Empowering AI Implementation on Edge

SLAC Neural Network Library (SNL) for FPGA

- Speaker Name / Title / Department

May 19, 2025

Abhilasha Dave*, Julia Gonski, Ryan Herbst, J.J Russell, Ryan Coffee, Larry Ruckman

*adave@slac.stanford.edu

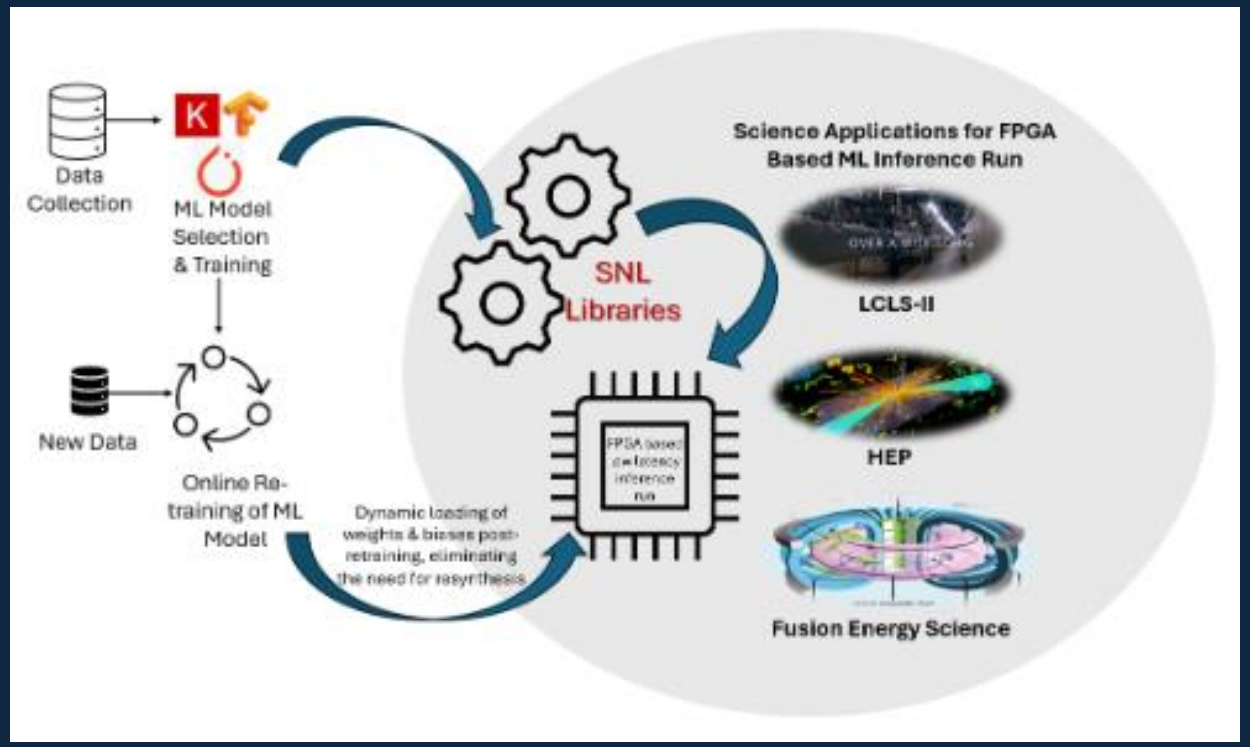
ML4FE Workshop

SLAC NATIONAL ACCELERATOR LABORATORY

Stanford University | U.S. DEPARTMENT OF ENERGY

Goal of SLAC Neural Network Library (SNL)

- SNL is well-suited for FPGA based HW environment requiring real-time, high-speed machine learning data processing.
- Examples include:
  - The 2D Beam Emission Spectroscopy (BES) system at DIII-D
  - Current and next-generation High Energy Particle (HEP) colliders



Key Points

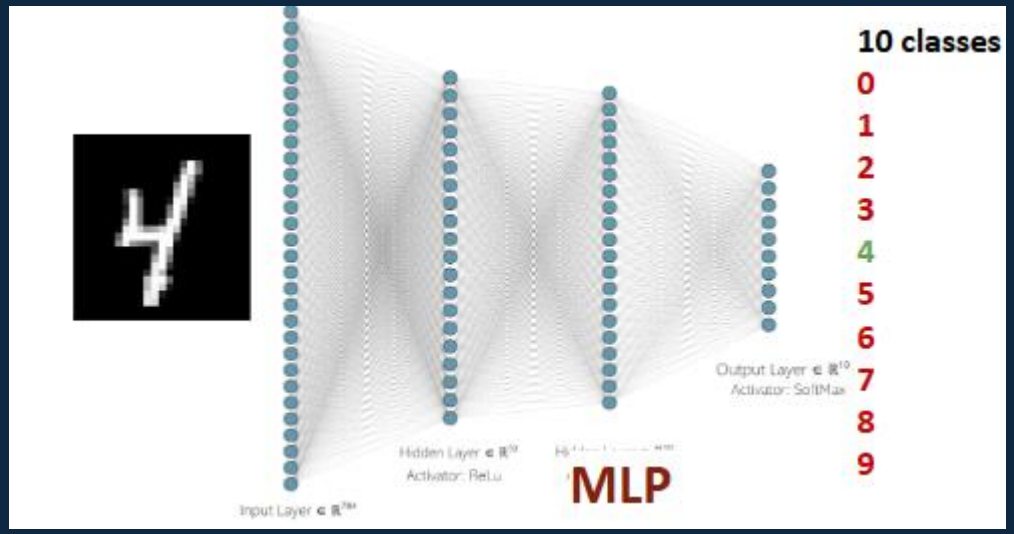
- Provides specialized set of libraries for deploying ML inference to FPGA
- Using High-Level-Synthesis (HSL): C++ programming of FPGA
- Supports Keras like API for layer definition
- **Dynamic reloading** of **weights** and **biases** to avoid re-synthesis
- Supports 10s of thousands of parameters or more depending on latency requirements for the inference model
- Total end to end latency of couple of **usec** to couple of **millisecond**.
- Streaming interface between layers.
- Allow for **pipeline** of the **data flow** for a **balance** of **latency vs frame rate**
- Library approach allows for user/application specific enhancements

