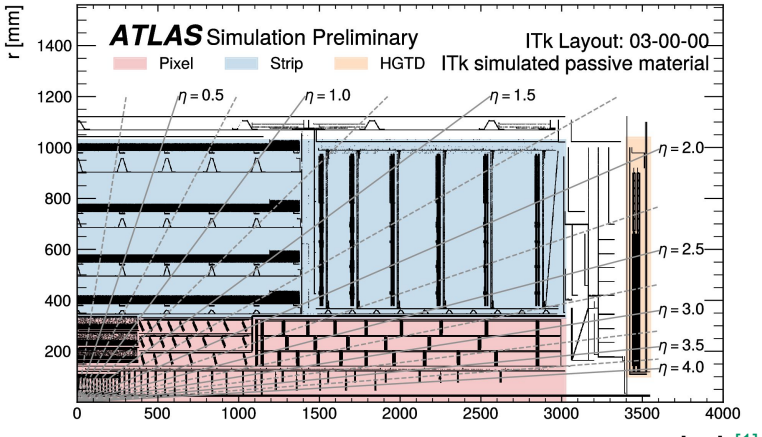
Online track reconstruction with graph neural networks on FPGAs for the ATLAS experiment

Sebastian Dittmeier on behalf of the ATLAS collaboration
Physikalisches Institut – Universität Heidelberg
CHEP 2024
Krakow, 19. – 25. October 2024

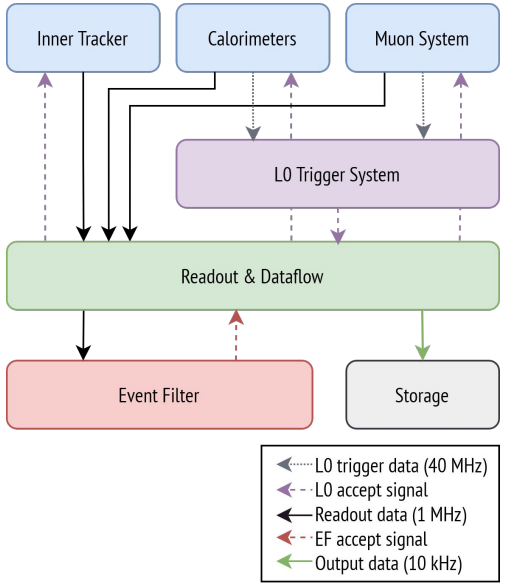
SPONSORED BY THE
Federal Ministry of Education and Research
FSP ATLAS
Erforschung von Universum und Materie

ATLAS Upgrade: ITk & TDAQ

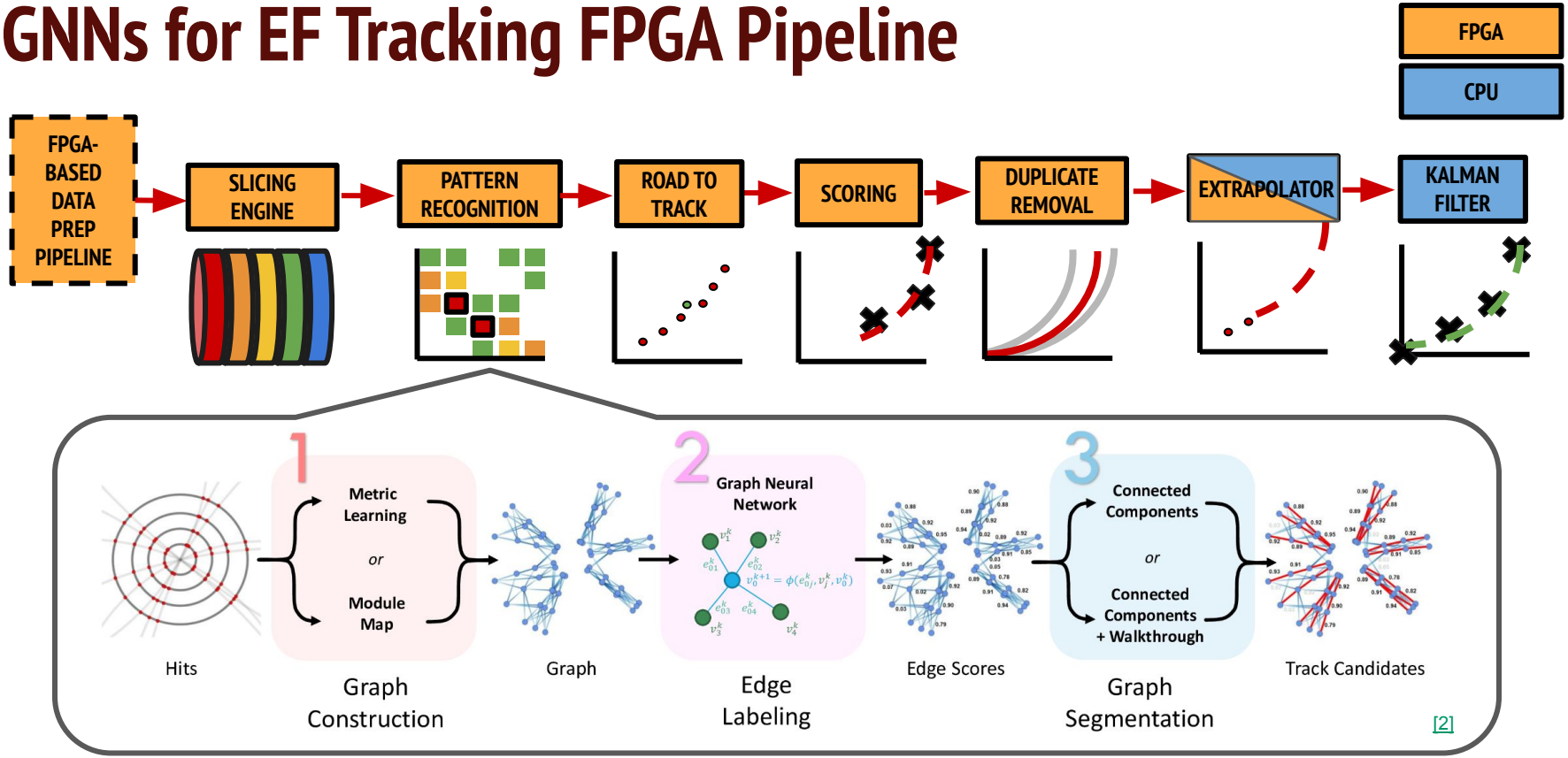
- Increase in pileup up to ⟨μ⟩ ≈ **200** @ HL-LHC
- **New** tracking detector: Inner Tracker (ITk)
  - Extended forward coverage
  - More readout channels
- **Upgrade**: Trigger and Data Acquisition
  - L0 trigger rate increase to **1 MHz (× 10)**
  - Event Filter accept rate increase **to 10 kHz (× 3)**
    - **ITk track reconstruction** computationally most expensive → power hungry!
    - Potentially **heterogeneous computing** farm with GPUs and/or FPGAs
    - **Data center class FPGAs**: testbed with AMD Alveo U250 & U55C



