Track reconstruction for the ATLAS Phase-II Event Filter using GNNs on FPGAs

Sebastian Dittmeier (Physikalisches Institut - Heidelberg University)
on behalf of the ATLAS TDAQ collaboration
CHEP 2023
May 08 – 12, 2023@ Norfolk, Virginia, USA


FSP ATLAS
Erforschung von Universum und Materie

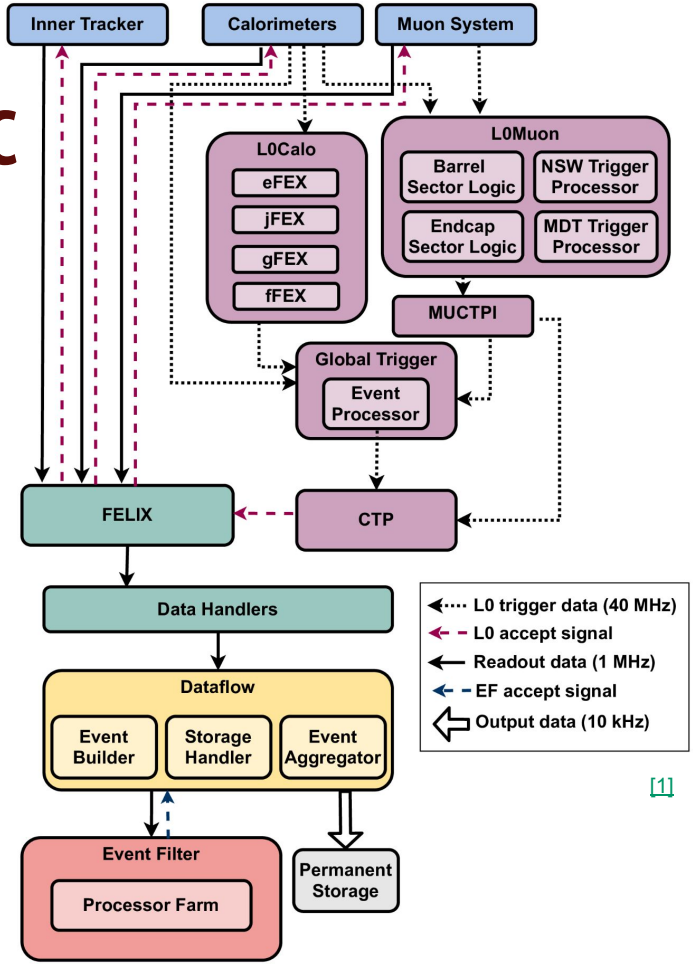
Event Filter Tracking for ATLAS @ HL-LHC

- Event Filter forms trigger decision after L0 accept
- **Track reconstruction** computationally intensive
  - **Track finding** biggest challenge
- Ongoing R&D on acceleration with **new algorithms** and **GPUs or FPGAs**

Table 2.4: *The CPU required in* HS06 × sec *to reconstruct* $t\bar{t}$ *MC events with* $\langle\mu\rangle = 140$ *and 200 in the ITk, using full-scan and regional (5% η-ϕ coverage) track selection cuts.*

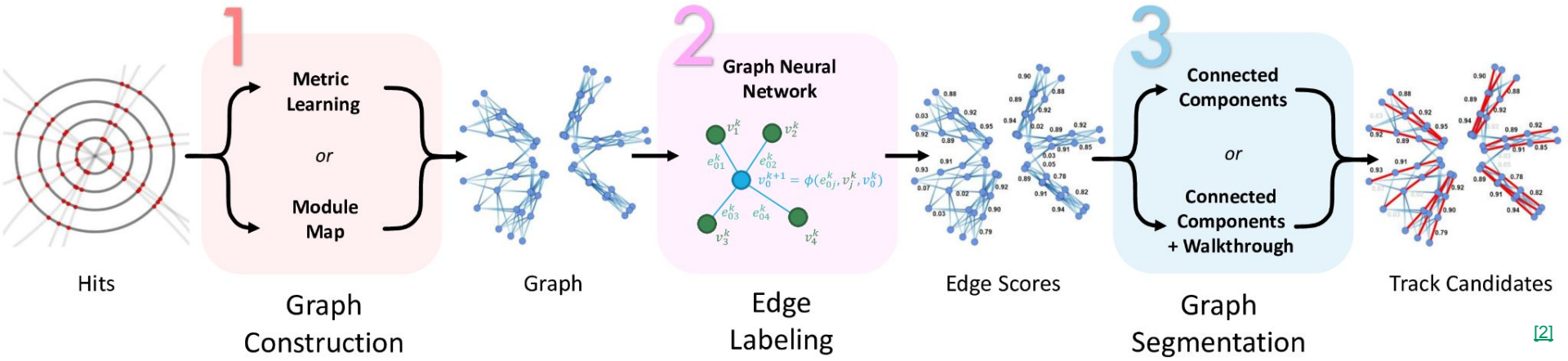
$\langle\mu\rangle$	Tracking	Byte Stream Decoding	Cluster Finding	Space Points	Si Track Finding	Total ITk
140	full-scan	2.2	6.1	1.0	13.4	22.7
	regional	0.33	0.90	0.15	1.11	2.49
200	full-scan	3.2	8.1	1.2	23.2	35.7
	regional	0.48	1.23	0.18	1.92	3.81

[1]



[1]

Finding Track Candidates with Graph Neural Networks



[2]

- Exa.TrkX / GNN4ITK pipeline with different methods per step (computational constraints)
- More talks at CHEP on Exa.TrkX and developments in GNN4ITK (1) and (2)
- FPGAs can potentially, compared to CPUs / GPUs:
  - **Speed up** inference considerably
  - **Reduce power** consumption

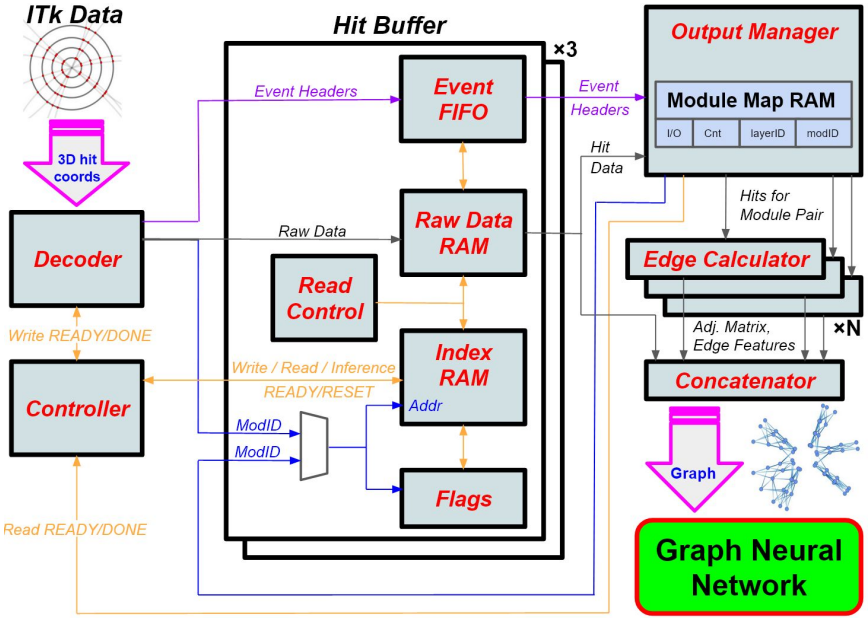
Challenge:
retaining performance with smaller models due to resource constraints on the FPGA

Considerations for GNNs on FPGAs for EF Tracking

Model architecture ↔ Translation using available tools (e.g. HLS4ML, FINN), dedicated hardware implementations

Hardware implementations under development

Model architecture ↔ Translation using available tools (e.g. HLS4ML, FINN), dedicated hardware implementations

