JAVIER DUARTE
(ON BEHALF OF MANY OTHERS)

OCTOBER 21, 2020

IRIS-HEP MEETING

GNN TRACKING ON FPGAS

MANY OTHERS

Aneesh Heintz*
Cornell Unviersity
Ithaca, NY 14850, USA

Vesal Razavimaleki*, Javier Duarte
University of California San Diego
La Jolla, CA 92130, USA

Gage DeZoort, Isobel Ojalvo, Savannah Thais
Princeton University
Princeton, NJ 08544, USA

Markus Atkinson, Mark Neubauer
University of Illinois at Urbana-Champaign
Champaign, IL 61820, USA

Lindsey Gray, Sergo Jindariani, Nhan Tran
Fermi National Accelerator Laboratory
Batavia, IL 60310, USA

Philip Harris, Dylan Rankin
Massachusetts Institute of Technology
Batavia, IL 60310, USA

Thea Aarrestad, Vladimir Loncar†, Maurizio Pierini, Sioni Summers
European Organization for Nuclear Research (CERN)
CH-1211 Geneva 23, Switzerland

Jennifer Ngadiuba
California Institute of Technology
Pasadena, CA 92115, USA

Mia Liu
Purdue University
West Lafayette, IN 47907, USA

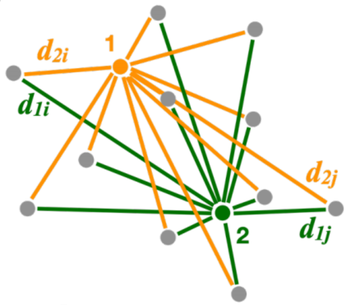
Edward Kreinar
HawkEye360
Herndon, VA 20170, USA

Zhenbin Wu
University of Illinois at Chicago
Chicago, IL 60607, USA

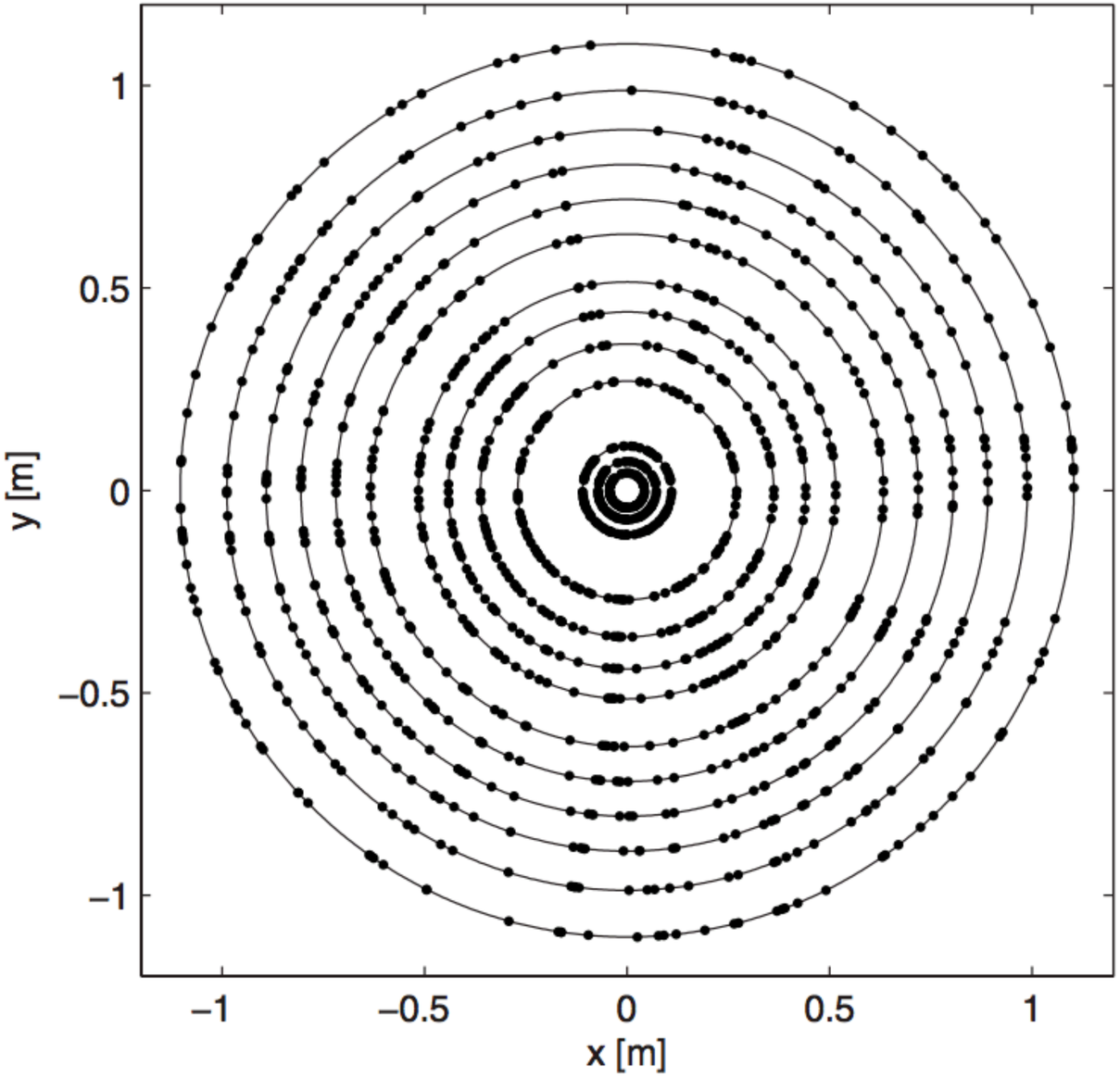
*IRIS-HEP fellows

INTRODUCTION AND PREVIOUS WORK

- GNN tracking [arXiv:1810.06111, arXiv:2003.11603] may scale better than traditional tracking algorithms and may be more easily parallelized on heterogenous computing resources like FPGAs
- Previous work [arXiv:2008.03601] has implemented simple GNNs on FPGAs for particle energy regression

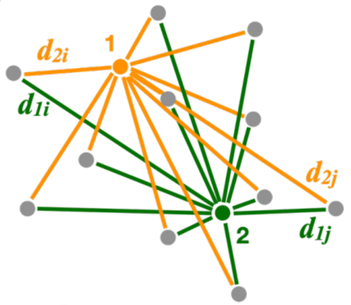


- This talk: two FPGA implementations of GNN segment classifiers using hls4ml and OpenCL

