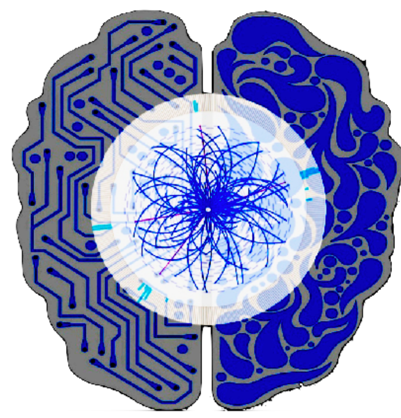


Overview

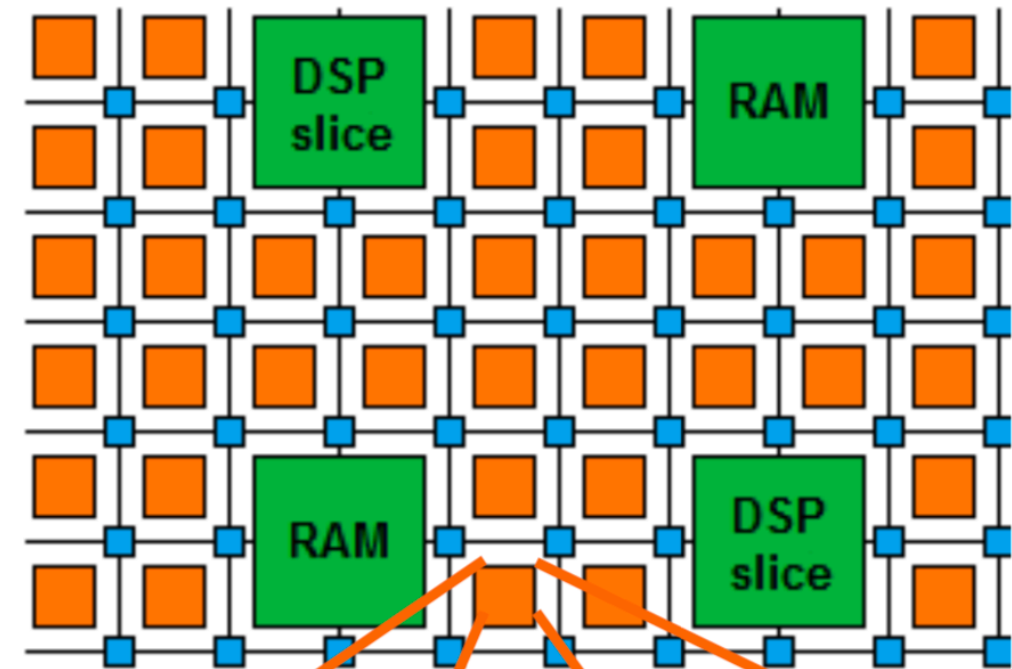
A3D3 Hardware Algorithm Co-development Seminar

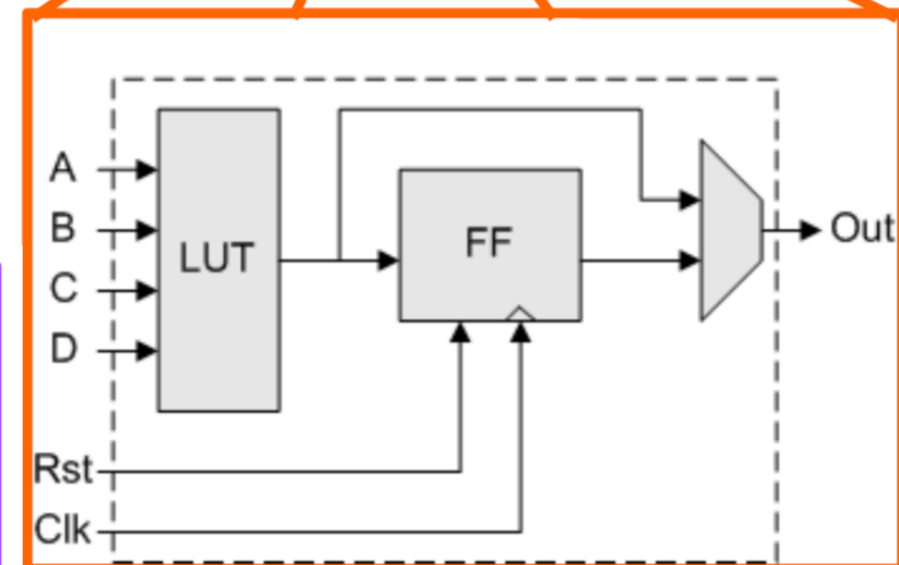
Deming Chen, Song Han, Philip Harris, Scott Hauck, Pan Li, **Dylan Rankin**

November 1st, 2021

What is an FPGA?

- Building blocks:
    - **Multiplier units (DSPs) [arithmetic]**
    - **Look Up Tables (LUTs) [logic]**
    - **Flip-flops (FFs) [registers]**
    - **Block RAMs (BRAMs) [memory]**
- Algorithms are wired onto the chip
- Run at high frequency - *hundreds of MHz, O(ns) runtime*
- Programming traditionally done in Verilog/VHDL
    - Low-level hardware languages
- Possible to translate C to Verilog/VHDL using High Level Synthesis (HLS) tools

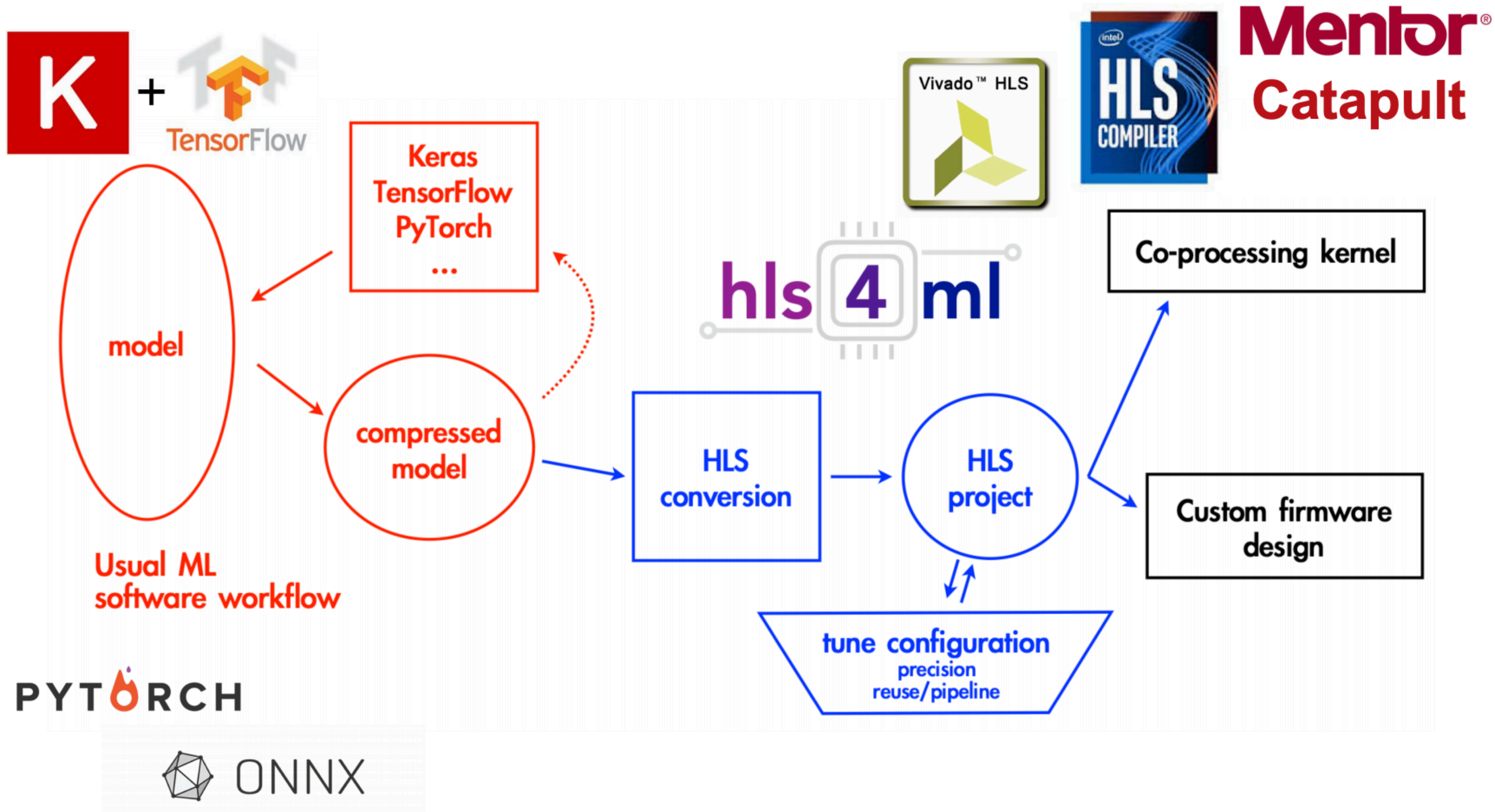




Virtex Ultrascale+ VU9P
6800 Multipliers
1M LUTs
2M FFs
75 Mb BRAM

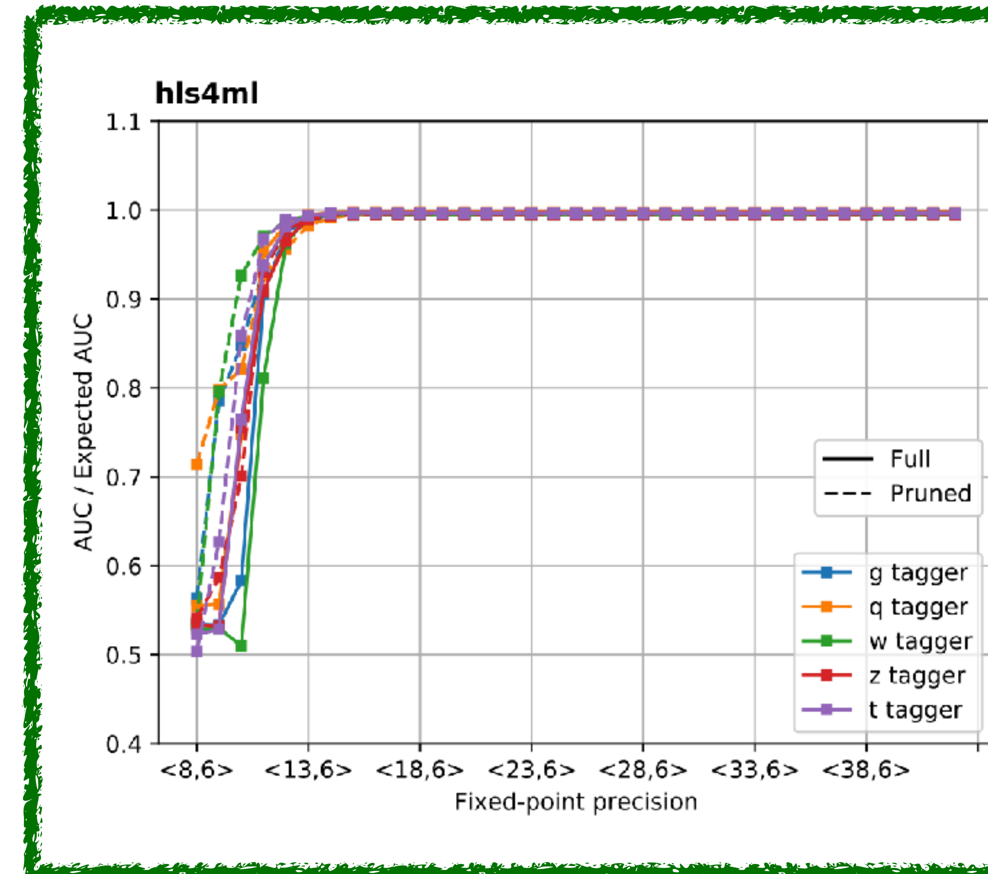
- hls4ml is a software package for creating implementations of neural networks for FPGAs and ASICs
    - https://fastmachinelearning.org/hls4ml/
    - arXiv:1804.06913
- Supports common layer architectures and model software, options for quantization/pruning
    - Output is a fully ready HLS project
- pip installable
- Customizable output
    - Tunable precision, latency, resources