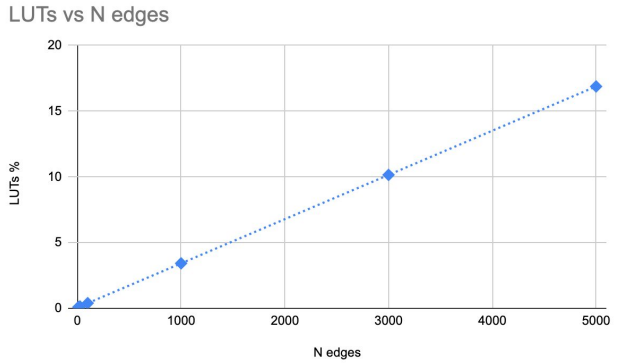


VHDL implementations under development:

- **Module Map for graph construction**
- Walkthrough for graph segmentation

Hardware implementations under development

Model architecture ↔ Translation using available tools (e.g. HLS4ML, FINN), dedicated hardware implementations



Resource estimate from synthesis (Stratix 10 GX) versus number of graph edges

VHDL implementations under development:

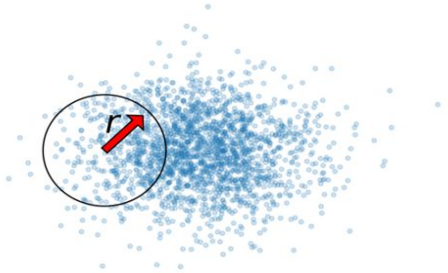
- Module Map for graph construction
- **Walkthrough for graph segmentation**

Considerations for GNNs on FPGAs for EF Tracking

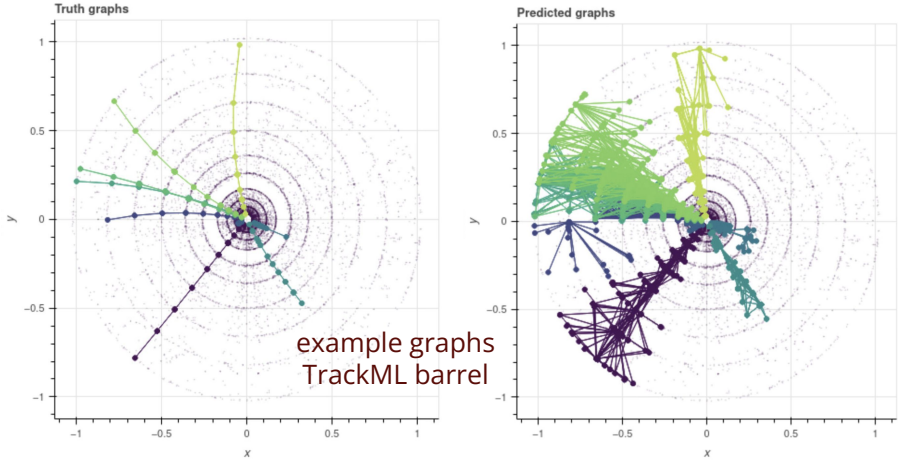
Model architecture	↔	Translation using available tools (e.g. HLS4ML, FINN), dedicated hardware implementations
Model optimization	↔	Smaller models, but also: **Quantization** + **Pruning** to meet resource constraints

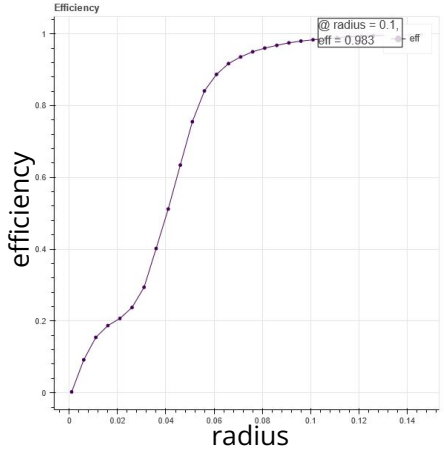
Model optimization studies

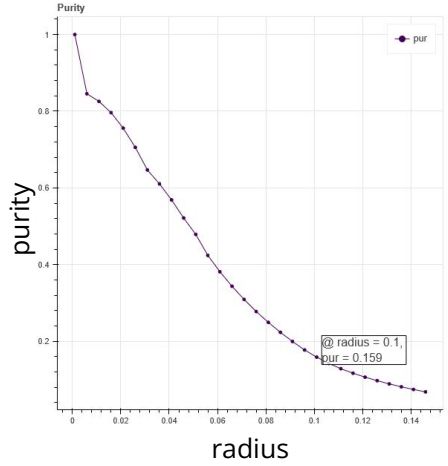
Graph construction using Metric Learning



- Goal: Limit amount of false edges while keeping true edges
- Embed hit points into latent space using a **multi-layer perceptron (MLP)**
- Connect hits that are close to each other in latent space
- Trade-off efficiency versus purity by cut on radius

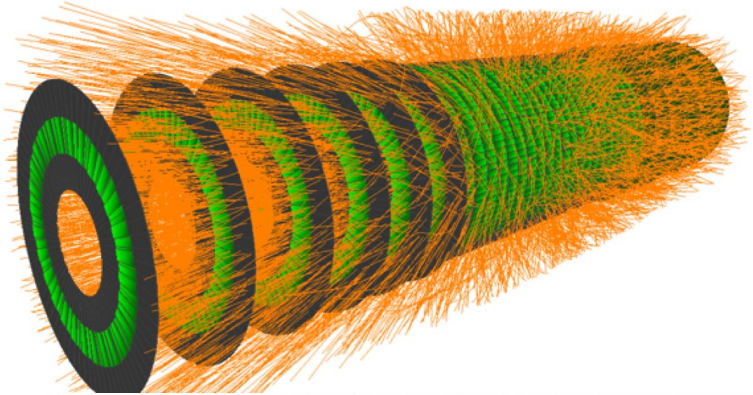


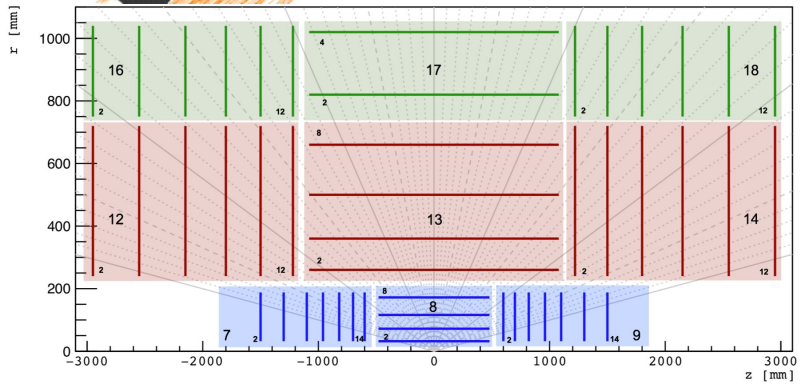




Quantization and Pruning studies with TrackML dataset

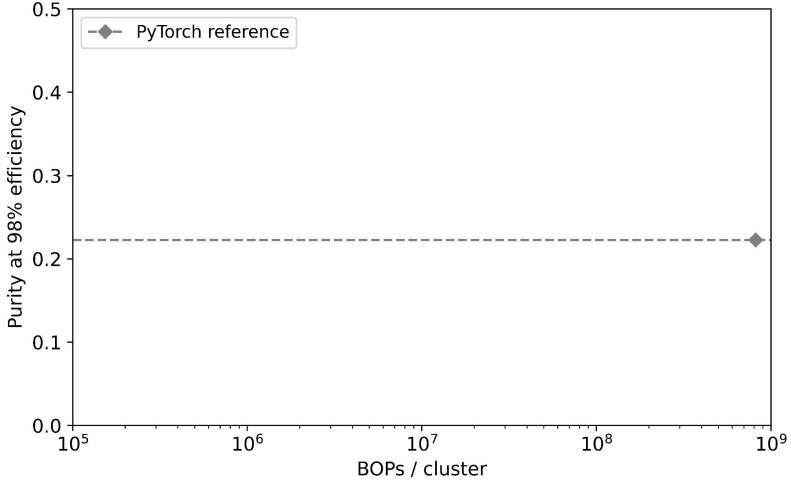
- Simplified detector geometry, adapted from early ATLAS ITk designs
- Pile-up 200 conditions like @ HL-LHC
- The following studies make use of a small pre-processed dataset of particles with $p_T > 1$ GeV
- Repository on Github: fork of Exa.TrkX pipeline