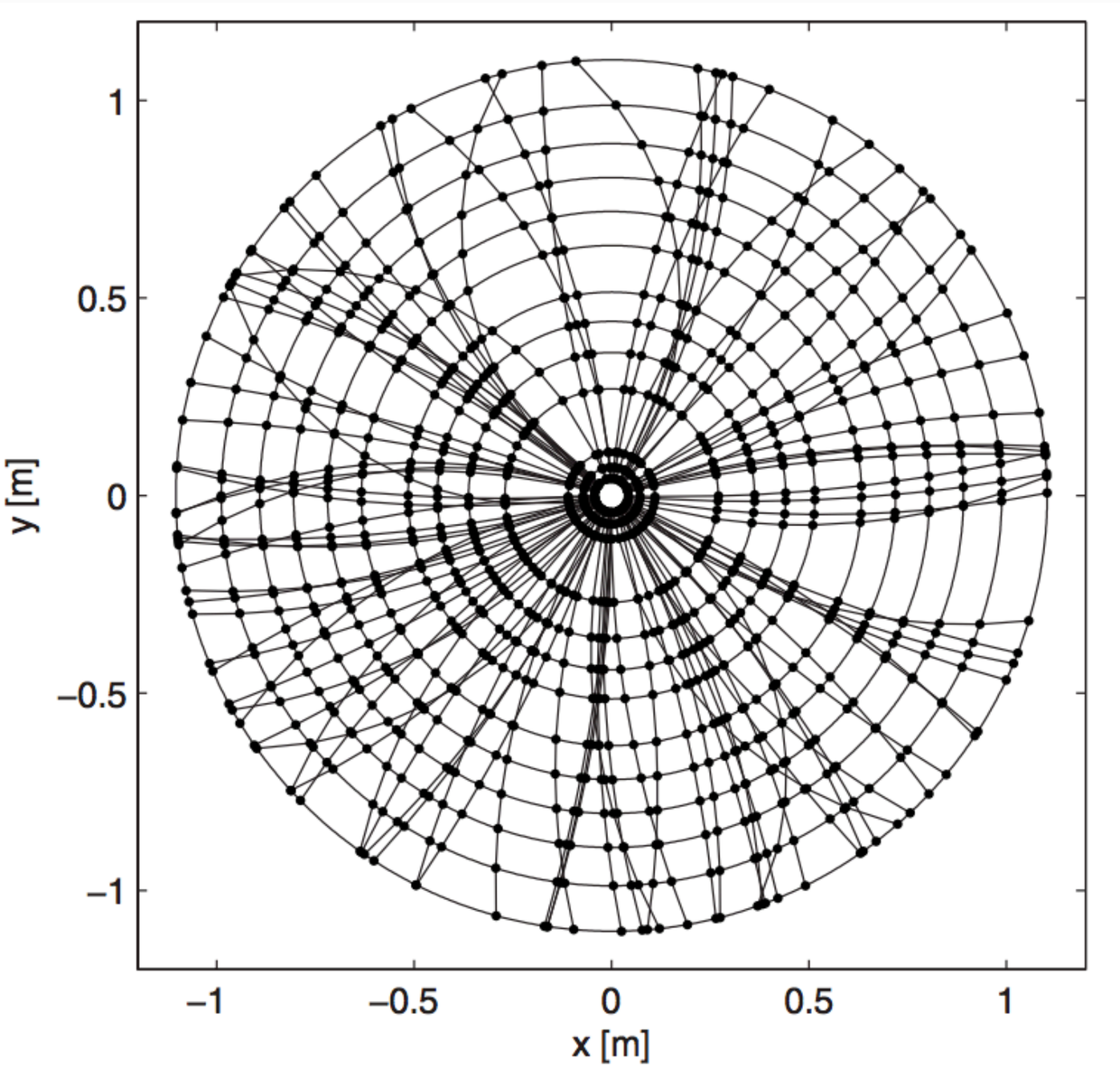


INTRODUCTION AND PREVIOUS WORK

- GNN tracking [arXiv:1810.06111, arXiv:2003.11603] may scale better than traditional tracking algorithms and may be more easily parallelized on heterogenous computing resources like FPGAs
- Previous work [arXiv:2008.03601] has implemented simple GNNs on FPGAs for particle energy regression

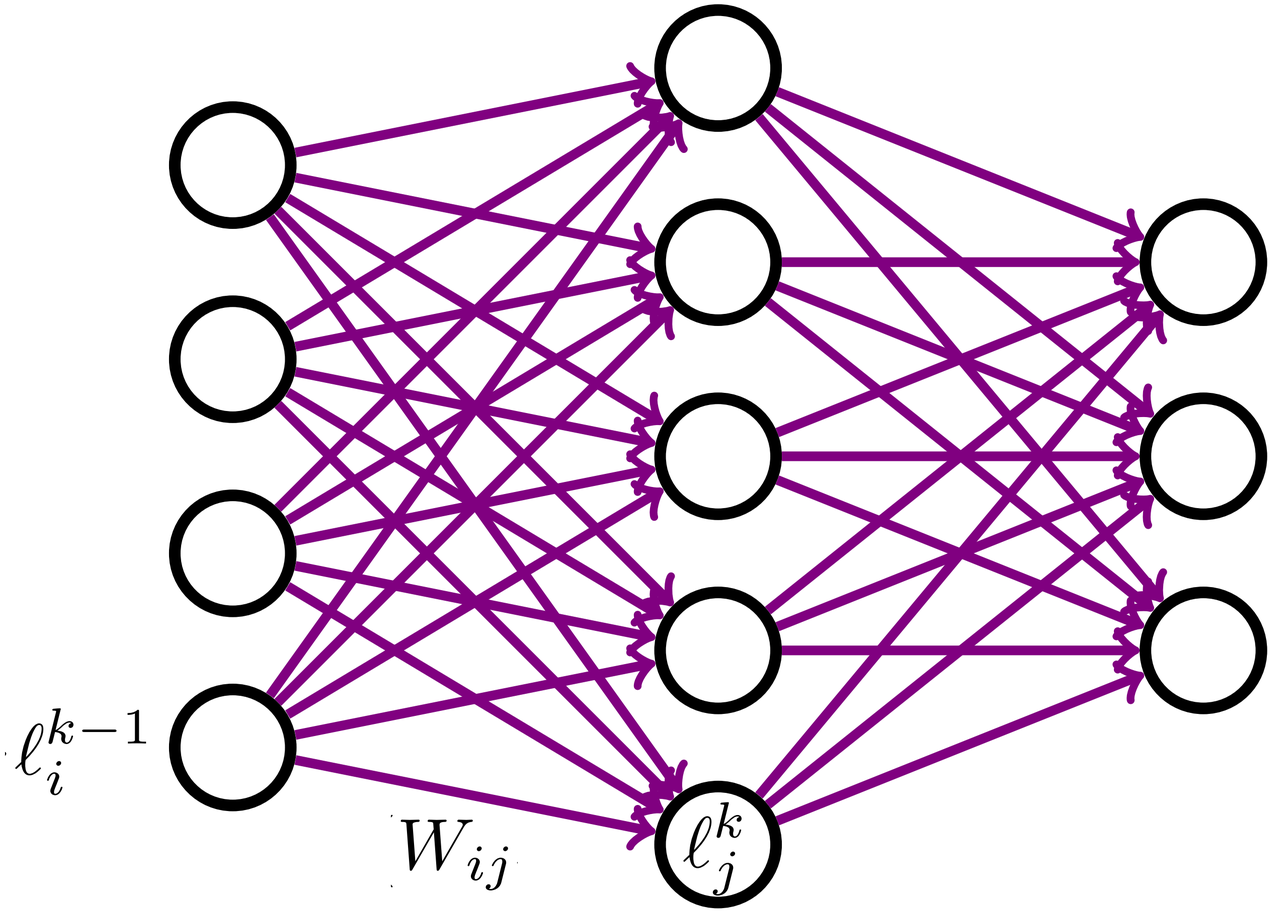


- This talk: two FPGA implementations of GNN segment classifiers using hls4ml and OpenCL