hls4ml Workflow

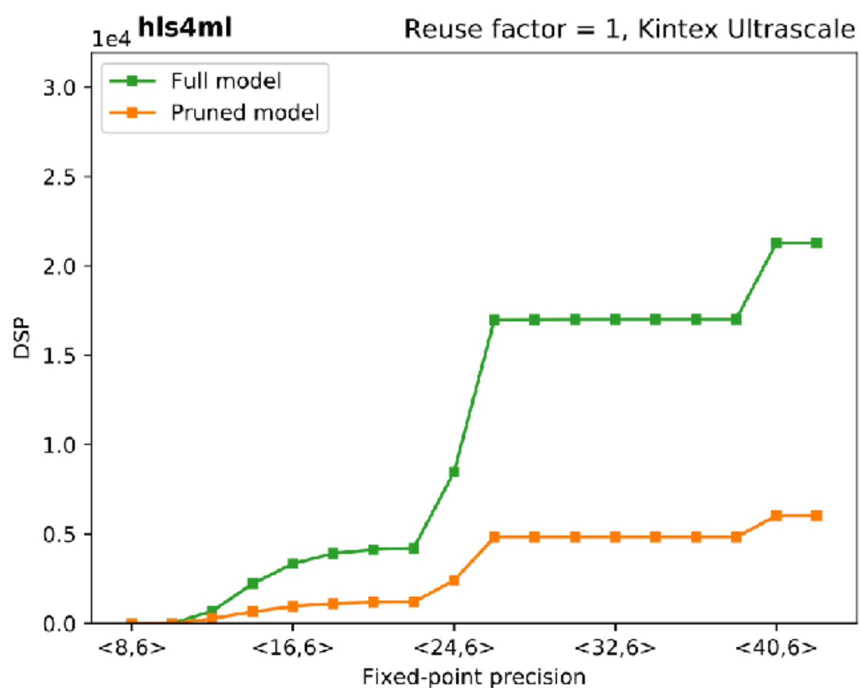


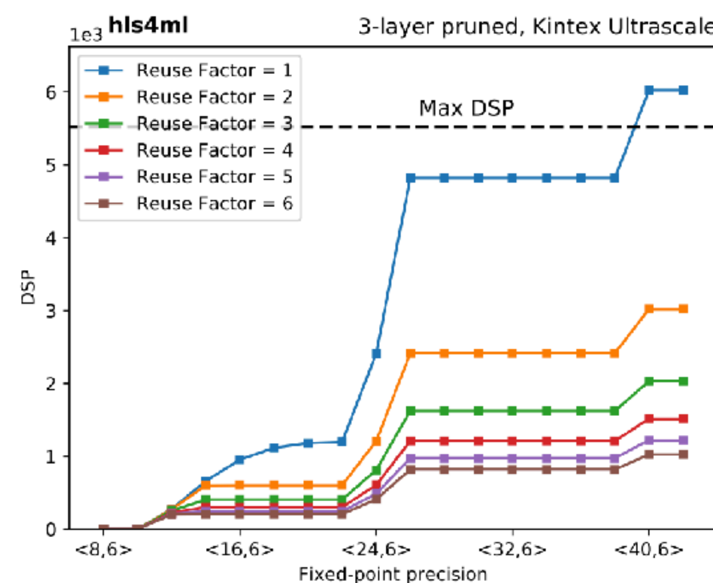
hls4ml Customization

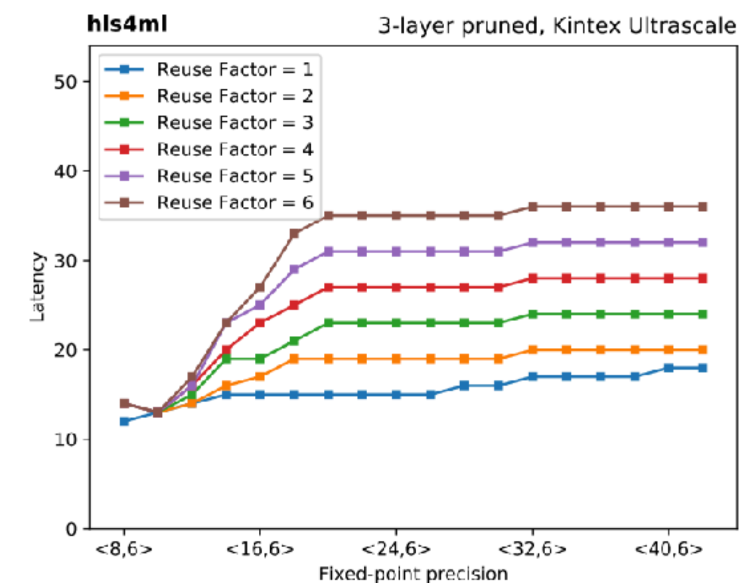
- Multiple different knobs to adjust design for desired performance/latency/resource usage
  - Pruning
  - Quantization
  - Reuse



3-layer dense network for jet tagging
(details in backup)

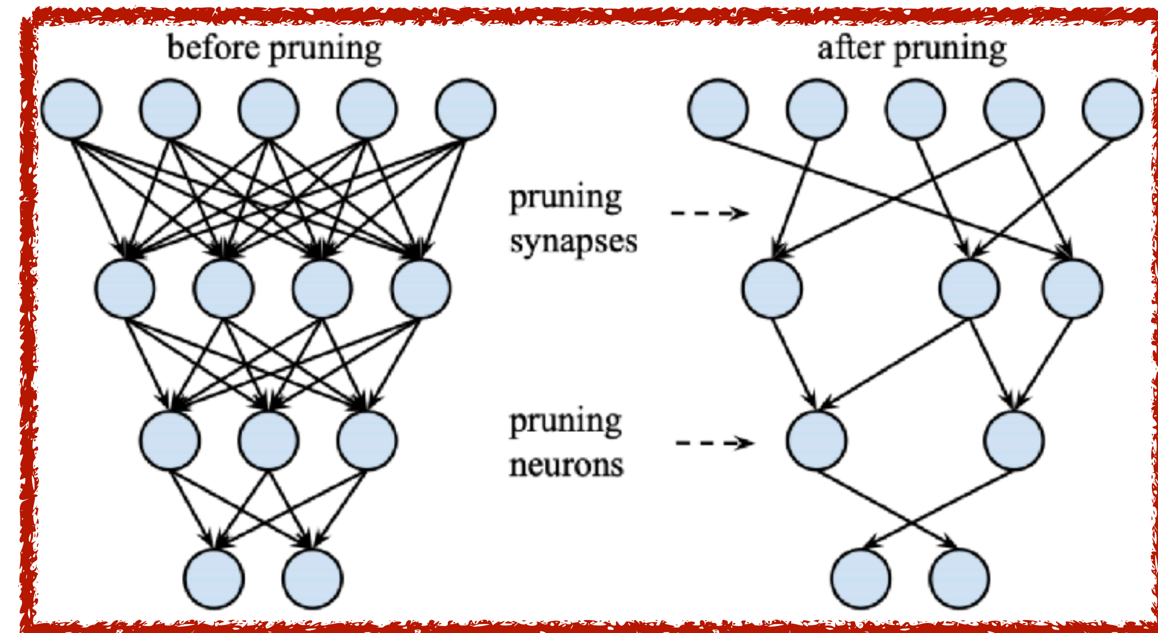




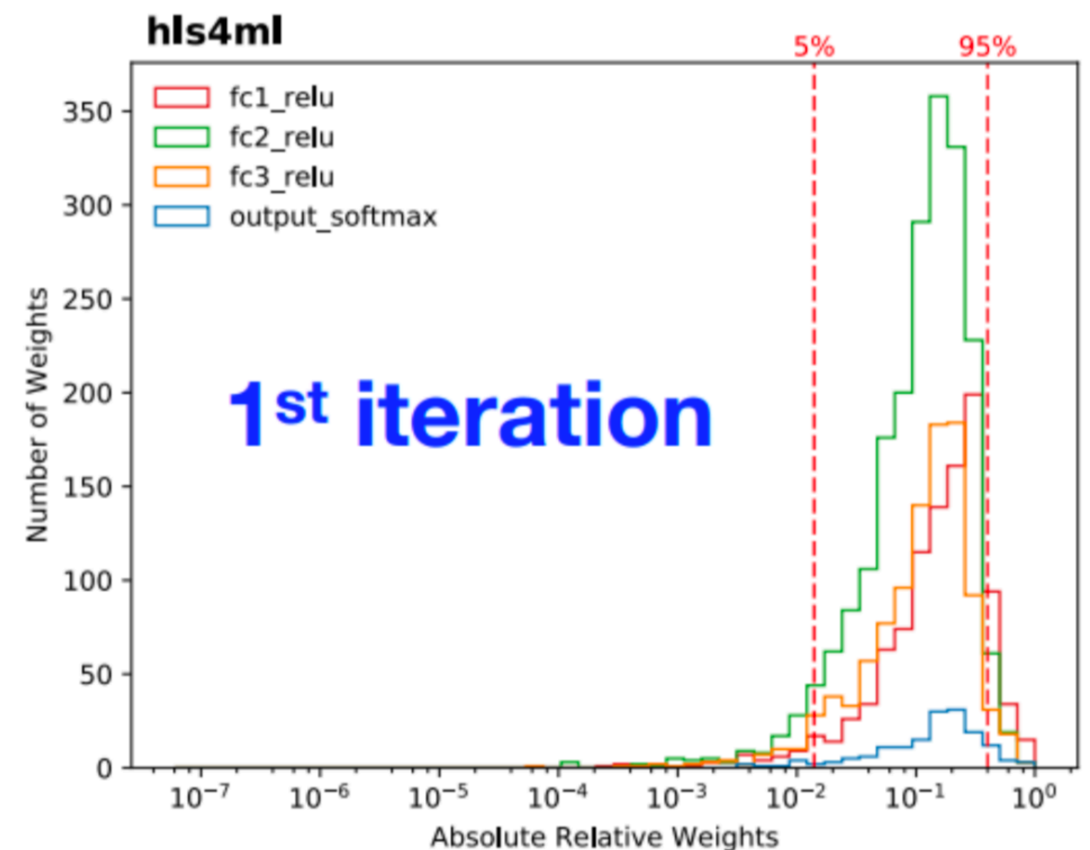


Pruning

- Are all the pieces a given network necessary?
- Many techniques for determining "best" way to prune
- hls4ml naturally supports a method of successive retraining and weight minimization
    - Use L1 regularization (penalty term in loss function for large weights)
    - Remove smallest weights
    - Repeat
- HLS automatically removes multiplications by 0

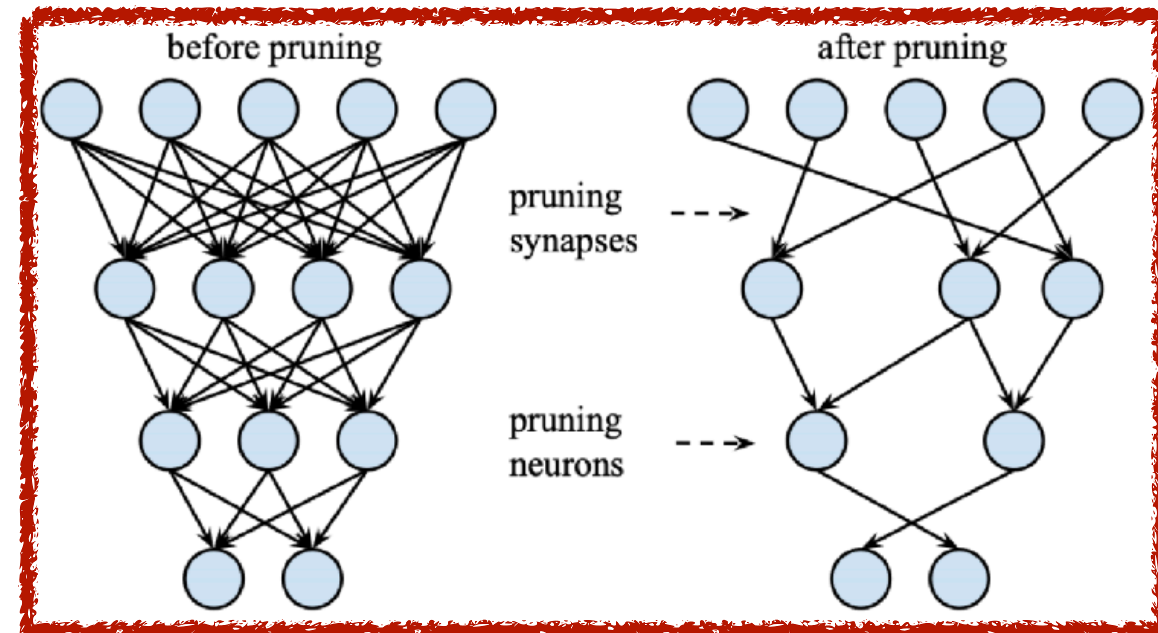


$$L_\lambda(\mathbf{w}) = L(\mathbf{w}) + \lambda \|\mathbf{w}\|$$

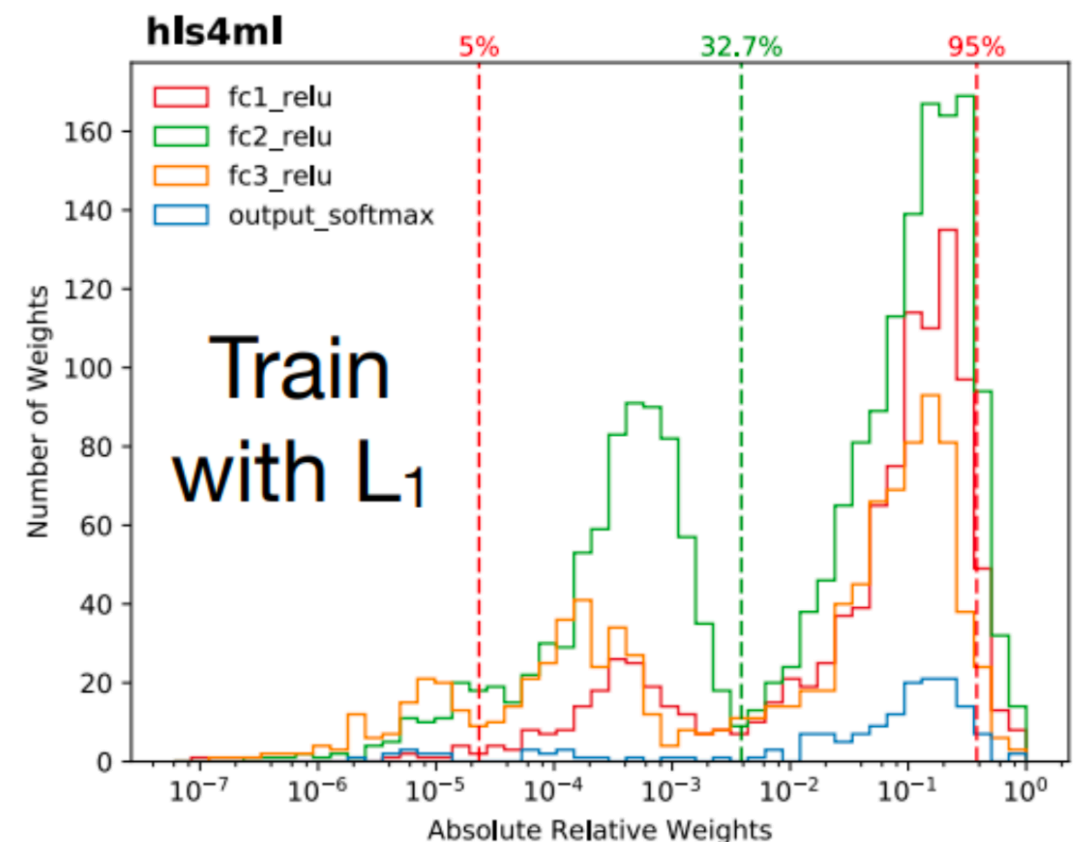


Pruning

- Are all the pieces a given network necessary?
- Many techniques for determining “best” way to prune
- hls4ml naturally supports a method of successive retraining and weight minimization
    - Use L1 regularization (penalty term in loss function for large weights)
    - Remove smallest weights
    - Repeat
- HLS automatically removes multiplications by 0