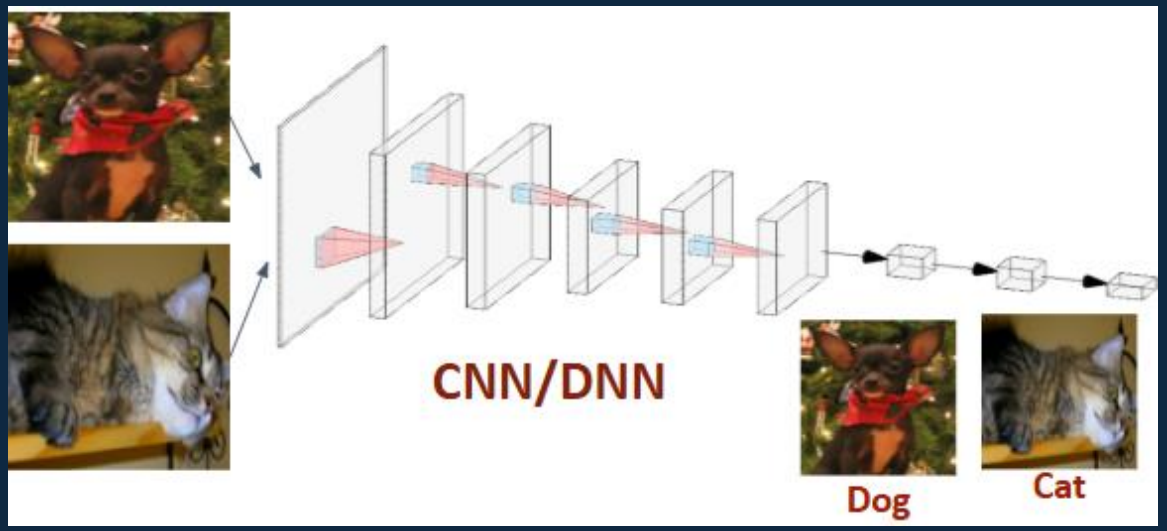
Buildable Neural Network for Inference Run using SNL & Future Support

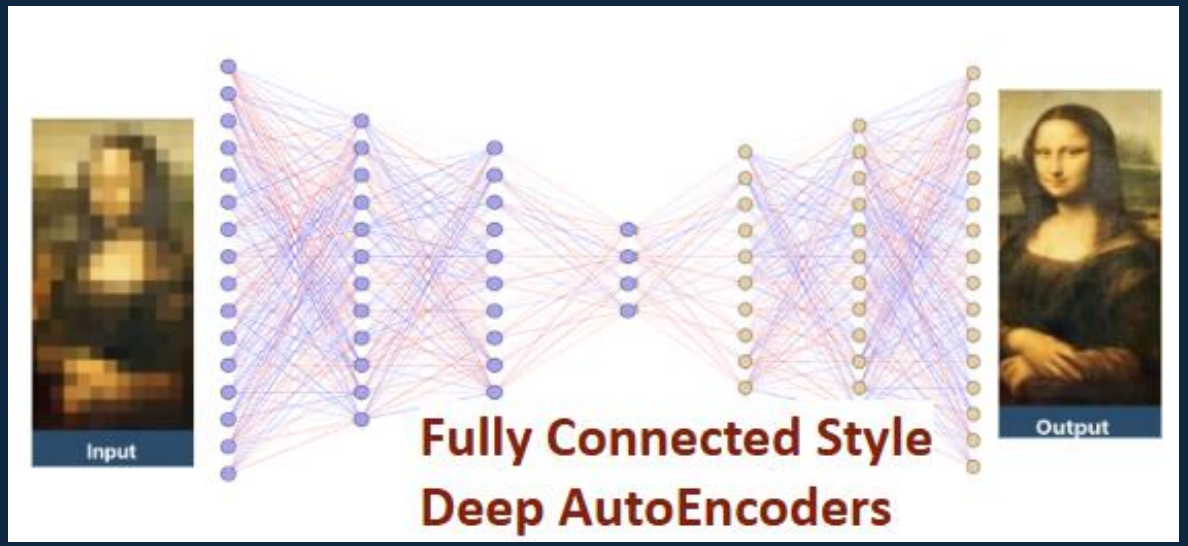
SNL continuously evolving library

Supported Libraries:

- **NN Layers:** Conv2D, Conv1D, MaxPooling2D, Average Pooling 2D, Average Pooling 1D, Dense
- **Activators:** Relu , LeakyRelu, PRelu SoftMax, Linear
- **Data Types:** Fixed point, Integer, Floating Point





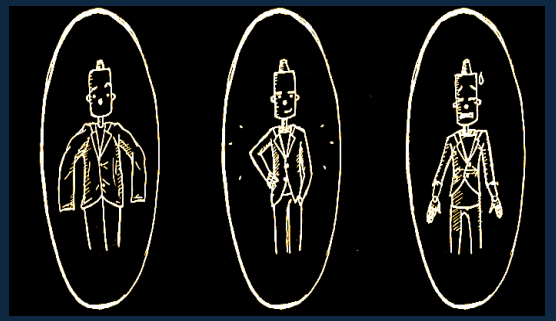


Edge Hardware Challenge

Usually, this Machine Learning models are oversized, and the biggest challenge is how to fit them on to FPGA/resource constrained HW?

Edge Hardware Challenge

A Few Strategies to Address the Challenge



*https://images.app.goo.gl/CJwExCAN8enBA3nh7

- Analyzing the model capacity & overfitting along with assessing the efficiency of parameter usage

*AI Generated Image

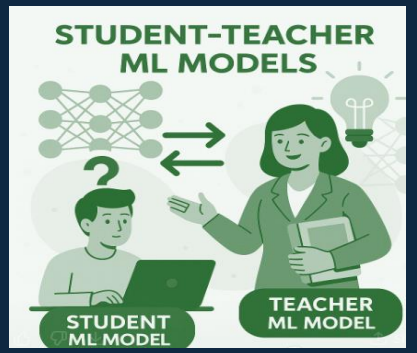
- Applying a quantization techniques to reduce the size of input, outputs, weights, biases
- Resulting into lightweight ML model

- Pruning technique to cutdown neuron and their weight connections.

Preprocessing of the input

*AI Generated Image

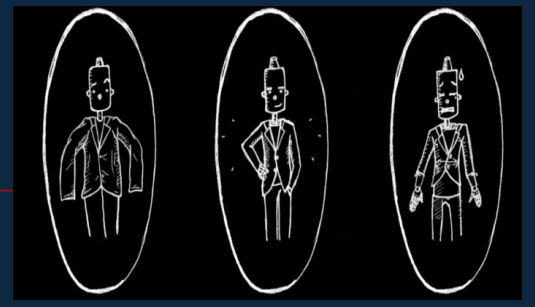
- Algorithmic research aimed at developing lightweight, efficient ML models optimized for resource-constrained hardware, enabling faster inference with minimal accuracy loss.