PyTorch Reference MLP

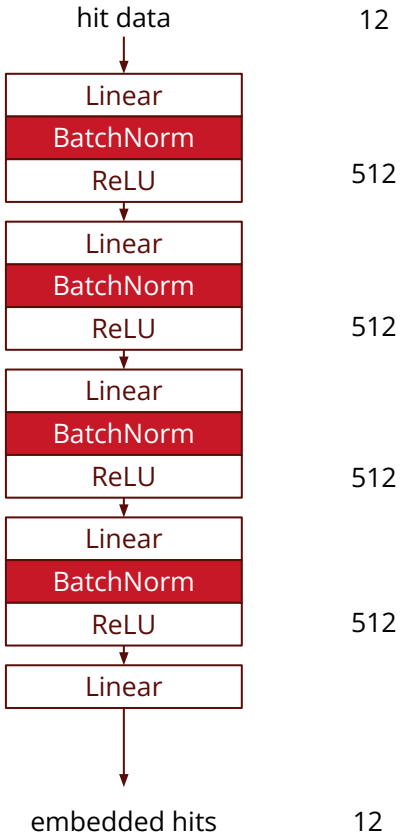
Evaluation using purity at fixed efficiency



Architecture adapted for FPGA target:
BatchNorm instead of LayerNorm, no output normalization

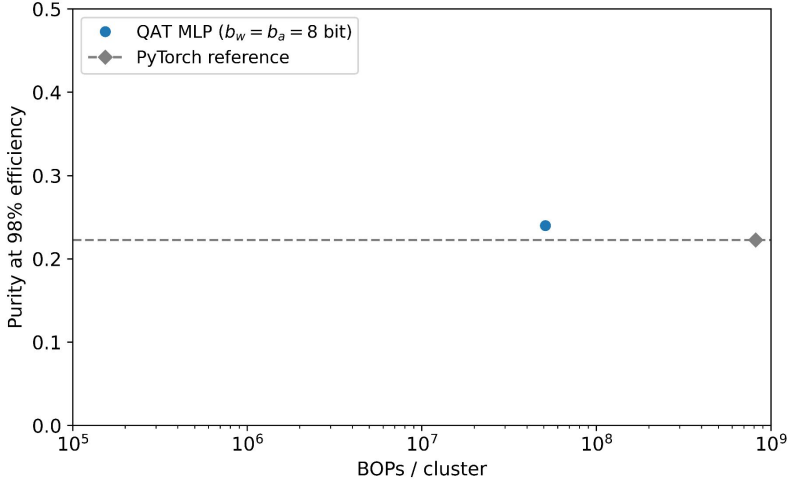
Model size evaluation in Bit Operations per cluster: BOPs $\propto (1-f_p) b_a b_w$

f_p: pruning fraction, b_a (b_w): bit width of activations (weights)



Quantization Aware Training

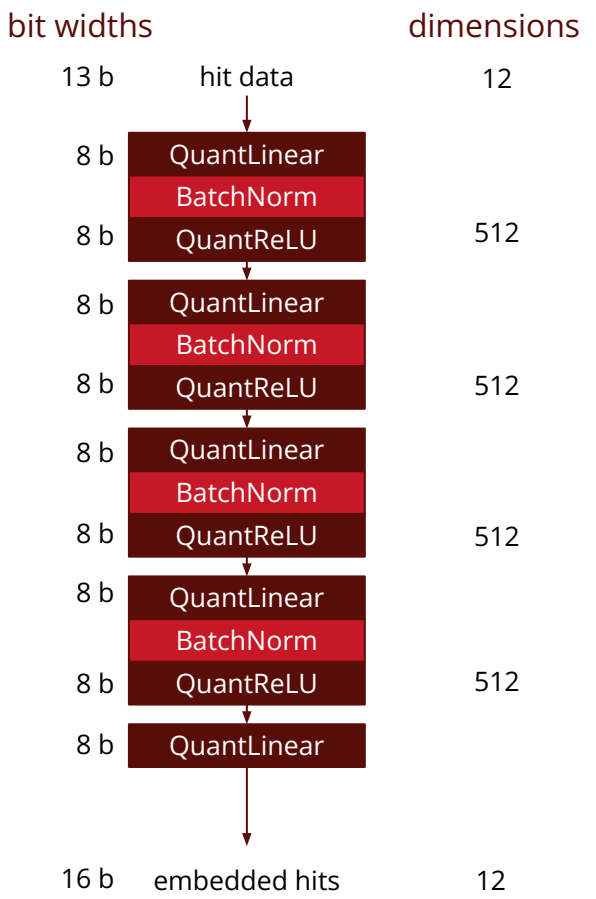
Using Brevitas (Xilinx), quantizing weights and activations to 8 bits



Hit input data using signed fixed point representation <2,10>

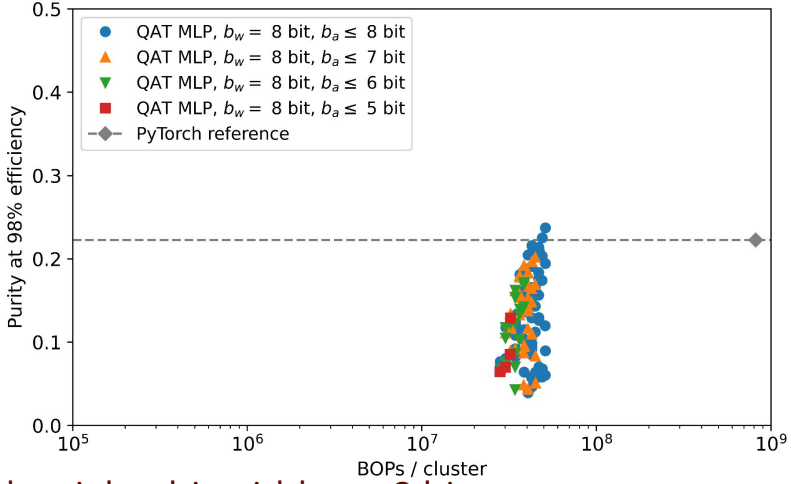
Model size evaluation in Bit Operations per cluster: BOPs $\propto (1-f_p) b_a b_w$

f_p: pruning fraction, b_a (b_w): bit width of activations (weights)



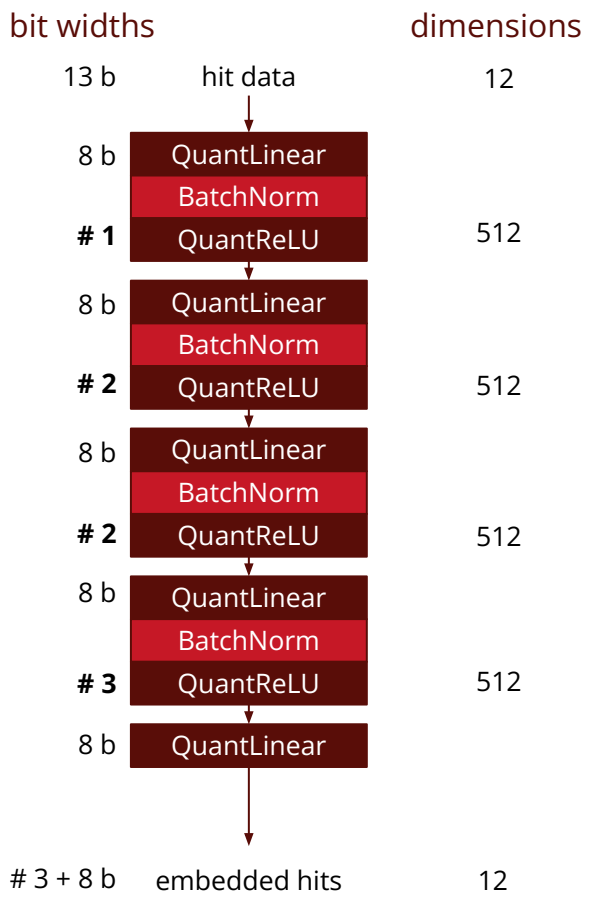
Quantization Aware Training

Sweep of activation bit widths b_a (2 to 8 bits)



