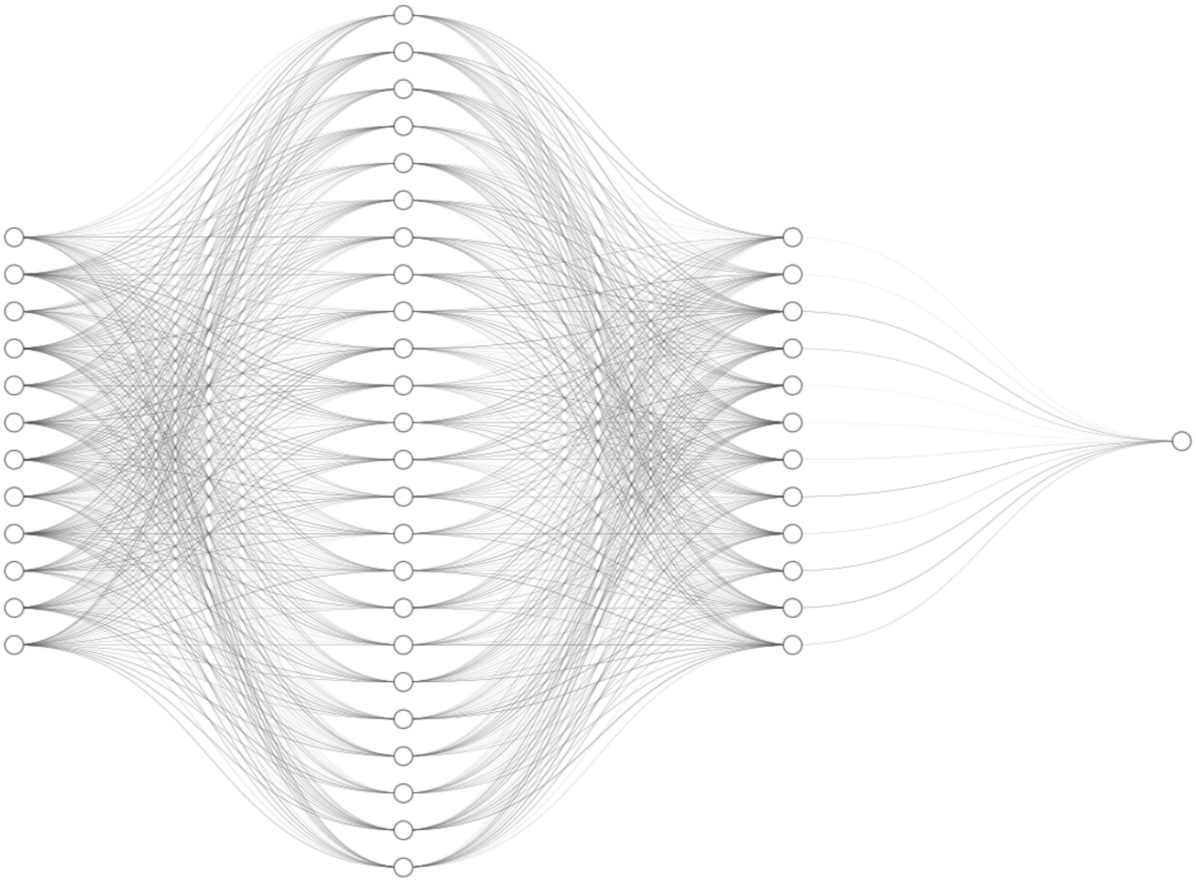
Model Architecture from Previous NNTRG Study (Nomaru 2023)