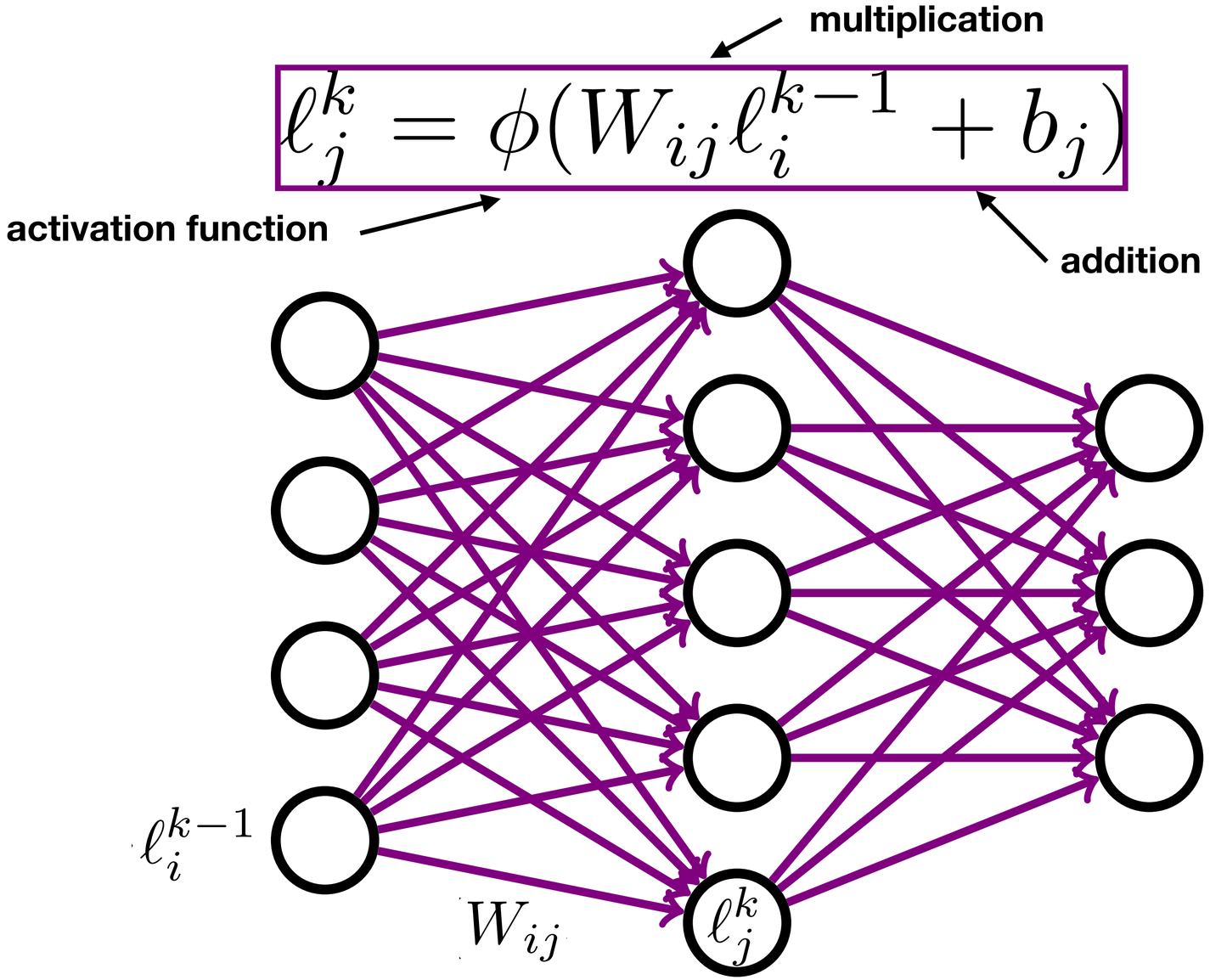


NEURAL NETWORK OPERATIONS

$$\ell_j^k = \phi(W_{ij}\ell_i^{k-1} + b_j)$$



NEURAL NETWORK OPERATIONS



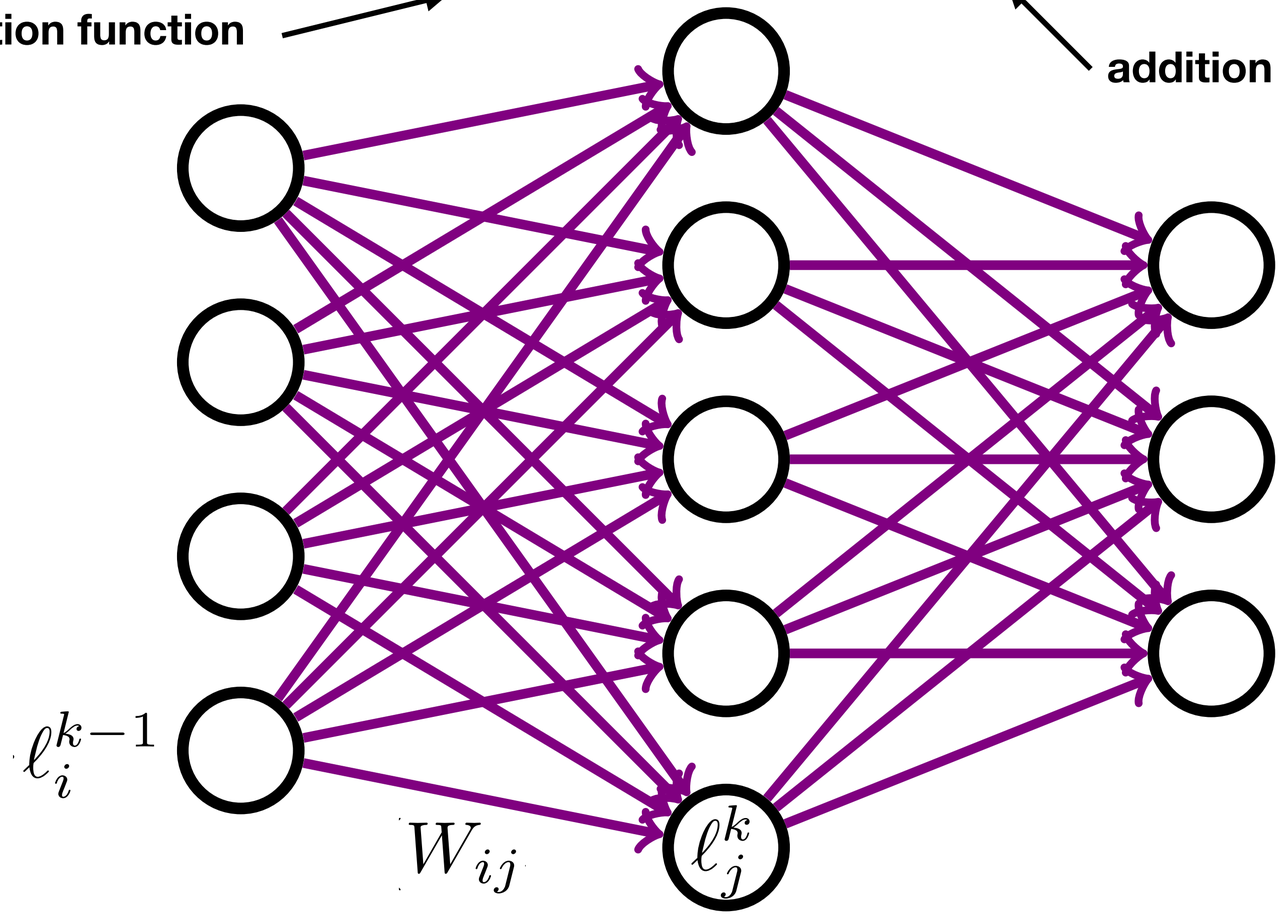
NEURAL NETWORK OPERATIONS

multiplication

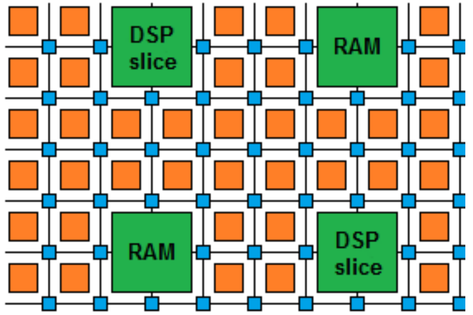
$$\ell_j^k = \phi(W_{ij}\ell_i^{k-1} + b_j)$$

activation function

addition

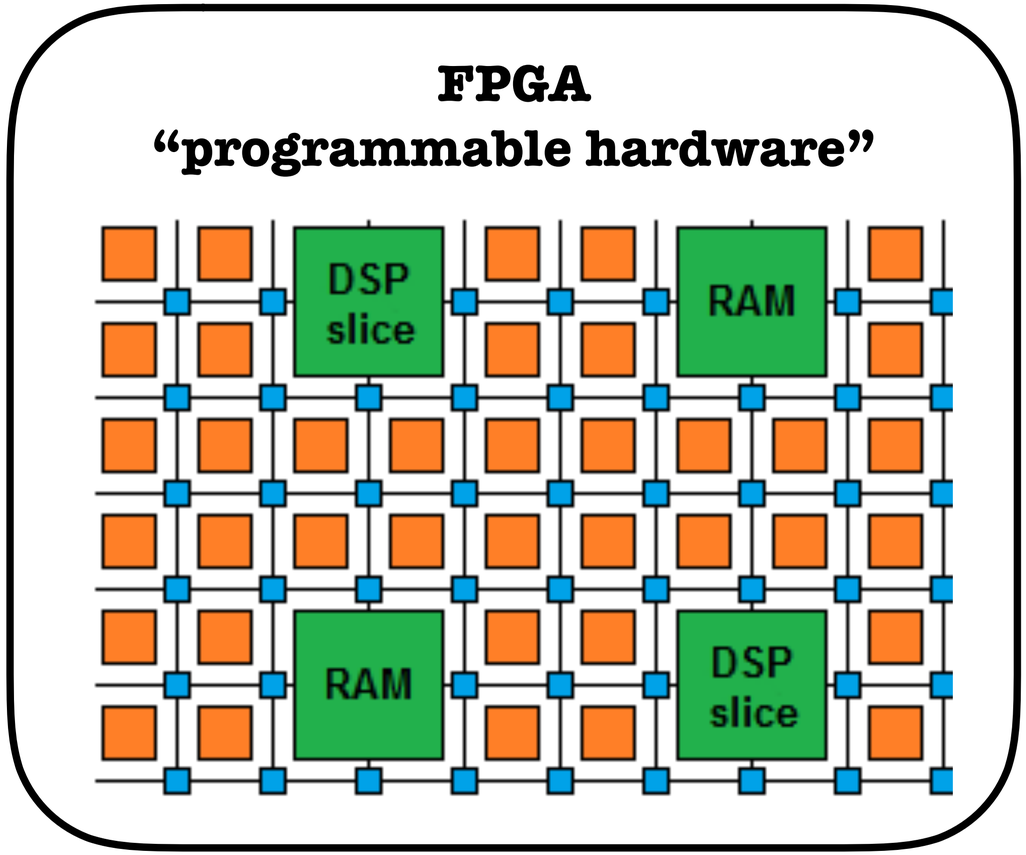


Maps nicely onto FPGA resources: high IO, DSPs, LUTs, etc.



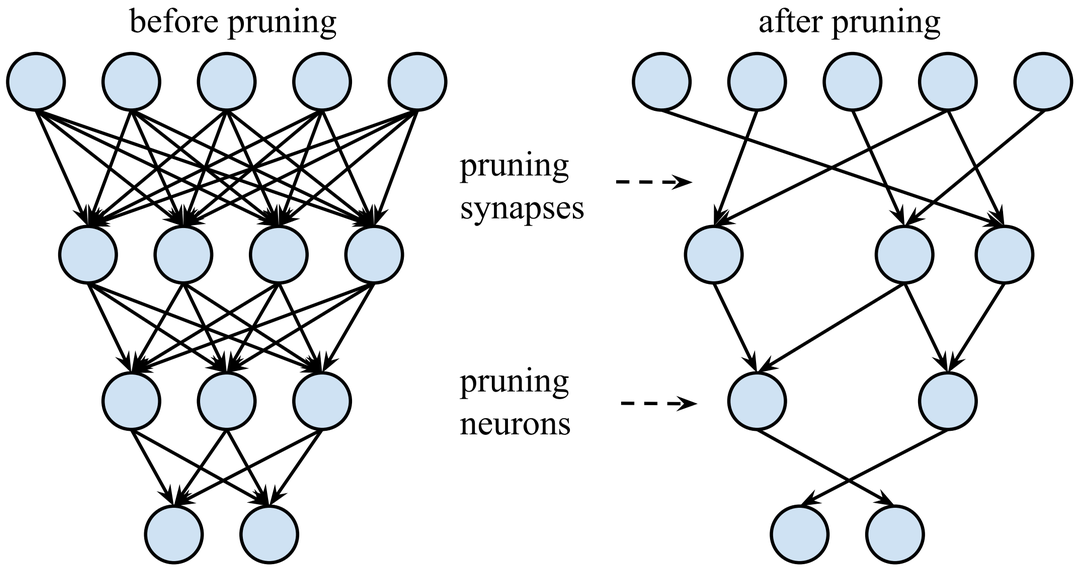
FPGAS: REPROGRAMMABLE HARDWARE

- Pros:
  - Reprogrammable interconnects between embedded components that perform multiplication (DSPs), apply logical functions (LUTs), or store memory (BRAM)
  - High throughput connections to other FPGAs: O(100) optical transceivers running at O(15) Gbps
  - Parallelization and pipelining
  - Low power
- Cons:
  - Usually requires domain knowledge to program (using VHDL/Verilog)