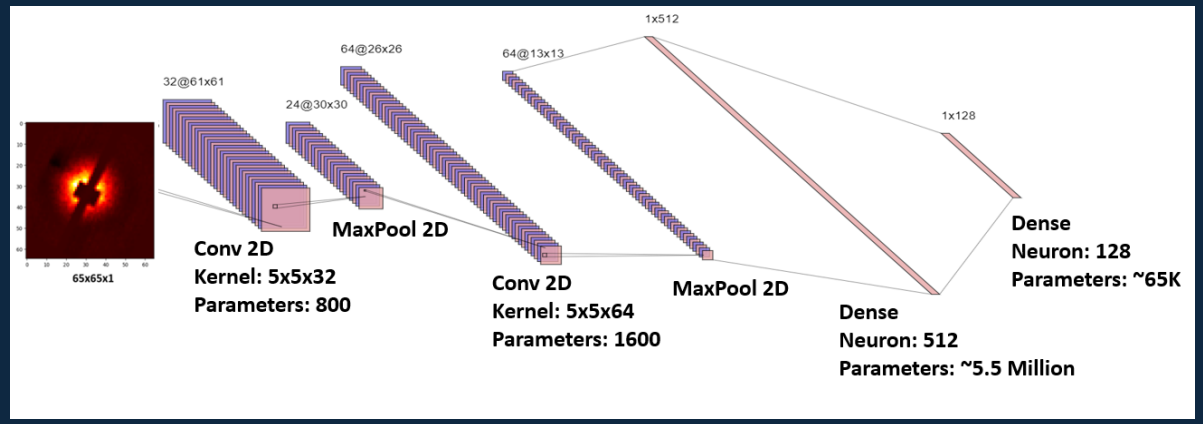


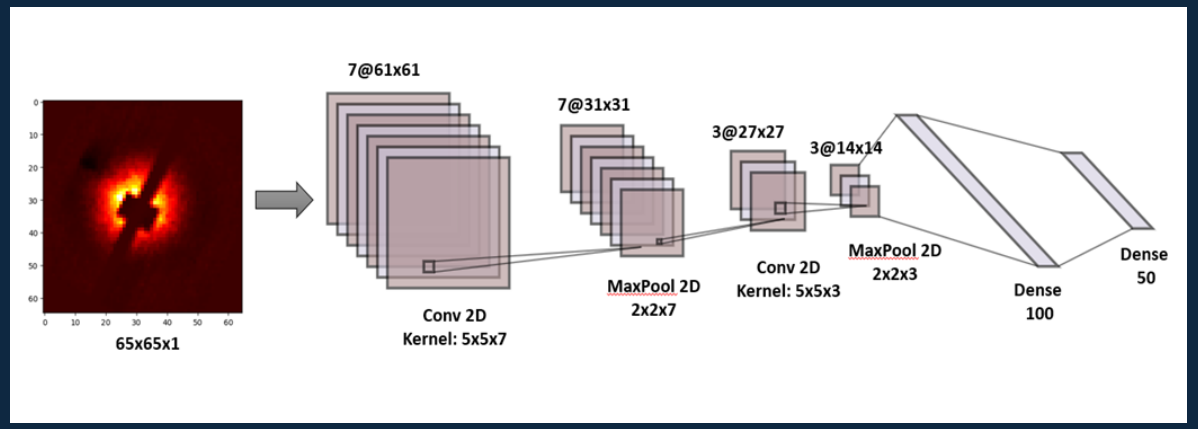
*AI Generated Image

- Knowledge Distillation: Transfer knowledge from a large 'teacher' model to a smaller 'student' model for faster, efficient inference

SNL Demonstration 1: SpeckleNN (LCLS – II)







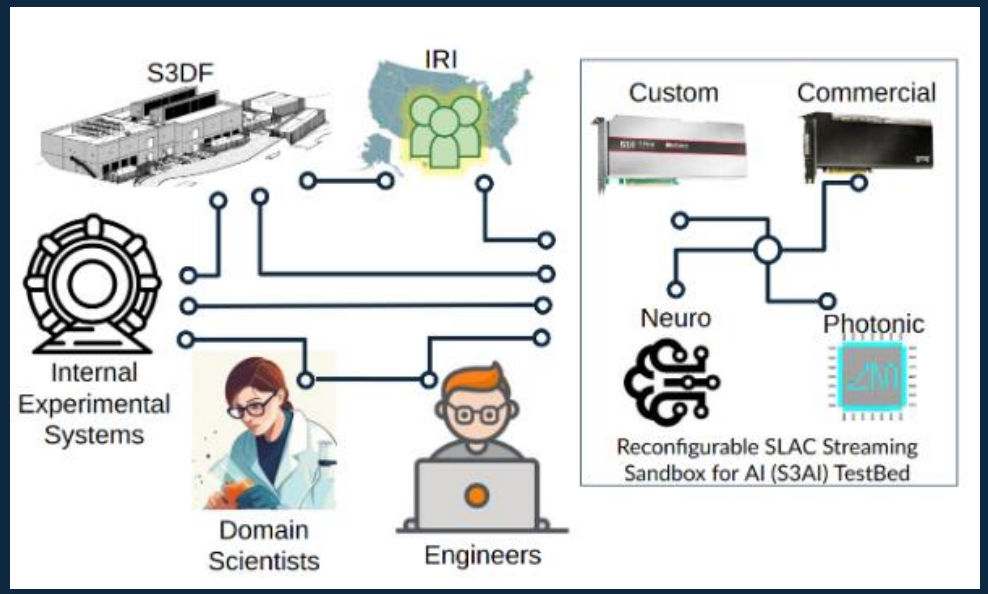
- Total Parameters:~**5.6Million**
- Data Volume Compression: **98%**
- Classification Accuracy: **94%**

~99% Model size reduction →

- Total Parameters:~**56K**
- Data Volume Compression: **98.8%**
- Classification Accuracy: **~91%**

https://arxiv.org/pdf/2502.19734
Accepted for publication in Frontiers in High Performance Computing; currently in production

SpeckleNN Performance Analysis on Different HW



- SLAC is creating a user facility to allow users to:
  - Develop and test their models using real edge hardware on real experiment data!
  - This provides a space to review and test models, making them lean and small, enabling deployment in Edge hardware.

```
(groqflow) adave@rdsrv420:~/speckleNN_Example$ python speckleNN.py
/home/adave/.local/lib/python3.10/site-packages/sklearn/utils/__ini
3.0 is required for this version of SciPy (detected version 1.21.6)
  from scipy.sparse import issparse



Building "speckleNN"
    ✓ Exporting PyTorch to ONNX
    ✓ Optimizing ONNX file
    ✓ Checking for Op support
    ✓ Converting to FP16
    ✓ Compiling model
    ✓ Assembling model

Woohoo! Saved to ~/.cache/groqflow/speckleNN
Calculated Time: 0.0007321834564208984
Your build's estimated performance is:
0.0000266 seconds
37655.9 inferences per second
Performance estimation completed.
(groqflow) adave@rdsrv420:~/speckleNN_Example$
```

