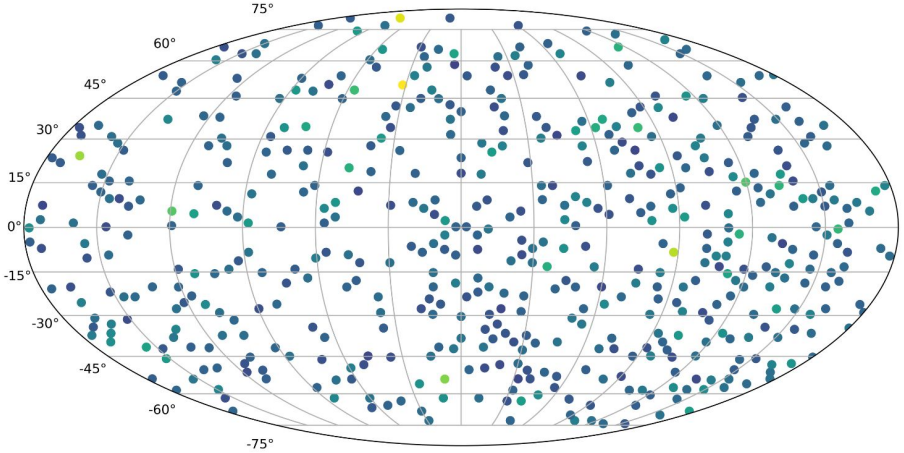
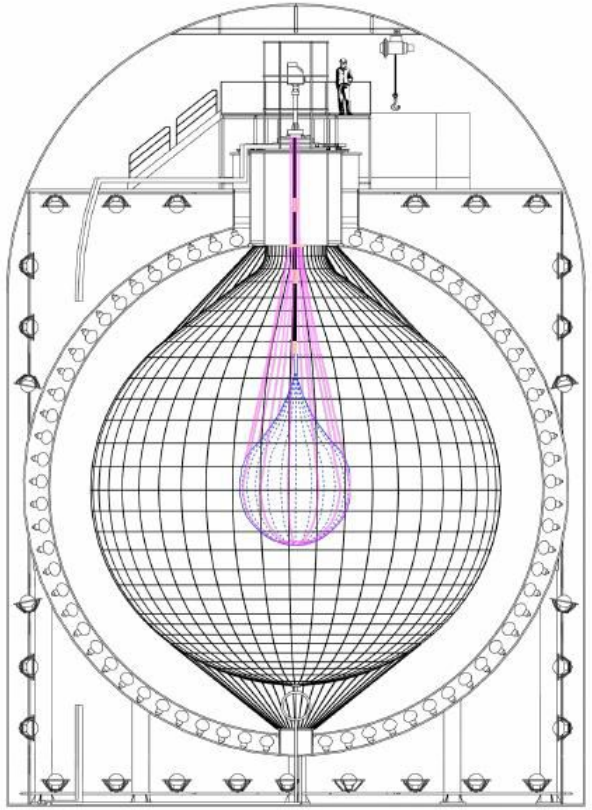
# Modern Machine Learning Model Deployment on FPGA for KamLAND-Zen

Zepeng Li

# KamLAND-Zen detector

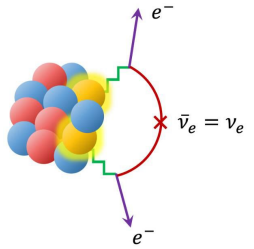
- Particles interact in the liquid scintillator and deposit energy. Energy is converted into light and detected by photo-multipliers.
- Energy resolution: 6.7%/sqrt(E (MeV))
- Vertex resolution: ~13.7 cm

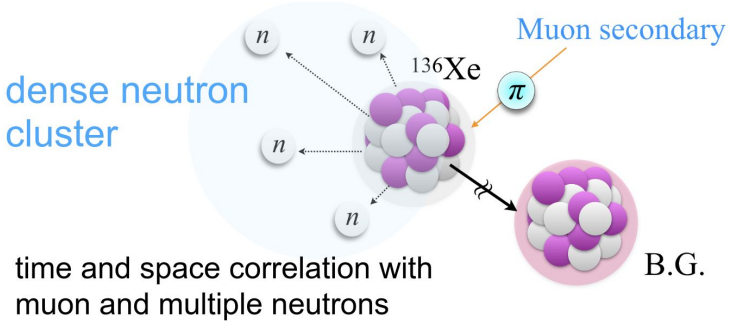


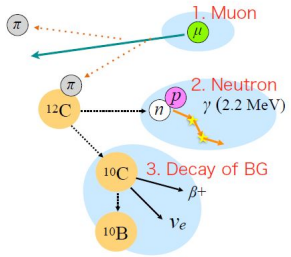



# Machine learning based background rejection in KamLAND-Zen

- Signal is 2.46 MeV electron events
- Primary backgrounds:
  - 2vbb decays
  - Long-lived cosmic muon spallation
- Minor backgrounds
  - Radioactive background
  - Solar neutrinos
  - Short-lived cosmic muon spallation