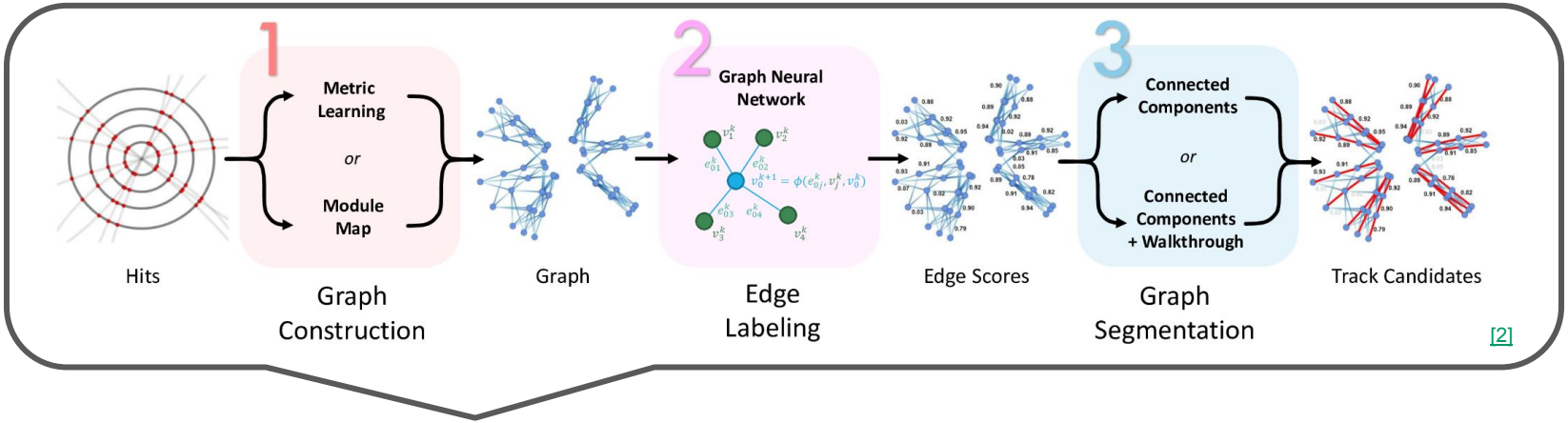
[1]



GNNs for EF Tracking FPGA Pipeline



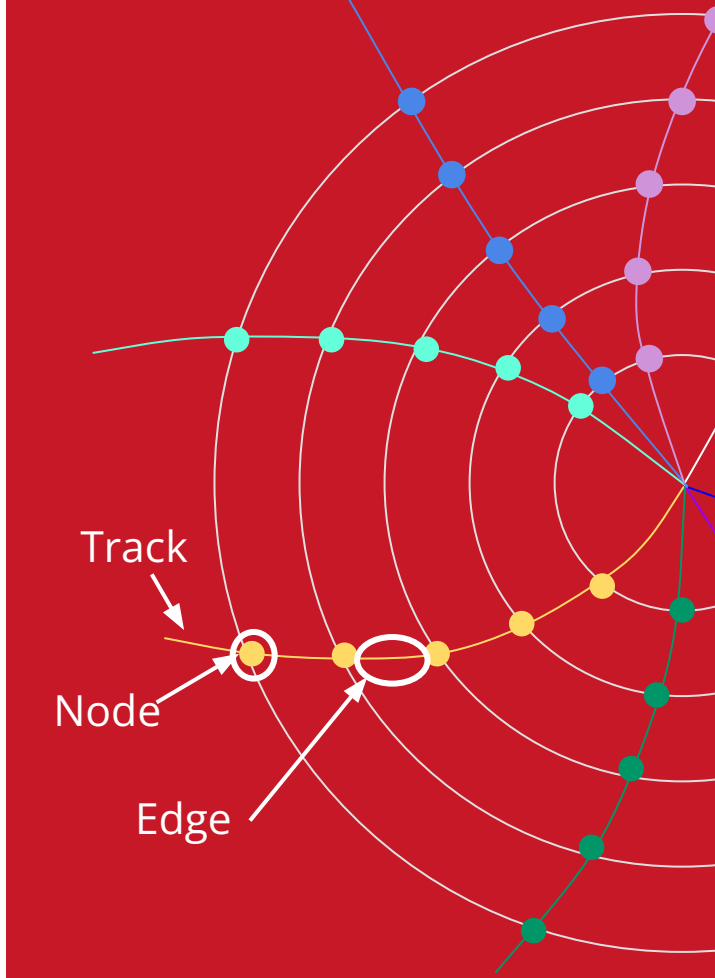
Content of this talk



- Metric Learning: FPGA deployment with a compressed model
- Graph Neural Network: model compression study
- Connected Components: FPGA deployment as a kernel
- Detector regionalization as an overall strategy to reduce graph sizes

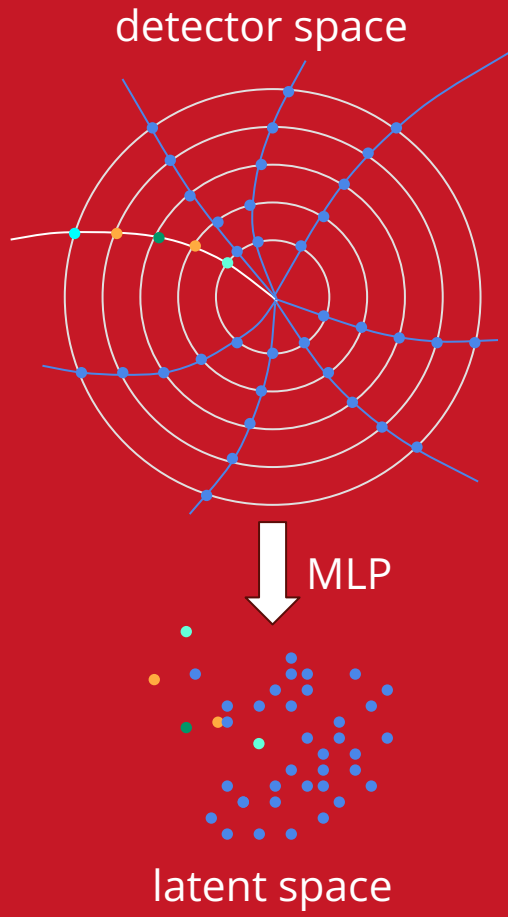
Graph Construction

- Graph structure
  - Represent each **hit** as a **node**
  - **Edges** suggest **two consecutive hits** of **same track**
  - Typical ITk event @ pileup ⟨μ⟩ = 200: ~ 300 000 hits
- Goal of graph construction:
  high efficiency and minimal graph size
- Module Map
  - Map of possible connections between detector modules derived from simulation
- Metric Learning
  - ML approach, trained on simulation
  - Multi-Layer Perceptron (MLP) embedding followed by radius clustering