Neural Network

Requirements

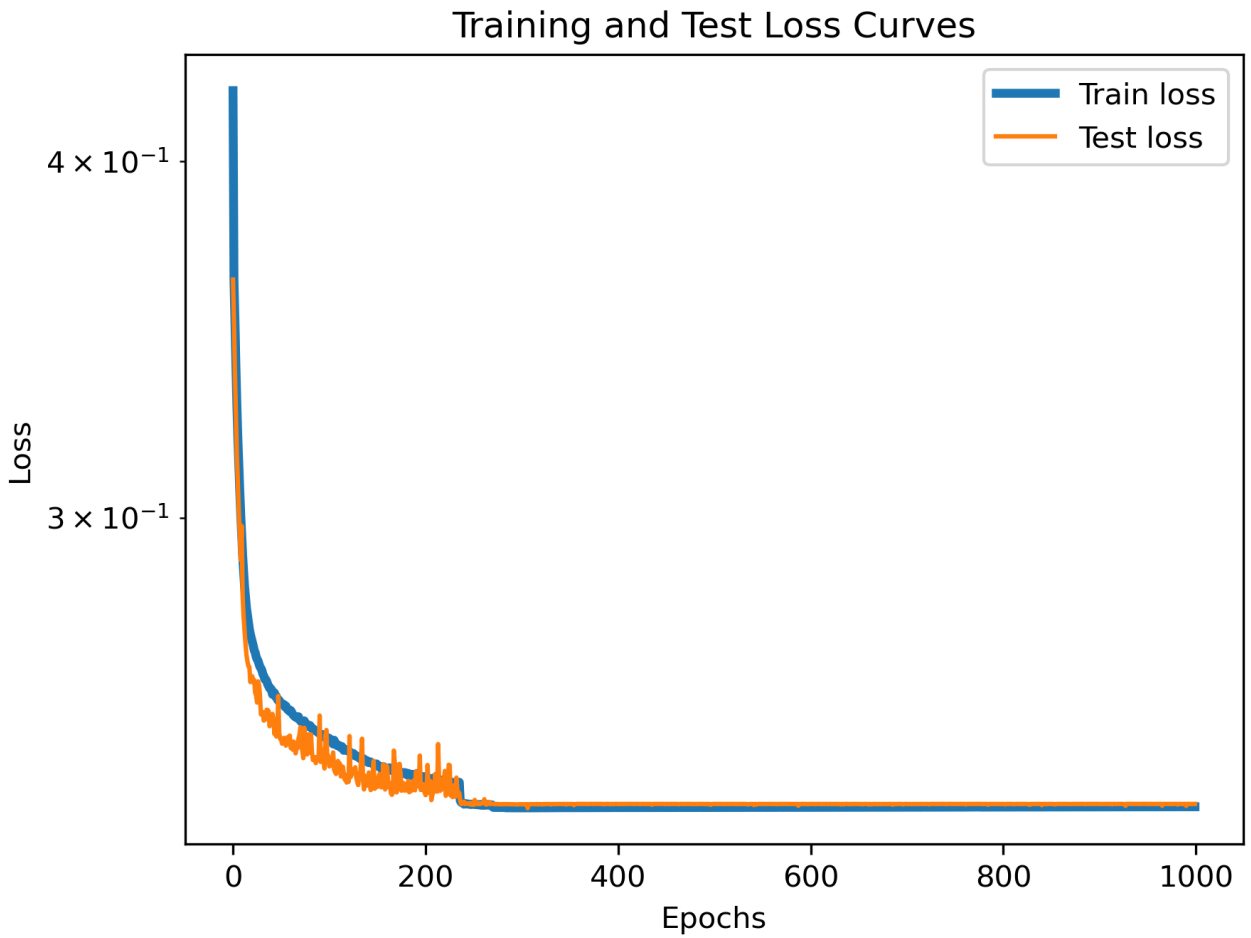
- To take in all of the data from the ECL for a τ-channel decay, we require a minimum of 24 nodes in the input layer.
- We require latency $< 500\ \mathrm{ns}$ due to limited buffer size.
- In principle, we aim for the lowest FPGA resource utilization possible without degrading efficiency or latency. In practice, the first target was simply $< 100\%$, and the present target is around 50%.