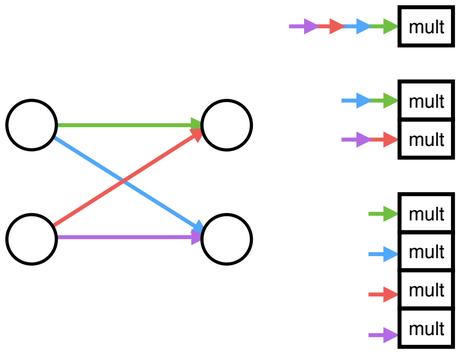


EFFICIENT MODEL HANDLES

1. **Compression**
   - **1.1. Smaller architectures**
   - 1.2. Pruning
     - Remove redundant connections



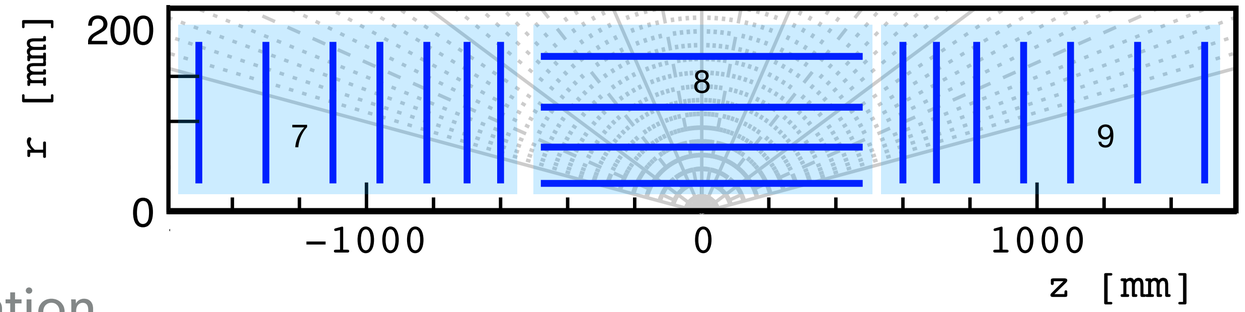
2. **Quantization**
   - **2.1. Post-training**
     - Reduce precision of trained model from 32-bit floating point to 16-bit fixed point, …
   - 2.2. Quantization-aware training (QAT)
     - Clip weights/activations during forward pass (but not backward pass) of training to emulate quantization
     - Can achieve as low as 6-bit, 4-bit, ternary, binary

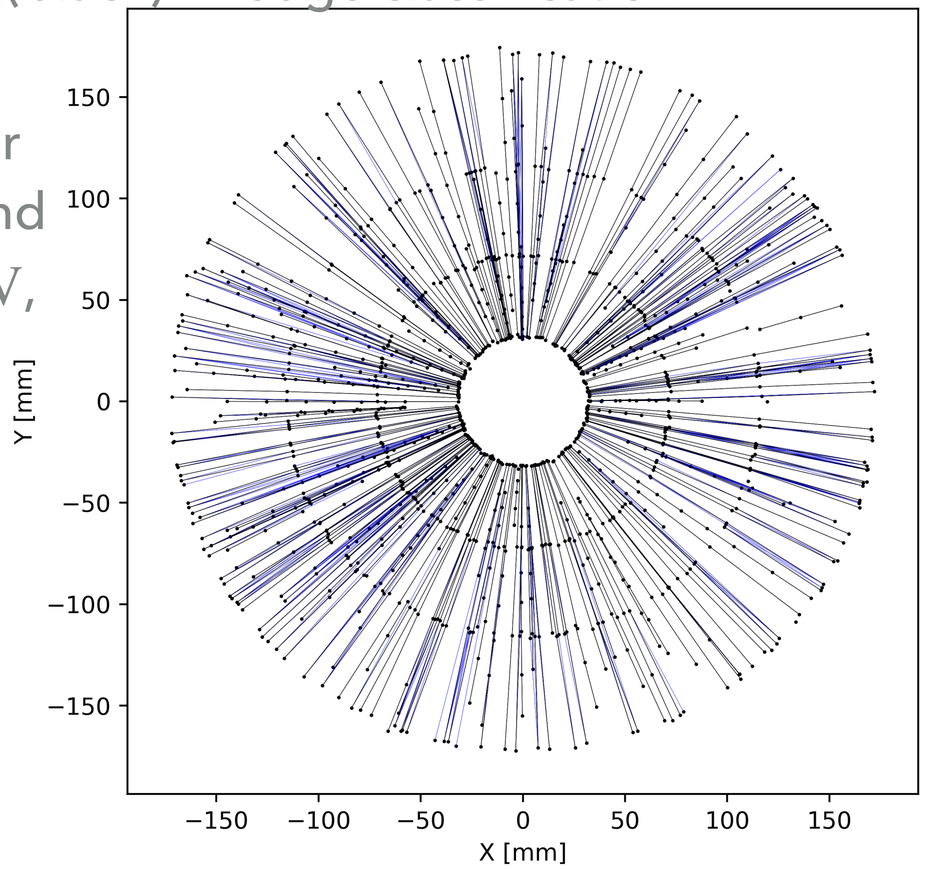


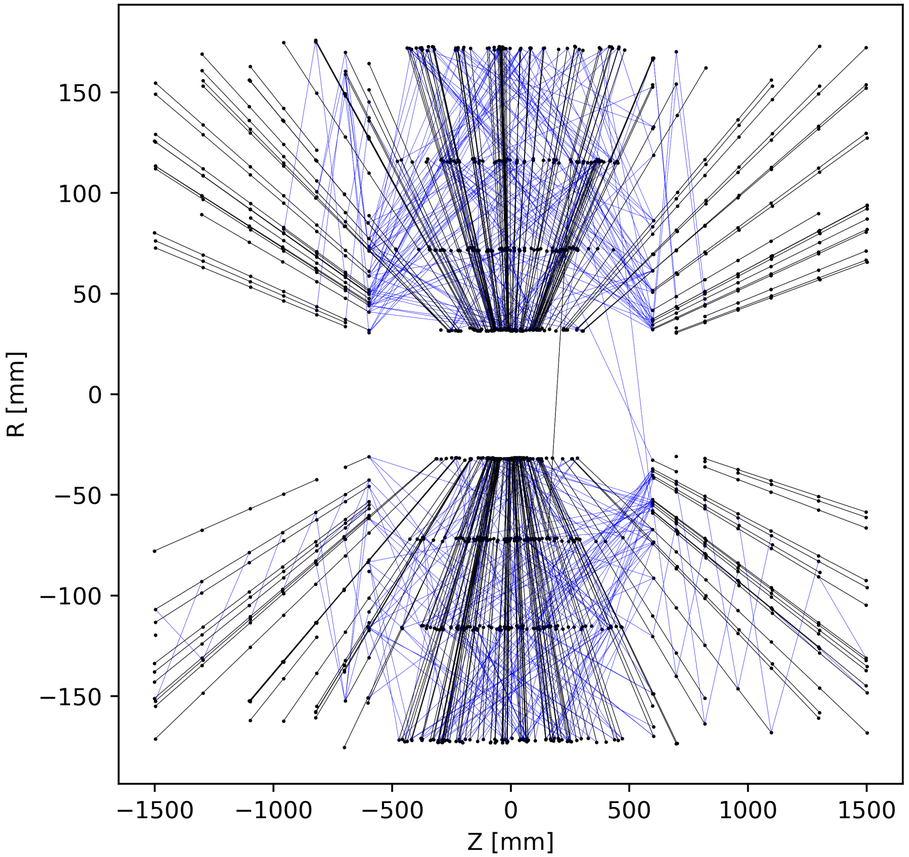
3. **Parallelization**
   - Balance parallelization (how fast) with resources needed (how costly)

GRAPH ENCODING OF DATA

- Kaggle TrackML dataset: https://www.kaggle.com/c/trackml-particle-identification/
  - Hits → nodes
  - Plausible track segments → edges
  - Good track segments (black) → edge classification targets
- Full event graphs of inner layers including barrel and endcaps with $p_T > 2$ GeV, $\Delta z < 15$ cm, and $\Delta\phi/\Delta r < 2.62 \times 10^{-4}$
- Contains ~1,100 nodes and ~1,500 edges
- Segment in ϕ and η for placement on FPGAs