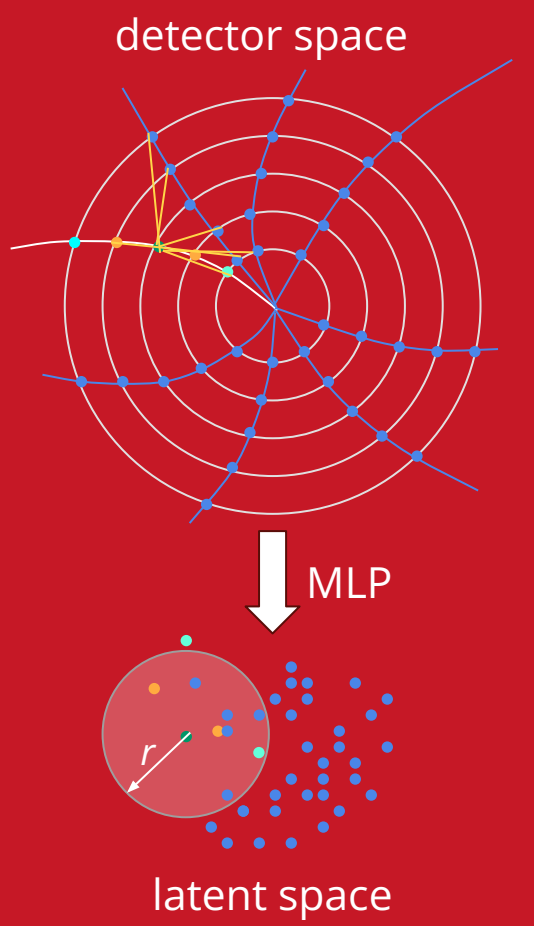
Graph Construction

- Graph structure
  - Represent each **hit** as a **node**
  - **Edges** suggest **two consecutive hits** of **same track**
  - Typical ITk event @ pileup $\langle\mu\rangle$ = 200: ~ 300 000 hits
- Goal of graph construction:
  high efficiency and minimal graph size
- Module Map
  - Map of possible connections between detector modules derived from simulation
- **Metric Learning**
  - ML approach, trained on simulation
  - **Multi-Layer Perceptron (MLP) embedding** followed by radius clustering



Graph Construction

- Graph structure
  - Represent each **hit** as a **node**
  - **Edges** suggest **two consecutive hits** of **same track**
  - Typical ITk event @ pileup $\langle\mu\rangle$ = 200: ~ 300 000 hits
- Goal of graph construction:
  high efficiency and minimal graph size
- Module Map
  - Map of possible connections between detector modules derived from simulation
- **Metric Learning**
  - ML approach, trained on simulation
  - **Multi-Layer Perceptron (MLP) embedding** followed by **radius clustering**



Metric Learning MLP on FPGA Workflow

MLP Definition
+ Training

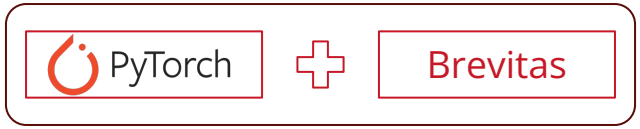
MLP architecture

Input features	3
Hidden layers	4
Hidden dimension	512
Normalization	Batch
Activation	ReLU
Output layer	Linear
Output features	12

Training data parameters

ITk simulation sample	$t\bar{t}$ @ $\langle\mu\rangle$ = 200
Target particles	primaries, no $e^{\pm}$
Target min. p_T	> 1 GeV
Target num. spacepoints	≥ 3

Metric Learning MLP on FPGA Workflow



MLP Definition
+ Training
Iterative Pruning

Quantization
Aware Training

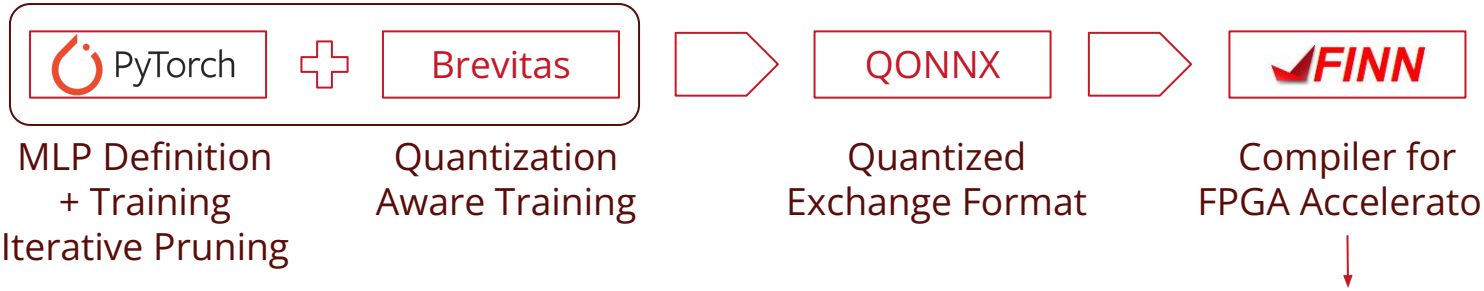
MLP architecture

Input features	3
Hidden layers	4
Hidden dimension	512
Normalization	Batch
Activation	ReLU
Output layer	Linear
Output features	12

800k Parameters
Floating-Point 32 bit

< 30k Parameters
Fixed-Point [4, 6] bit

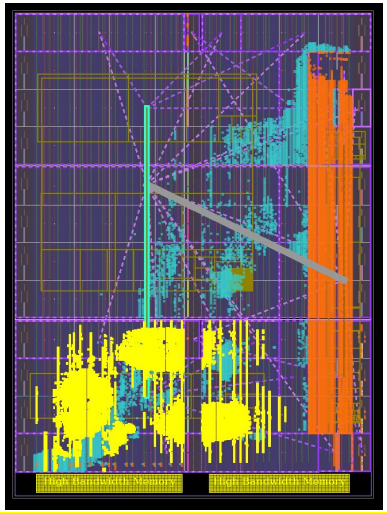
Metric Learning MLP on FPGA Workflow



FINN build targeting **300 kHz inferences** @ 300 MHz clock

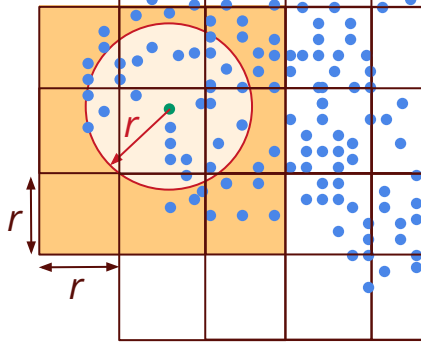
- Throughput measured @ 385 kHz inferences
- f_{max} > 300 MHz
- Resource utilization of streaming dataflow partition

Resource	Utilization (% of full U280)
LUT	25k (1.9 %)
FF	63k (2.4 %)
BRAM	166 (8.2 %)
URAM	0
DSP	**1547 (17 %)**



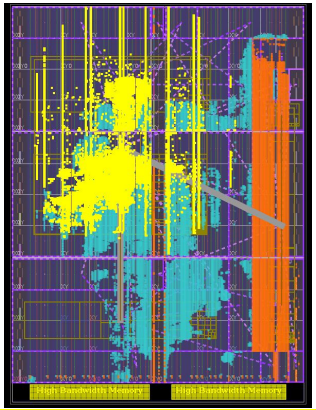
Streaming Dataflow Partition on U280

Metric Learning Radius Clustering on FPGA

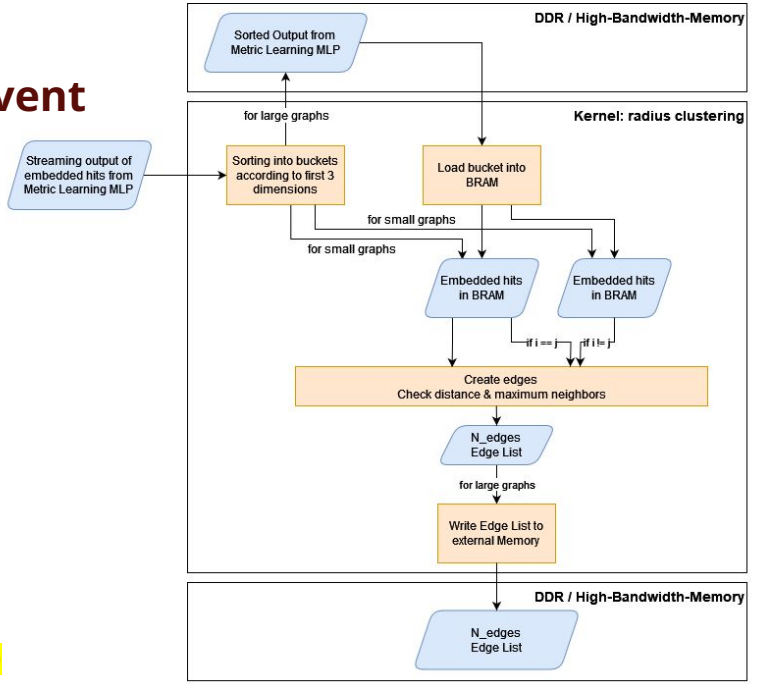


- **Sort data** in buckets and **compute** 12-dim **distances** → **create edge list**
- **HLS kernel** implemented with Vitis 2023.1
- Very preliminary resources for sizing of **full ITk event**
  Heavy on BRAM as MLP output features were not yet optimized for sorting data into buckets
- Timing not yet optimized
- Functionally verified