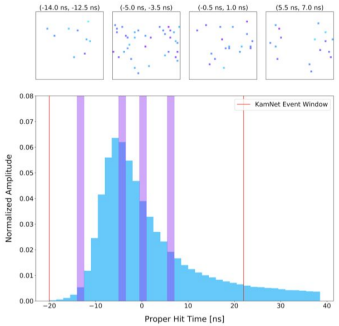


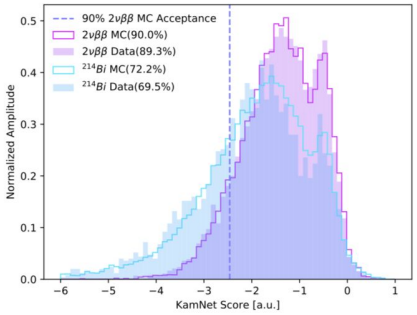



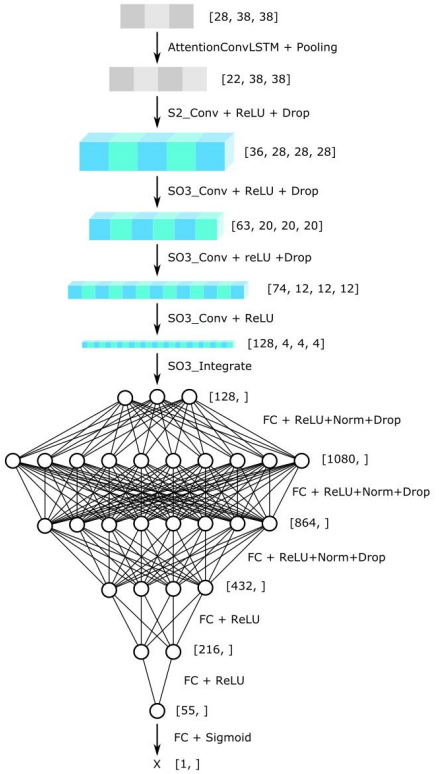



# Machine learning based background rejection in KamLAND-Zen

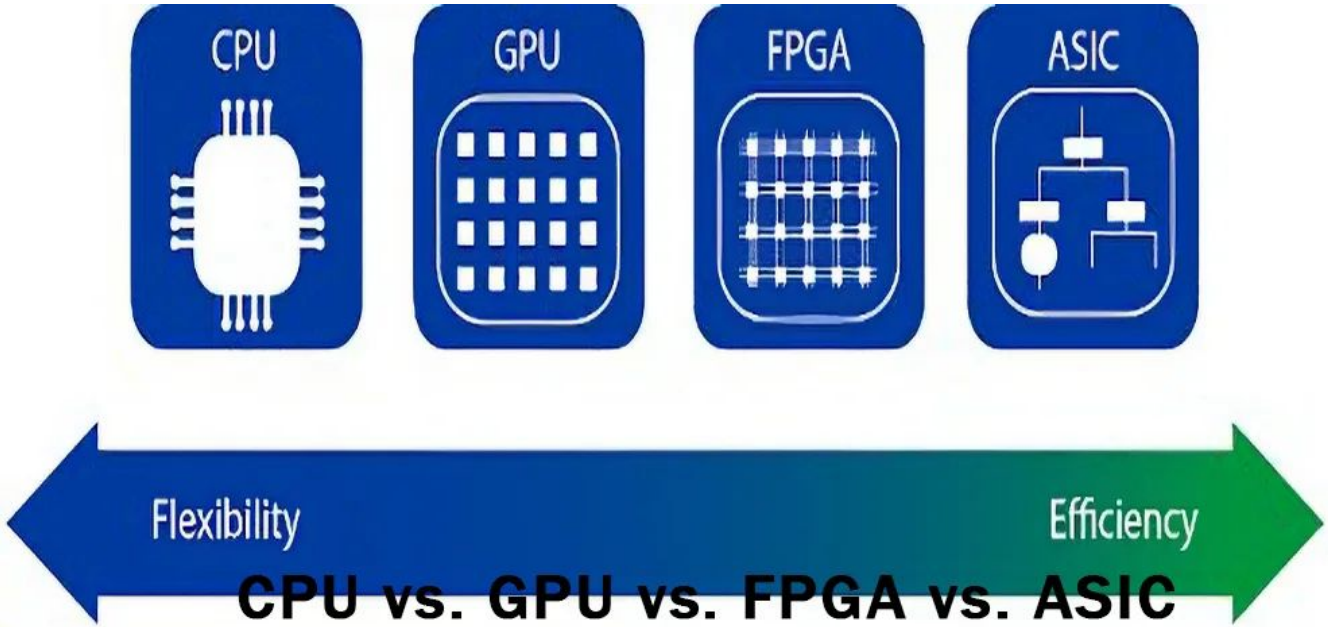
- A novel deep learning model to distinguish backgrounds and signals
- KamNettakes a time-series of 2-D hit maps and returns a single-valued KamNetScore
- Convolutional-LSTM (Long-Short Term Memory) Layer with attention module
  - Learns to identify and focus in on important sections of the event
- Spherical Convolution
  - Utilizes spherical symmetry to learn complex features



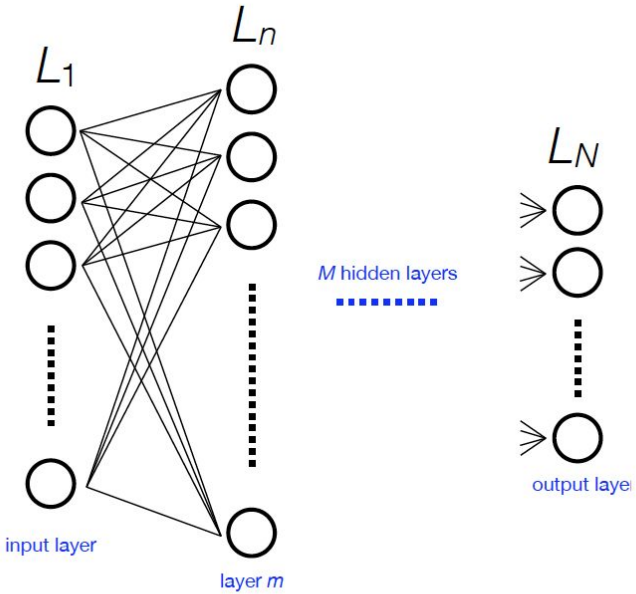
# Different Integrated Circuits




Offline analysis

Data acquisition

# Machine learning deployment

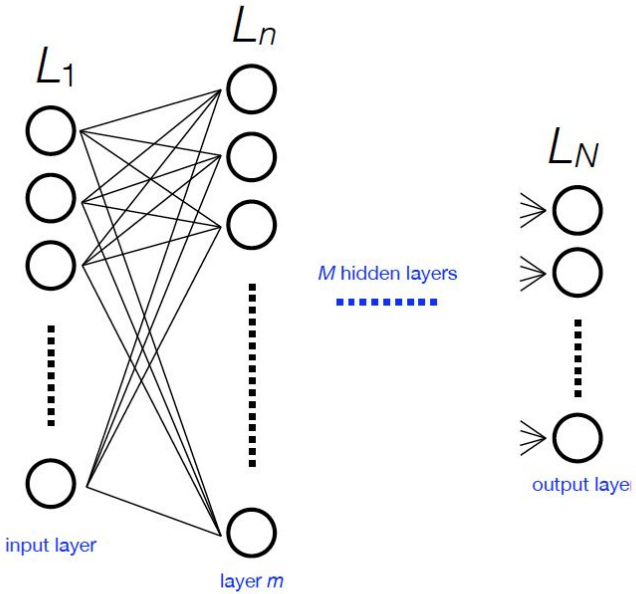



$$\mathbf{x}_n = g_n(\mathbf{W}_{n,n-1}\mathbf{x}_{n-1} + \mathbf{b}_n)$$

Activation function multiplications addition

- Most of computations are matrix/vector multiplication.
- On CPU, it is performed by looping over elements.
- Matrix multiplication is broken into independent operations in parallel on GPU.
- Hardware programming is not needed on either CPU or GPU.