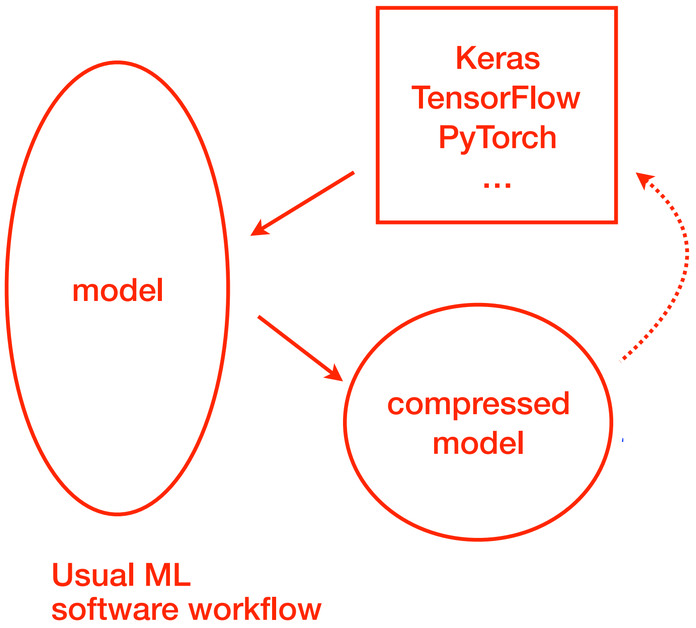






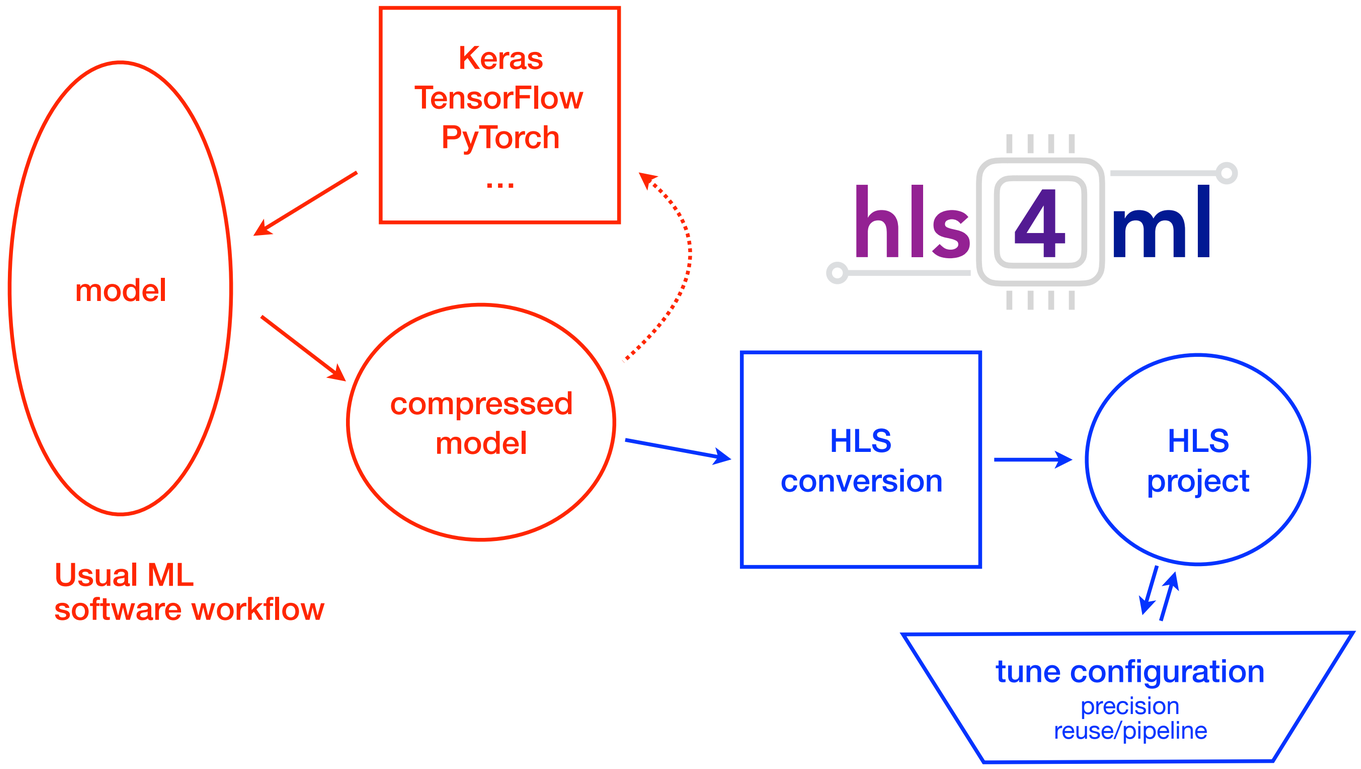
HLS4ML

- **hls4ml** for physicists or ML experts to translate **ML algorithms** into **FPGA firmware**



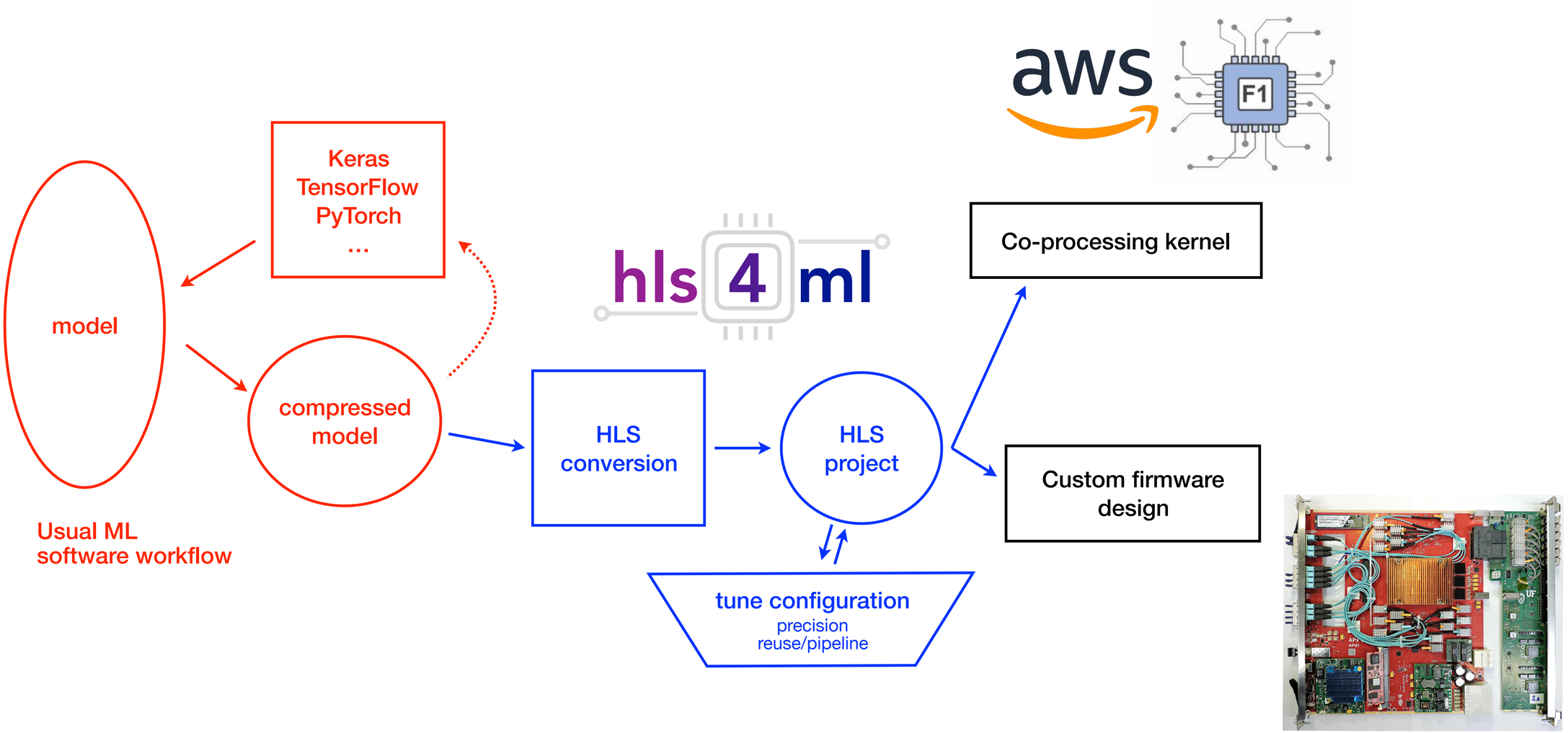
HLS4ML

- **hls4ml** for physicists or ML experts to translate **ML algorithms** into **FPGA firmware**

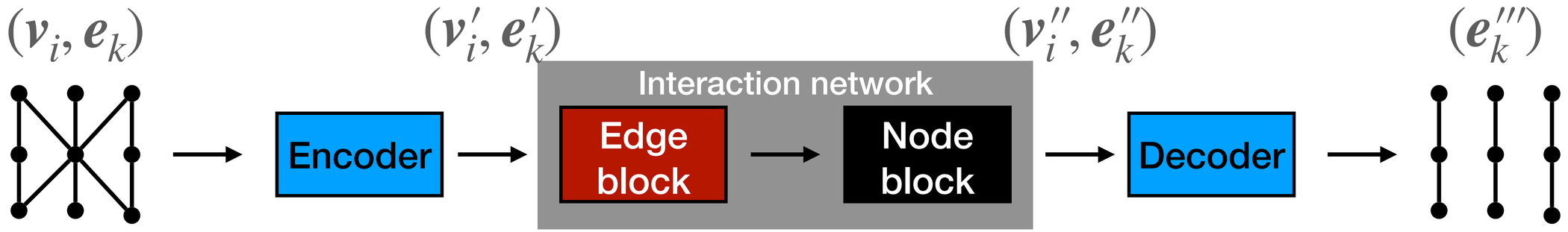


HLS4ML

- **hls4ml** for physicists or ML experts to translate **ML algorithms** into **FPGA firmware**