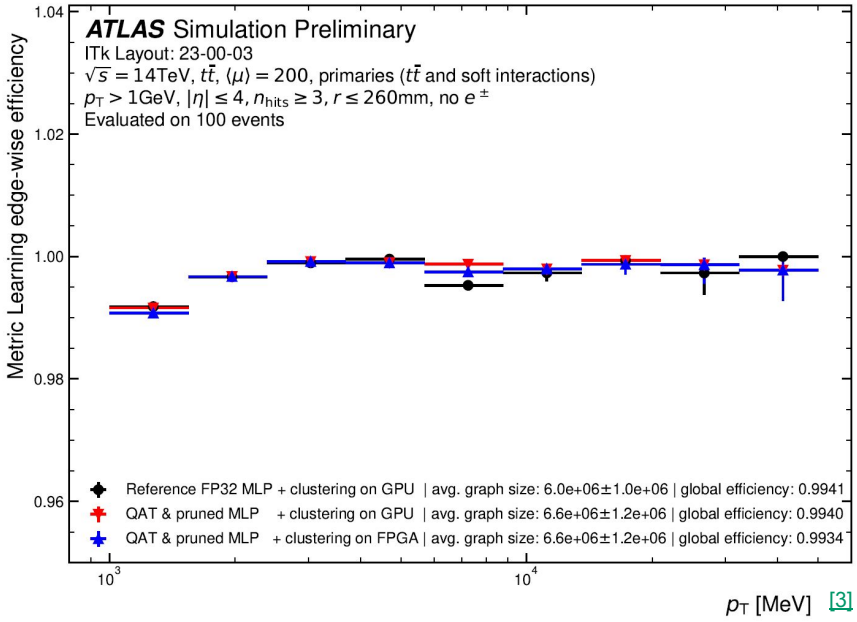
Resource	Utilization (% of full U280)
LUT	27k (2.1 %)
FF	35k (1.3%)
BRAM	**871 (43 %)**
URAM	0
DSP	140 (1.6 %)



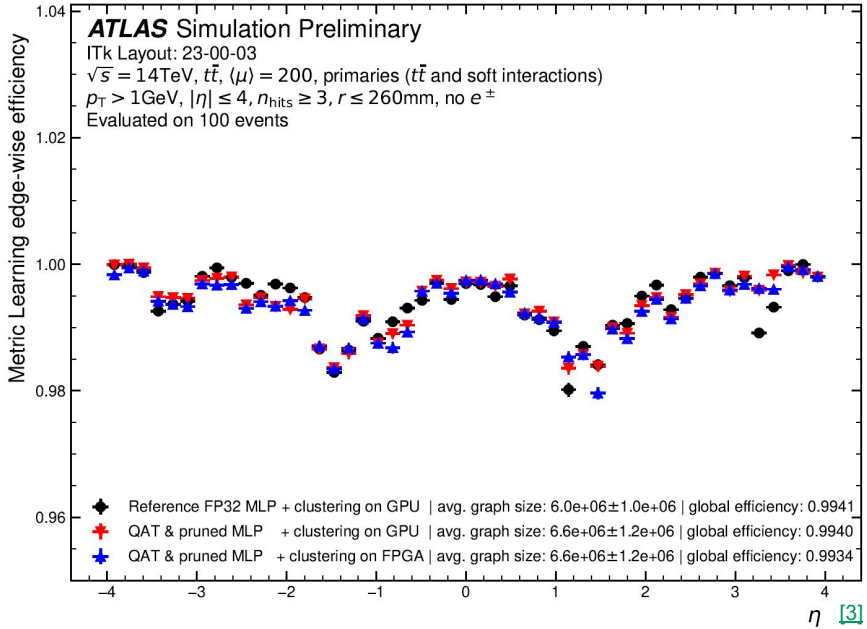
Radius Clustering on U280



Metric Learning – Physics Performance FPGA vs GPU



[3]



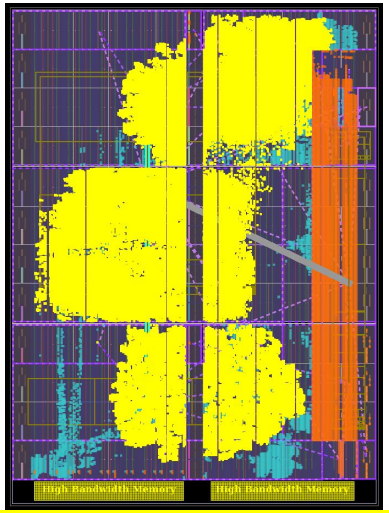
[3]

- Graphs created with compressed model are **~ 10 % larger for same efficiency**
- Differences FPGA vs GPU mostly due to QONNX export

Computing Performance Optimizations

FINN build targeting **300 MHz inferences** @ 300 MHz clock requires higher sparsity (here ~ 15k Parameters)

Resource	Utilization (% of full U280)
LUT	**352k (27 %)**
FF	107k (4.1 %)
BRAM	**0**
URAM	0
DSP	**133 (1.5 %)**

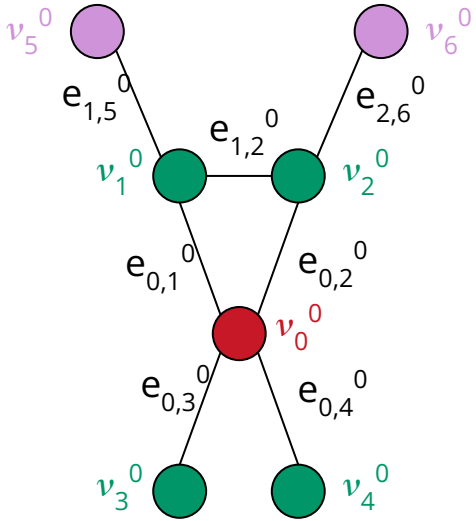


Streaming Dataflow Partition on U280 for a fully unrolled model

- Throughput measured @ 20 MHz (f_{max} = 201 MHz), most likely limited by under-sized FIFOs
- Model was trained on old samples, even smaller model has been trained, FINN build pending
- Metric Learning **optimized for sorting data into buckets** by **additional loss term: uncorrelated and wide uniform distributions** for 3 dimensions used for sorting
- This will reduce bucket size → expect significant speed-up and BRAM utilization reduction by more than a factor of 10 for the **radius clustering kernel**