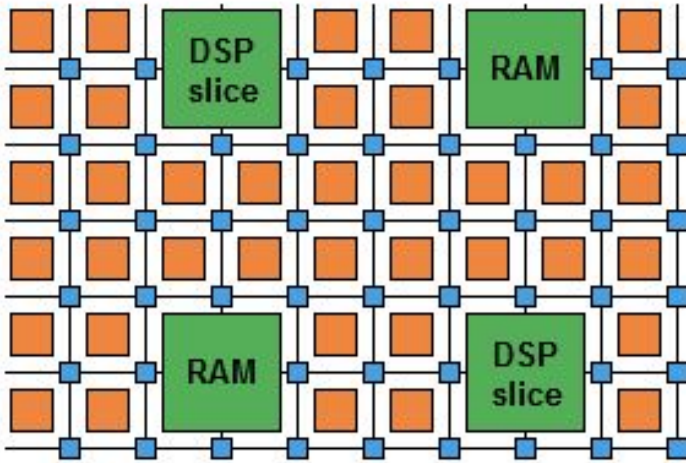
# Machine learning deployment




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

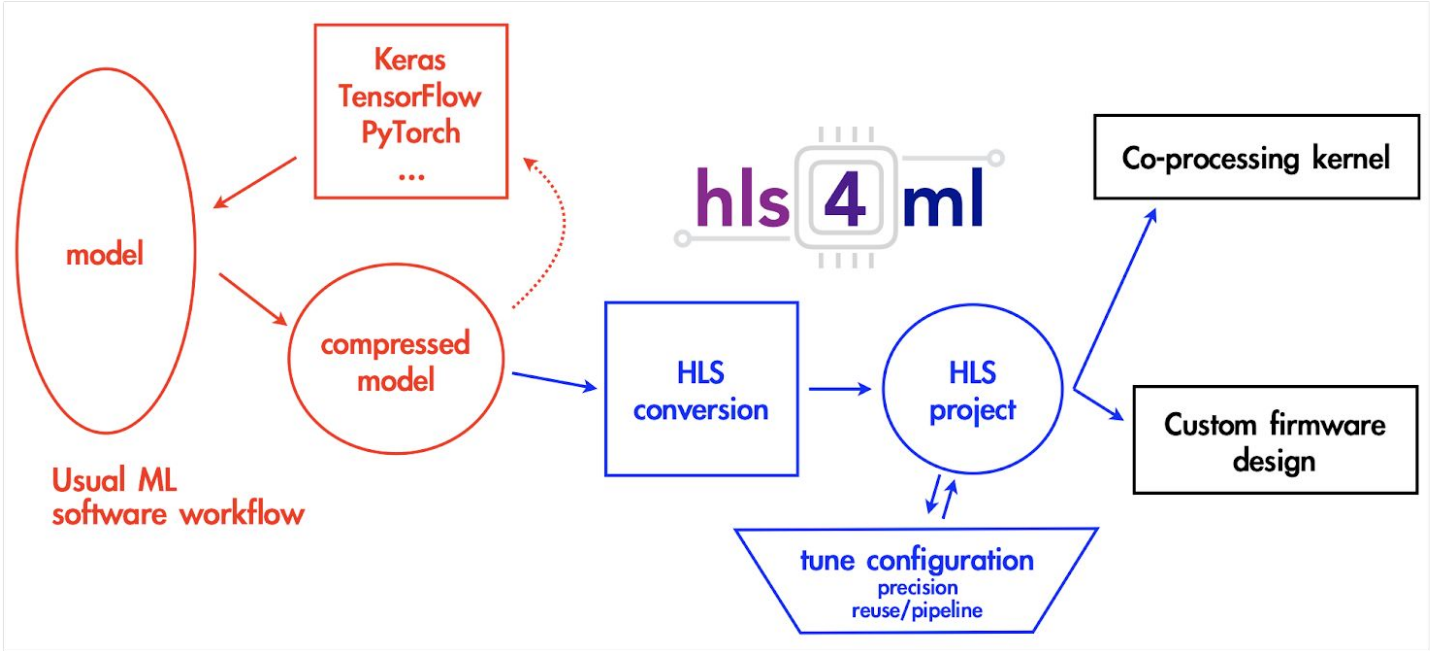
Activation function multiplications addition

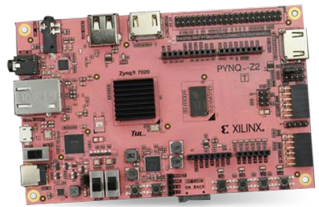
FPGA




BRAMs: precomputed activation functions
DSP: multiplication
Logic cells: addition

# HSL4ML

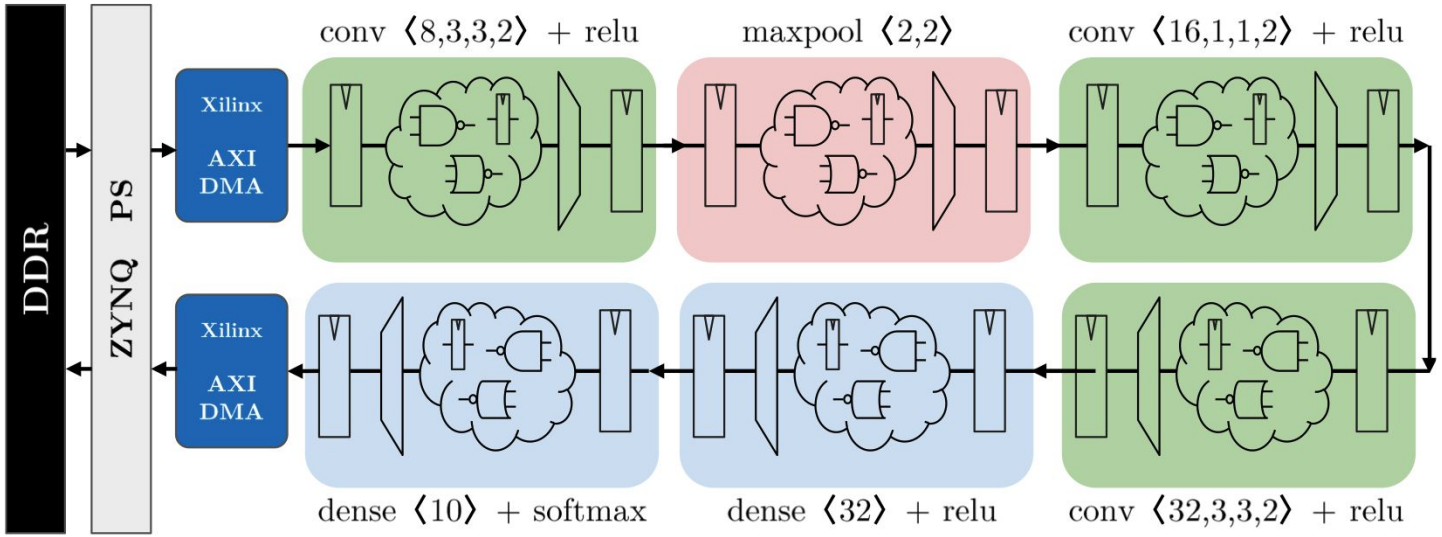


Keras, tensorflow, pytorch, and ONNX

Vivado, HLS compiler

# HLS4ML




Layer-by-layer implementation of neural networks in HLS4ML.