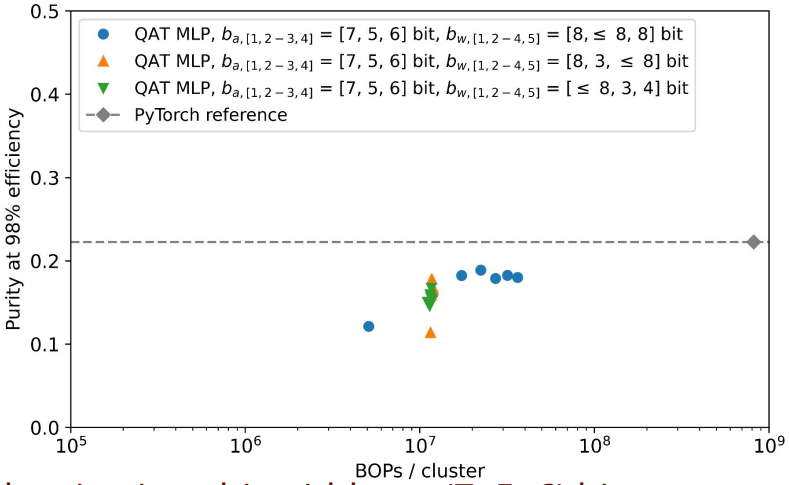
Fixed weights bit widths at 8 bits
3 independent activation parameters (#1, #2, #3)

Model size evaluation in Bit Operations per cluster: BOPs $\propto (1-f_p) b_a b_w$
f_p: pruning fraction, b_a (b_w): bit width of activations (weights)

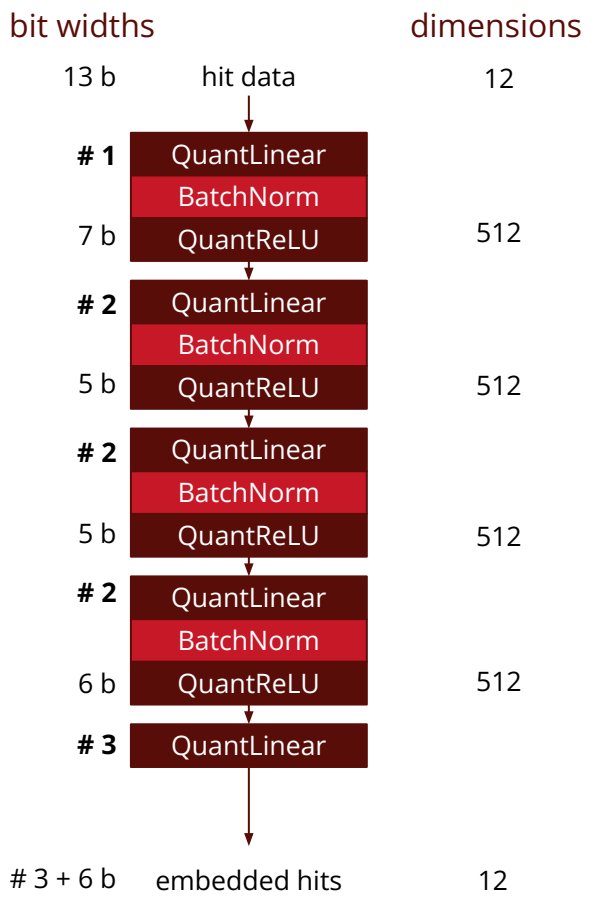


Quantization Aware Training

Scan of weight bit widths b_w (2 to 8 bits)



Fixed activations bit widths at (7, 5, 6) bits,
3 independent weight parameters (#1, #2, #3)

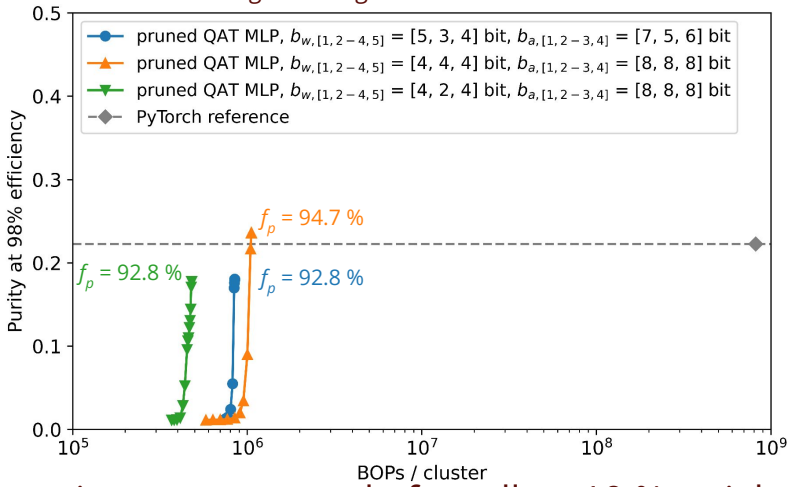


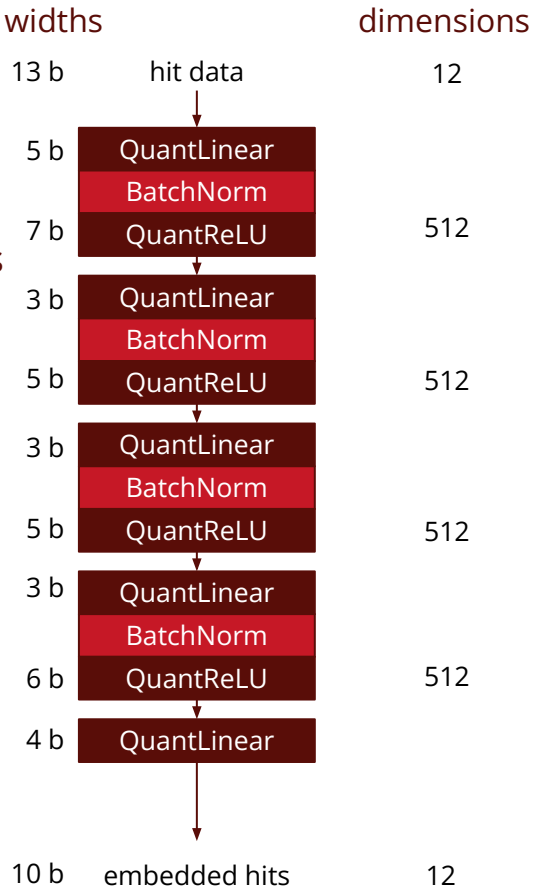
Model size evaluation in Bit Operations per cluster: BOPs $\propto (1-f_p) b_a b_w$

f_p: pruning fraction, b_a (b_w): bit width of activations (weights)

Adding iterative pruning to QAT

Using L1 Loss $L_{training} = L_{hinge} + L_1$, $L_1 = \lambda \sum |w_i|$, w_i: weights in linear layers



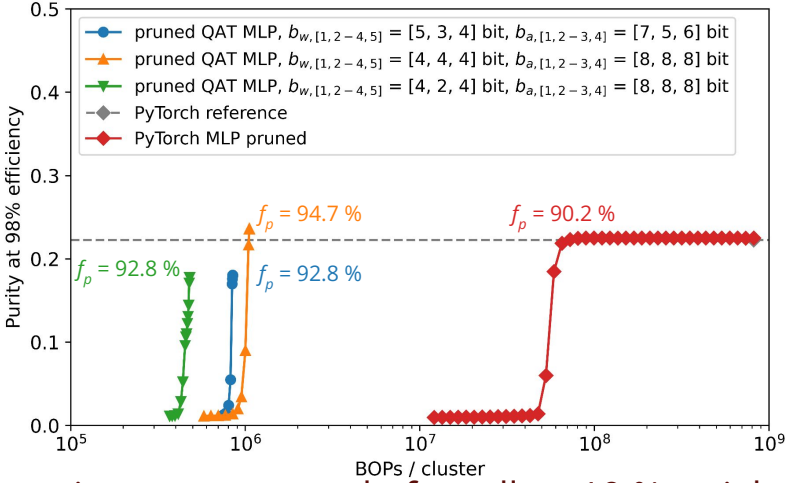


Per pruning step: removal of smallest 10 % weights (L1 unstructured)
Pruning every 180 epochs or if validation loss is stable for 10 epochs