Edge Labeling with Graph Neural Network

Geometric deep learning algorithm to classify edges as true or false



Repeat for *N* message passing steps

$$e_{x,y}^1 = \phi_e(v_x^0, v_y^0, e_{x,y}^0) \qquad v_x^1 = \phi_n(v_x^0, \sum e_{x,y}^1)$$

ϕ_e: Edge update MLP

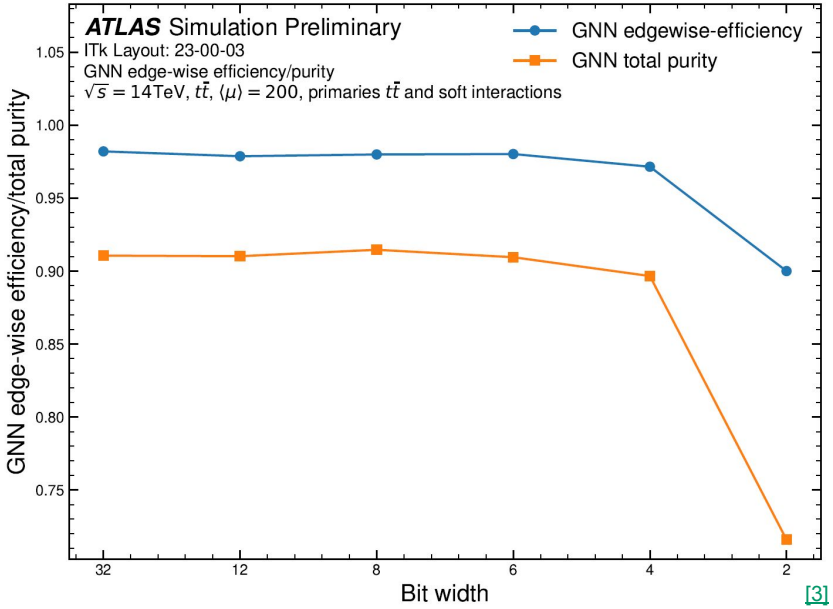
ϕ_n: Node update MLP

$\sum$: Aggregation function

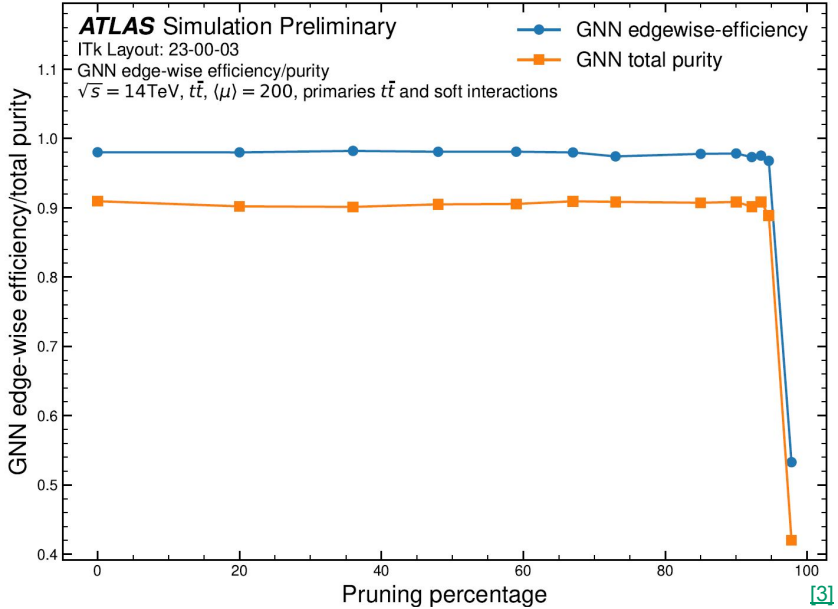
v_x^k = features of node *x* at iteration *k*

$e_{x,y}^k$ = features of edge between nodes *x* and *y* at iteration *k*

Graph Neural Network – Model Compression



[3]



[3]

PyTorch
PyG
+
Brevitas

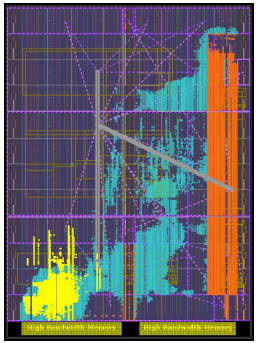
GNN Definition + Training Iterative Pruning

Quantization Aware Training

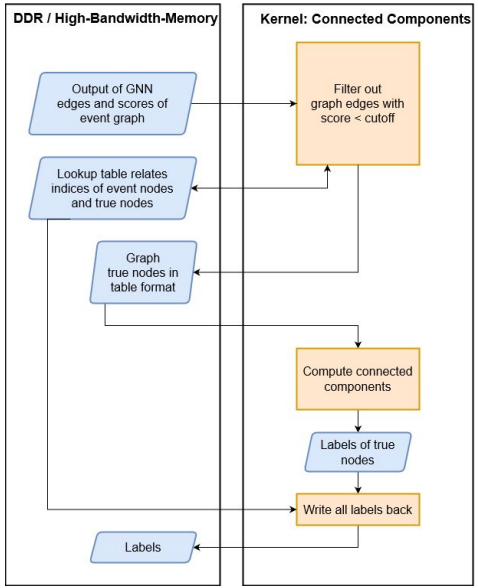
- GNN with 48 hidden dim + 8 message passing steps
- For pruning: quantization fixed to 6 bits
- Great potential for model compression

Graph Segmentation – Connected Components on FPGA

- Remove edges scored below threshold
- Store remaining edges in a memory efficient edge table
- Compute connected components
- HLS Kernel implemented with Vitis 2023.1
- Measured preliminary execution time:
  193 ms for a realistic full detector event
- Identified future optimizations
  - In- and output data structures
  - Full event-pipeline
  - Arbitrary precision data types



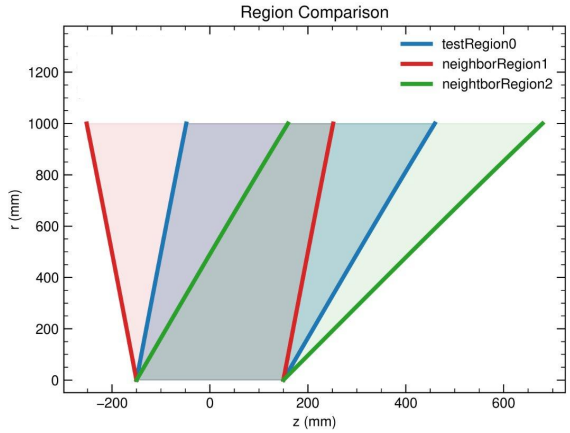
Connected Components on U280

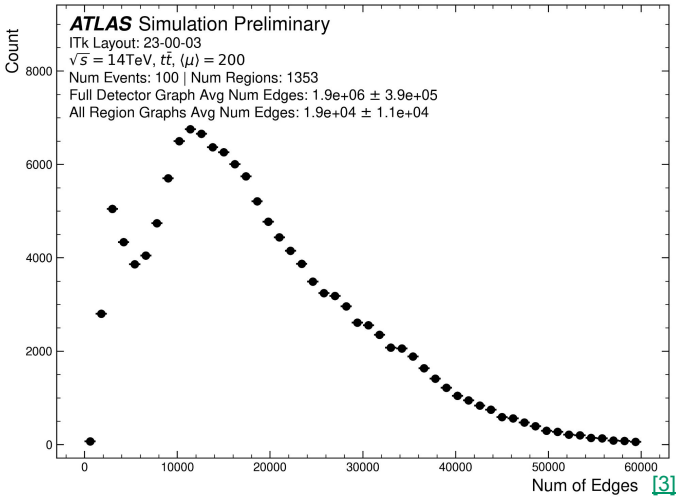


Resource	Utilization (% of full U280)
LUT	13k (1%)
FF	17k (0.7%)
BRAM	181 (9%)
URAM	0
DSP	0