Model Structure

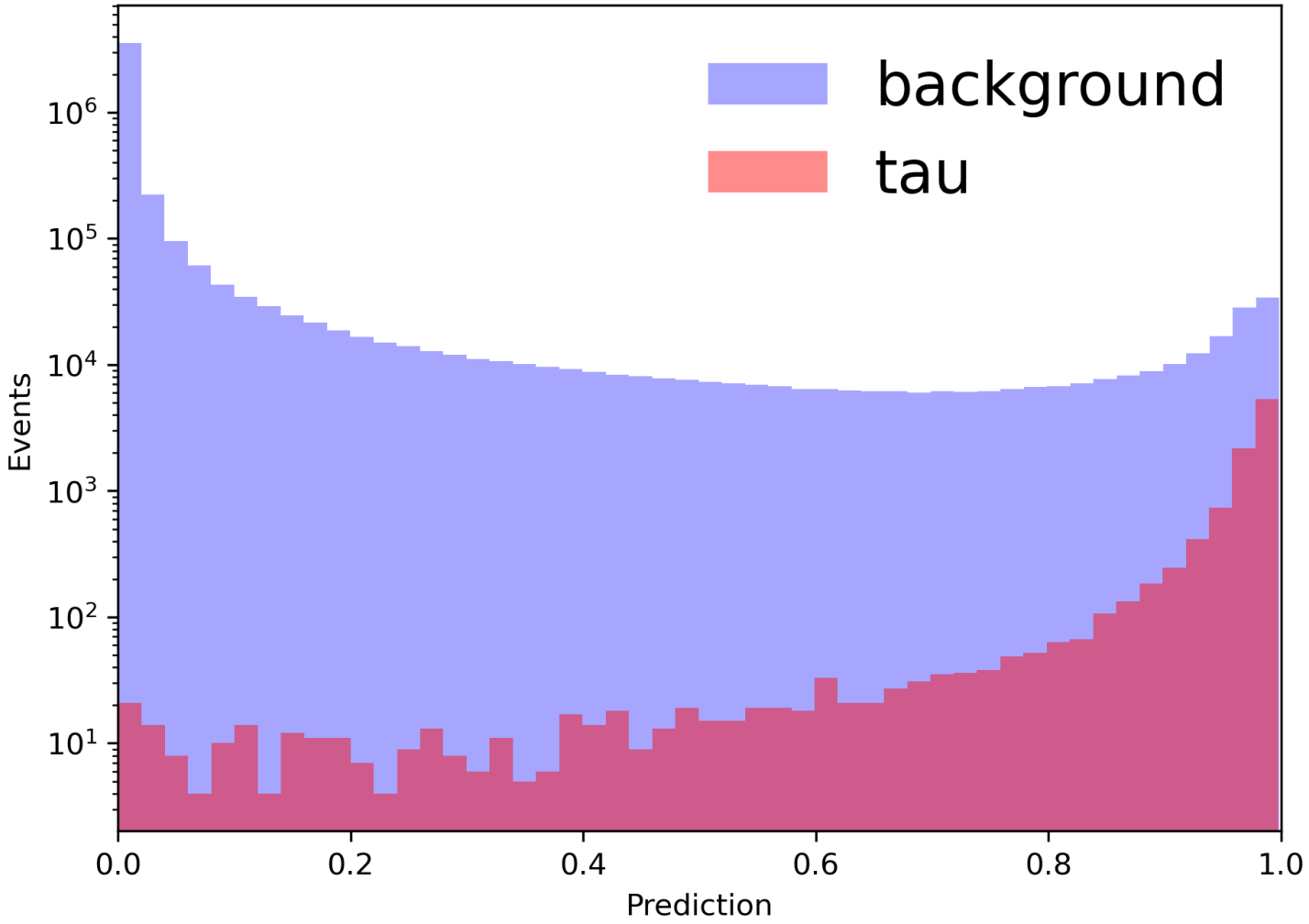


A feed-forward dense neural network with two hidden layers

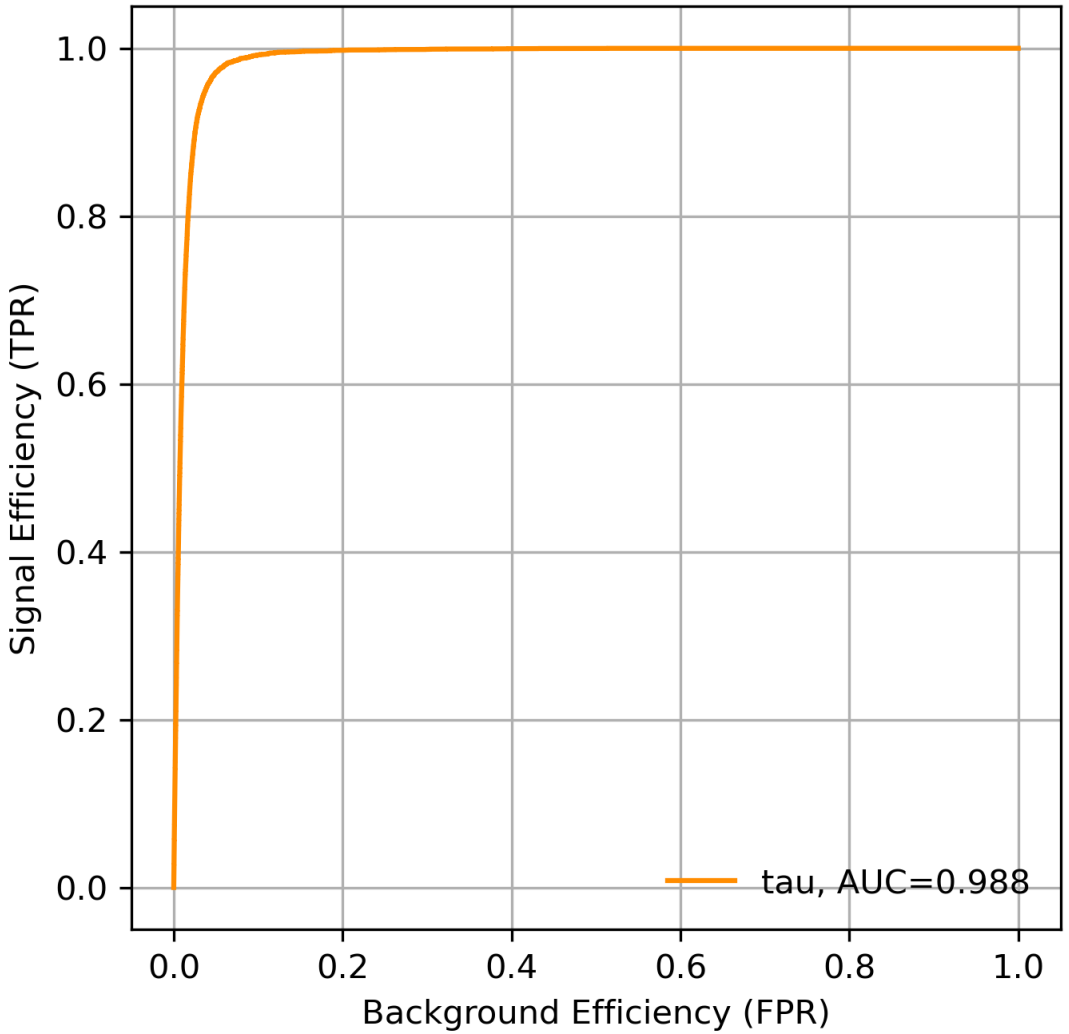
Training


Training and Test Loss Curves
Train loss
Test loss
Loss
4×10^{-1}
3×10^{-1}
0
200
400
600
800
1000
Epochs

Evaluation


background
tau
Events
10^6
10^5
10^4
10^3
10^2
10^1
0.0
0.2
0.4
0.6
0.8
1.0
Prediction