Model size evaluation in Bit Operations per cluster: BOPs $\propto (1-f_p) b_a b_w$
f_p: pruning fraction, b_a (b_w): bit width of activations (weights)

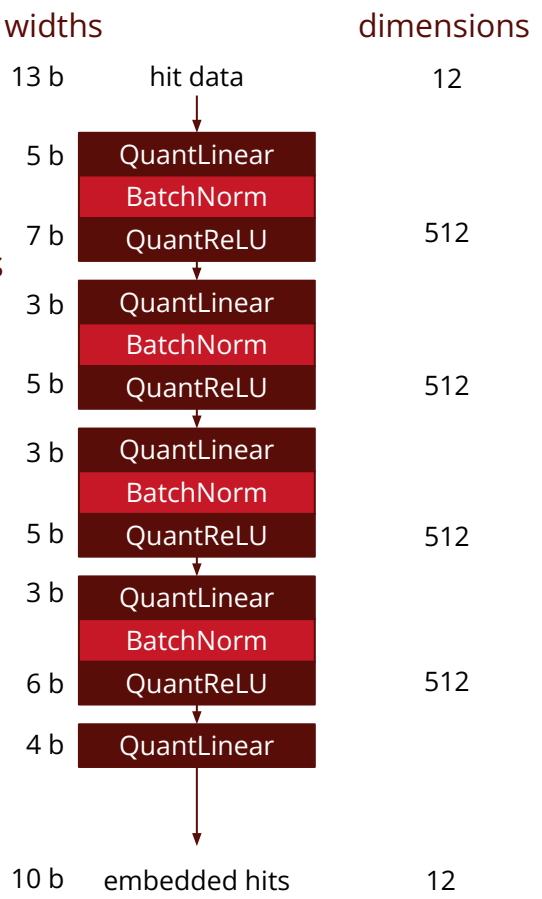
Comparing pruned QAT MLP vs MLP

Using L1 Loss $L_{training} = L_{hinge} + L_1$, $L_1 = \lambda \sum |w_i|$, w_i: weights in linear layers



Per pruning step: removal of smallest 10 % weights (L1 unstructured)
Pruning every 180 epochs or if validation loss is stable for 10 epochs

Model size evaluation in Bit Operations per cluster: BOPs $\propto (1-f_p) b_a b_w$
f_p: pruning fraction, b_a (b_w): bit width of activations (weights)



Further Considerations

Model architecture	↔	Translation using available tools (e.g. HLS4ML, FINN), dedicated hardware implementations
Model optimization	↔	Smaller models, but also: **Quantization** + **Pruning** to meet resource constraints
Event / graph size	↔	Segmentation of graphs , restrict on sub-detectors, or external memory / HBM to meet memory constraints
Throughput	↔	Online track reconstruction for the trigger EF system rates: 1 MHz (regional), 150 kHz (global)
Deployment and integration	↔	Accelerator cards and tools (Xilinx, Intel), heterogeneous software platforms (e.g. oneAPI)

Conclusions and Outlook

- Ongoing studies to implement GNNs on FPGAs for the ATLAS Event Filter @ HL-LHC
- Graph construction methods under investigation
  - Metric learning MLP quantization and pruning studies look promising
  - Pending next steps:
    - Translation for FPGA and complete algorithmic implementation
    - Application to ATLAS ITk simulation
  - Planned comparison with VHDL Module Map implementation
- More ongoing studies:
  - Performance of smaller interaction networks for ATLAS ITk
  - Graph segmentation into detector regions
  - MLP and Interaction Network to FPGA translation

Conclusions and Outlook

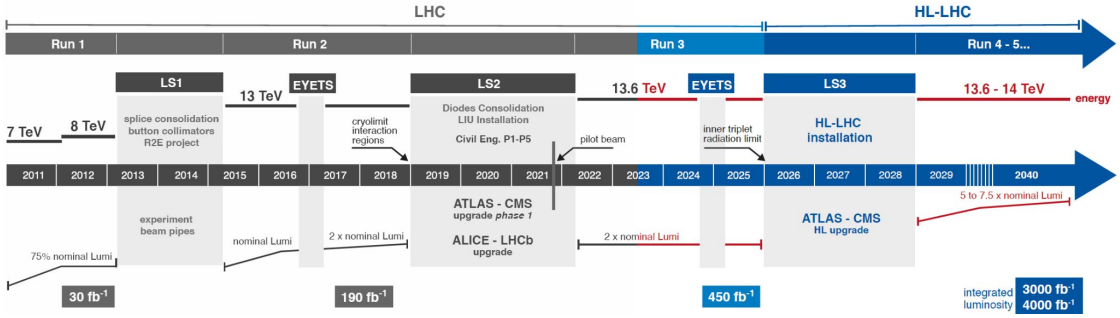
- Ongoing studies to implement GNNs on FPGAs for the ATLAS Event Filter @ HL-LHC
- Graph construction methods under investigation
  - Metric learning MLP quantization and pruning studies look promising
  - Pending next steps:
    - Translation for FPGA and complete algorithmic implementation
    - Application to ATLAS ITk simulation
  - Planned comparison with VHDL Module Map implementation
- More ongoing studies:
  - Performance of smaller interaction networks for ATLAS ITk
  - Graph segmentation into detector regions
  - MLP and Interaction Network to FPGA translation

Thank you for your attention!

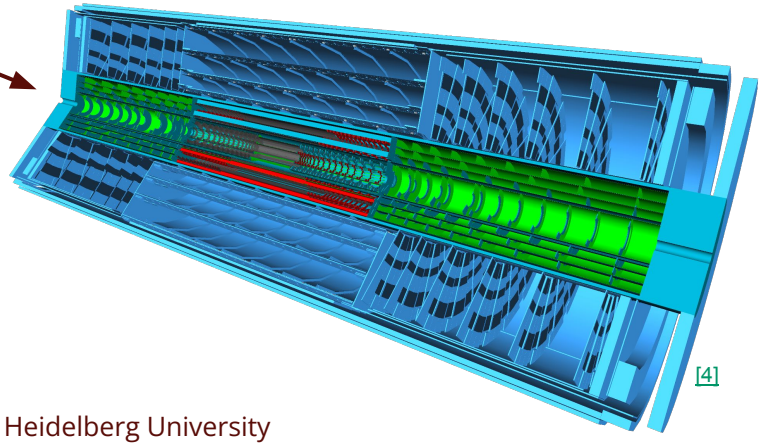
Backup

ATLAS upgrade for the High-Luminosity LHC

- Peak instantaneous luminosity of $7.5 \times 10^{34}\text{cm}^{-2}\text{s}^{-1}$ leads to average pile-up $\langle\mu\rangle \approx 200$
- **Upgrades include:**
  - **New** all-silicon **tracking detector**: Inner Tracker (ITk)
  - Upgrade of **Trigger** and **Data Acquisition** System (TDAQ) [3]
  - And more (HGTD, muon system, calorimeter)



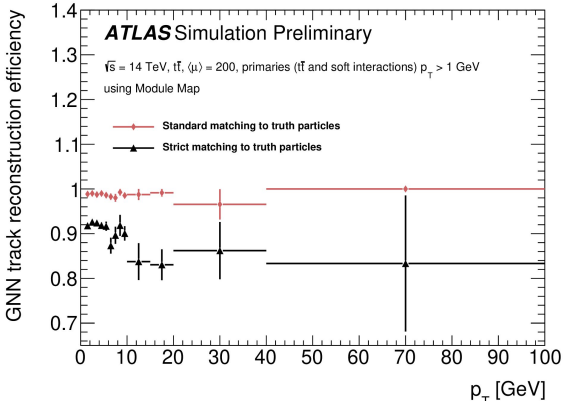
[4]