Detector Regionalization

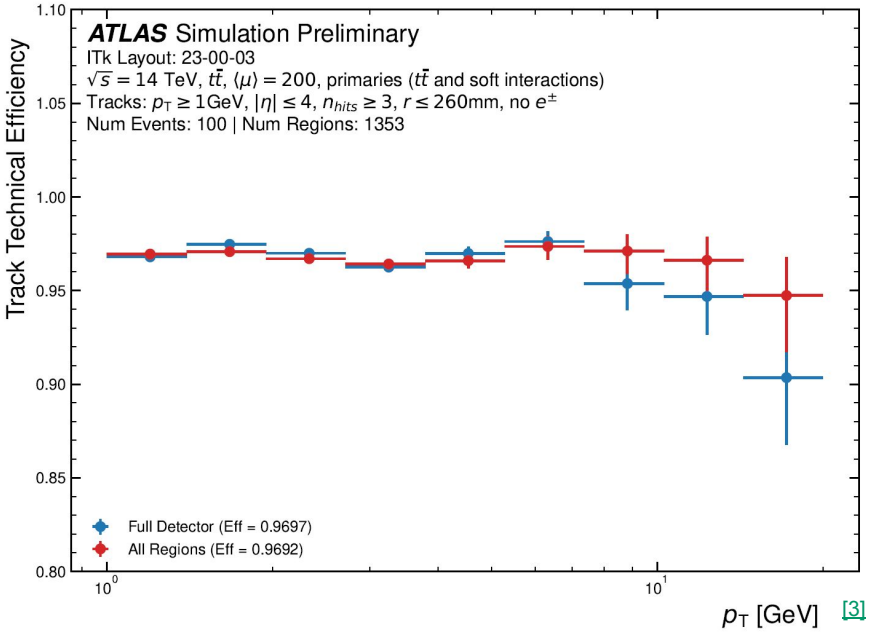
- Strategy to fit graphs into FPGA on-chip memory
- Region Definitions
  - 0.2 η x 0.2 ϕ with z/ϕ spread and overlap to account for track curvature
  - 1353 regions with deep overlap
    → each true track inside at least one region
- **Regional graph construction** implementation with Module Map > 99.5 % edge-wise efficiency
- Results in **average graph size reduction** by a **factor** of **100** compared to full detector



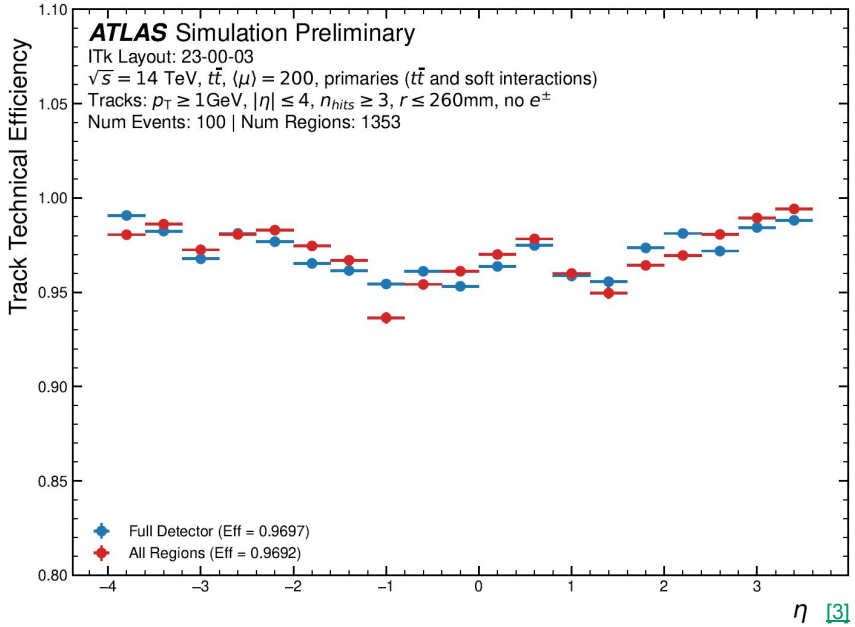


[3]

Detector Regionalization – Track Technical Efficiency



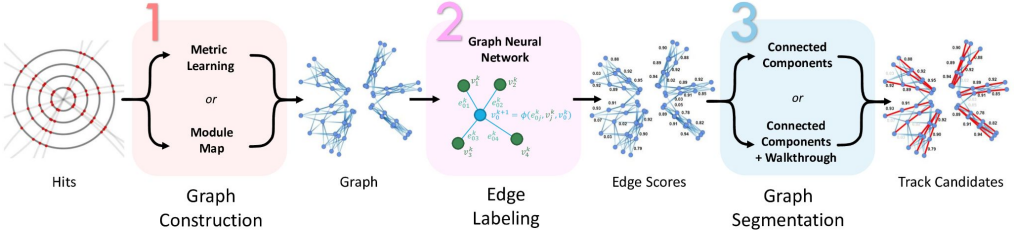
[3]



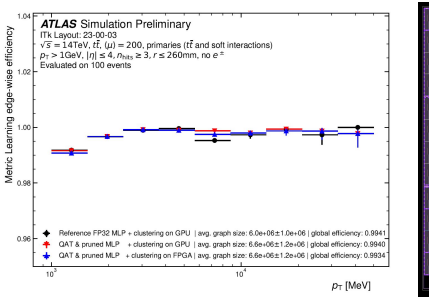
[3]

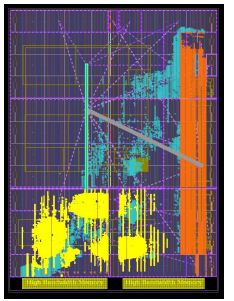
- Interaction Network trained on full detector events
- No significant difference in efficiency for inference regional vs full detector
- Overlaps lead to significant increase of duplicate tracks → duplicate removal

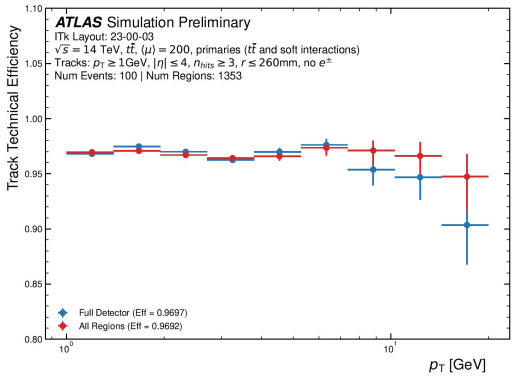
Summary



- **Heterogeneous online computing farm** under consideration for the **ATLAS Event Filter at HL-LHC**
- Development of **GNN-based track finding on FPGAs**
- Preliminary standalone implementations of FPGA algorithms for graph construction and segmentation **available and functionally verified**
- Started compression and FPGA implementation of Interaction Network
- **Validated detector regionalization** as an approach to fit graphs into on-chip memory







Backup

References

[1] ATLAS Inner Tracker Layout 03-00-00
https://atlas.web.cern.ch/Atlas/GROUPS/PHYSICS/PLOTS/ITK-2023-001/