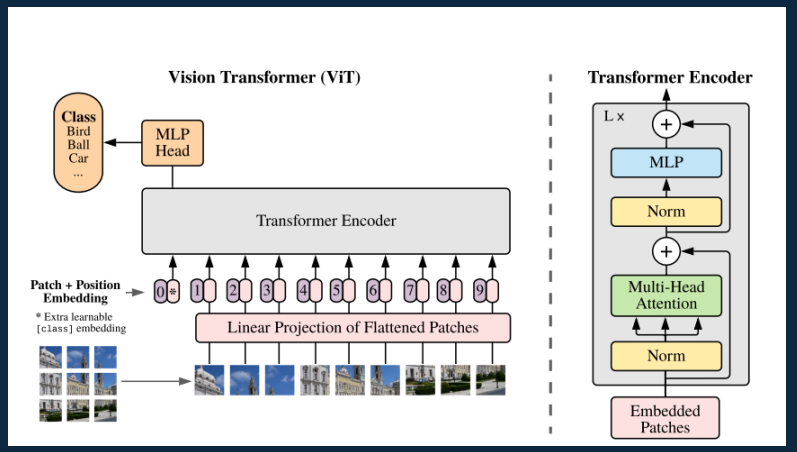
Inference Run Framework	Latency/image (us)	Power (W)
SNL-based FPGA SpackleNN	45.05 (5ns clock period)	9.4
GPU A100	400	73
Preliminary Groq System Time	732.18	~57

Challenge:
What is the fair comparison of performance between different HW systems?

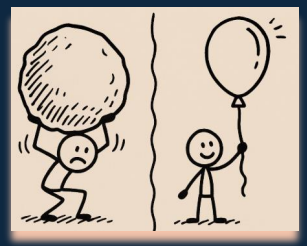
SNL Demonstration 2: ATLAS LAr Calorimeter (LHC)

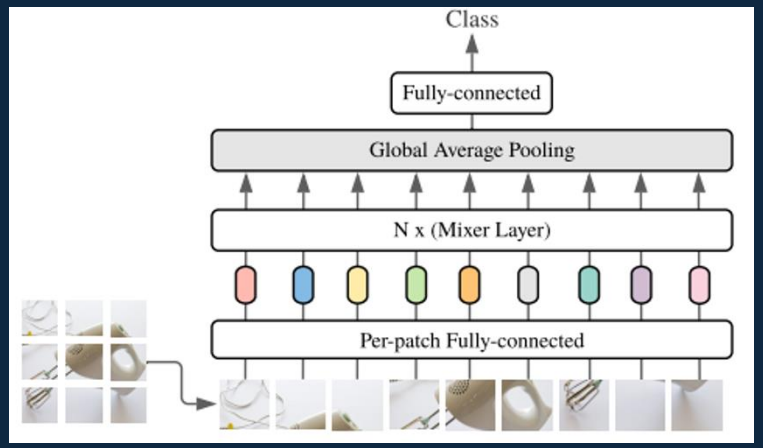
Problem: certain kinds of beyond the Standard Model physics can be detected only through cell-level analysis of calorimeter showers

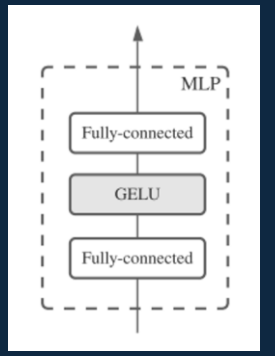
- Dataset developed by Columbia University ATLAS group studying the decay of light axion-like particles into overlapping LAr showers with "substructure"



Trasformer model is implemented by Columbia University ATLAS group

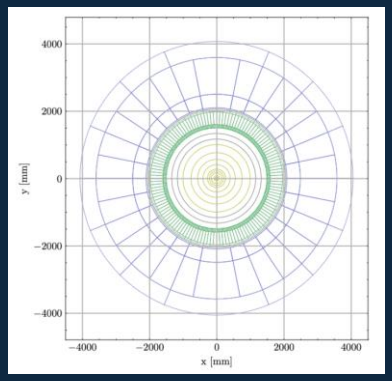


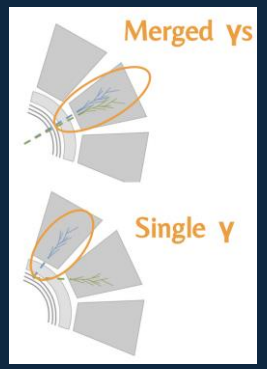




- Replaces self-attention with two **independent MLPs**:
  - **Token-mixing** MLP (across sequence length)
  - **Channel-mixing** MLP (across features)
- Alternates between **transposing token and channel dimensions** to allow information mixing.

"MLP-Mixer: An all-MLP Architecture for Vision", 2105.01601

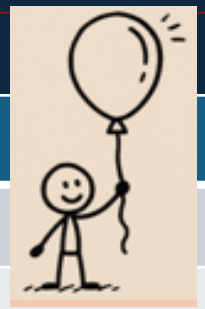


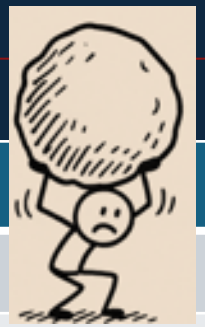


- **No Q/K/V matrices or pairwise comparisons**
- **Does not require full-token memory access**
  - for mixing each MLP processes independently across dimensions.
- **No softmax or dot-product for attention**, avoiding quadratic complexity.
- Achieves **competitive representational power** with **linear compute/memory cost**

➢ **But transformers are computationally expensive… any edge application is challenging**