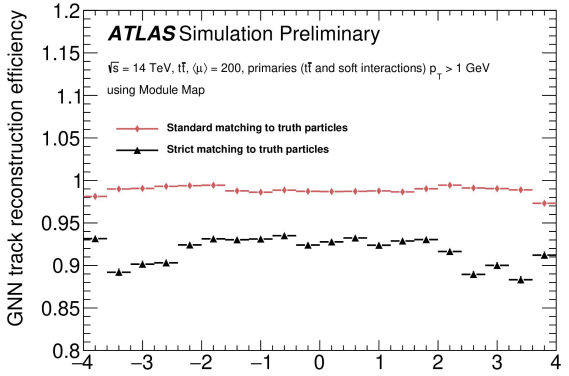
[4]

The approach: Graph Neural Networks on FPGAs

- Finding track candidates with **Graph Neural Networks**
- **Demonstrated good performance** with ATLAS ITk simulation [5] (GNN4ITK)
- Reference to GNN4ITK talks
- FPGAs can potentially, compared to CPUs / GPUs:
  - **Speed up** inference considerably
  - **Reduce power** consumption



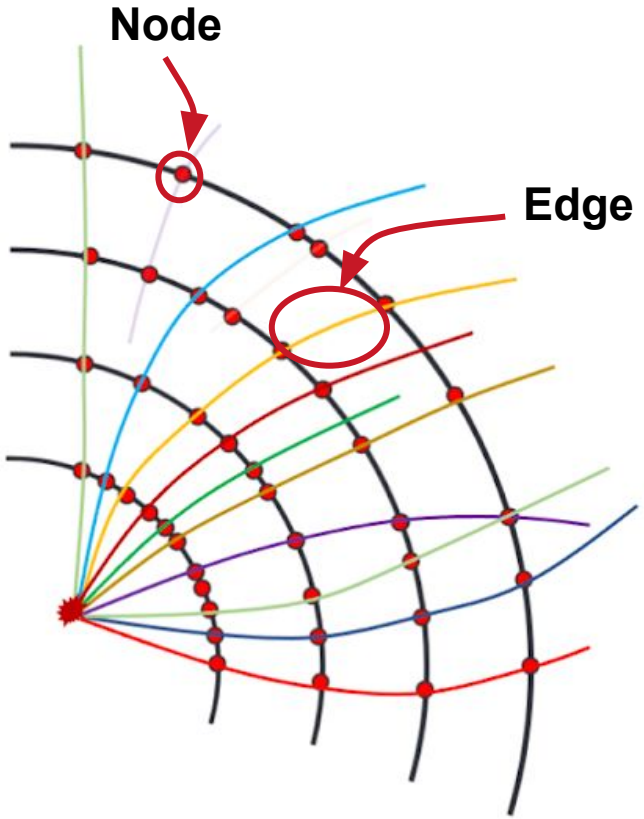
[2]



[2]

Collision events as graphs

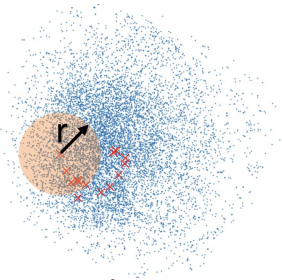
- Graphs consist of a set of nodes and edges
- Take an event in a detector:
  - Represent each **hit** as a **node**
  - Connect nodes by edges
  - **Edges** suggest **two hits** belong to the **same track**
- 3 levels of information:
  - Node-level (position, energy deposited, etc.)
  - Edge-level (belongs to track or not, geometric info, etc.)
  - Graph-level (event, detector region, etc.)
- Similarly, predictions can be made with a GNN on node-level, edge level, and graph-level
  - In this case, make edge-level predictions to construct tracks (i.e. are edges true or false?)

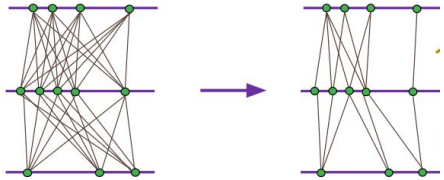


Graph construction methods

Goal: Limit amount of false edges while keeping true edges

- **Metric learning**
  - Embed hit points into latent space using a **multi-layer perceptron (MLP)**
  - Connect hits that are close to each other in latent space
- **Module map**
  - For each module pair, find max and min of geometric values
  - Apply geometric cuts for each pair, and construct map of possible connections/edges
  - Store in permutation invariant matrix



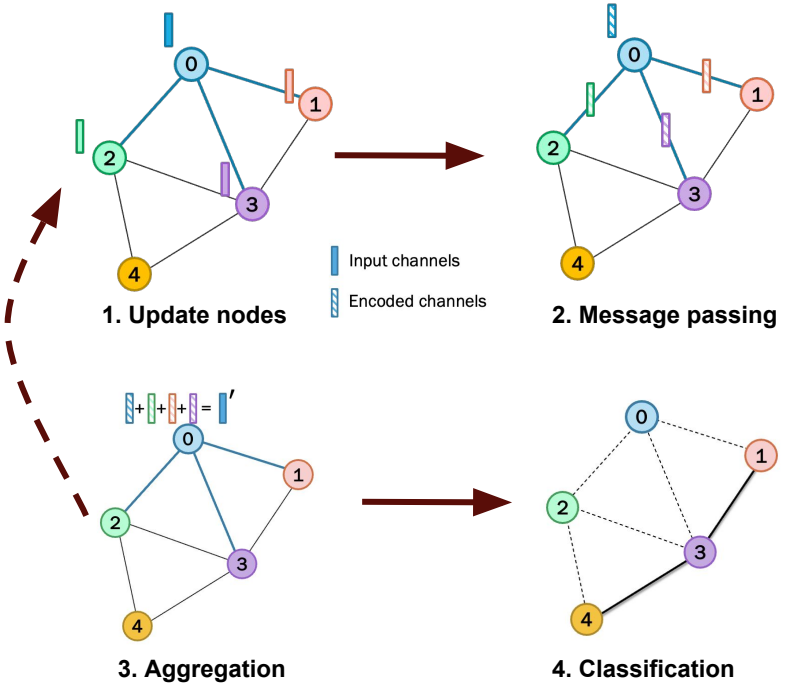


Edge classification (GNN)

Goal: classify edges as “True” or “False”
- could they belong to a track?

GNNs rely on **Message Passing**:

1. Node vectors (properties of node) are updated through an update function (**MLP**)
2. Information is passed along edges to neighbouring nodes
3. All messages are aggregated (summed) at each node
4. After one or more message passing steps, use classifier to make edge-level predictions

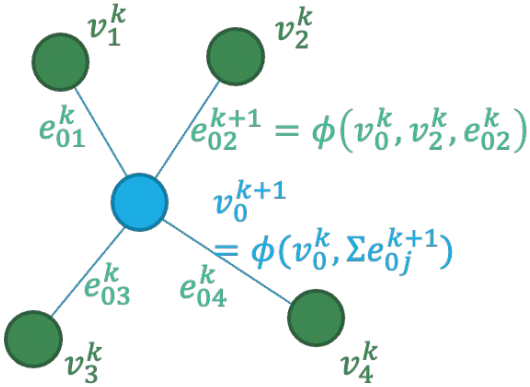


Edge classification (GNN)

Use type of GNN called **“Interaction Network”**

- Interaction Network adds an extra step to the message passing algorithm
- “Edge network” updates edge features, allowing two nodes to form unique relationships
- Improves quality of edge-level predictions

v_i^k node features
e_{ij}^k edge features
at iteration k



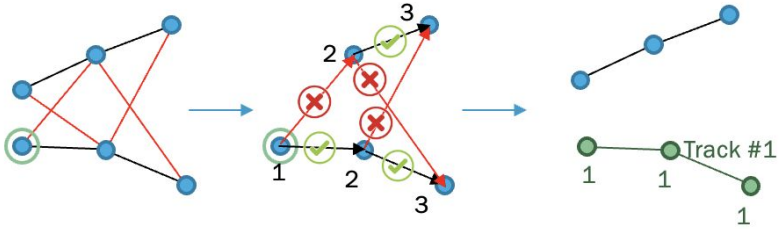
Finally, edge scores are assigned to all edges in an event