HLS4ML MODEL



$$e_k' = \phi_1^e(e_k) \quad e_k'' = \phi_2^e(e_k', v_{r_k}', v_{s_k}') \quad v_i'' = \phi_2^v(\bar{e}_i'', v_i') \quad e_k''' = \phi_3^e(e_k'')$$

$$v_i' = \phi_1^v(v_i) \quad \bar{e}_i'' = \rho^{e\to v}(E_i')$$

- Simplified Exa.TrkX NeurIPS 2019 segment classifier [arXiv:2003.11603]
  - Encoder transforms node and edge features into an 8-dim latent space
  - Edge block computes “messages” (updated edge features) between nodes
  - Node block updates node features based on these messages
  - Decoder transforms edge features into edge weight classifier
- IN may be iterated, but we use 1 iteration + small latent space to keep model small

ϕ_1^e :NN(8,ReLU,8,ReLU)
ϕ_1^v :NN(8,ReLU,8,ReLU)
ϕ_2^e :NN(8,ReLU,8,ReLU)
ϕ_2^v :NN(8,ReLU,8,ReLU)
ϕ_3^e :NN(8,ReLU,8,ReLU,8,ReLU,1,sigmoid)

HLS4ML IMPLEMENTATION


```
template<class data_T, class index_T, class res_T, typename CONFIG_T>
  void IN_edge_module(
                      data_T    Re[CONFIG_T::n_edge][CONFIG_T::n_hidden],
                      data_T    Rn[CONFIG_T::n_node][CONFIG_T::n_hidden],
                      index_T    receivers[CONFIG_T::n_edge][1],
                      index_T    senders[CONFIG_T::n_edge][1],
                      res_T     L[CONFIG_T::n_edge][CONFIG_T::n_hidden],
                      res_T     Q[CONFIG_T::n_node][CONFIG_T::n_hidden],
                      typename CONFIG_T::dense_config1::weight_t  core_edge_w0[3*CONFIG_T::n_hidden*CONFIG_T::n_hidden],
                      typename CONFIG_T::dense_config1::bias_t    core_edge_b0[CONFIG_T::n_hidden],
                      typename CONFIG_T::dense_config2::weight_t  core_edge_w1[CONFIG_T::n_hidden*CONFIG_T::n_hidden],
                      typename CONFIG_T::dense_config2::bias_t    core_edge_b1[CONFIG_T::n_hidden])
```

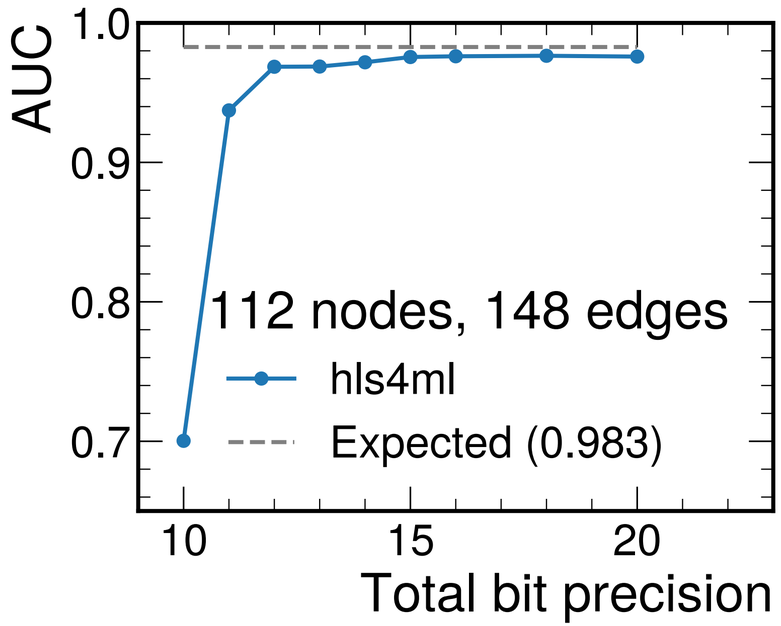
Edge block

```
#pragma HLS PIPELINE II=CONFIG_T::reuse_factor //*CONFIG_T::n_edge
IN_edge_loop: for(int i = 0; i < CONFIG_T::n_edge; i++) {
  #pragma HLS UNROLL
  index_T r = receivers[i][0];
  index_T s = senders[i][0];
  data_T l_logits[2*CONFIG_T::n_hidden];
  #pragma HLS ARRAY_PARTITION variable=l_logits complete dim=0
  nnet::merge<data_T, CONFIG_T::n_hidden, CONFIG_T::n_hidden>(Re[i], Rn[r], l_logits);
  data_T l[3*CONFIG_T::n_hidden];
  #pragma HLS ARRAY_PARTITION variable=l complete dim=0
  nnet::merge<data_T, 2*CONFIG_T::n_hidden, CONFIG_T::n_hidden>(l_logits, Rn[s], l);

  data_T L0_logits[CONFIG_T::dense_config1::n_out];
  #pragma HLS ARRAY_PARTITION variable=L0_logits complete dim=0
  nnet::dense_large_basic<data_T, data_T, typename CONFIG_T::dense_config1>(l, L0_logits, core_edge_w0, core_edge_b0);
```

- Implementation: https://github.com/vesal-rm/hls4ml/tree/graph_pipeline/example-prjs/graph/gnn_simple
  - “Receiver” and “sender” arrays used to dynamically index node feature matrix
  - “Zero-padding” used to set max graph size (edges and nodes)
  - Pipelining with initiation interval = reuse factor at edge/node-block-level

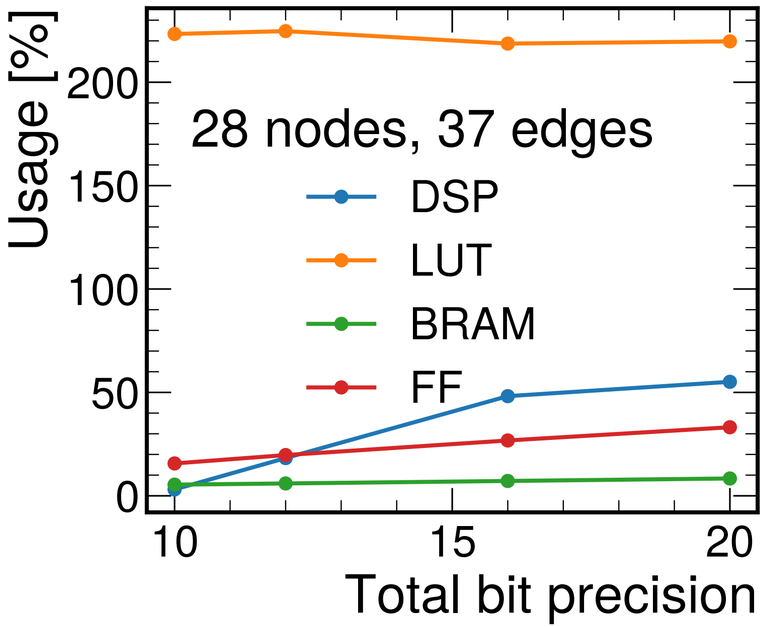
HLS4ML BIT PRECISION



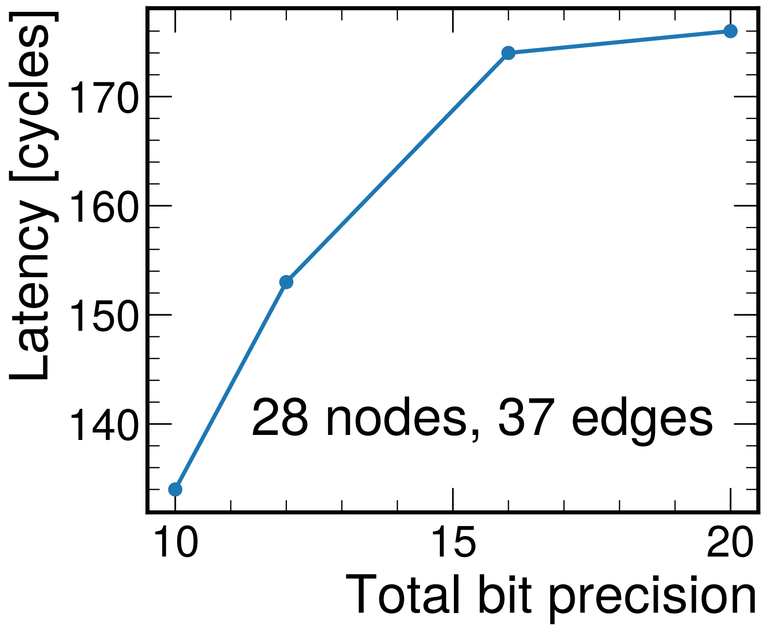
- Small GNN can correctly classify track segments with AUC ~ 0.983
- First, scan the fixed point precision bit width <total,integer> = <X,6>
  - Good performance with <12,6>
  - Note: QAT can likely reduce this to 6 bits or less