Comparison of Transformer and MLP-Mixer

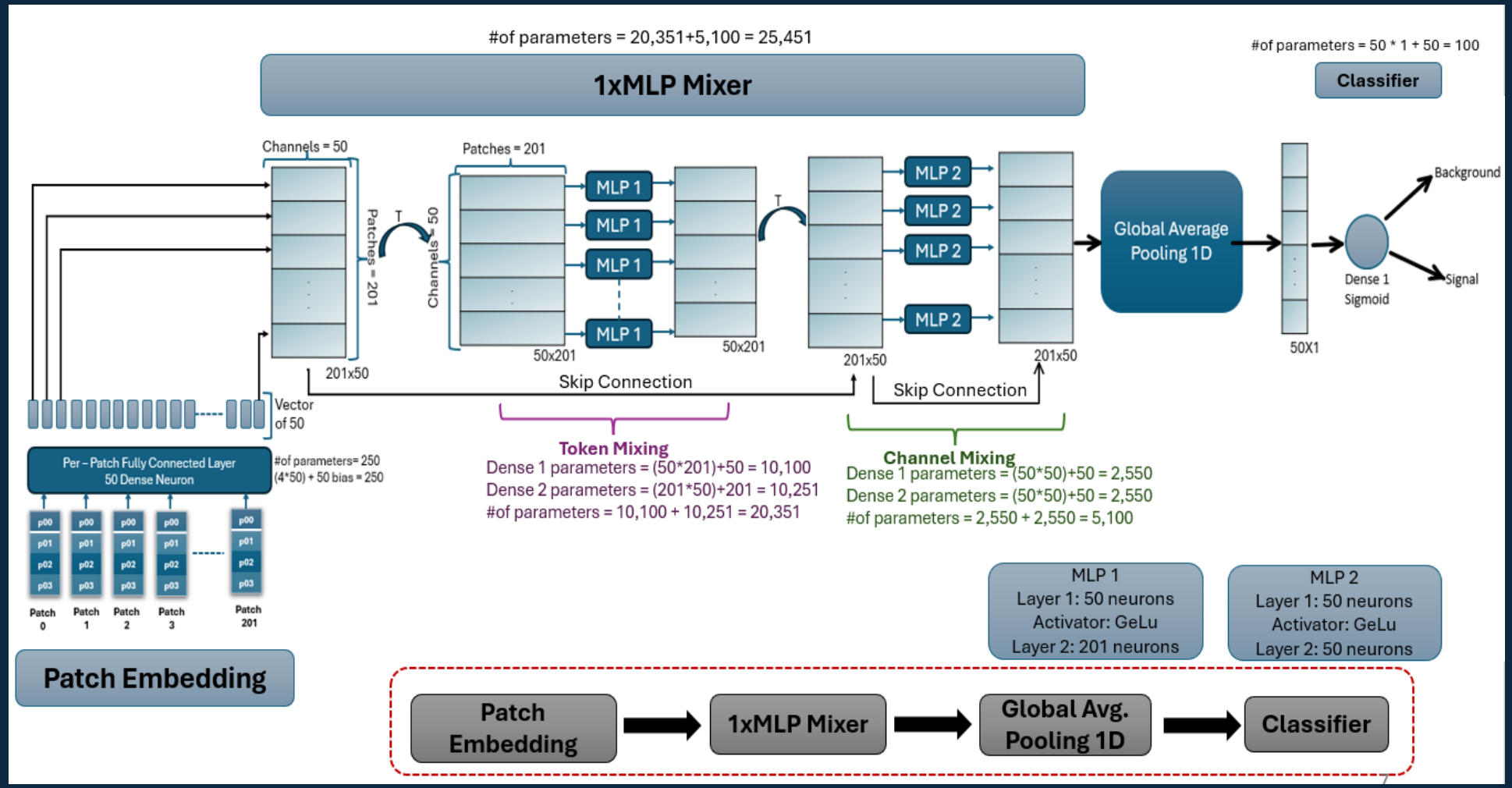




	MLP-Mixer	Transformer (Columbia)
Accuracy on Latest Dataset	0.85 AUC	0.91 AUC
FLOPs (Floating-Point Operations) Metrics to evaluate the computational cost of models	4,100,400	10,492,200
Training Time on GPU	~3 to 7min	~1.5hrs
Aspects		
💻 Speed	Faster inference/training	Slower
⚡ Efficiency	Lower power use, good for edge/real-time	More costly on devices
🎯 Accuracy	Might be lower (depends on task)	Might be higher (more capacity)
📦 Deployment	Easier to deploy on limited hardware	Requires more resources

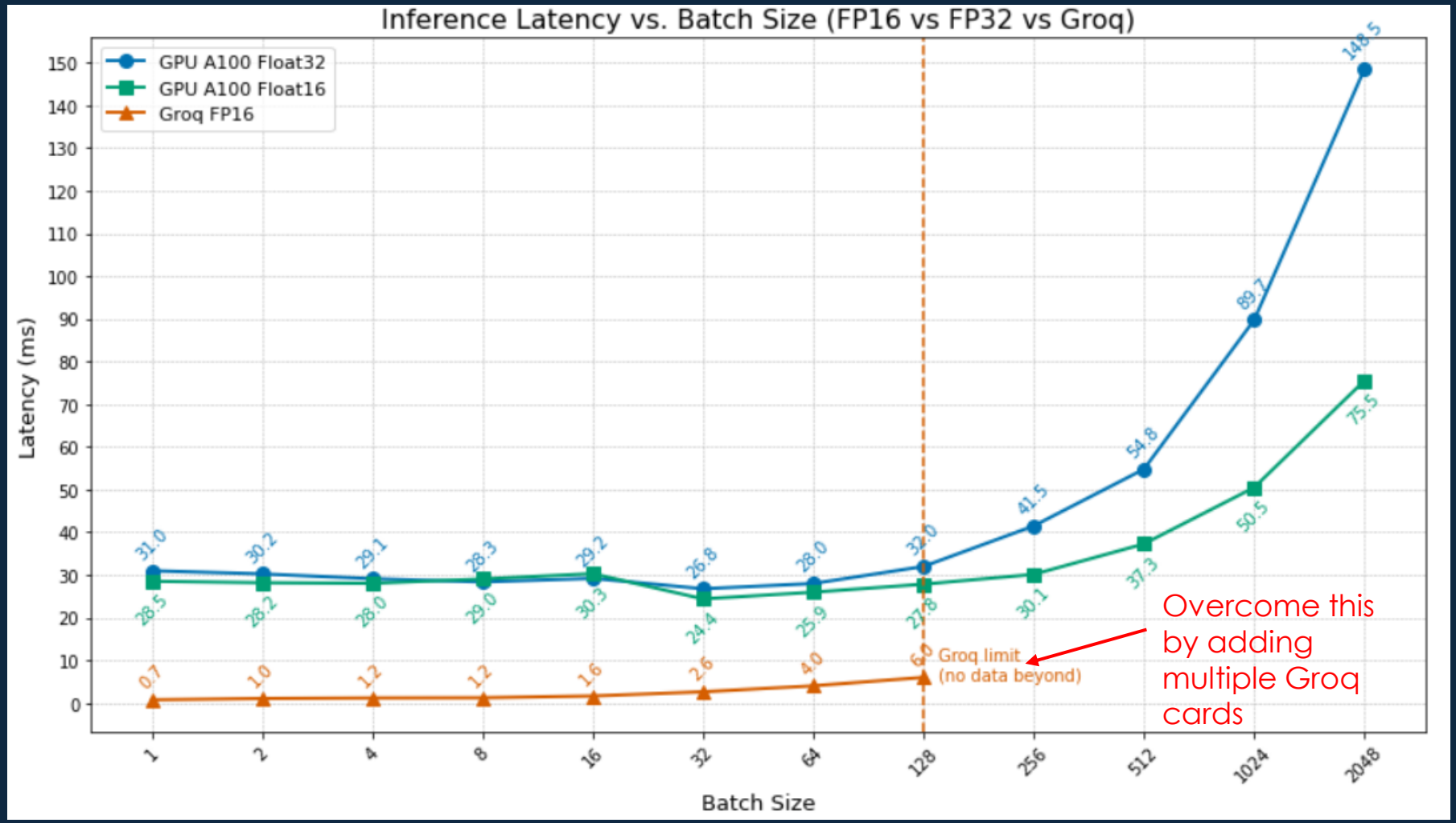
*These are very preliminary results unpublished work (2025)