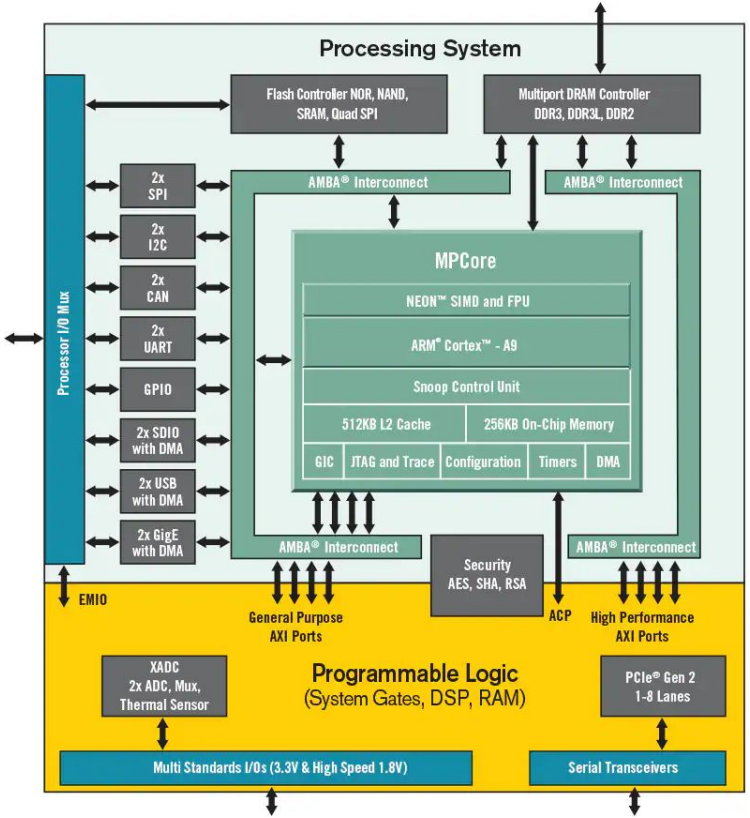
Multiplier could be reused inside a layer.

# Modern ML models on FPGA

| layer name | output size | 18-layer | 34-layer | 50-layer | 101-layer | 152-layer |
|---|---|---|---|---|---|---|
| conv1 | 112×112 | 7×7, 64, stride 2 | | | | |
| conv2_x | 56×56 | 3×3 max pool, stride 2 | | | | |
| | | $\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$ | $\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$ | $\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$ | $\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$ | $\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$ |
| conv3_x | 28×28 | $\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$ | $\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$ | $\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$ | $\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$ | $\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$ |
| conv4_x | 14×14 | $\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$ | $\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$ | $\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$ | $\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$ | $\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$ |
| conv5_x | 7×7 | $\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$ | $\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$ | $\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$ | $\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$ | $\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$ |
| | 1×1 | average pool, 1000-d fc, softmax | | | | |
| FLOPs | | $1.8\times10^9$ | $3.6\times10^9$ | $3.8\times10^9$ | $7.6\times10^9$ | $11.3\times10^9$ |

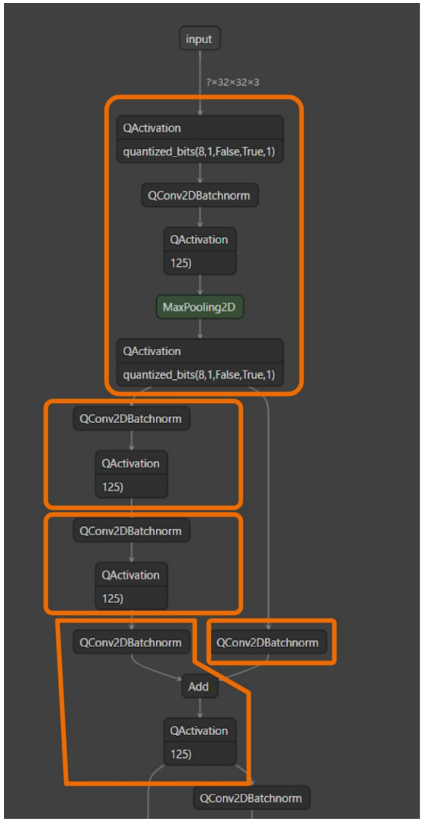
HLS4ML does not support modern neural networks.