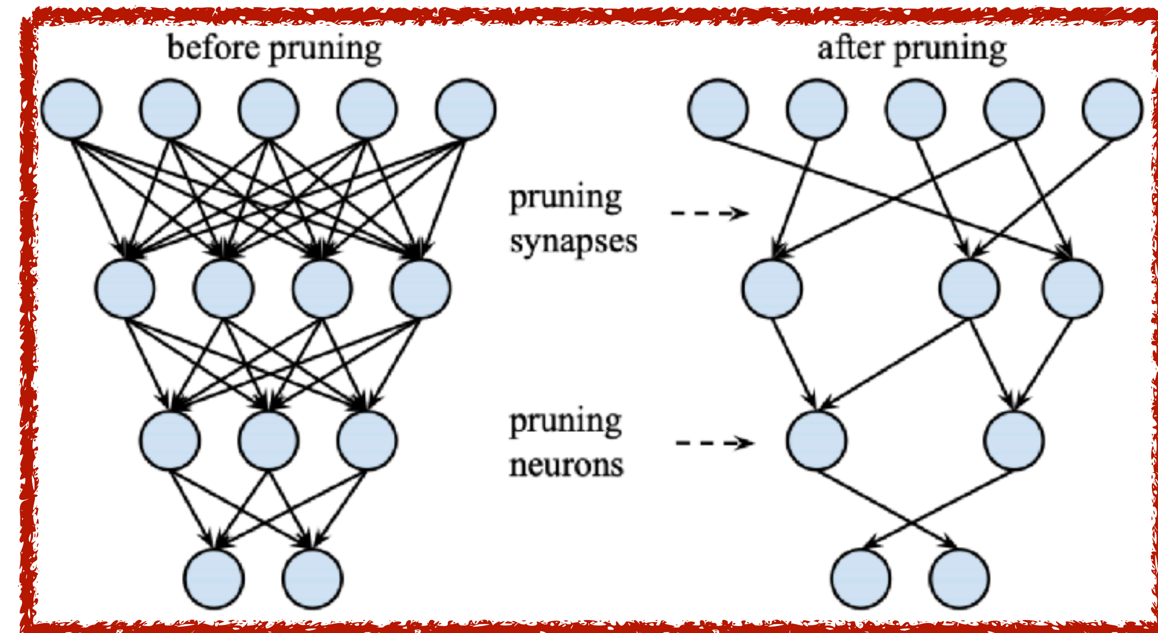


$$L_\lambda(\mathbf{w}) = L(\mathbf{w}) + \lambda \|\mathbf{w}\|$$

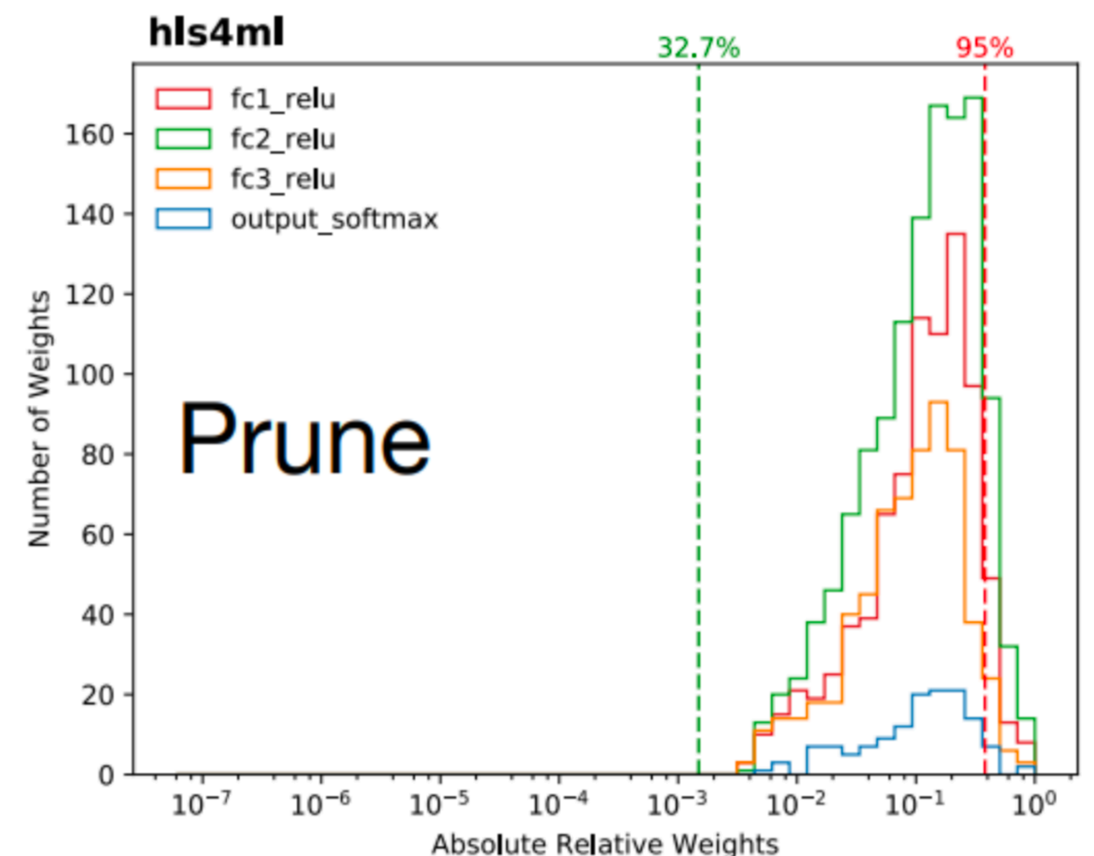


Pruning

- Are all the pieces a given network necessary?
- Many techniques for determining "best" way to prune
- hls4ml naturally supports a method of successive retraining and weight minimization
  - Use L1 regularization (penalty term in loss function for large weights)
  - Remove smallest weights
  - Repeat
- HLS automatically removes multiplications by 0

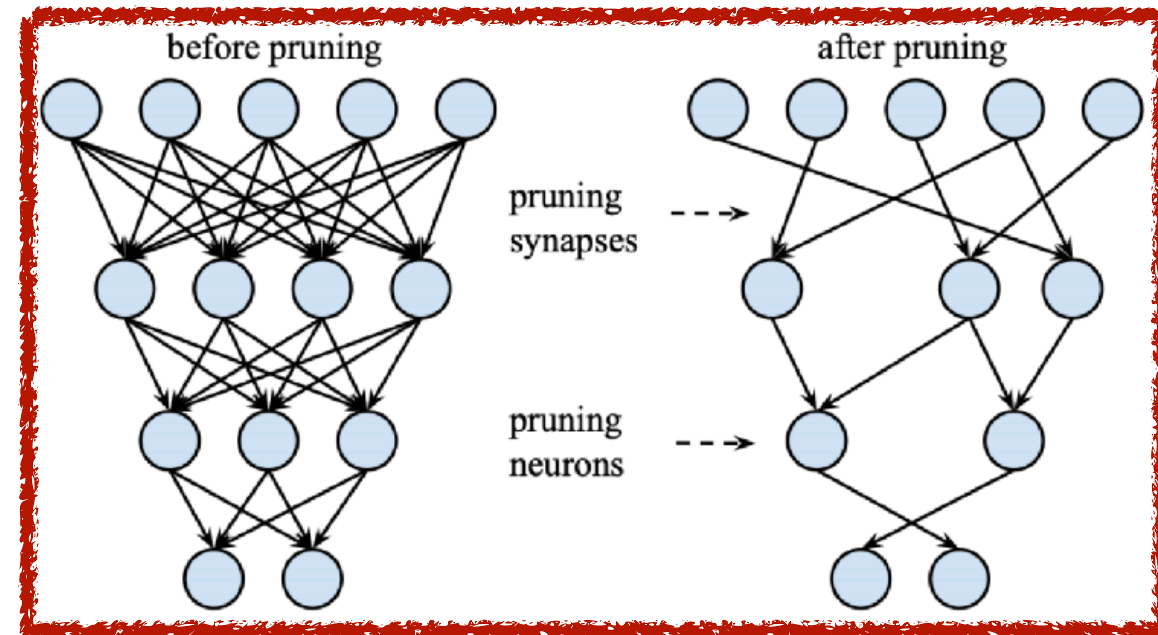


$$L_\lambda(\mathbf{w}) = L(\mathbf{w}) + \lambda\|\mathbf{w}\|$$

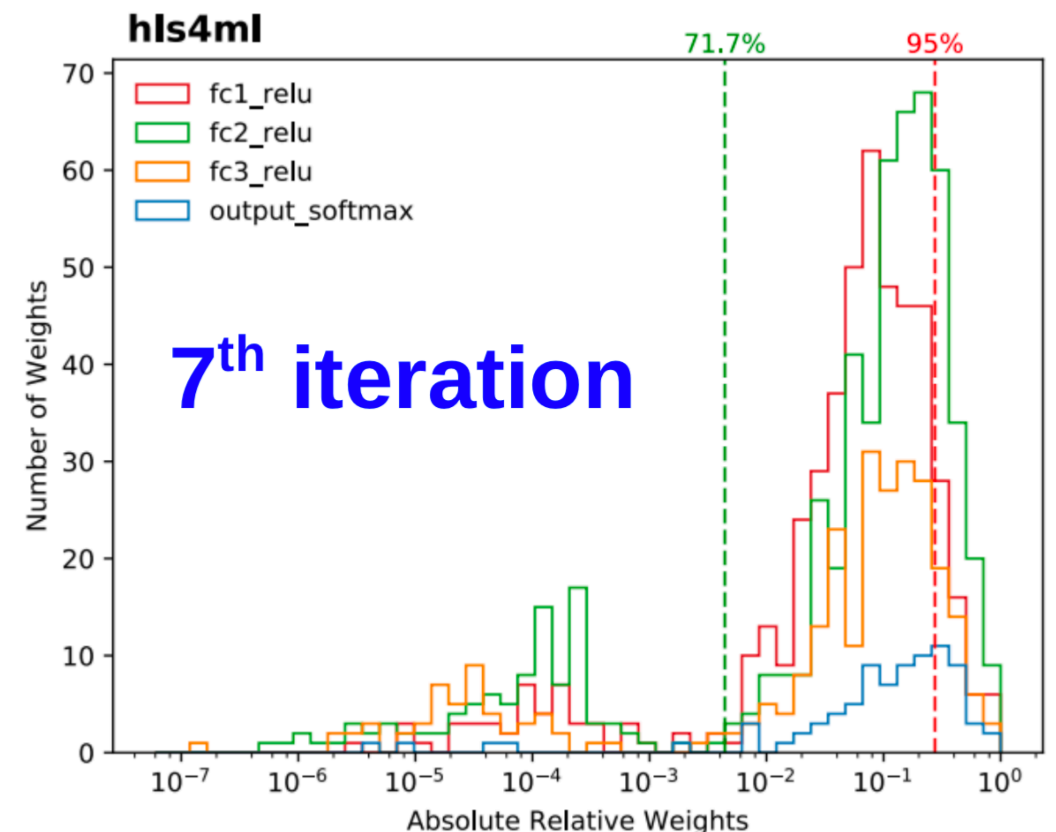


Pruning

- Are all the pieces a given network necessary?
- Many techniques for determining "best" way to prune
- hls4ml naturally supports a method of successive retraining and weight minimization
    - Use L1 regularization (penalty term in loss function for large weights)
    - Remove smallest weights
    - Repeat
- HLS automatically removes multiplications by 0