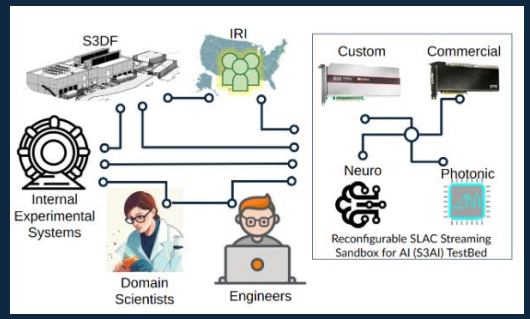
MLP Mixer Architecture for Calorimeter ML



*This is based on unpublished work (2025)

Running the MLP Mixer Architecture on Different Hardware



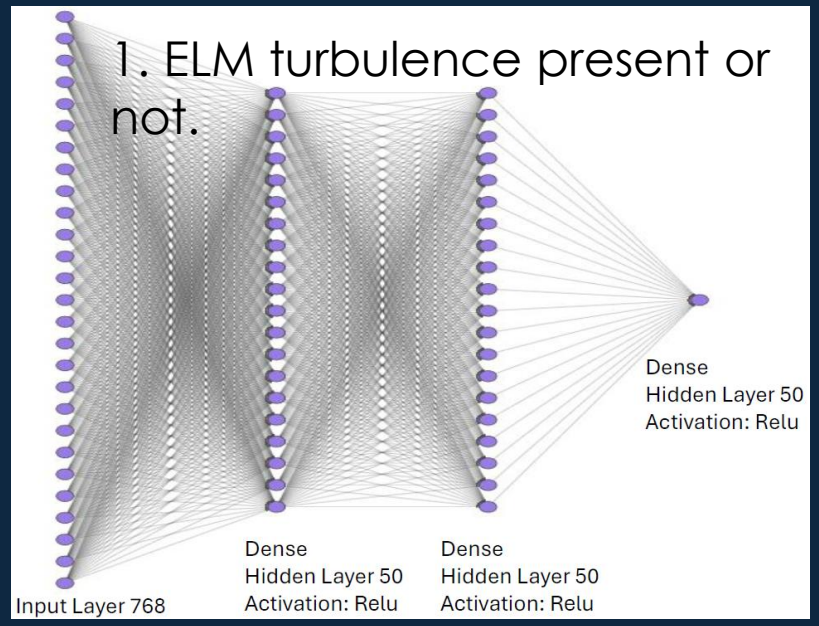


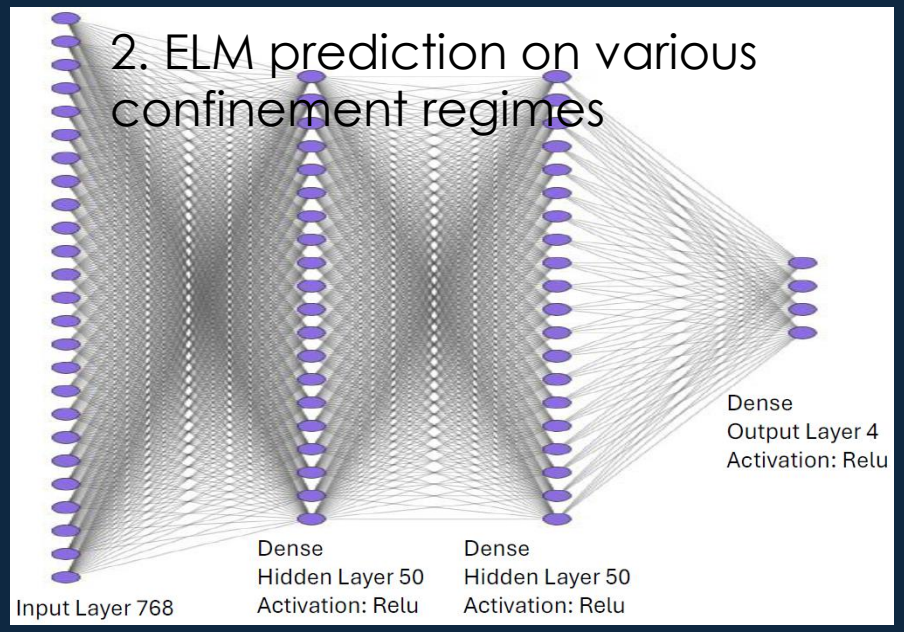
- Future work:
  - SNL based FPGA implementation of MLP-Mixer
  - Also run the transformer model inference run on this HW

Challenge:
What is the fair comparison of performance between different HW systems?

This is based on unpublished work (2025)

SNL Demonstration 3: ELM Prediction and Regime Classification in the 2D BES System at DIII-D





FPGA-Based Real-Time ML Inference Using SNL for ELM Prediction and Regime Classification in the 2D BES System at DIII-D

Abhilasha Dave[1], Ryan Coffee[1], Ryan Herbst[1], James Russell[1], Larry Ruckman[1], Jalal Butt[2], Keith Erickson[2], David Smith[3]
[1]SLAC National Accelerator Laboratory, Menlo Park, CA, USA
[2]Princeton Plasma Physics Laboratory, Princeton, NJ, USA
[3]University of Wisconsin, Madison, WI, USA
Unpublished manuscript, September 9, 2024