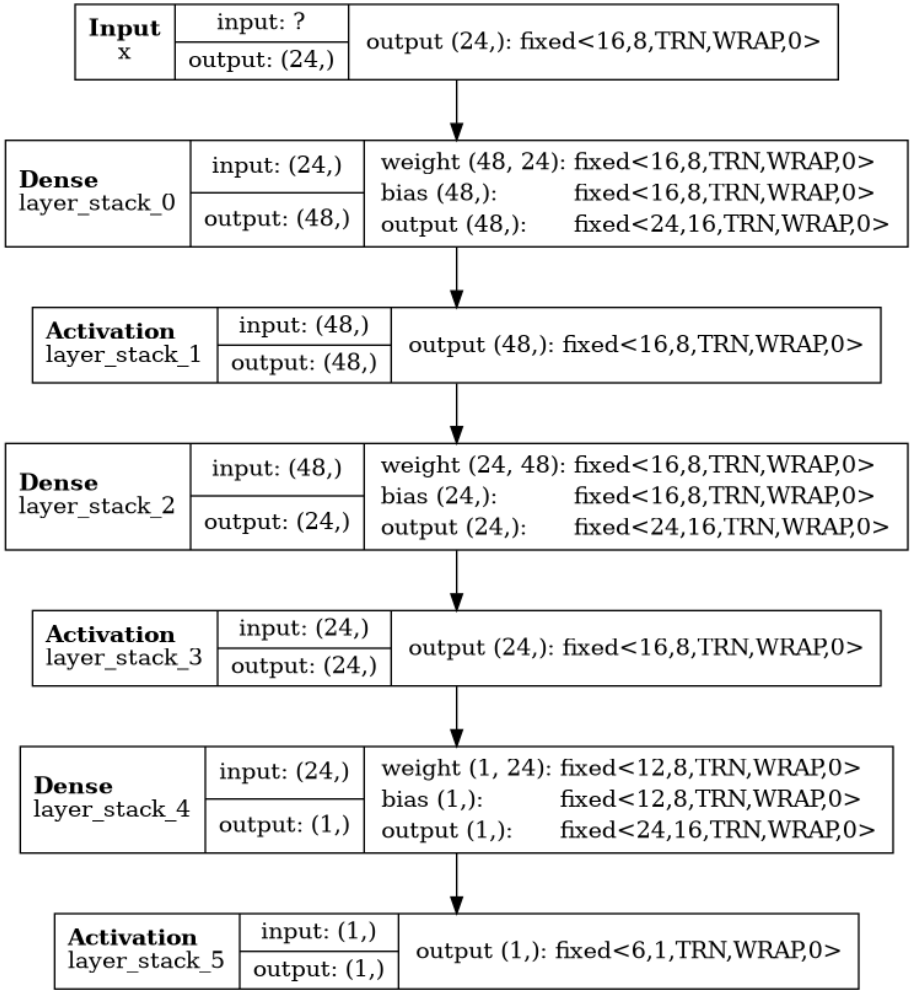
Efficiency


1.0
0.8
0.6
0.4
0.2
0.0
Signal Efficiency (TPR)
0.0
0.2
0.4
0.6
0.8
1.0
Background Efficiency (FPR)
tau, AUC=0.988