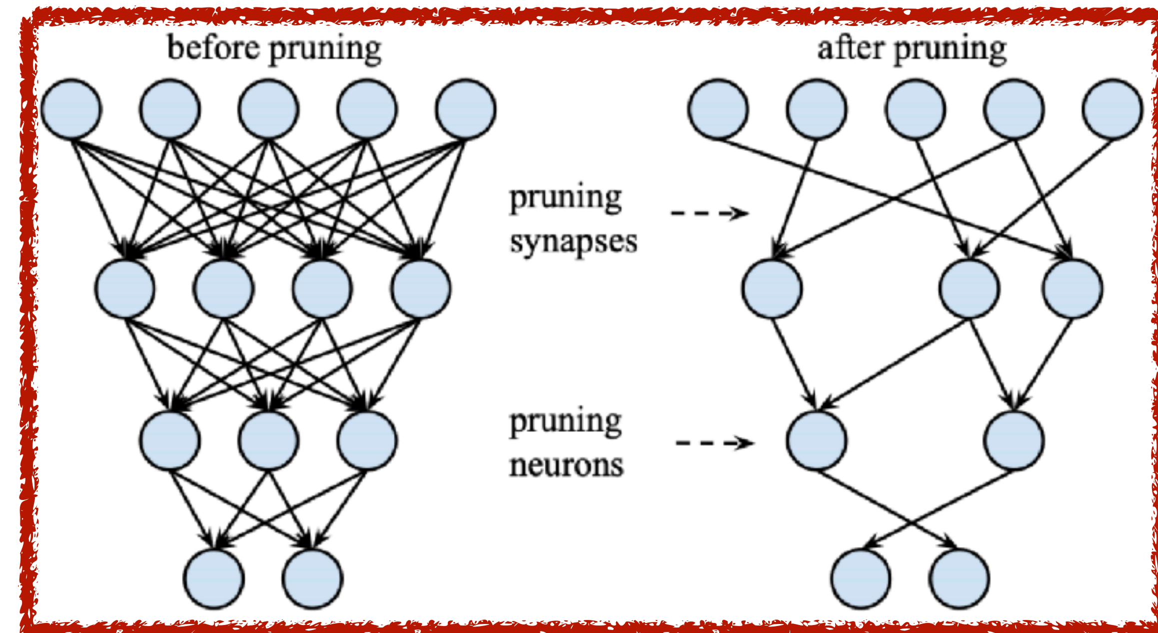


$$L_\lambda(\mathbf{w}) = L(\mathbf{w}) + \lambda\|\mathbf{w}\|$$

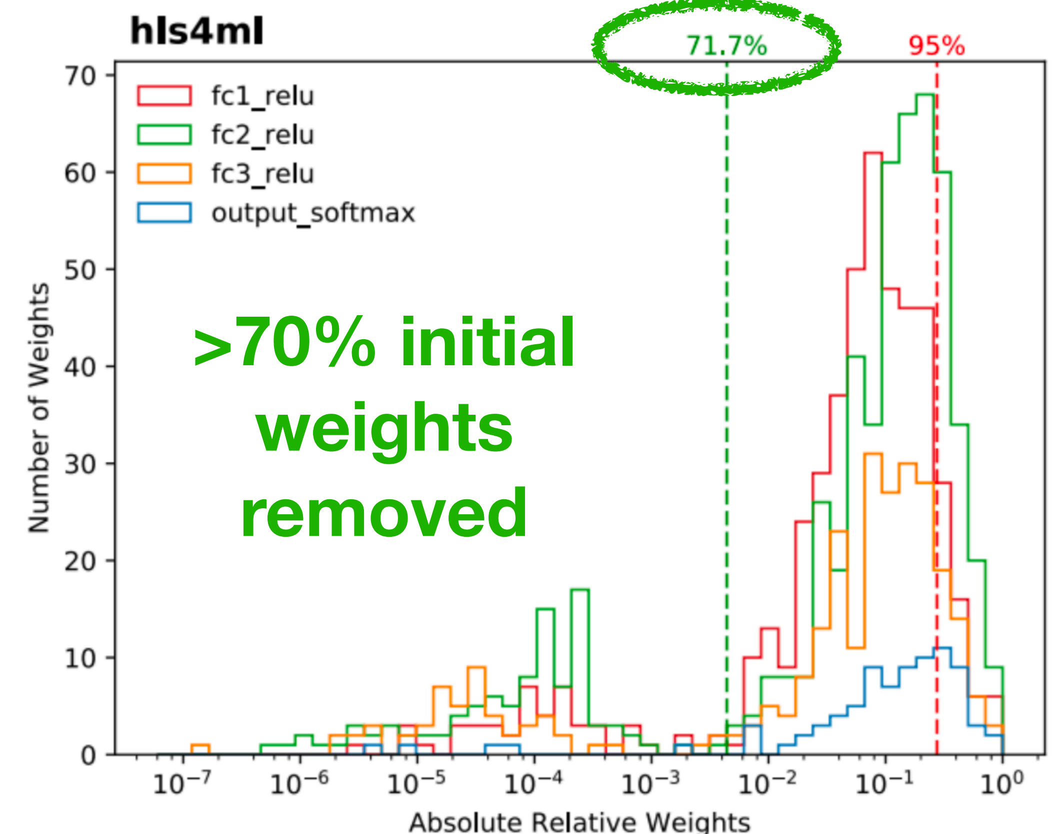


Pruning

- Are all the pieces a given network necessary?
- Many techniques for determining "best" way to prune
- hls4ml naturally supports a method of successive retraining and weight minimization
    - Use L1 regularization (penalty term in loss function for large weights)
    - Remove smallest weights
    - Repeat
- HLS automatically removes multiplications by 0

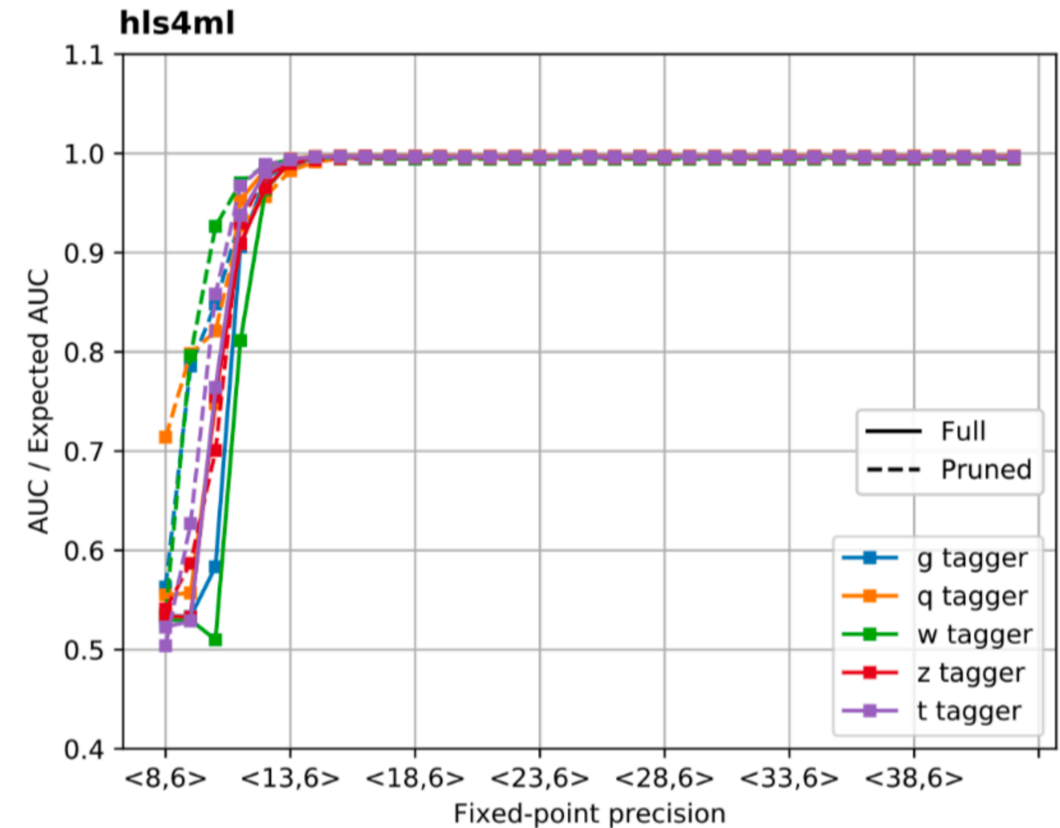


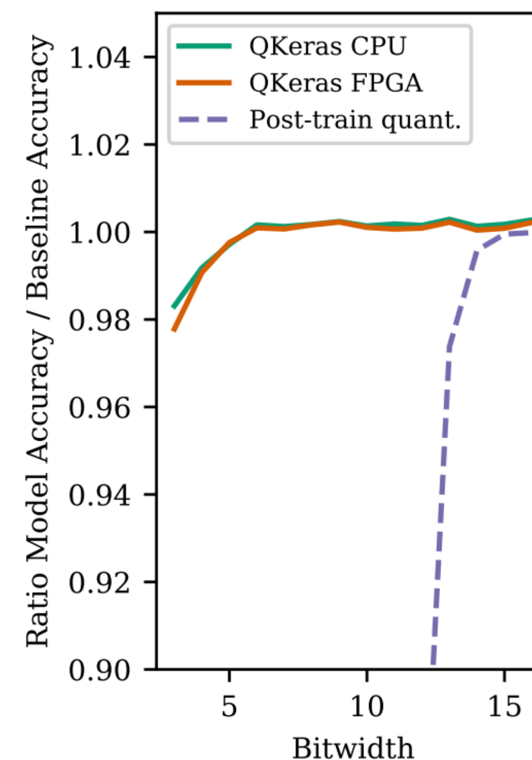
$$L_{\lambda}(\mathbf{w}) = L(\mathbf{w}) + \lambda \|\mathbf{w}\|$$

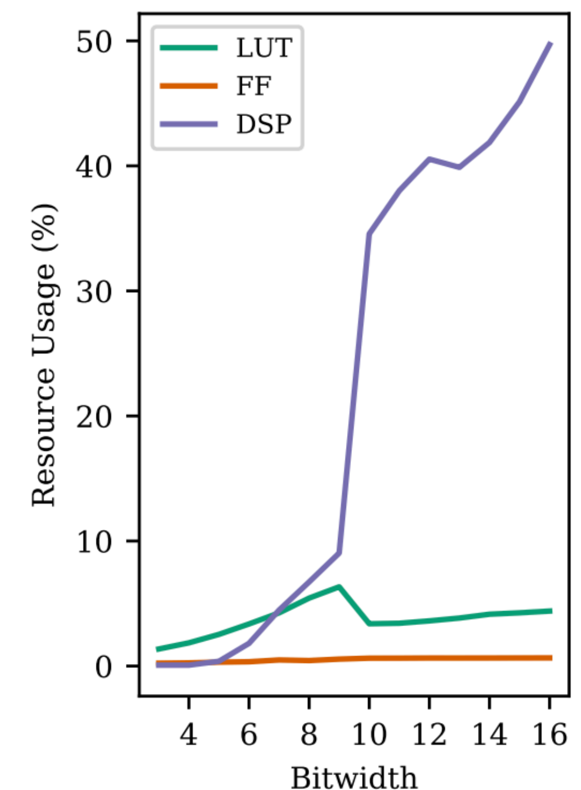


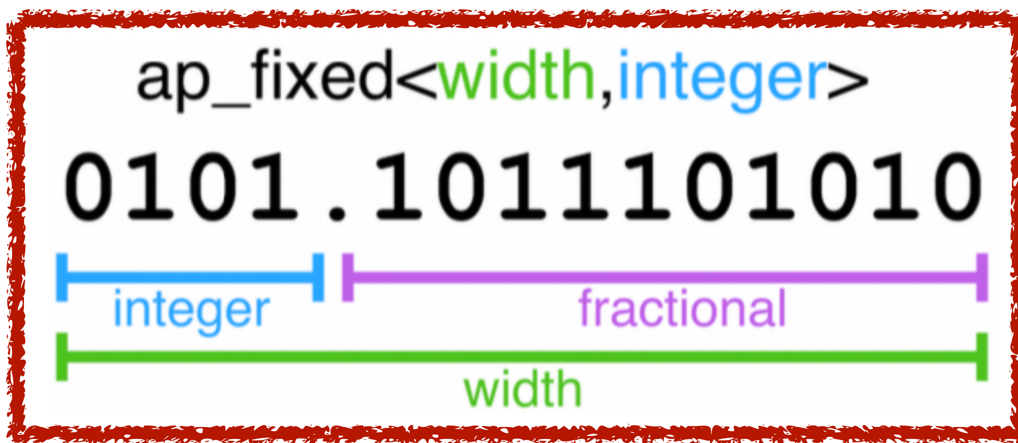
Quantization

- hls4ml uses fixed-point classes for all computations
- Precision can be adjusted as needed (impacts accuracy, performance, resources)
    - Can be combined with other customizations
- Binary & Ternary neural networks take this to very low precision: [2020 Mach. Learn.: Sci. Technol]
- Quantization-aware training - QKeras + support in hls4ml: [arXiv:2006.10159]