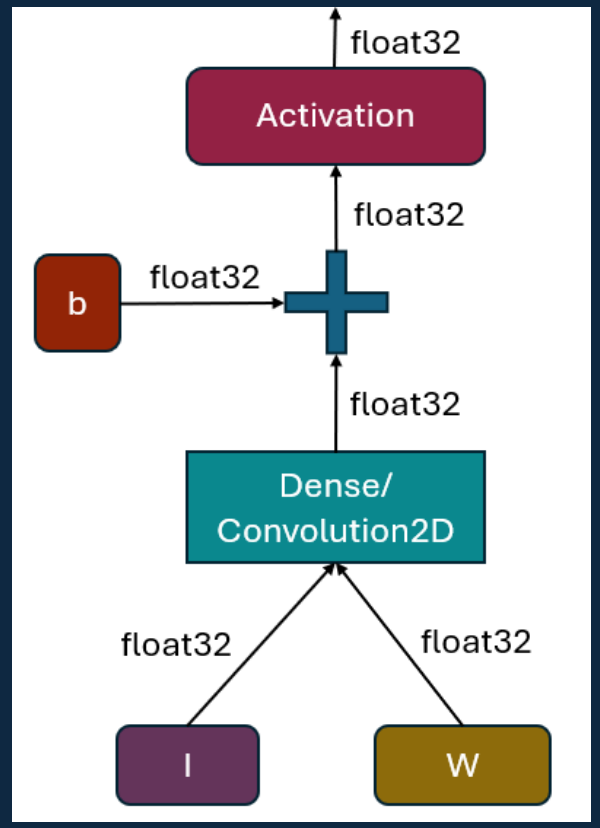
- SNL based FPGA implementation of ML inference for the 2D Beam Emission Spectroscopy (BES) system at DIII-D.
- BES system measures localized pedestal dynamics during edge-localized mode (ELM) events.
- Monitors **edge turbulence dynamics** across **various confinement regimes** (L-mode, H-mode, QH-mode, and wide pedestal QH-mode).

➢ **Key takeaway:** A single SNL based FPGA synthesized model can support multiple applications by **dynamically uploading task-specific weights and biases** between experiments.