HLS4ML LATENCY & RESOURCES

Xilinx KU115 FPGA



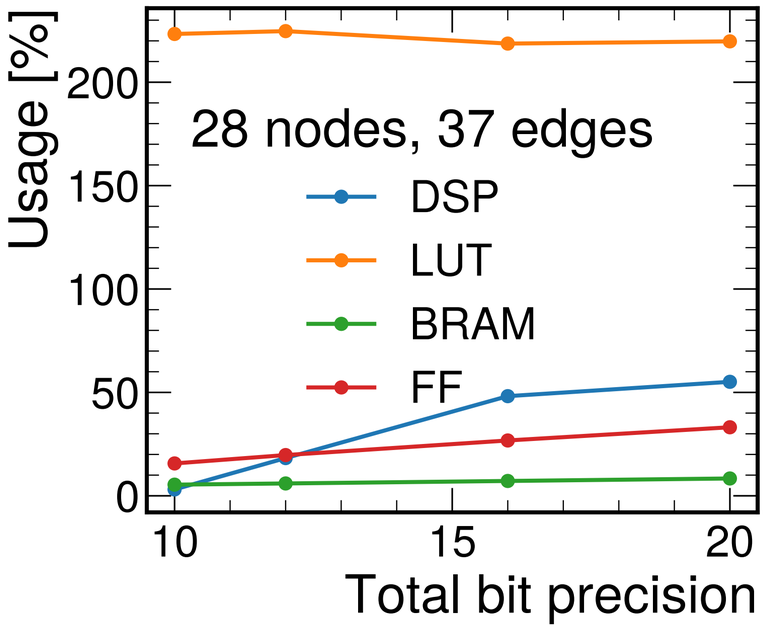
1 cycle = 5 ns



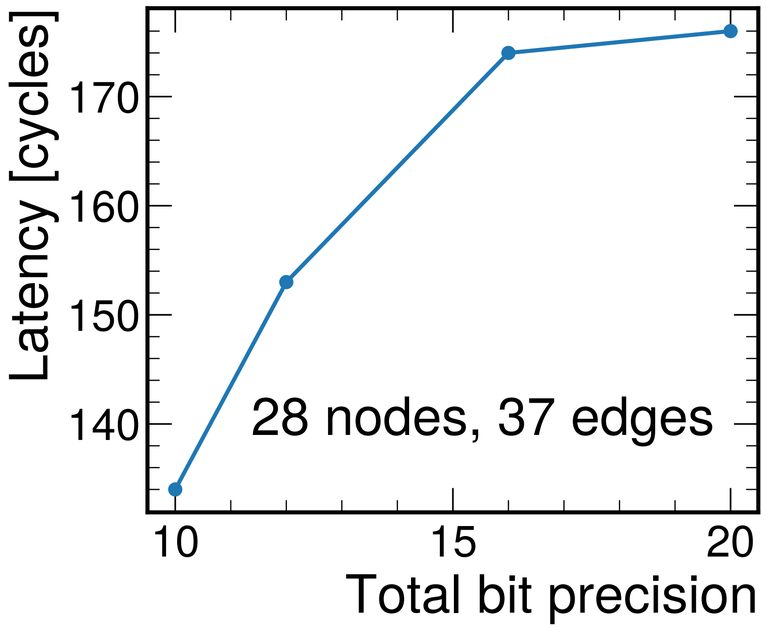
Note: LUTs are overestimated in HLS

HLS4ML LATENCY & RESOURCES

Xilinx KU115 FPGA



1 cycle = 5 ns

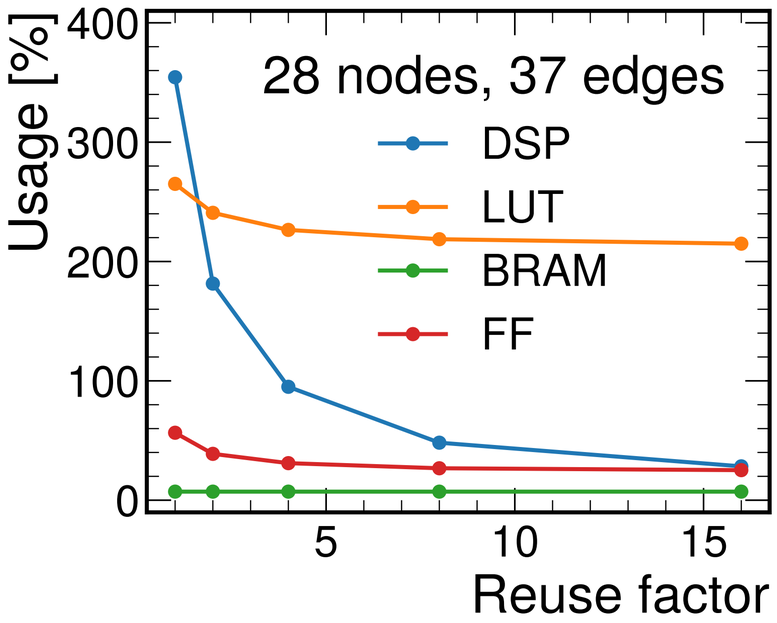


Note: LUTs are overestimated in HLS

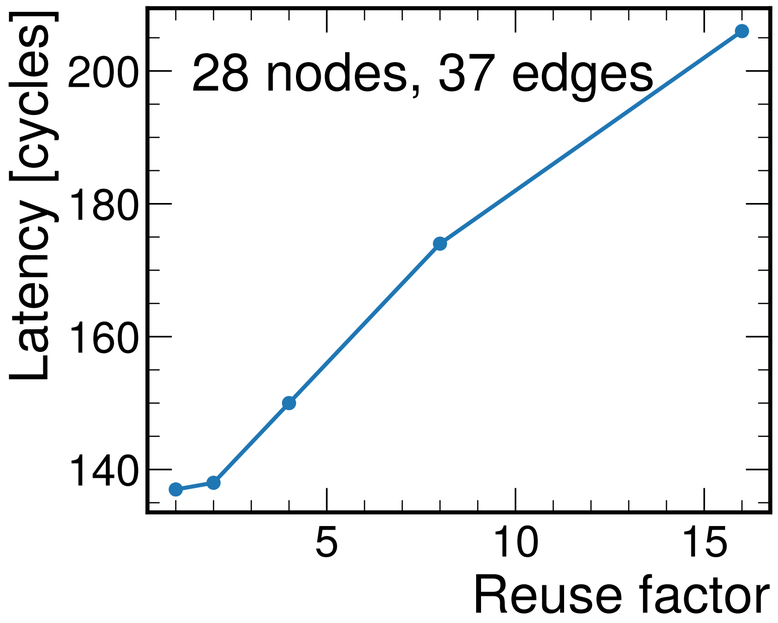
- To iterate faster, show results for 1/64 of a detector (fixed graph size of 28 nodes, 37 edges with zero-padding – corresponds to the 95th percentile of graph sizes)
- hls4ml implementation has < 1 μs latency
- Scan versus bit precision shows lower bit width results in smaller area, faster execution

HLS4ML LATENCY & RESOURCES

Xilinx KU115 FPGA



1 cycle = 5 ns



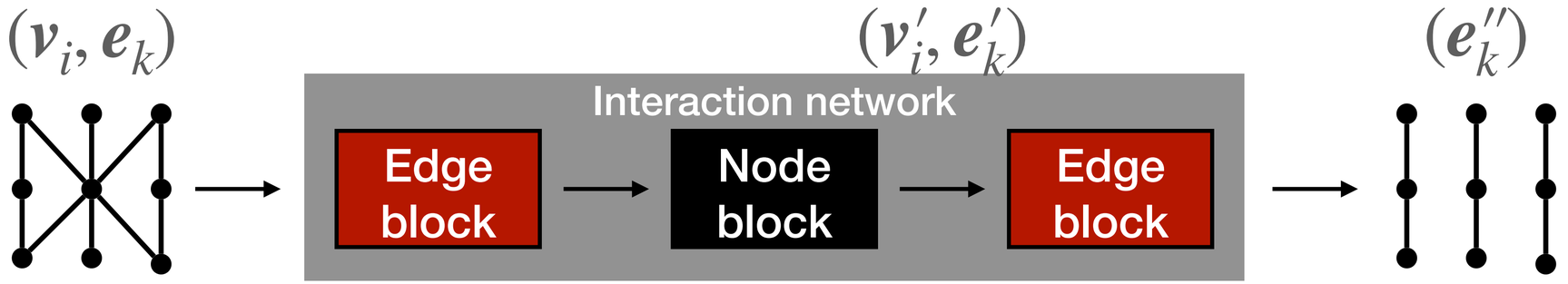
Note: LUTs are overestimated in HLS

- To iterate faster, show results for 1/64 of a detector (fixed graph size of 28 nodes, 37 edges with zero-padding – corresponds to the 95th percentile of graph sizes)
- hls4ml implementation has < 1 μs latency
- Scan versus bit precision shows lower bit width results in smaller area, faster execution
- Scan versus reuse factor shows trade-off between resource usage and latency