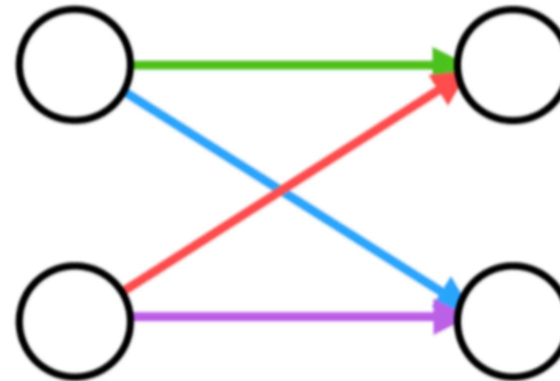


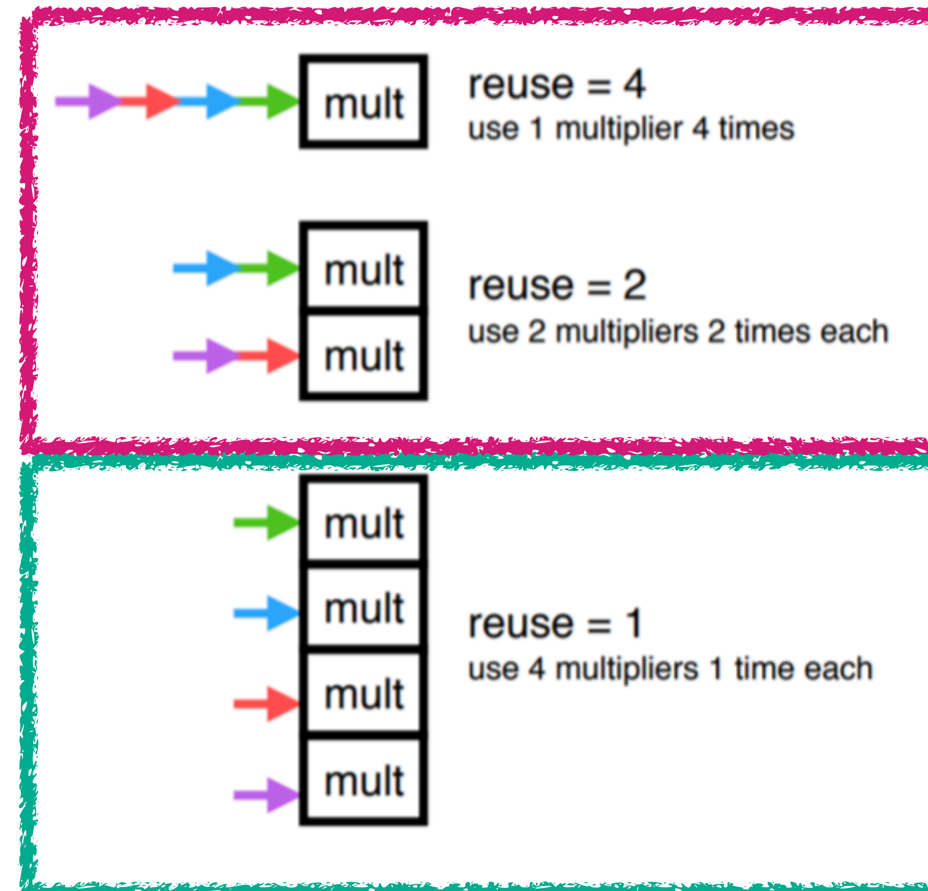


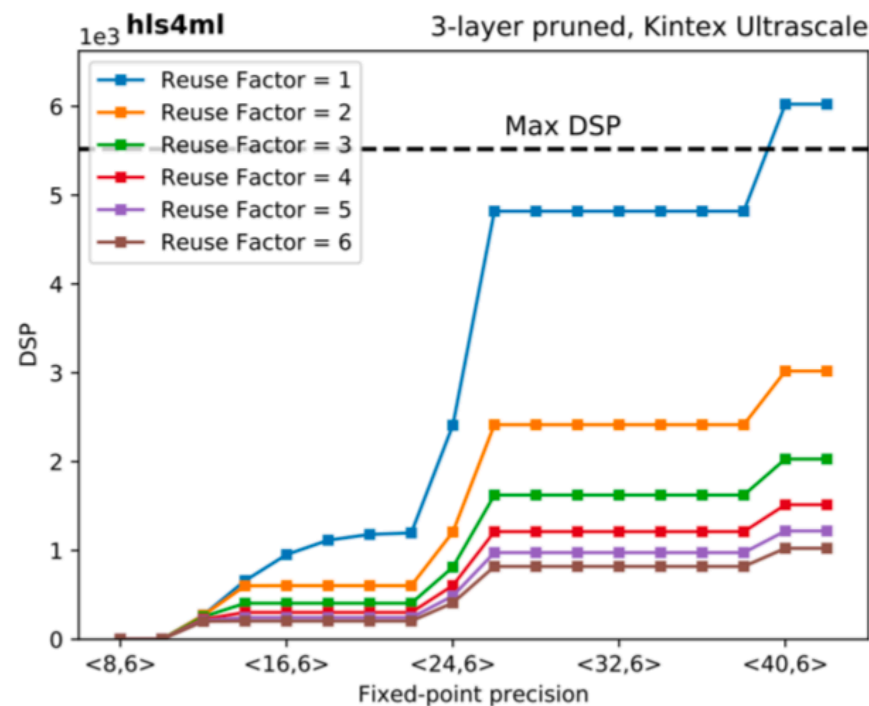


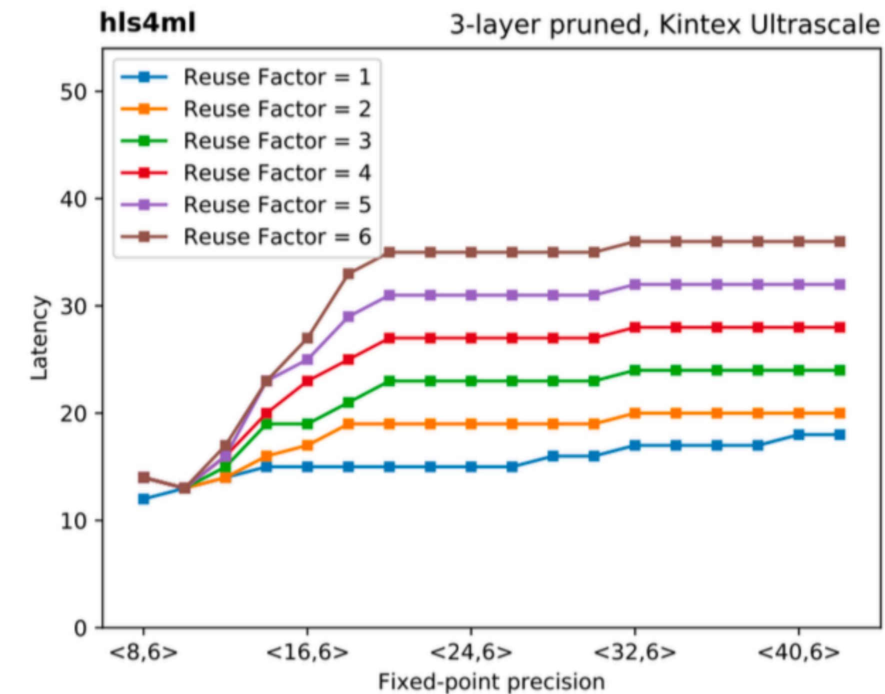
Reuse

- For lowest latency, compute all multiplications at once
  - Reuse = 1 (fully parallel) → latency = # layers)
- Larger reuse implies more serialization
- Allows trading higher latency for lower resource usage



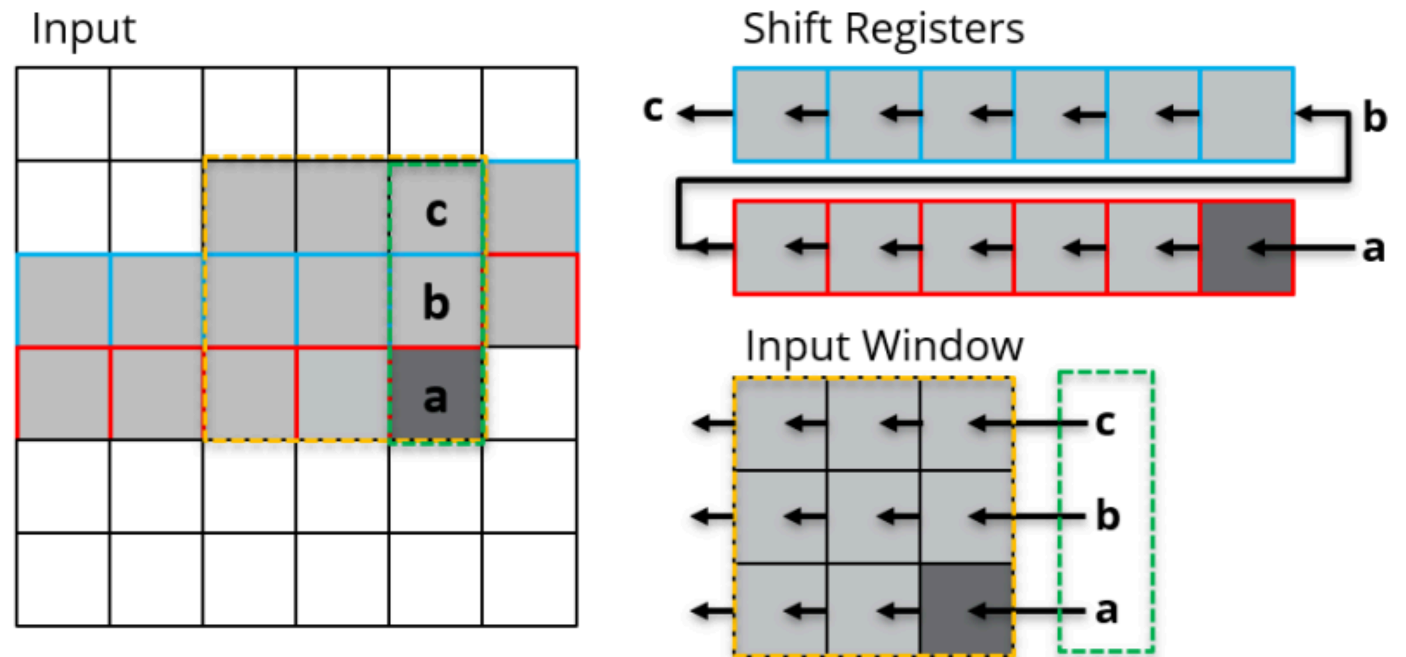


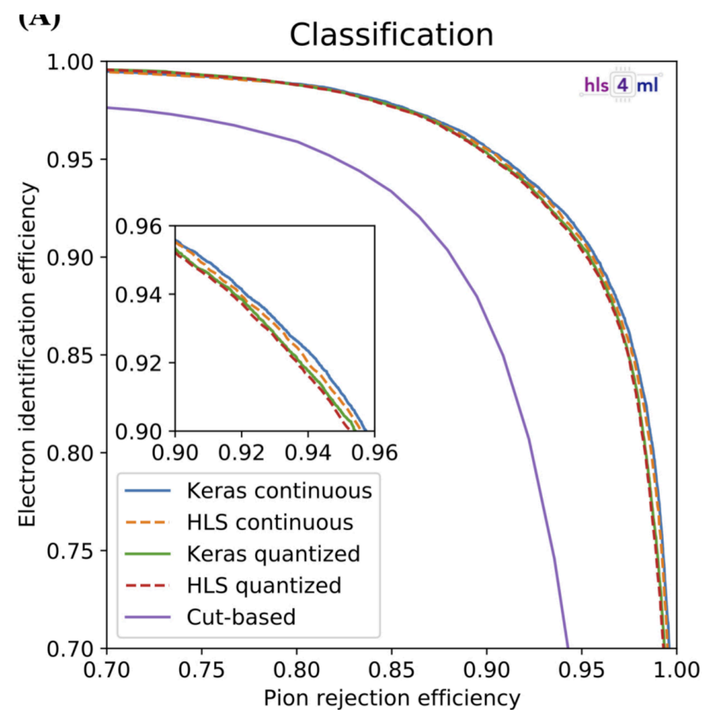


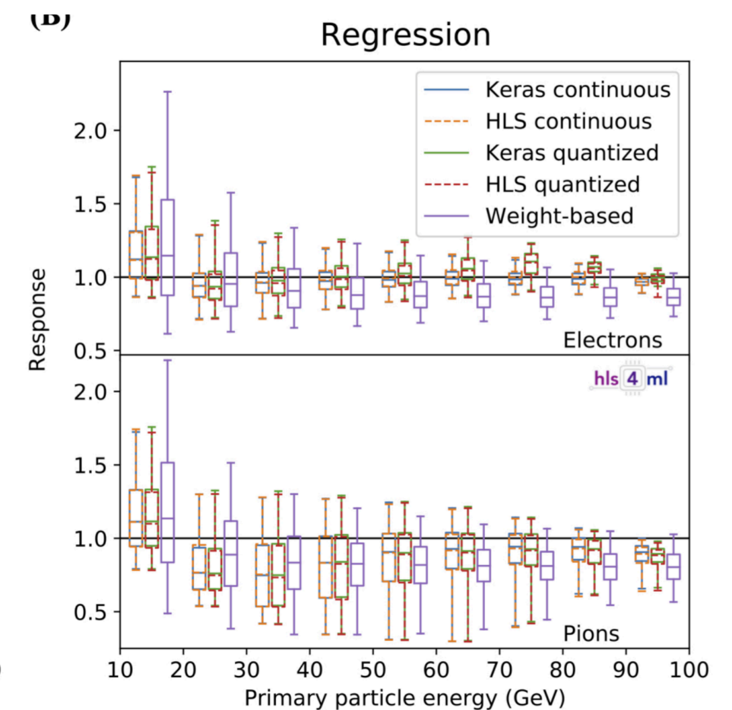


Other hls4ml Highlights

- Large CNN support [arXiv:2101.05108]
    - Good resource scaling
- Boosted Decision Trees: [JINST 15 P05026 (2020)]
- GarNet / GravNet: [arXiv: 2008.03601]







Tutorials

- Tutorial series developed to introduce users to hls4ml, ML on FPGAs
  - https://github.com/fastmachinelearning/hls4ml-tutorial
  - Jupyter notebook-based
- Great way to learn about the tools capabilities
- Also a hands-on series (in-person / virtual)

Convert the model to FPGA firmware with hls4ml

Now we will go through the steps to convert the model we trained to a low-latency optimized FPGA firmware with hls4ml. First, we will evaluate its classification performance to make sure we haven't lost accuracy using the fixed-point data types. Then we will synthesize the model with Vivado HLS and check the metrics of latency and FPGA resource usage.