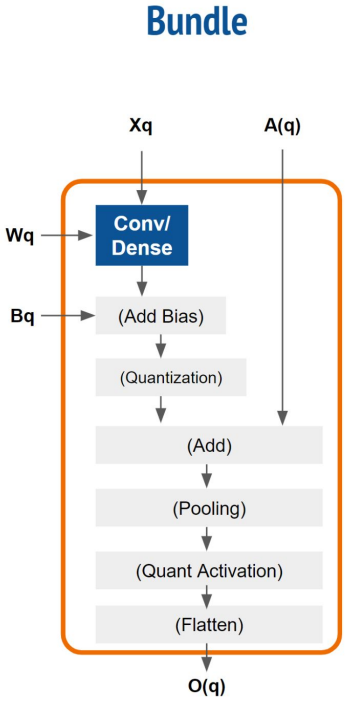
# ZYNQ heterogeneous SoC

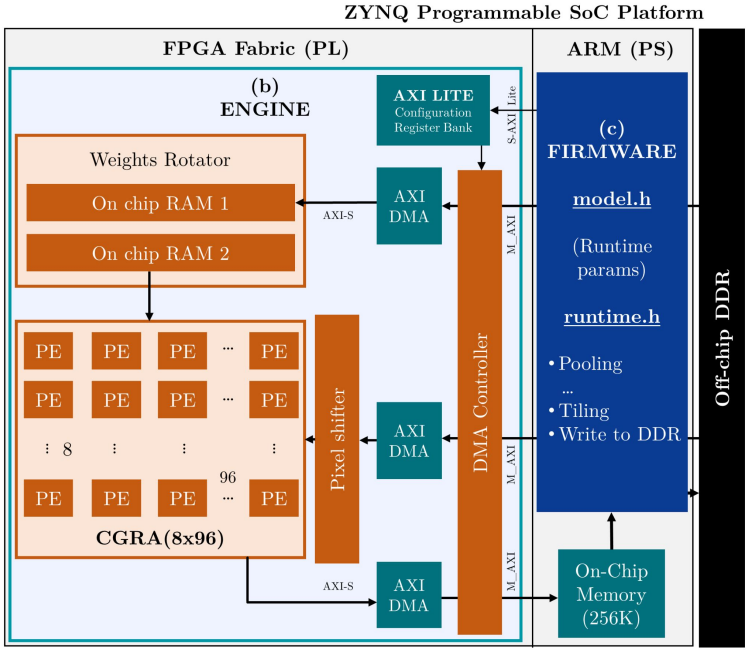



- System on Chip: Arm CPU + FPGA
- Advanced eXtensible Interface (AXI) provides for high bandwidth and low latency connections between elements.

# Modern ML models on FPGA using CGRA4ML




Parameterizable Coarse-Grained Reconfigurable Array

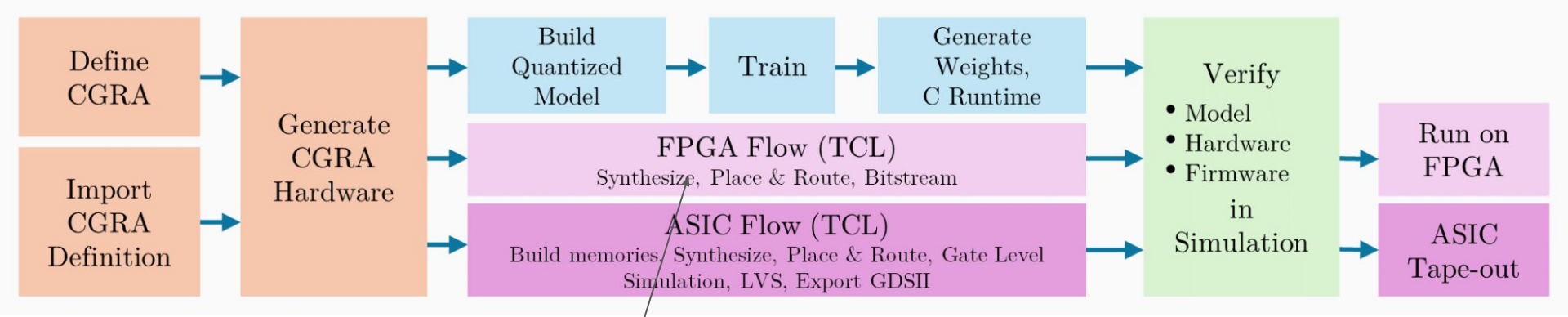
PE: processing element

https://arxiv.org/pdf/2408.15561

# Modern ML models on FPGA using CGRA4ML

Model development and optimization in python

Vitis




Reconfigurable for different tasks and FPGAs

Vivado

# Results

| Model | ResNet-50 | PointNet |
|---|---|---|
| Bits | 4 | 4 |
| PEs | (7,96) | (32,32) |
| Frequency (MHz) | 250 | 250 |
| FFs | 101706 | 69277 |
| LUTs | 82200 | 100076 |
| BRAMs | 6 | 4.5 |
| Static Power (W) | 0.700 | 0.700 |
| Dynamic Power (W) | 3.847 | 3.840 |
| Total Power (W) | 4.547 | 4.540 |
| GOPs/W | 37.3 | 56.8 |

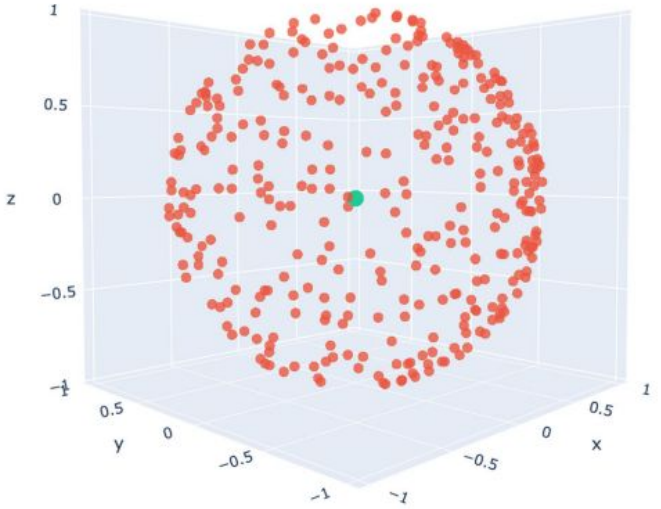
TABLE III
IMPLEMENTATION OF RESNET-50 AND POINTNET ON ZCU104 FPGA

# Methods

# PointNET event reconstruction

- Data can be thought of as a point cloud
  - Geometric semantics
  - Invariant to permutations (x, y, z encoded)
- Use the PointNET architecture (Qi et al 2017)

2126 PMTs (ID + OD)




(Fu et al 2024)