OPENCL

- OpenCL is a C-based platform-agnostic language framework for writing programs that execute across heterogeneous platforms like CPUs, GPUs, DSPs, FPGAS, and other processors or hardware accelerators
- Host CPU with “kernels” that are accelerated on the FPGA
- Some methods to improve efficiency:
  - 2D local memory tiling and 2D register blocking to reduce the redundancy and latency of reading from globally shared off-chip memory
  - Matrix multiplication kernels pad each matrix
  - **Double buffering** allows host to process and transfer data while kernel executes concurrently
  - All loops iterations executed in the kernels are “unrolled” to run in parallel
- Tested with an Arria 10 GX 1150 FPGA

OPENCL MODEL



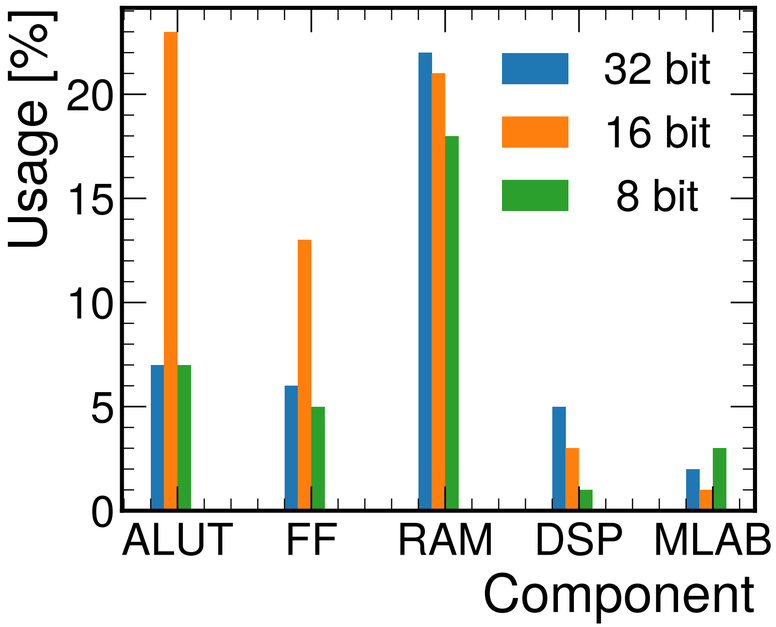
$$e_k' = \phi_2^e(v_{r_k}, v_{s_k})$$
$$\bar{e}_i' = \rho^{e\to v}(E_i)$$
$$v_i' = \phi_2^v(\bar{e}_i', v_i) \quad e_k'' = \sigma(\phi_2^e(e_k', v_{r_k}', v_{s_k}'))$$

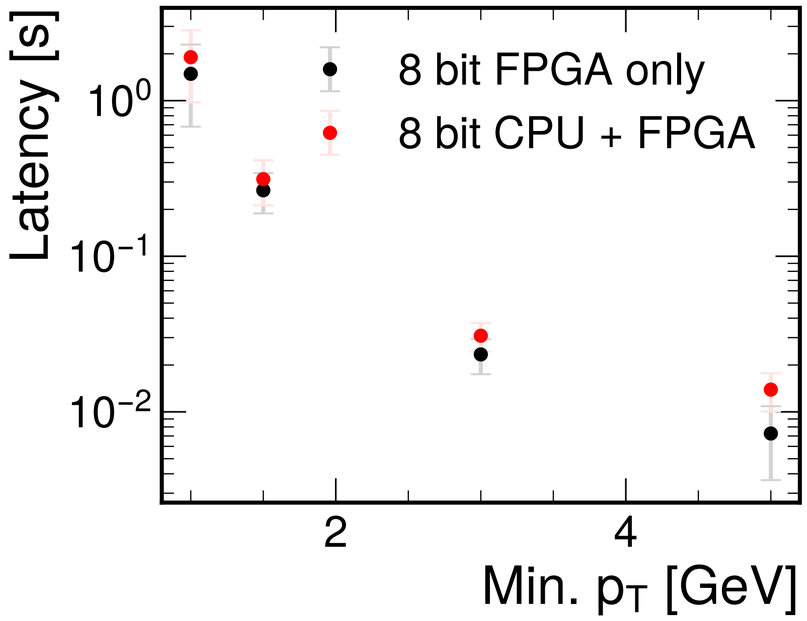
ϕ_2^e :NN(250,ReLU,250,ReLU,250,ReLU,1,ReLU/sigmoid)

ϕ_2^v :NN(200,ReLU,200,ReLU,3ReLU)

- Similar to HEP.TrkX CTD 2018 segment classifier [arXiv:1810.06111]
  - No encoder/decoder step, but same edge block and node block
  - Edge (node) block uses a larger latent space of 250-dim (200-dim)

OPENCL LATENCY & RESOURCES

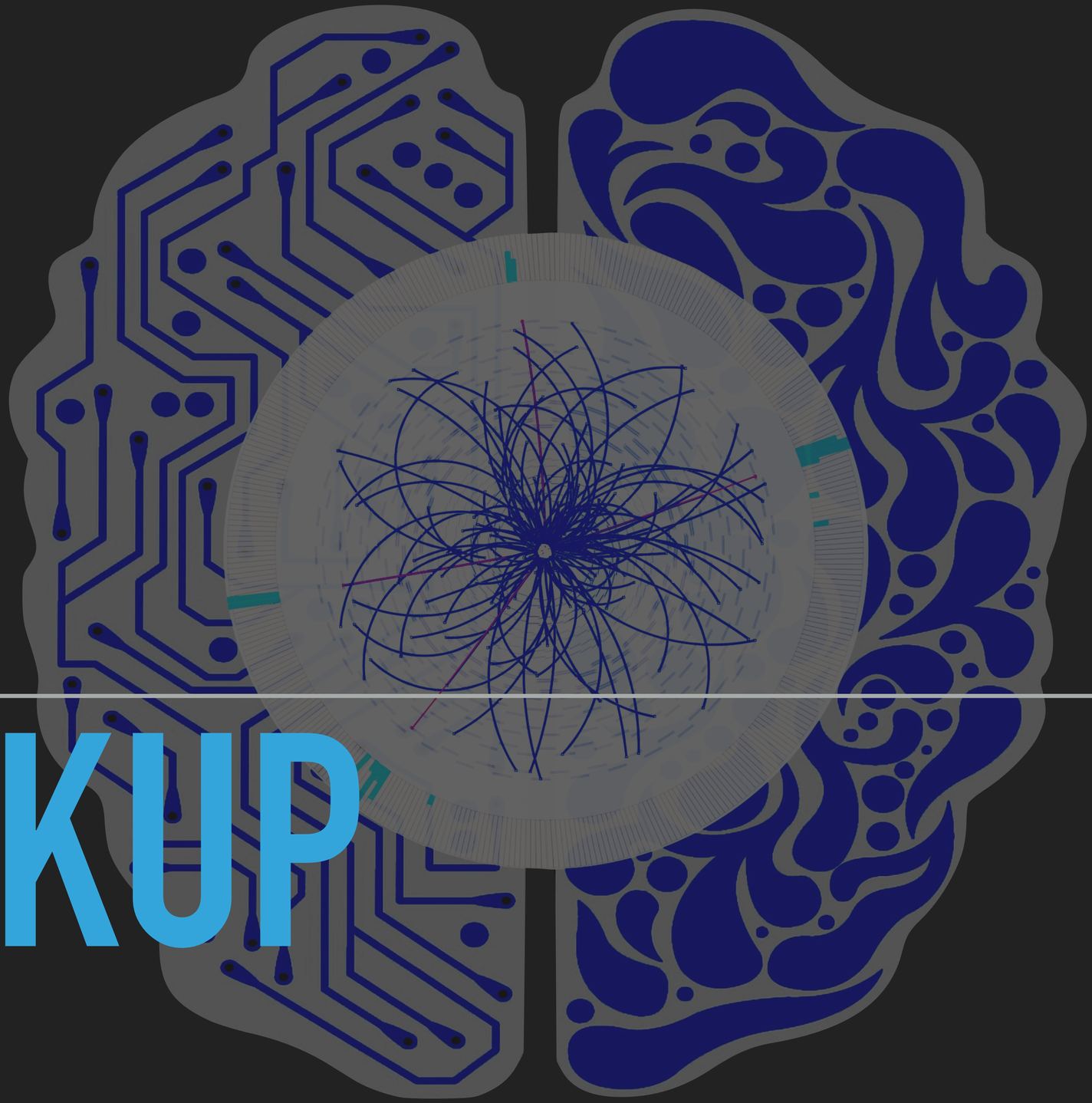




- OpenCL implementation scales up more easily to larger graph sizes (smaller minimum p_T)
- Latency is longer (includes CPU–FPGA I/O), but resource usage is quite manageable

SUMMARY AND OUTLOOK

- Two complementary implementations of GNNs on FPGAs
- Current performance is promising both for trigger-level applications (hls4ml) and coprocessing applications (OpenCL and hls4ml)
- OpenCL implementation can scale more easily to larger graphs/models, while hls4ml implementation may have a latency/throughput advantage
- Future work
  - Extend and scale up the implementations to make them more flexible to test different types of GNNs
  - Benchmark the implementations and architectures against each other



BACKUP