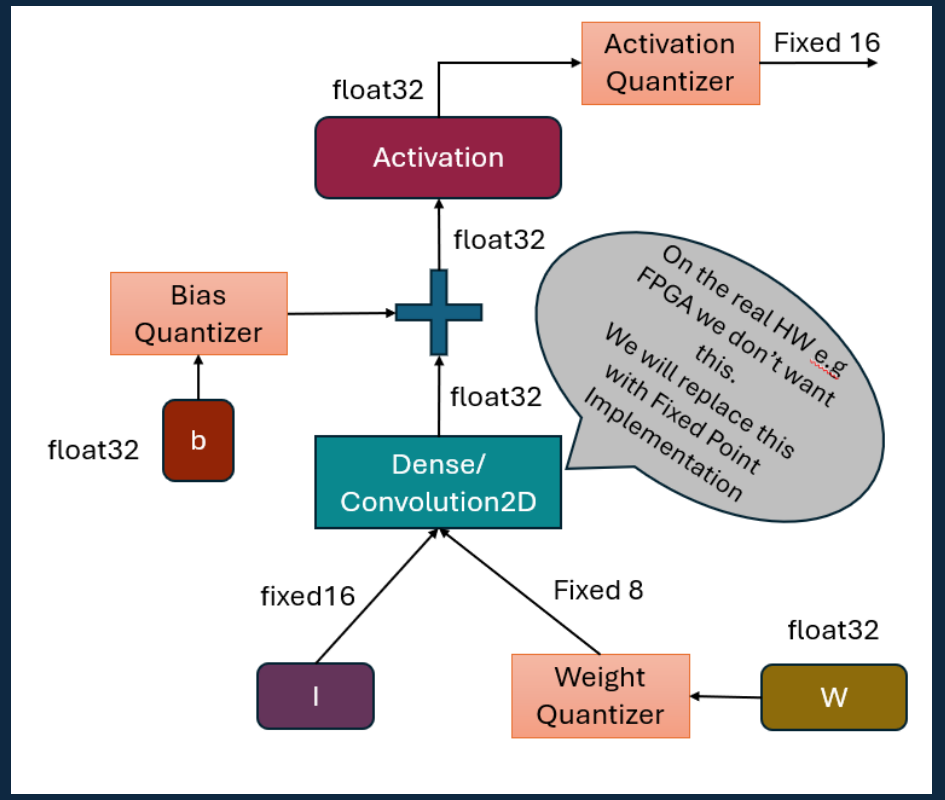
*LCLS SRD AMO: Ryan Coffee

Model Quantization

- Simulated Quantization in Qkeras



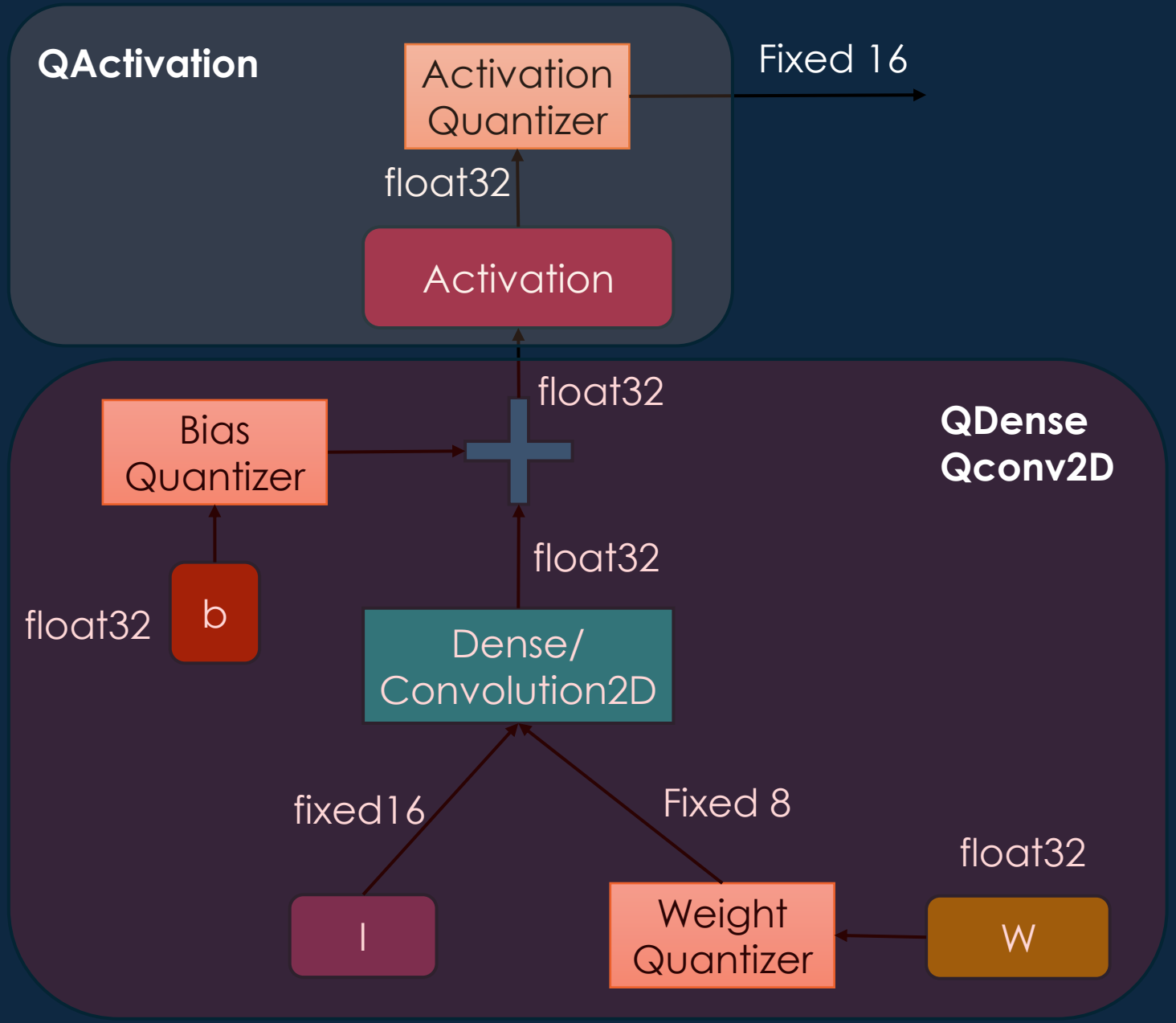
Keras



QKeras

- SNL will follow similar to Qkeras quantization implimentation
- Google: https://arxiv.org/pdf/1712.05877
- Qkeras&HLS4ML: https://arxiv.org/pdf/2006.10159v1

Qkeras Wraps the actual Keras operation



Layers Implemented in QKeras

- QDense
- QConv1D
- QConv2D
- QDepthwiseConv2D
- QSeparableConv1D (depthwise + pointwise convolution, without quantizing after the depthwise step)
- QSeparableConv2D (depthwise + pointwise convolution, without quantizing after the depthwise step)
- QMobileNetSeparableConv2D (extended from MobileNet SeparableConv2D quantizes the activation values after the depthwise step)
- QConv2DTranspose

https://github.com/google/qkeras/tree/master

How the Quantizers Works?

- There are two different types of quantizers available in Qkeras
  - Auto Scale
  - Power of Two Quantizer

Auto Scale Quantizer

- **Scaling is decided by:**
  - Take the maximum val from the input vals and divided that number by the number of bins you want
  - $s = \frac{\max(w \text{ or } b)}{n-1}$
- **Quantizer:**
  - Divide the input by the scale, round to nearest integer and multiply that integer number by the scale
  - $q(I) = int\left(\frac{w \text{ or } b}{s}\right) * s$
- This is a problem for activation for weights okay

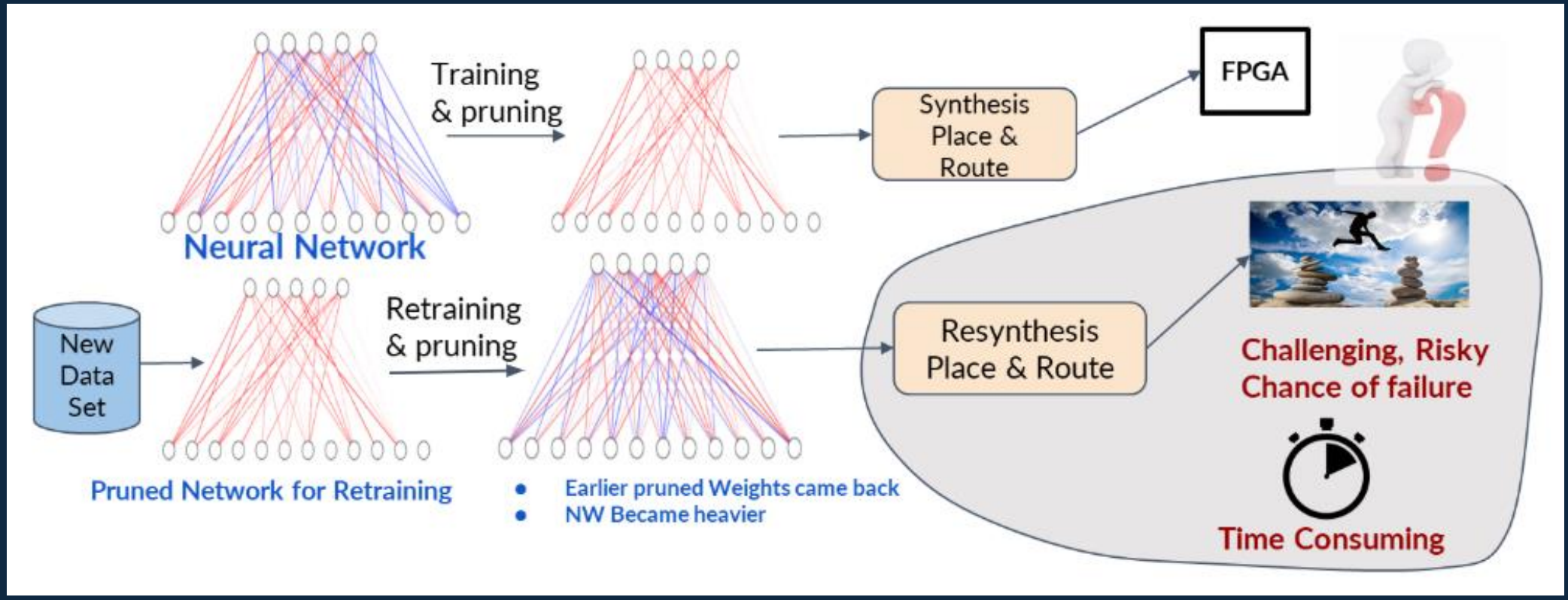
Power of Two Scale (PoT)

- **Scaling is decided by**:
  - Take the number of bits
  - $s = \frac{1}{2\hat{}bits}$
- **Quantizer:**
  - Now you can do the sift operation for division and multiplication which is cheaper for HW
  - Q(I) = Int(I >> log2(s)) << log2(s)
- This is good for activation and in general HW implementation

Challenge with Neural Network Pruning (Not Supported in SNL)

•Some AI-to-FPGA frameworks take the weights and biases and pruning portions of the network structure to save resources

- **Re-synthesis** is required for each new training set
- **Risk of the FPGA implementation failing** due to increase resources usage, timing failures or massive change in internal interconnect structure



Thank You