Use threshold value (e.g. ~0.5) to discard false edges

Track reconstruction methods

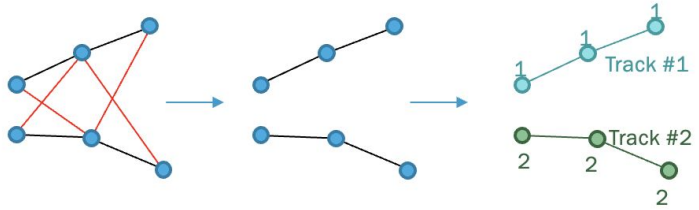
Method 1: Walkthrough

- Identify starting node
- Traverse edges with high scores
- Longest path found
  → track candidate



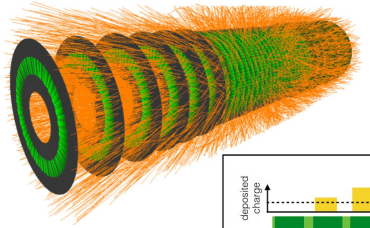
Method 2: Connected components

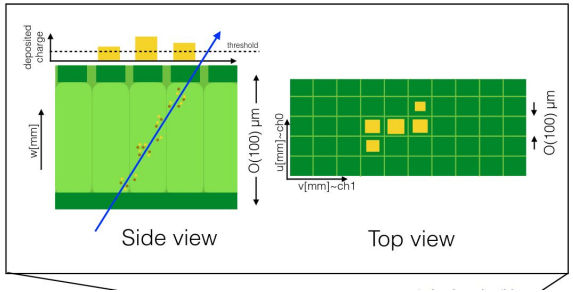
- For edges above threshold score, identify connected paths
- Assign component index to nodes



Method 3: Connected components followed by walkthrough

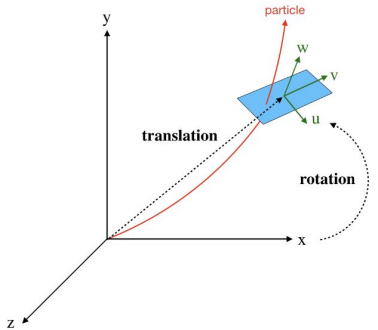
A closer look at the data set





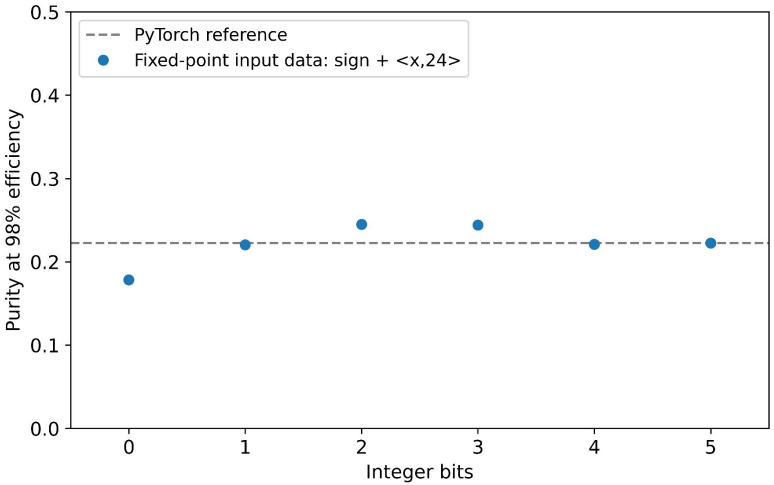
- **TrackML** Detector Example
- Input to MLP for metric learning (graph construction)
- Pre-processed dataset of particles $p_T > 1$ GeV
  - Input data: cluster data with 12 input features
    - `cell_count`: the number of cells in the cluster
    - `cell_val`: the total amount of energy deposited in the cluster
    - `leta, lphi, lx, ly, lz` : the angle of the vector representing the cluster shape, in local coordinates
    - `geta, gphi`: the angle of the shape vector in global coordinates
    - `r, phi, z`: cluster coordinates
- Detector information:
  - Pixels: 50 µm × 50 µm
  - Short strips: 80 µm × 1200 µm
  - Long strips: 120 µm × 10800 µm
- $t\bar{t}$ production overlaid with 200 soft QCD interactions
- More details about the TrackML data set can be found here [6]

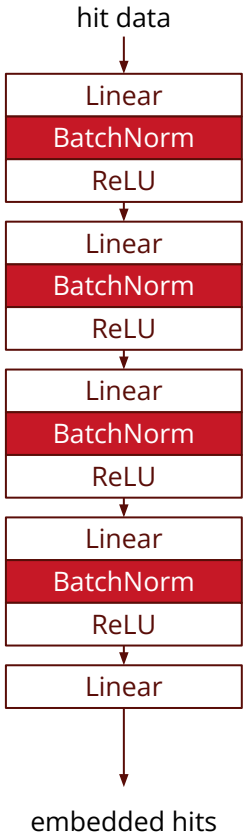
Described in https://arxiv.org/abs/2012.04533



Integers or kind of quantized numbers

Input data quantization (integer)

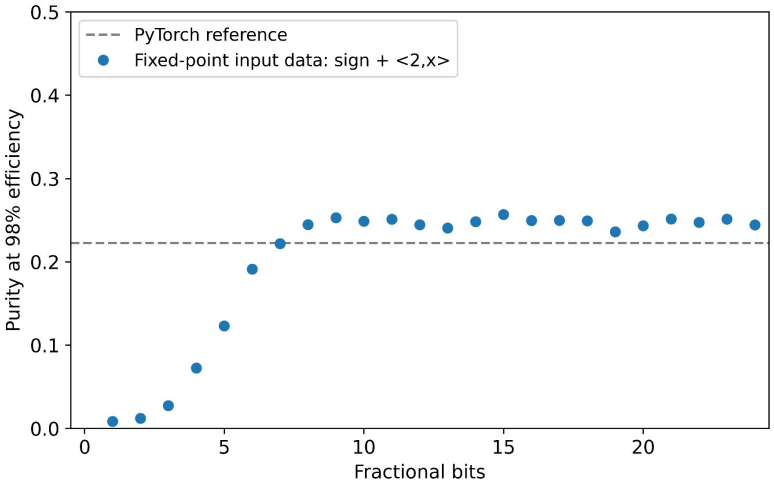


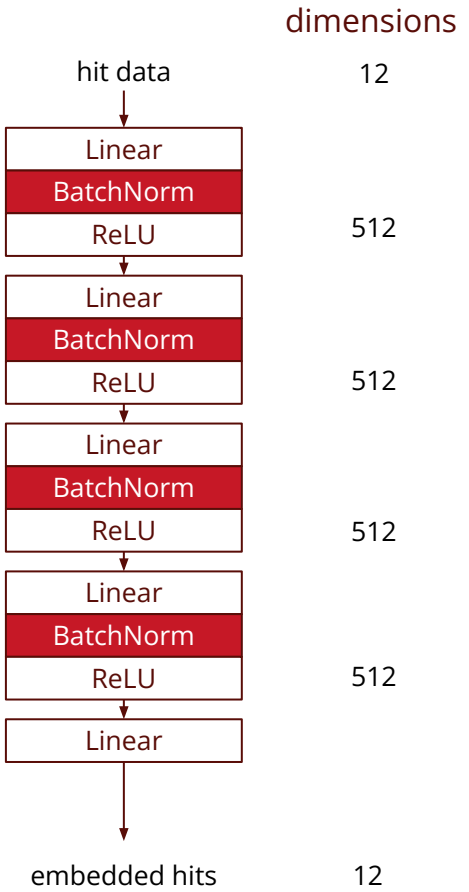


Model size evaluation in Bit Operations per cluster: BOPs $\propto (1-f_p) b_a b_w$

f_p: pruning fraction, b_a (b_w): bit width of activations (weights)

Input data quantization (fractional)