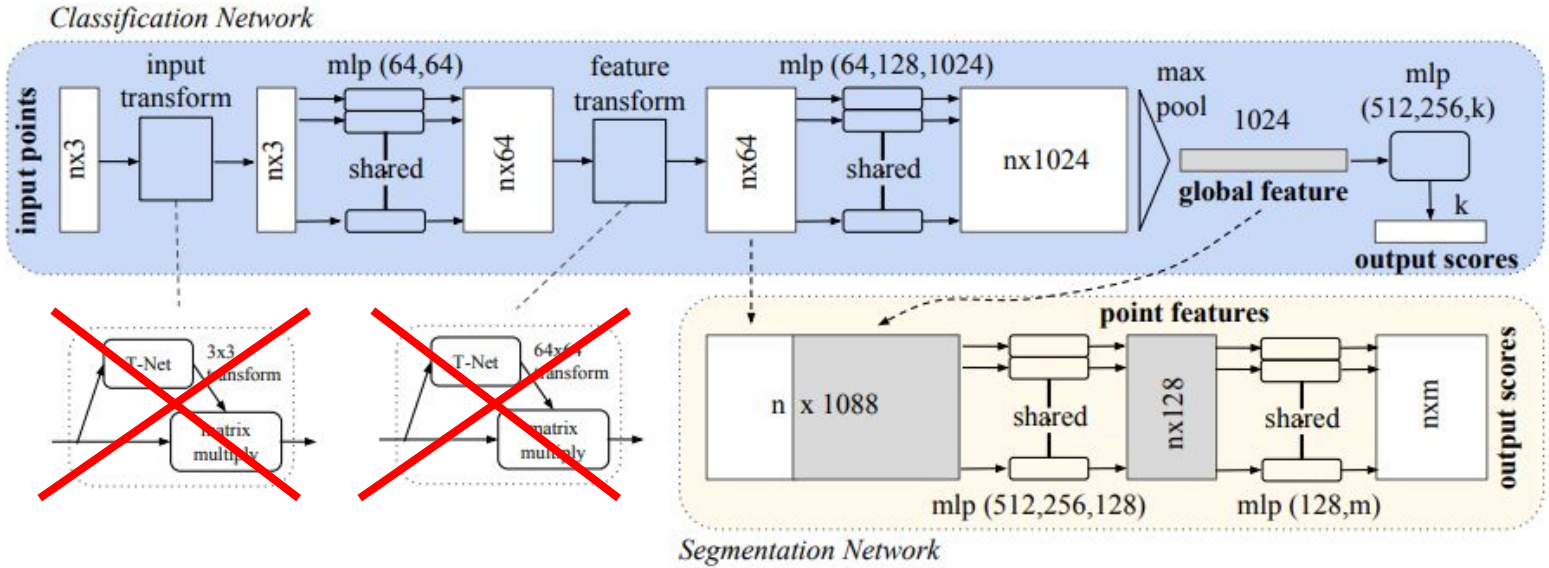
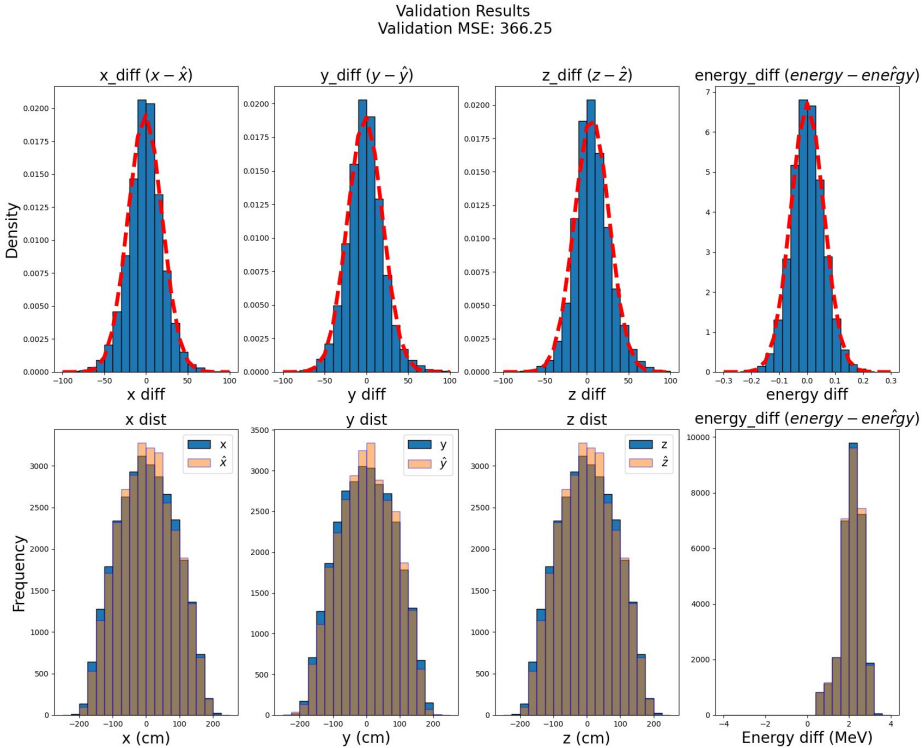