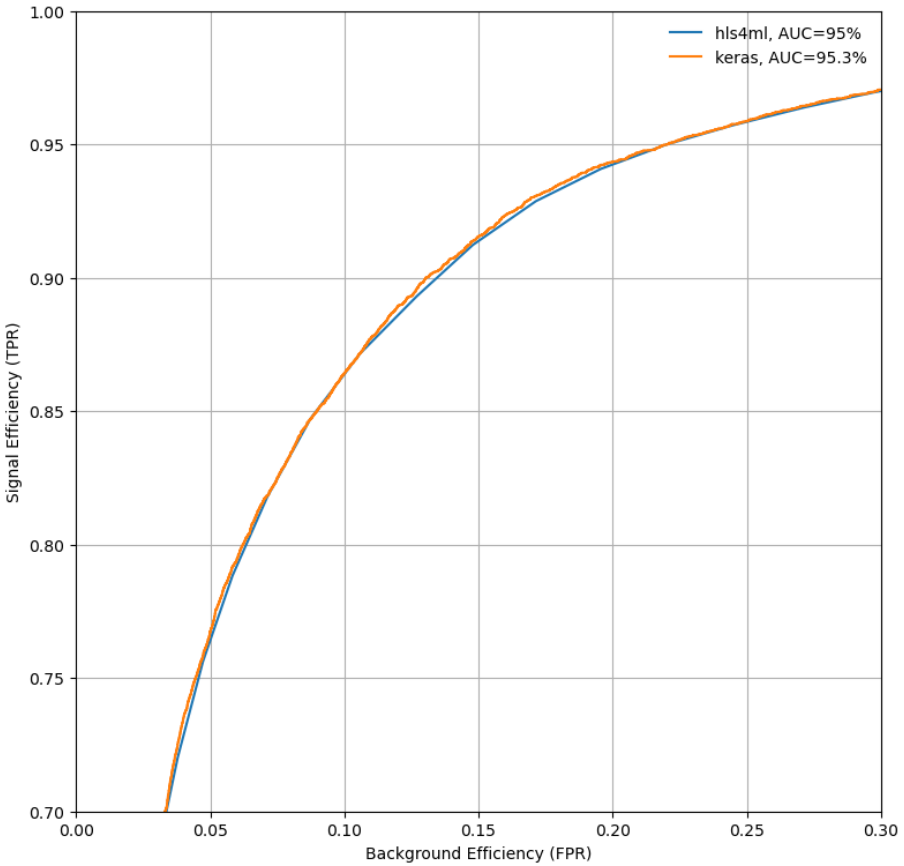
High-Level Synthesis

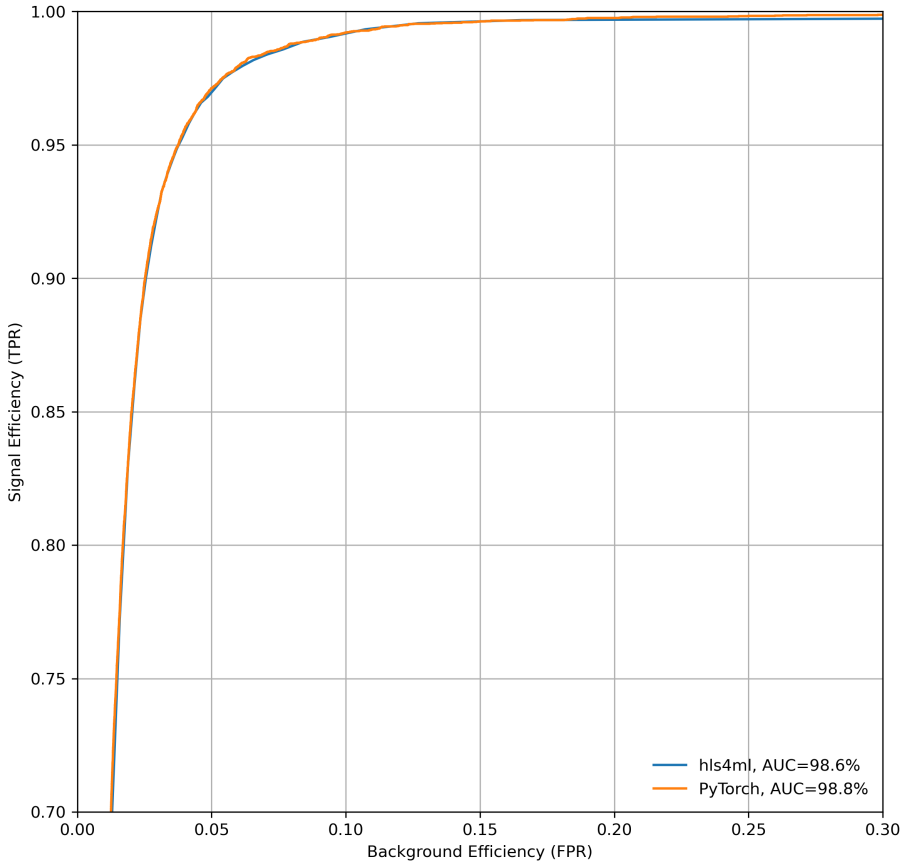
Synthesis

- We generate FPGA firmware from the PyTorch model using high-level synthesis (hls4ml).
- In this process, we can configure bitwidths for weights, biases, and activations for each layer independently.
- We can choose to optimize for either latency or hardware utilization depending on relative constraints for each.



Performance





Nomaru 2023 → Current

Latency & Utilization Estimates

Latency (cycles)		Latency (absolute)		Interval		Pipeline
min	max	min	max	min	max	Type
20	23	0.100 us	0.115 us	4	4	dataflow

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	594	-
FIFO	0	-	725	4060	-
Instance	261	587	26626	51320	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	1296	-
Register	-	-	144	-	-
Total	261	587	27495	57270	0
Available	2842	672	891424	445712	0
Utilization (%)	9	87	3	12	0

Optimization

Quantization (Methods)