[2] Track finding performance plots for a Graph Neural Network pipeline on ATLAS ITk Simulated Data
https://atlas.web.cern.ch/Atlas/GROUPS/PHYSICS/PLOTS/IDTR-2022-01/

[3] ATLAS Experiment – Public Results: Approved plots for the EF Tracking project
https://twiki.cern.ch/twiki/bin/view/AtlasPublic/EFTrackingPublicResults

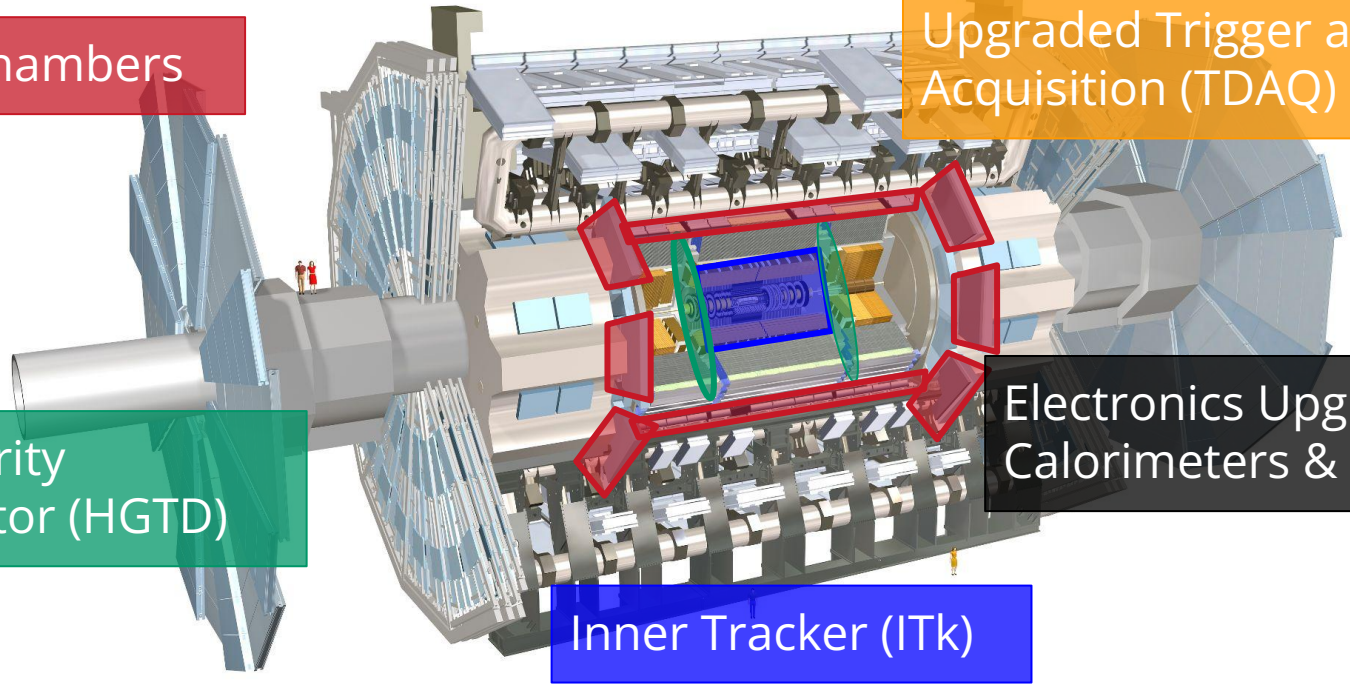
Related contributions @ CHEP

- Development of an FPGA based track reconstruction pipeline for the ATLAS Event Filter [Thursday, 14:24 (track 2)]
- Performance of the ATLAS GNN4ITk Particle Track Reconstruction GPU pipeline [Monday, (poster)]
- Improving Computational Performance of ATLAS GNN Track Reconstruction Pipeline [Thursday, 16:33 (track 3)]
- High Performance Graph Segmentation for ATLAS GNN Track Reconstruction [Thursday, 16:51 (track 3)]
- Energy-efficient graph-based algorithm for tracking at the HL-LHC [Thursday, 17:45 (track 3)]

Upgrade of the ATLAS Experiment for the HL-LHC

New Muon Chambers

Upgraded Trigger and Data Acquisition (TDAQ) System



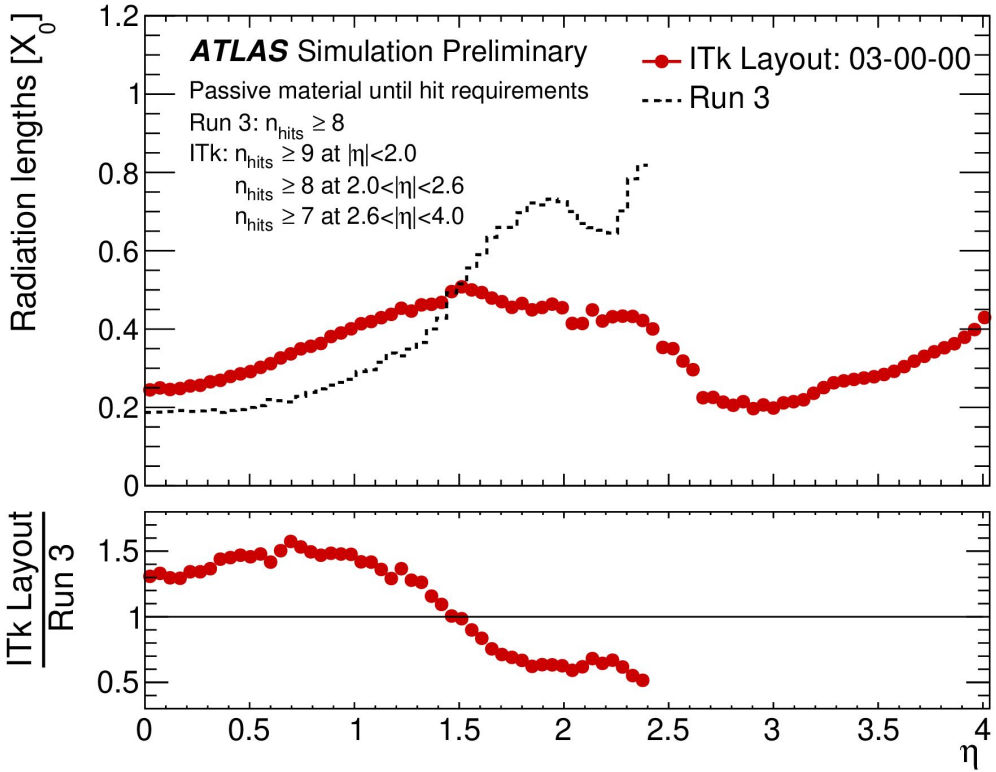
High Granularity Timing Detector (HGTD)