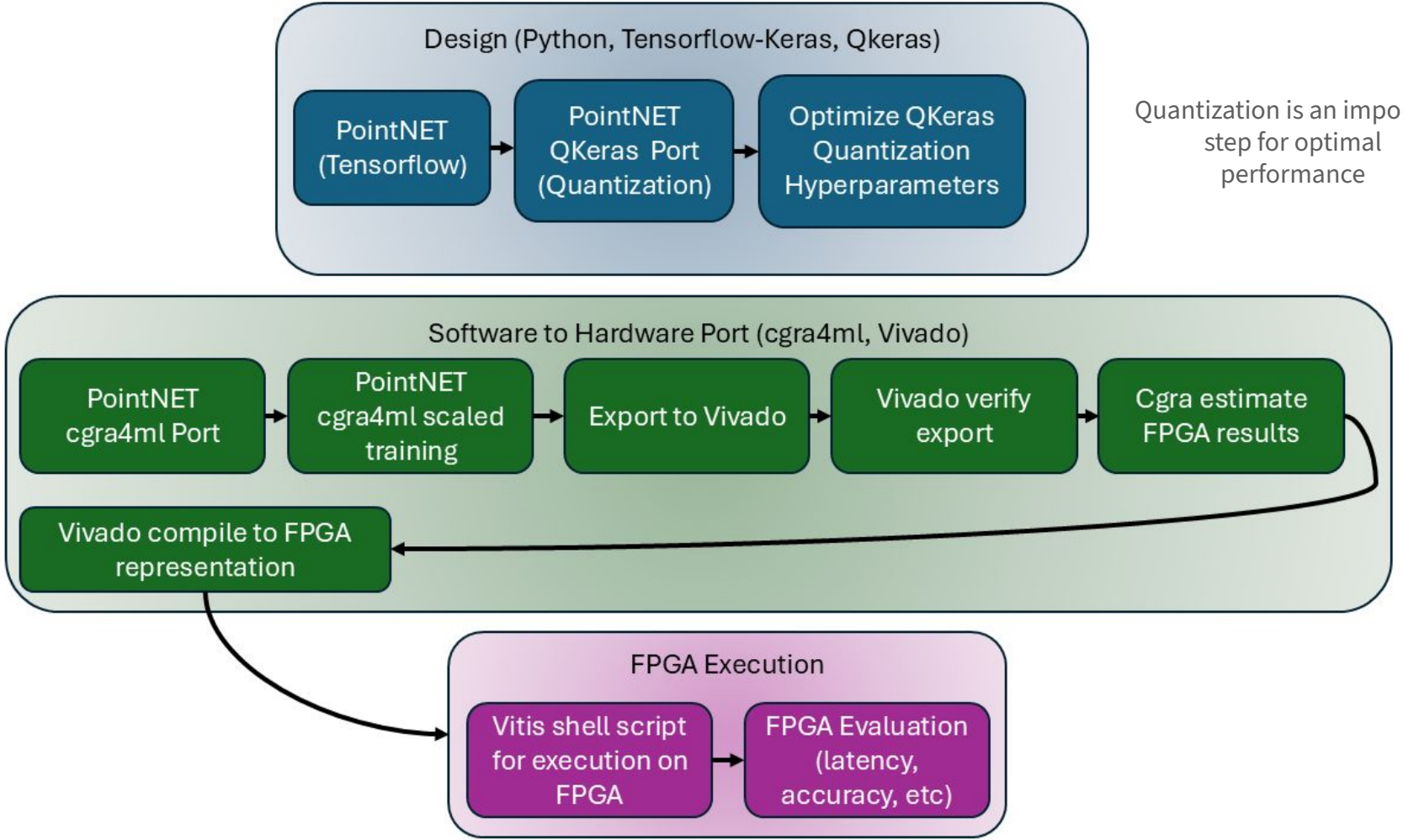

Figure 2. **PointNet Architecture.** The classification network takes $n$ points as input, applies input and feature transformations, and then aggregates point features by max pooling. The output is classification scores for $k$ classes. The segmentation network is an extension to the classification net. It concatenates global and local features and outputs per point scores. "mlp" stands for multi-layer perceptron, numbers in bracket are layer sizes. Batchnorm is used for all layers with ReLU. Dropout layers are used for the last mlp in classification net.

(Qi et al 2017)

Model is (sequential):
3 CNNs
Global Max Pool
3 Fully Connected

# Design Stage Reconstruction Results






Quantization is an important step for optimal performance

# Quantization

# Why Quantization?

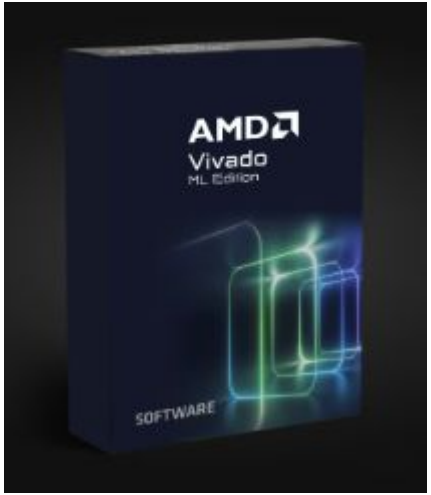
- Need to compress model
- Need to understand optimal hardware parameters
- Training QKeras establishes baseline for comparing hardware and software compression

```
class UserModel(XModel):
    def __init__(self, sys_bits, x_int_bits, *args, **kwargs):
        super(...) x_int_bits, *args, **kwargs)
                (variable) b0: Any
        self.b0 = XBundle(
            # core=XDense(
            #    k_int_bits=0,
            #    b_int_bits=0,
            #    units=64,
            #    act=XActivation(sys_bits=sys_bits, o_int_bits=0, type='relu', slope=0)
            # )
            core=XConvBN(
                k_int_bits=0,
                b_int_bits=0,
                filters=64,
                kernel_size=1,
                act=XActivation(sys_bits=sys_bits, o_int_bits=0, type='relu', slope=0)
            ),
        )
```

cgra port of PointNET

# Software to Hardware: Key Tools

- cgra4ml (UCSD Computer Science)
  - Converts Tensorflow model to Vivado-friendly format
  - Open-source and available on Github
- Vivado (AMD)
  - Model export verification and simulation
  - Synthesizes FPGA representation
- Vitis (AMD)
  - C-wrapper for FPGA model execution