Make an hls4ml config & model

The hls4ml Neural Network inference library is controlled through a configuration dictionary. In this example we'll use the most simple variation, later exercises will look at more advanced configuration.

In [ ]:
```
import hls4ml
config = hls4ml.utils.config_from_keras_model(model, granularity='model')
print("-----------------------------------")
print("Configuration")
plotting.print_dict(config)
print("-----------------------------------")
hls_model = hls4ml.converters.convert_from_keras_model(model,
                                                       hls_config=config,
                                                       output_dir='model_1/hls4ml_prj',
                                                       fpga_part='xcu250-figd2104-2L-e')
```

Let's visualise what we created. The model architecture is shown, annotated with the shape and data types

In [ ]:
```
hls4ml.utils.plot_model(hls_model, show_shapes=True, show_precision=True, to_file=None)
```

Compile, predict

Now we need to check that this model performance is still good. We compile the hls_model, and then use `hls_model.predict` to execute the FPGA firmware with bit-accurate emulation on the CPU.

In [ ]:
```
hls_model.compile()
X_test = np.ascontiguousarray(X_test)
y_hls = hls_model.predict(X_test)
```

Compare

That was easy! Now let's see how the performance compares to Keras:

In [ ]:
```
print("Keras  Accuracy: {}".format(accuracy_score(np.argmax(y_test, axis=1), np.argmax(y_keras, axis=1))))
print("hls4ml Accuracy: {}".format(accuracy_score(np.argmax(y_test, axis=1), np.argmax(y_hls, axis=1))))

fig, ax = plt.subplots(figsize=(9, 9))
_ = plotting.makeRoc(y_test, y_keras, le.classes_)
plt.gca().set_prop_cycle(None) # reset the colors
_ = plotting.makeRoc(y_test, y_hls, le.classes_, linestyle='--')

from matplotlib.lines import Line2D
lines = [Line2D([0], [0], ls='-'),
         Line2D([0], [0], ls='--')]
from matplotlib.legend import Legend
leg = Legend(ax, lines, labels=['keras', 'hls4ml'],
            loc='lower right', frameon=False)
ax.add_artist(leg)
```

Synthesize

Now we'll actually use Vivado HLS to synthesize the model. We can run the build using a method of our `hls_model` object. After running this step, we can integrate the generated IP into a workflow to compile for a specific FPGA board. In this case, we'll just review the reports that Vivado HLS generates, checking the latency and resource usage.

This can take several minutes.

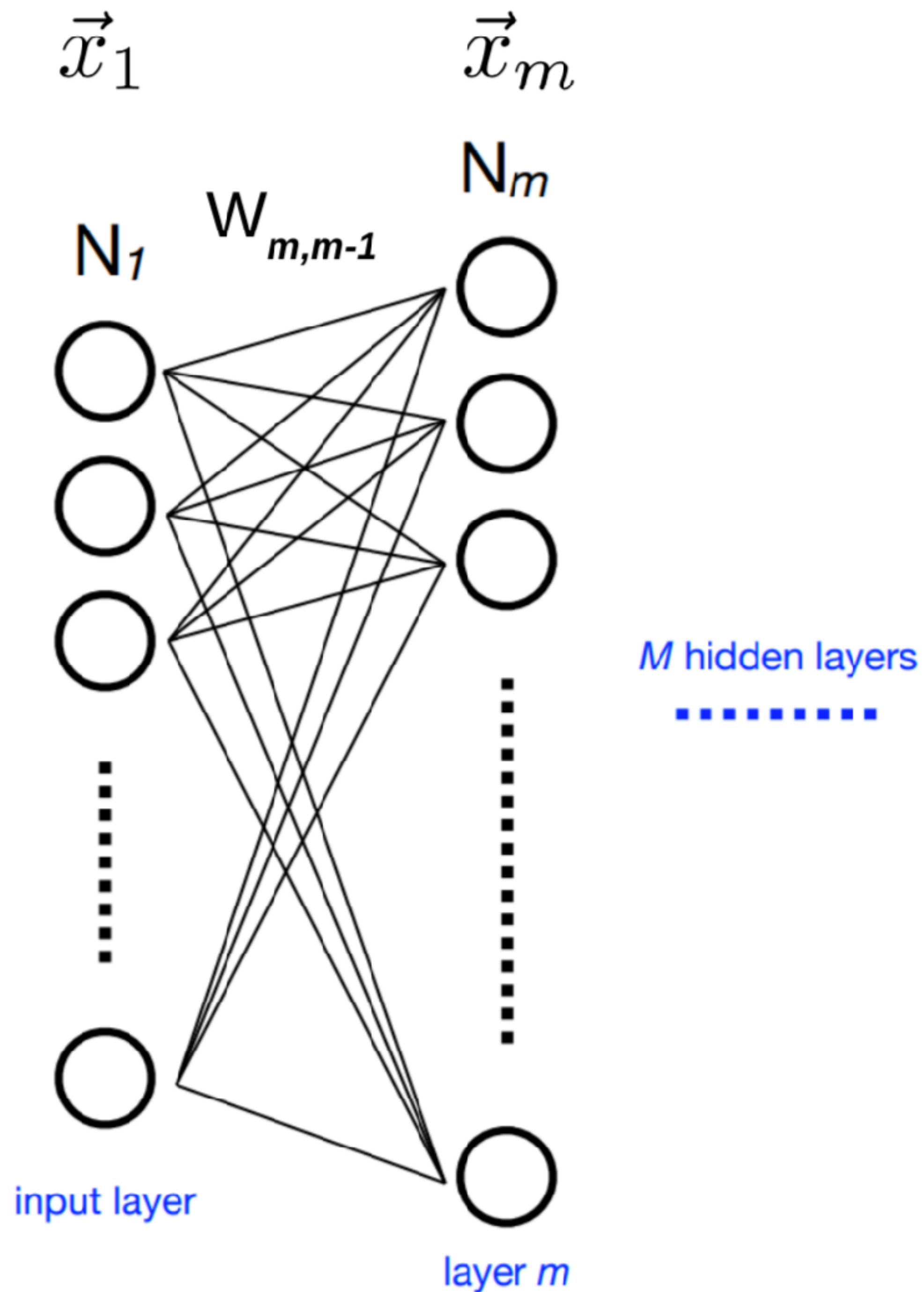
While the C-Synthesis is running, we can monitor the progress looking at the log file by opening a terminal from the notebook home, and executing:

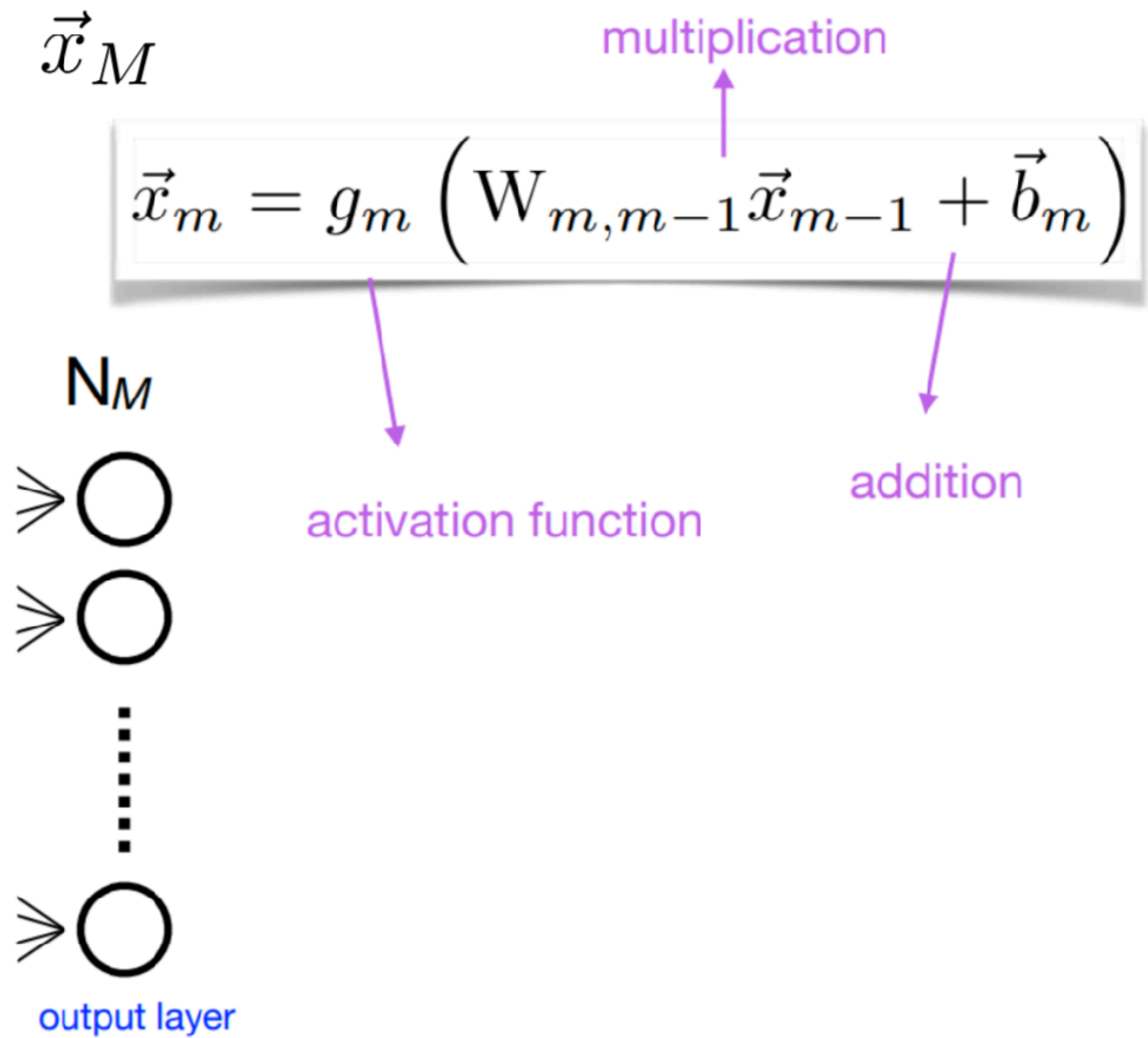
```
tail -f model_1/hls4ml_prj/vivado_hls.log
```