Uniform Quantization

$$Q(r) = \text{Int}(r/S) - Z$$

Symmetric & Asymmetric Quantization

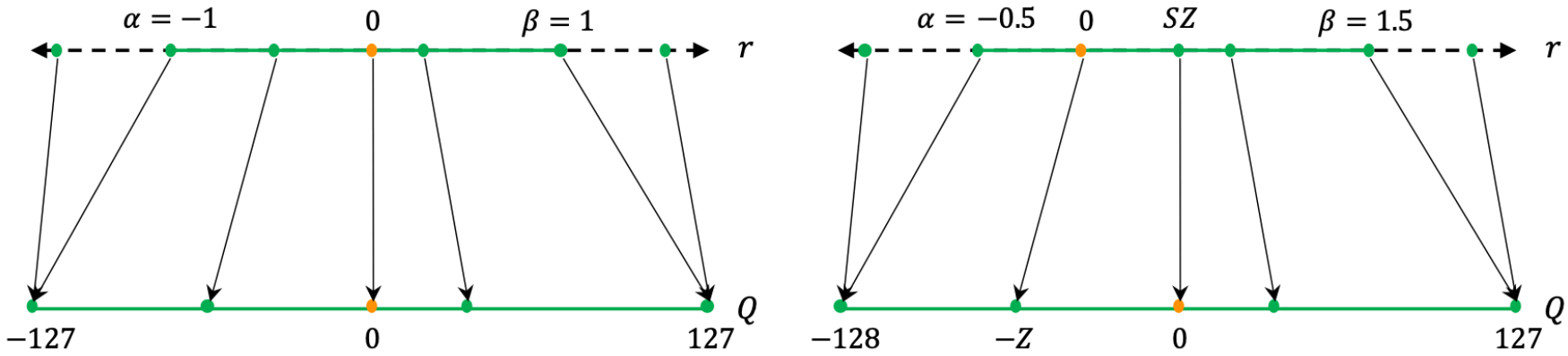


Figure 2: *Illustration of symmetric quantization and asymmetric quantization. Symmetric quantization with restricted range maps real values to [-127, 127], and full range maps to [-128, 127] for 8-bit quantization.*

(Gholani et. al 2021)

Quantization (PyTorch)

Post-Training Static Quantization (int8)

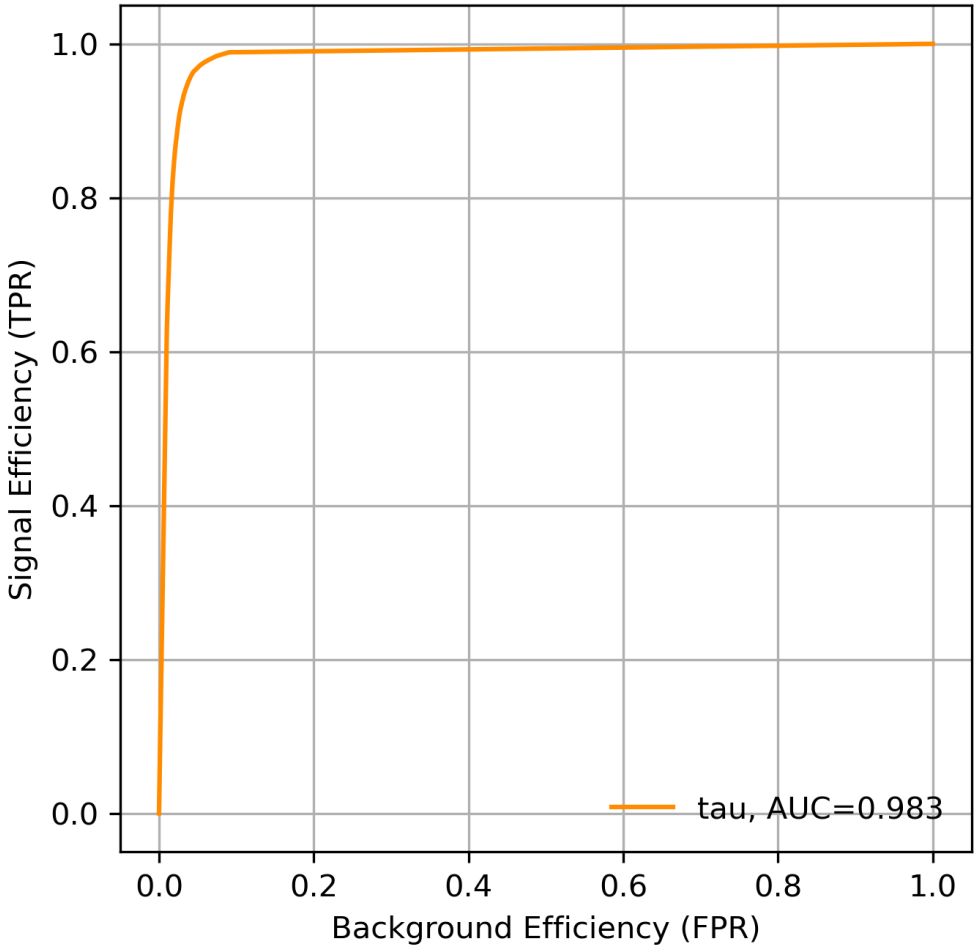
```
================================================
Quantized Model Size(Mb): 0.009874
Unquantized Model Size(Mb): 0.012074
Quantized Model is smaller by 18.22%.
Accuracy of Quantized Model: 204341.34179510426
Accuracy of Unquantized model: 204341.38712601995
Average Inference time of Quantized Model: 36.60524570941925
Average Inference time of Unquantized Model: 37.378313064575195
Quantized Model is faster by 2.07%.
================================================
```

Problem: Natively quantized PyTorch modules are not supported by hls4ml.