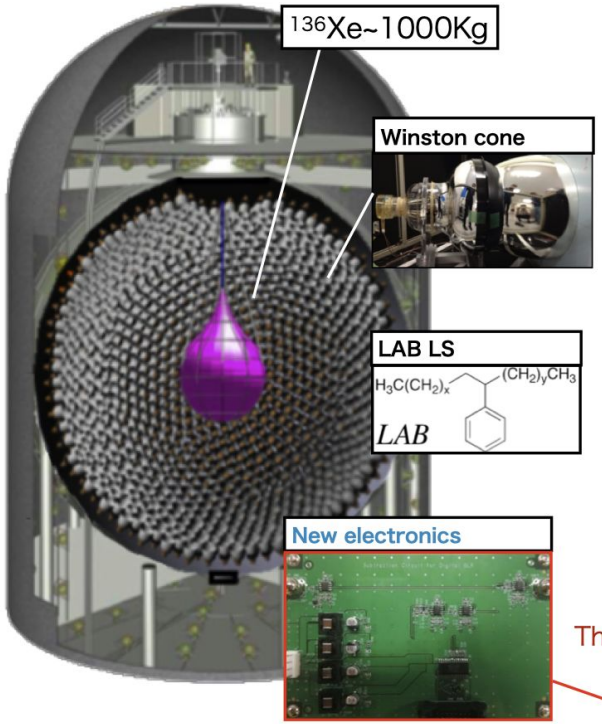



AMD Vitis

*Images courtesy of AMD*

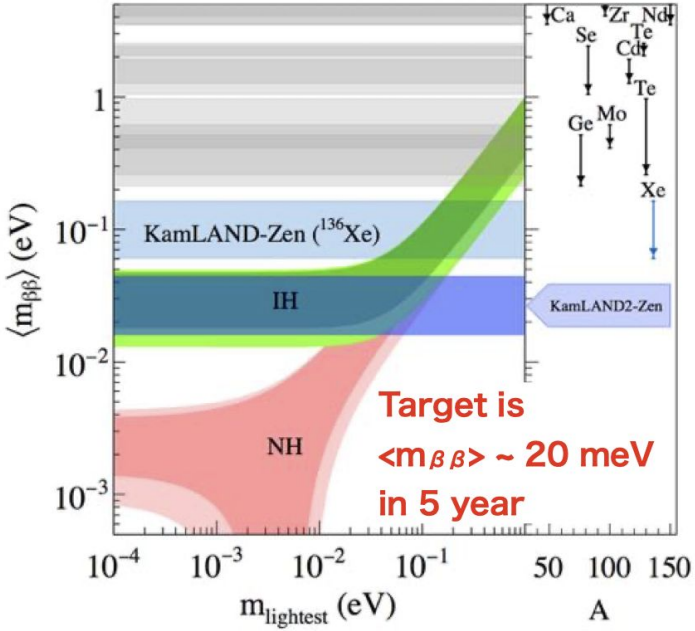
# Results

# KamLAND2-Zen




This talk

From H. Ozaki



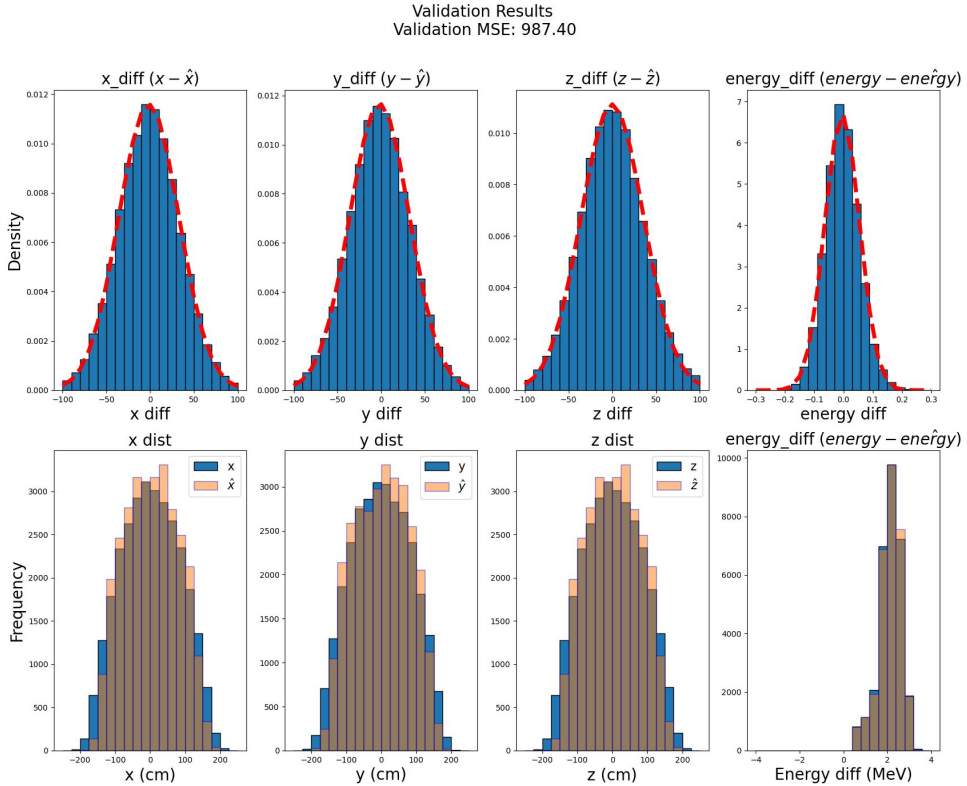

We aim to increase light collection by more than 5 times!

$\sigma$ (2.6MeV) = 4% → 2%

Other options(Scintillation film, Imaging detector, pressurized xenon ..) in development

**We are developing new electronics with wide dynamic range!**

# PointNET cgra Port Accuracy Results



# Reconstruction Results Summarized

| Experiment | Avg. Validation MSE | X Error (cm) | Y Error (cm) | Z Error (cm) | E Error (MeV) |
|---|---|---|---|---|---|
| Traditional KLZ (Li Thesis) | N/A | 17 | 17 | 17 | 0.14 |
| QKeras | 366.25 | 20 | 21 | 21 | 0.06 |
| Cgra4ml | 987.40 | 34 | 34 | 36 | 0.06 |

# RFSoC4x2

- ZYNQ Ultrascale+ FPGA in lab
- AMD Kit

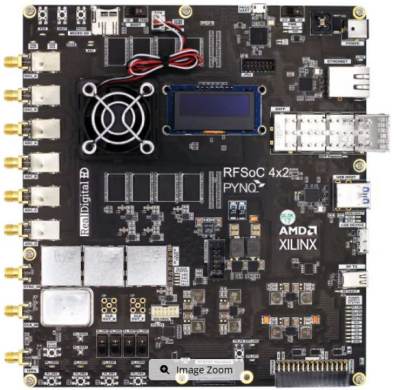
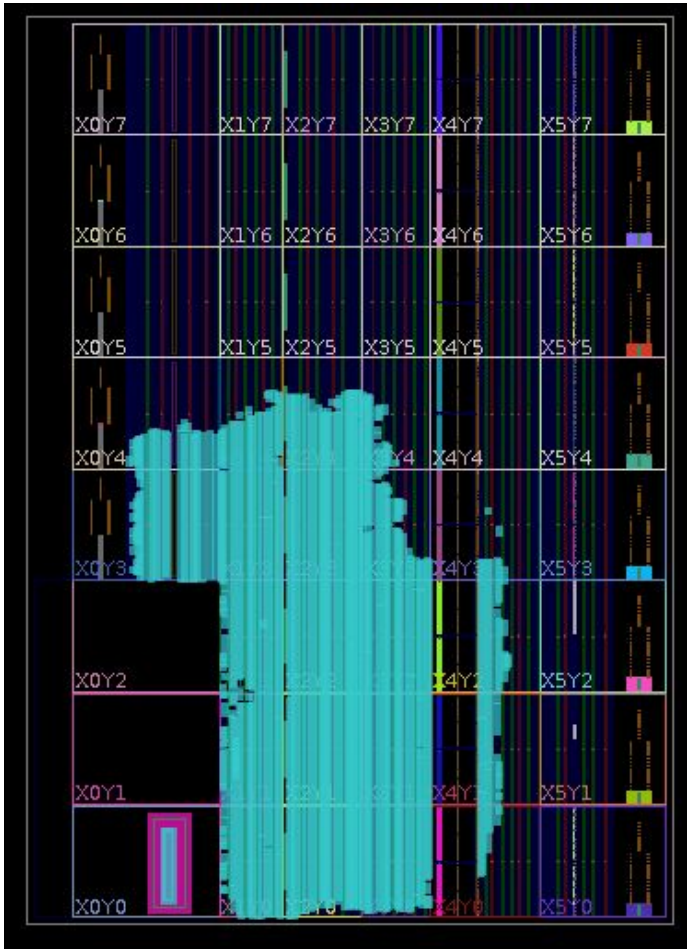



*Image courtesy of AMD*




Vivado synthesis of model

https://ml4physicalsciences.github.io/2024/files/NeurIPS_ML4PS_2024_153.pdf

| Version | Latency (ms/batch) @ 20 runs |
|---|---|
| Trained | 6980.9 |
| Untrained | 6996 |

~436.3 ms
inference per event

# Work to reduce the latency

- A simpler model without losing much accuracy.
- Use PMT cluster as a single point instead of a single PMT as input.
- Optimize quantization

# Summary

- CGRA4ML provide a framework for modern ML model deployment on FPGA.
- PointNET is an effective way of reconstructing detector physics in KLZ
- We can deploy PointNET onto an FPGA to make single-event inference on the order of 100s of ms