In [ ]:
```
hls_model.build(csim=False)
```

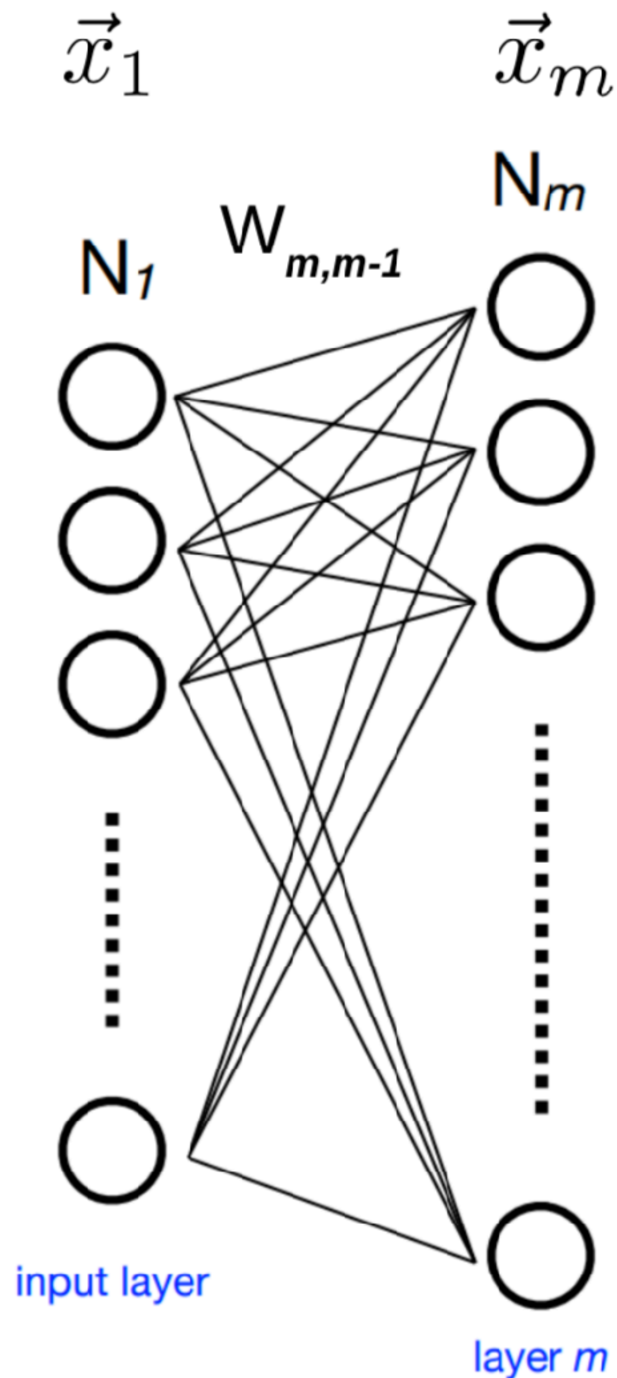
BACKUP

Inference on FPGAs

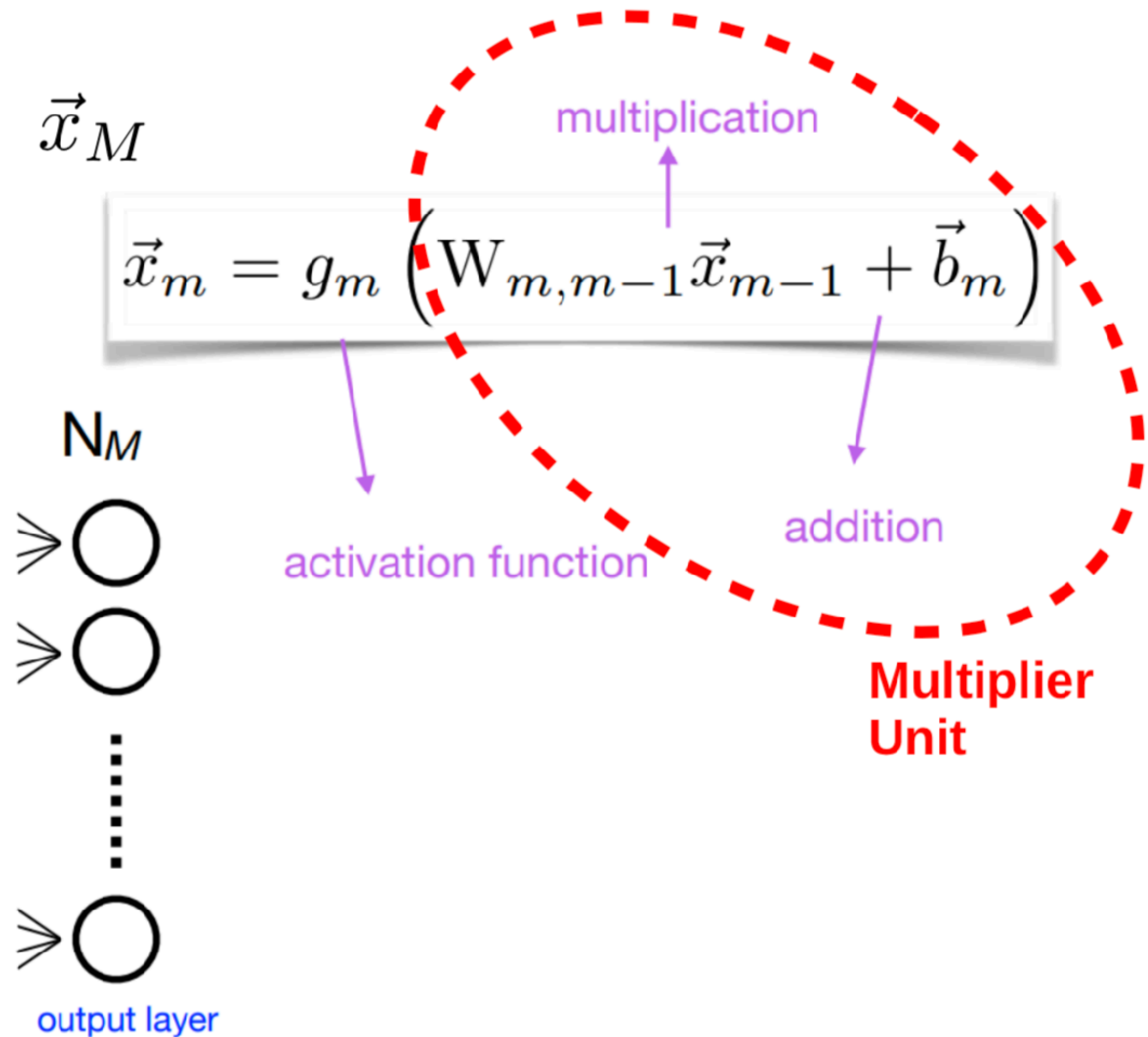




Inference on FPGAs

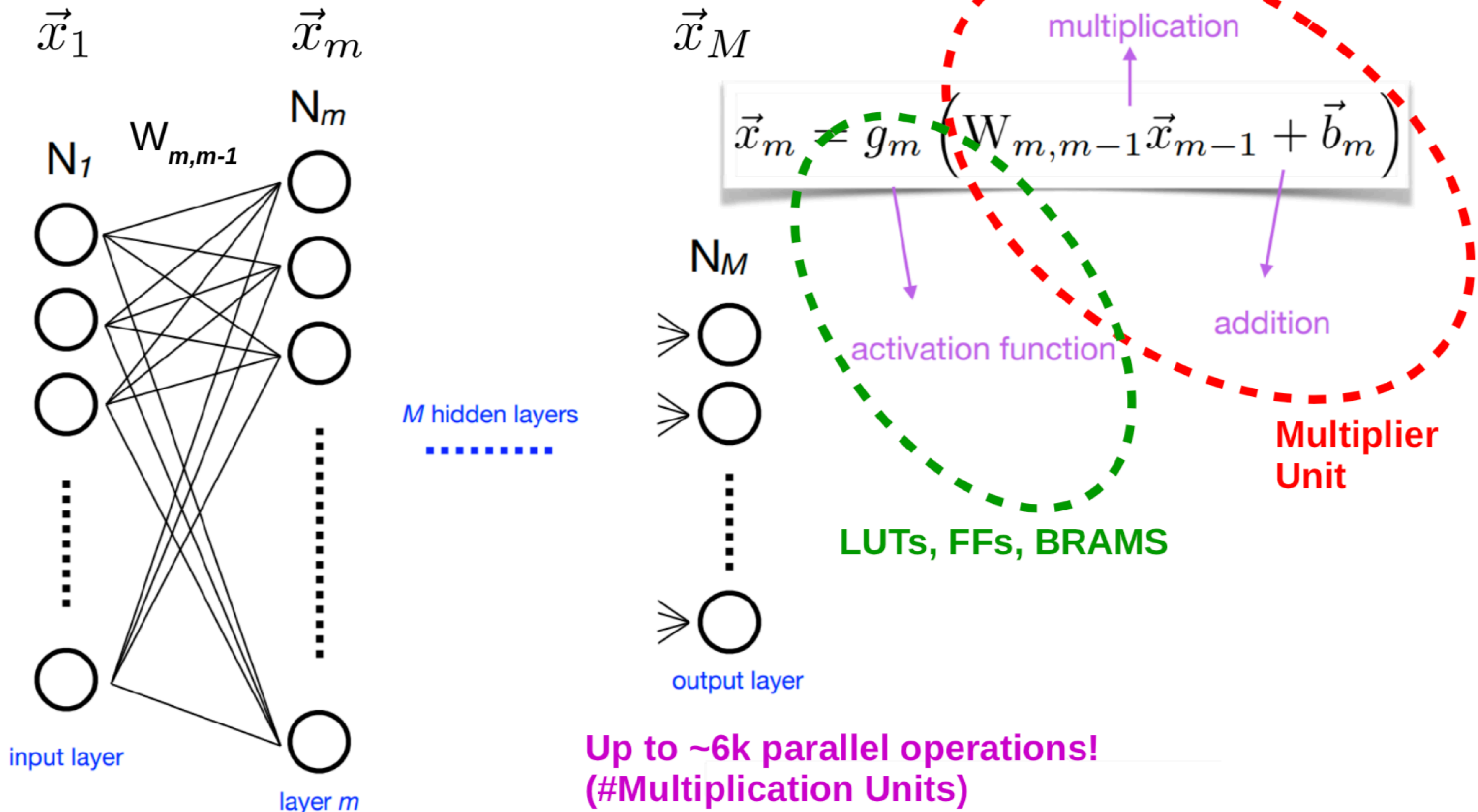


M hidden layers

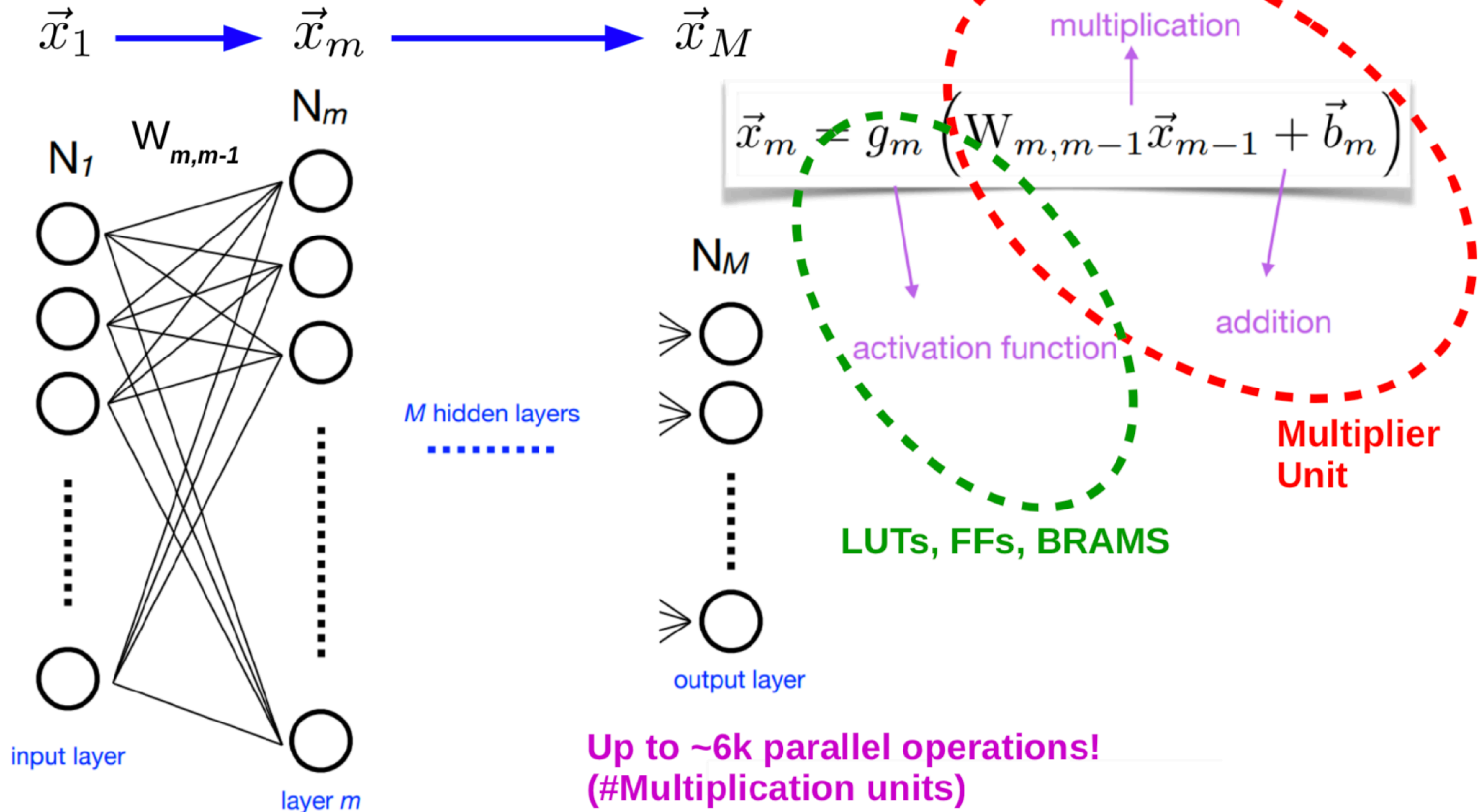


Up to ~6k parallel operations!
(#Multiplication Units)