Solution: Use **Brevitas** and export to Open Neural Network Exchange (ONNX) format before synthesis.

Quantization (Brevitas)

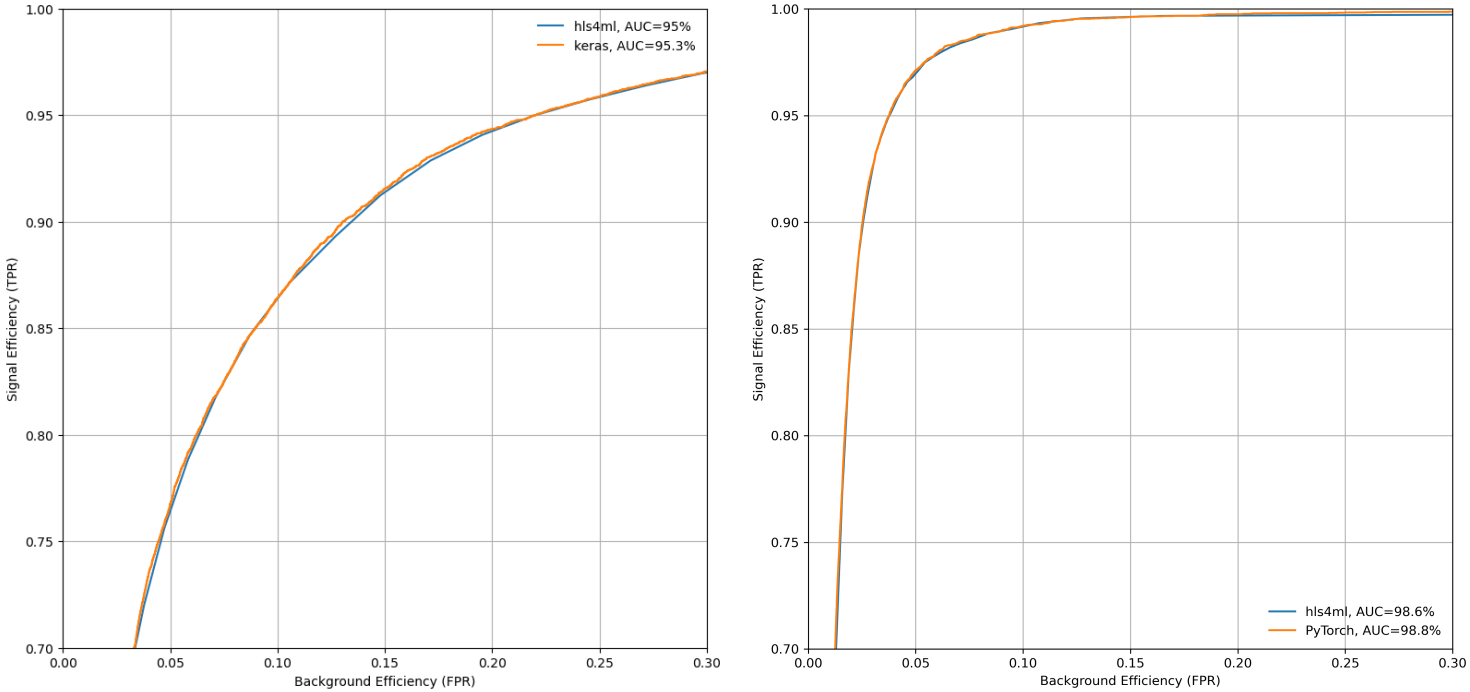
Quantization-Aware Training (int8)



Conclusion

Summary

- Continuing development of new ML-based τ firmware trigger using ECL data
- Current model shows considerable improvement over previous study, which already outperforms existing trigger algorithms



$$\text{AUC} \approx 0.95 \rightarrow \text{AUC} \approx 0.99$$

Next Steps

(Continued) Optimization

- Obtain updated latency and resource utilization estimates
- Re-train quantized model with data from loose trigger run
- Explore advanced quantization techniques (**HQG**)
- Hyperparameter optimization (**RayTune/Optuna**)

(Continued) Evaluation

- Evaluate model performance after QAT with new data
- Compute efficiencies for each final state separately

Firmware Implementation

- Generate IP core using Vivado/Vitis HLS