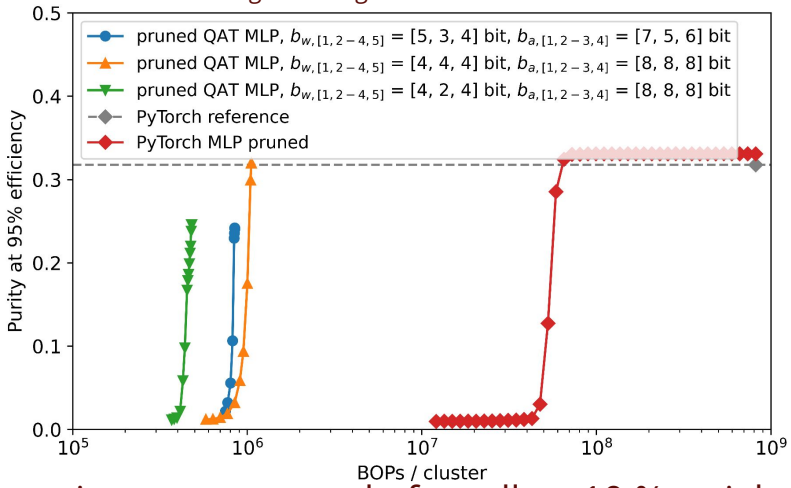


Model size evaluation in Bit Operations per cluster: BOPs $\propto (1-f_p) b_a b_w$

f_p: pruning fraction, b_a (b_w): bit width of activations (weights)

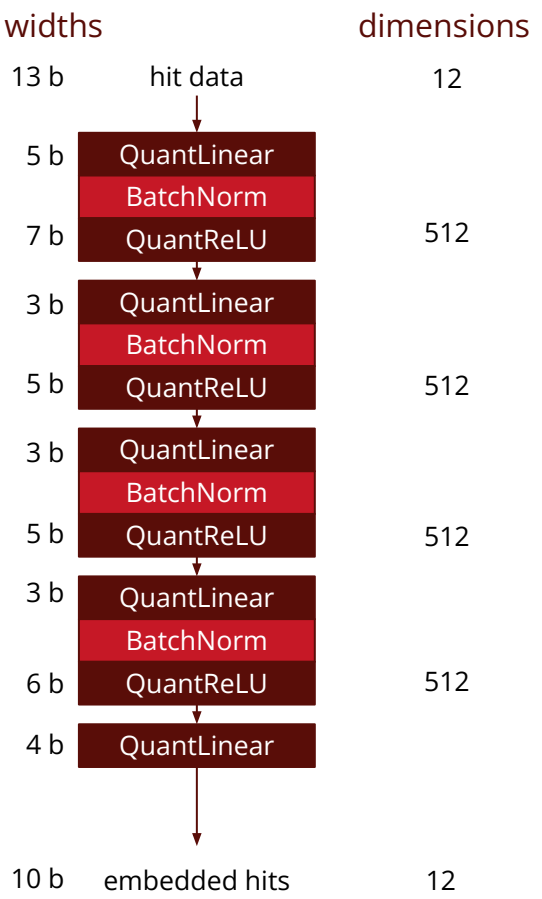
Pruned QAT MLP vs MLP @ 95%

Using L1 Loss $L_{training} = L_{hinge} + L_1$, $L_1 = \lambda \sum |w_i|$, w_i: weights in linear layers



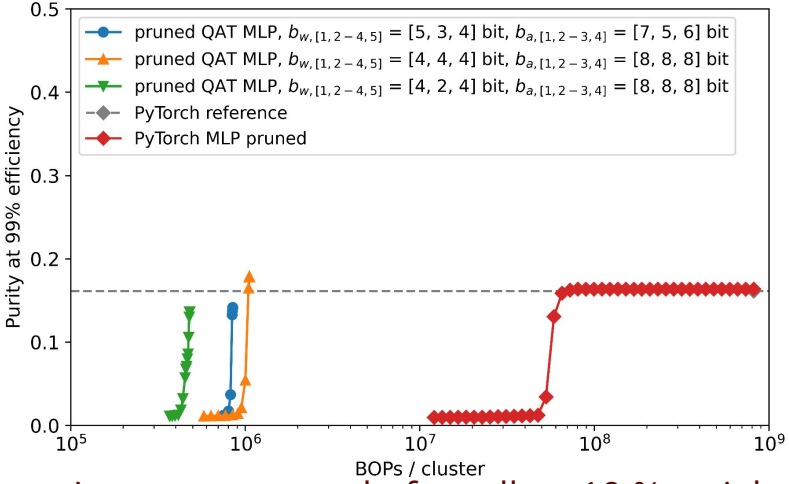
Per pruning step: removal of smallest 10 % weights
Pruning every 180 epochs or if validation loss is stable for 10 epochs

Model size evaluation in Bit Operations per cluster: BOPs $\propto (1-f_p) b_a b_w$
f_p: pruning fraction, b_a (b_w): bit width of activations (weights)



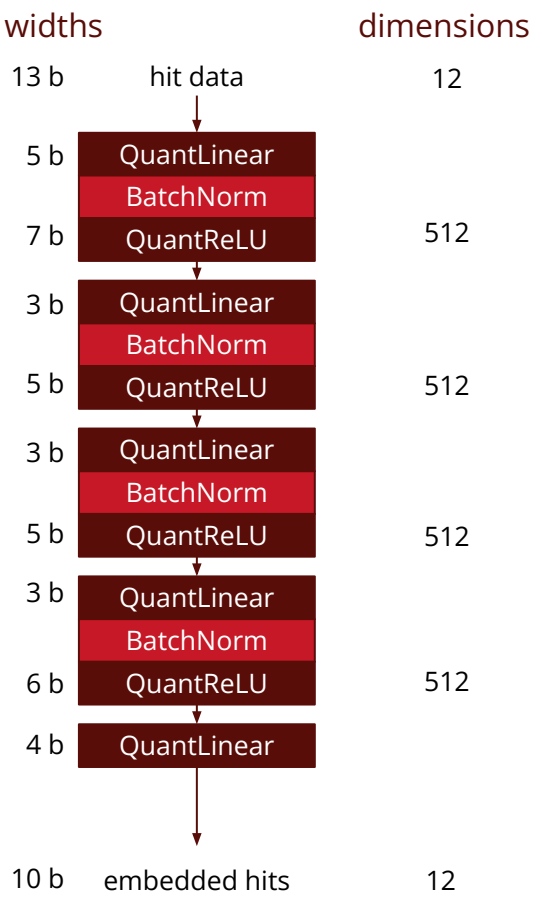
Pruned QAT MLP vs MLP @ 99%

Using L1 Loss $L_{training} = L_{hinge} + L_1$, $L_1 = \lambda \sum |w_i|$, w_i: weights in linear layers



Per pruning step: removal of smallest 10 % weights
Pruning every 180 epochs or if validation loss is stable for 10 epochs

Model size evaluation in Bit Operations per cluster: BOPs $\propto (1-f_p) b_a b_w$
f_p: pruning fraction, b_a (b_w): bit width of activations (weights)



Exploration of HLS4ML

- Translation from offline GNN tools to FPGA implementation required
- Graph construction - Metric Learning MLP as test bed
  - Translation of metric learning MLP successful
    PyTorch → ONNX → HLS4ML
  - Comparable performance,
    evaluated for 10 events
  - Efficiency and purity on edge level,
    connecting 2 hits belonging to the same track

Model Implementation	Efficiency	Purity
PyTorch	0.986	0.221
HLS converted from PyTorch	0.974	0.222

Exploration of tools for MLP and GNN translation

- OpenHLS: study for metric learning MLP
  - https://arxiv.org/abs/2302.06751
  - https://github.com/makslevental/openhls
- FINN: Fast, Scalable Quantized Neural Network Inference on FPGAs
  - https://github.com/Xilinx/finn
- FlowGNN: A Dataflow Architecture for Universal Graph Neural Network Inference via Multi-Queue Streaming
  - https://arxiv.org/abs/2204.13103
  - https://github.com/sharc-lab/FlowGNN