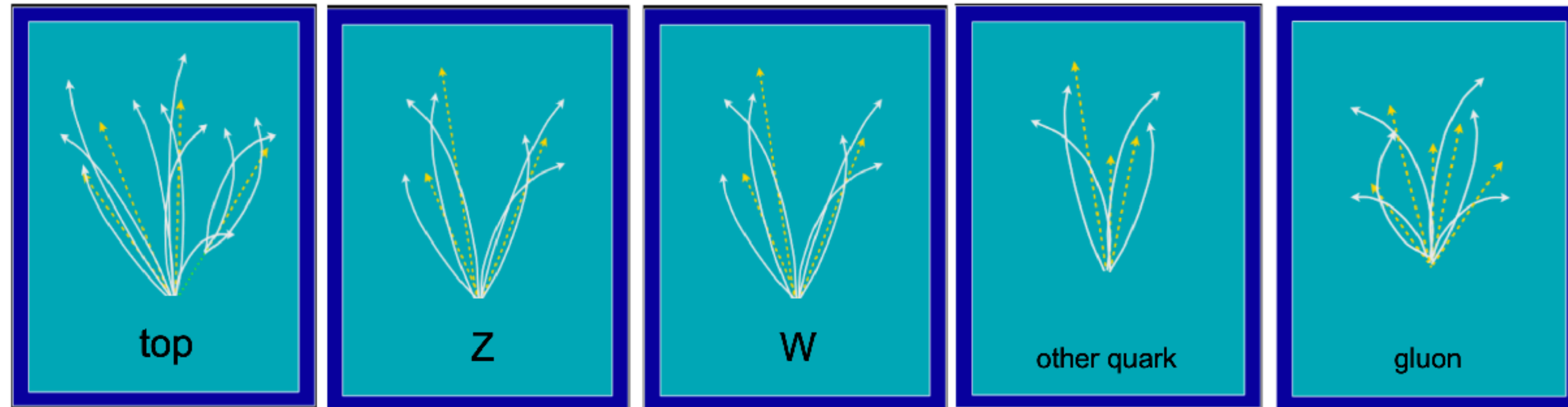
Inference on FPGAs

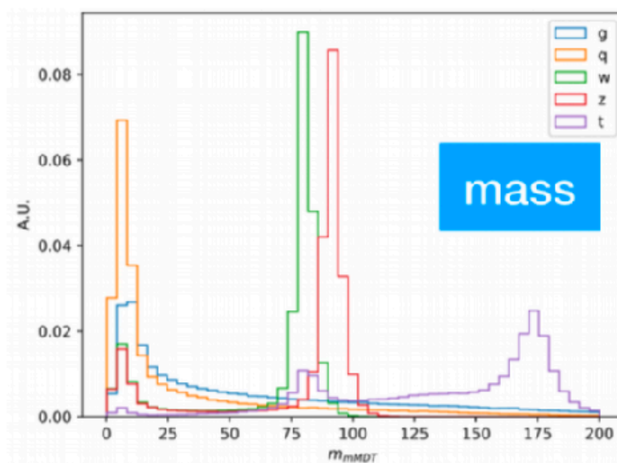


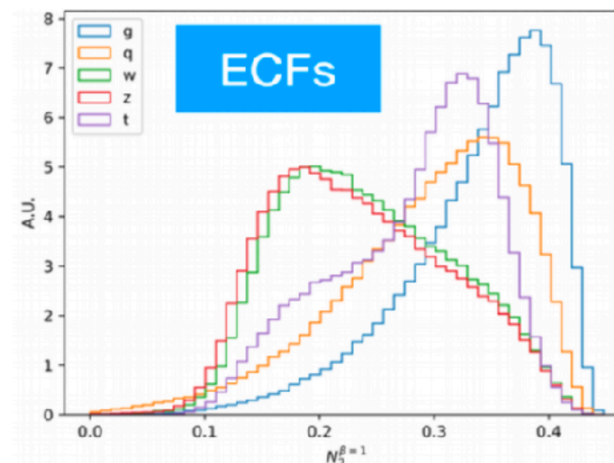
Inference on FPGAs

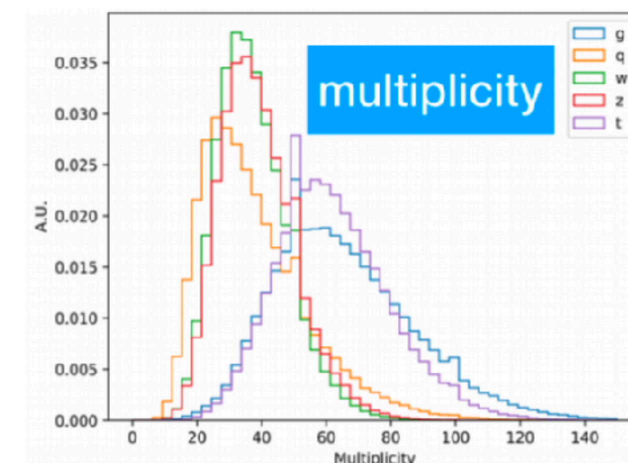


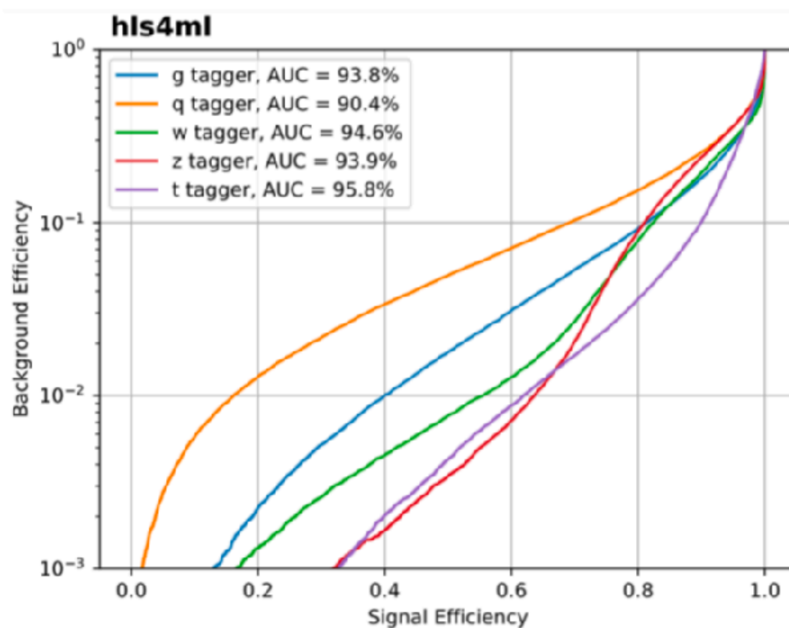
Example Network - Jet Tagging





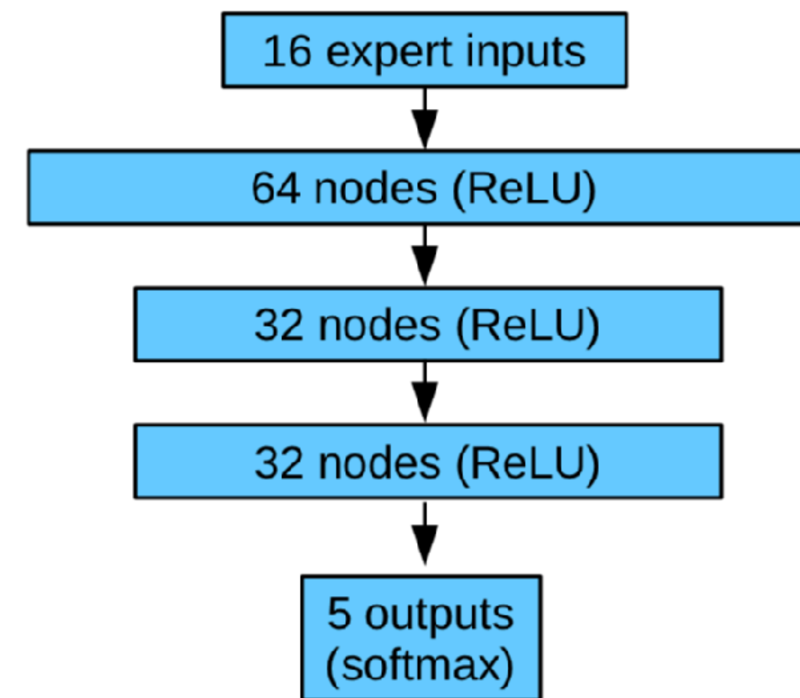






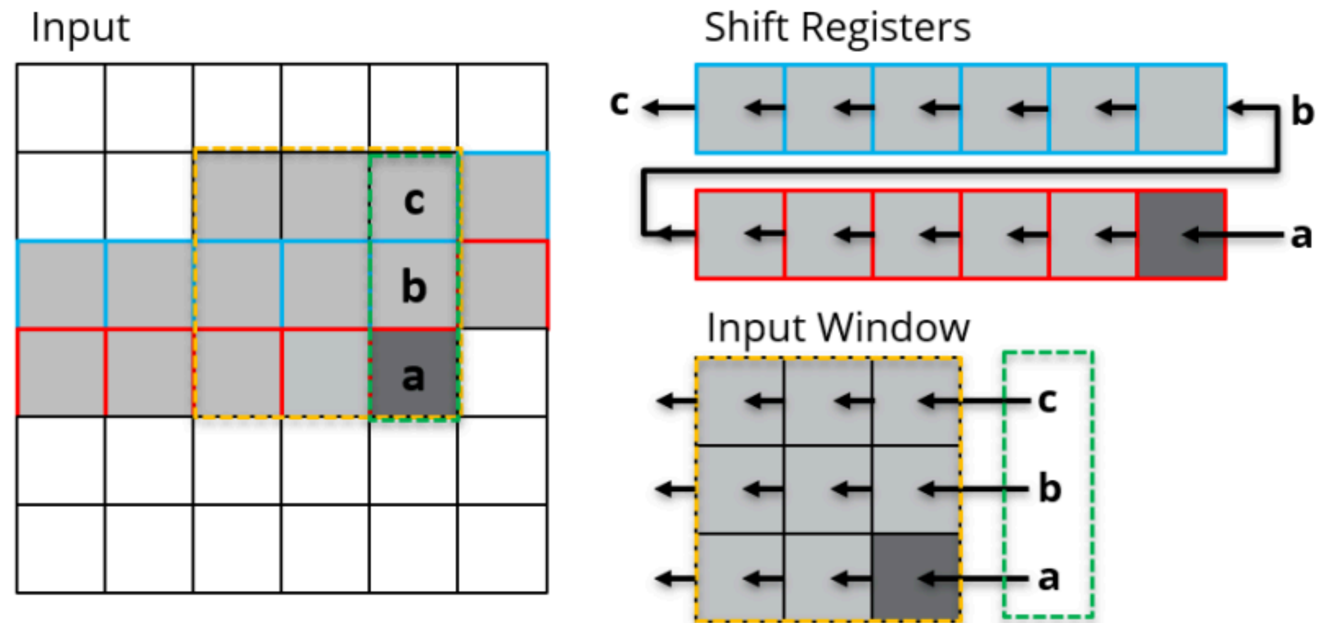
Observables

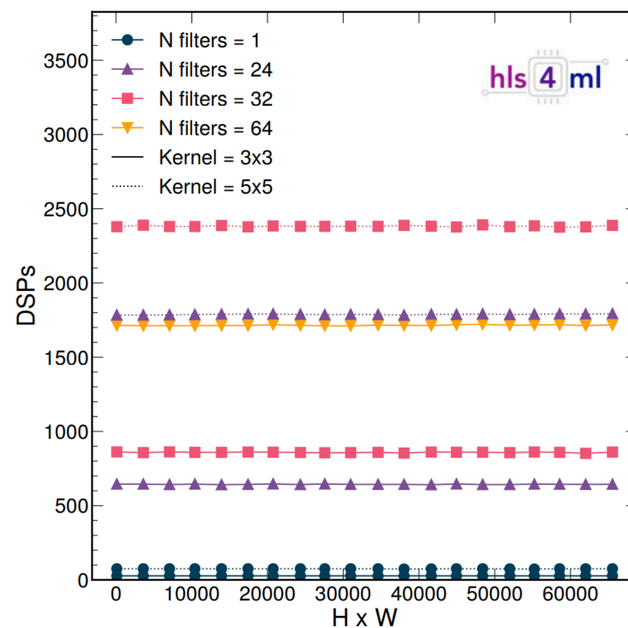
m_{mMDT}
$N_2^{\beta=1,2}$
$M_2^{\beta=1,2}$
$C_1^{\beta=0,1,2}$
$C_2^{\beta=1,2}$
$D_2^{\beta=1,2}$
$D_2^{(\alpha,\beta)=(1,1),(1,2)}$
$\sum z \log z$
Multiplicity

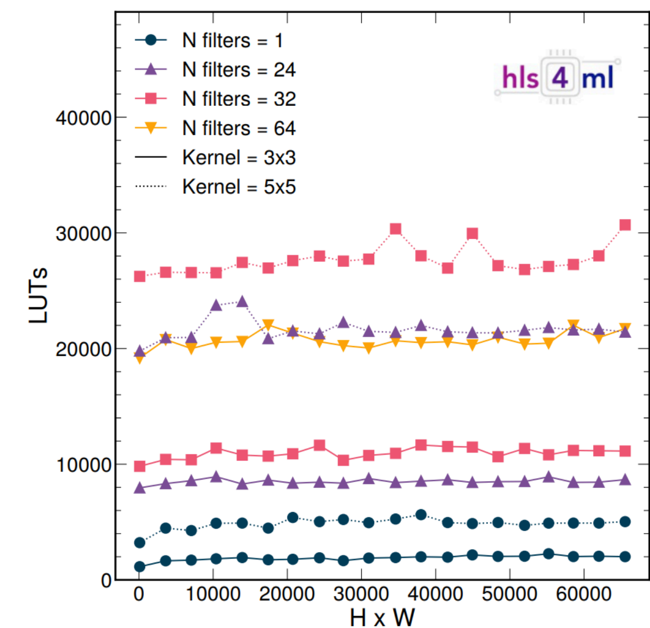


CNNs

- Special adjustments necessary to implement convolutional networks on FPGAs
  - HLS struggles with very long (nested) loops
- hls4ml is now able to synthesize large CNNs with good resource scaling
- Further optimizations possible for lower latencies
- arXiv:2101.05108







GarNet

- Graph networks have become very popular for complex geometric problems
    - Iterative nature difficult for FPGAs
- Modified GarNet architecture implemented in hls4ml
    - arXiv:2008.0360
- Model developed for HGCal cluster ID and energy regression
    - Able to run in under 1 μs, fit within a single VU9P SLR