Electronics Upgrades for Calorimeters & Muon

Inner Tracker (ITk)

https://cds.cern.ch/record/1095924

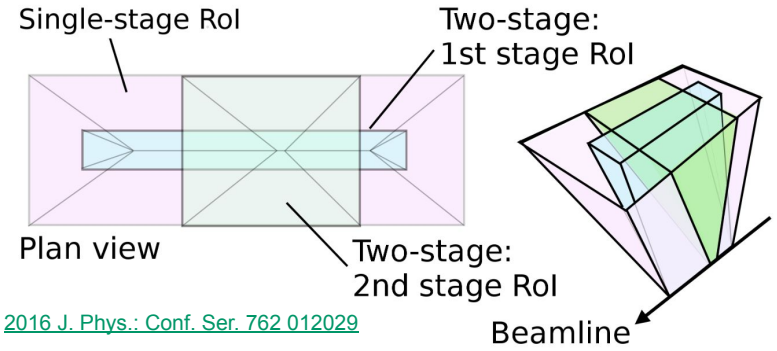
ATLAS ITk material



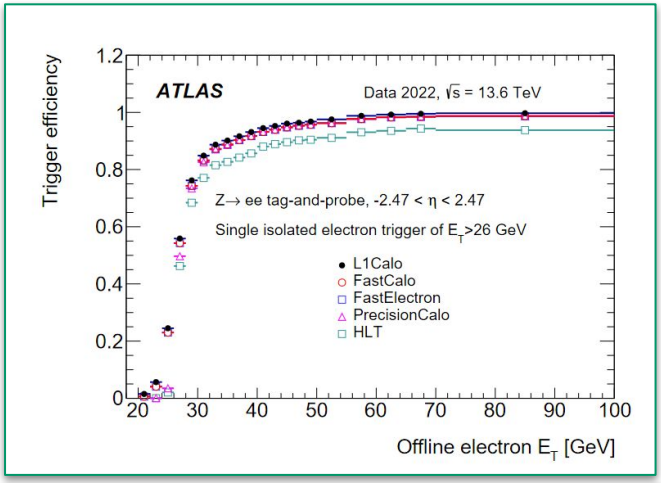
https://atlas.web.cern.ch/Atlas/GROUPS/PHYSICS/PLOTS/ITK-2023-001/

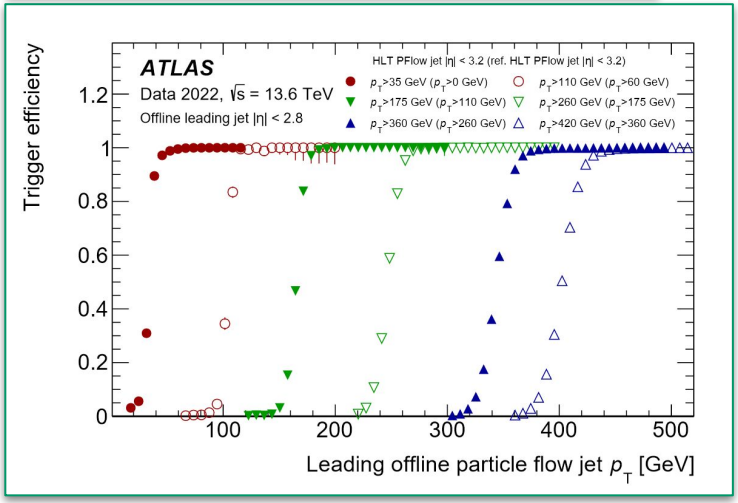
Tracking for the ATLAS Trigger

- **Tracking** is a **crucial** element of the ATLAS **trigger**
- Leptons (isolation), b-physics: **Regions of Interest** (100 kHz)
- Jets and Missing p_T: **Full detector** (14 kHz)



2016 J. Phys.: Conf. Ser. 762 012029

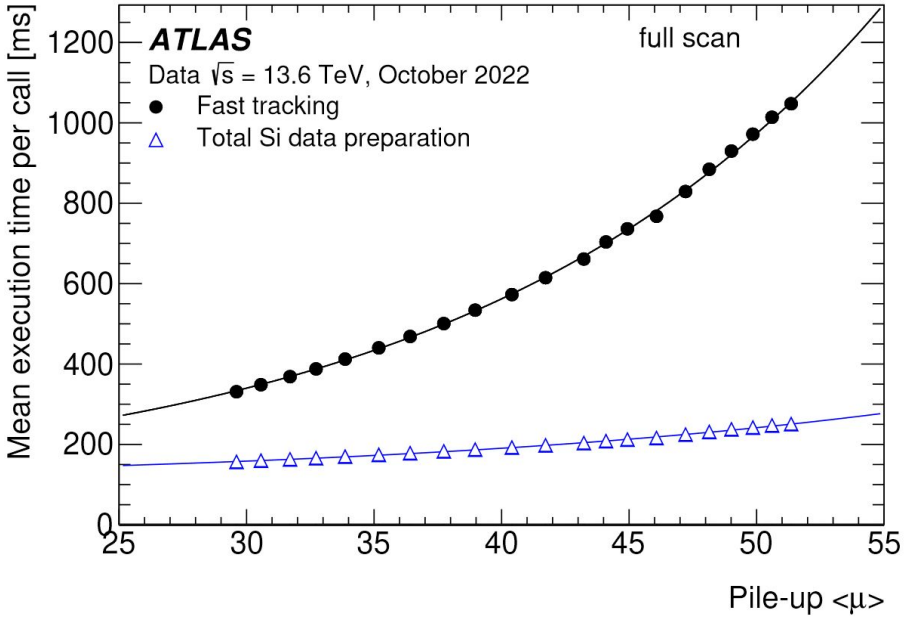




G. Aad et al 2024 JINST 19 P06029

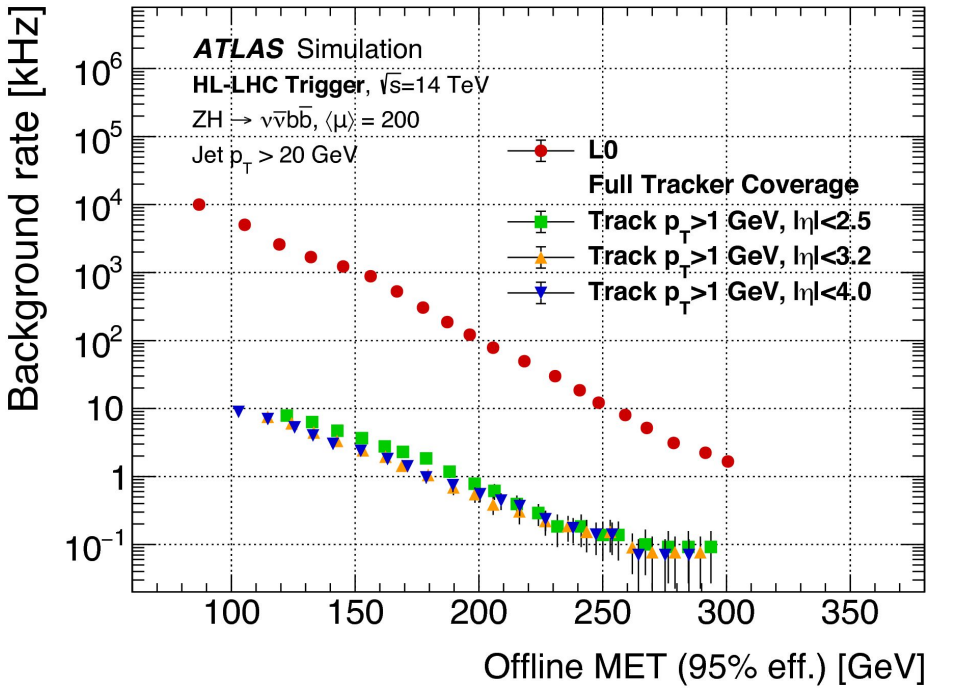
Kalman Filter at Trigger Level

- Very well suited for **precision track fitting**
- Using full **material** and **magnetic** field information takes CPU time → **approximations** for track finding
- Even then: biggest CPU consumer in ATLAS trigger system by far!
- **Execution time scales worse than linearly** with number of simultaneous interactions



G. Aad et al 2024 JINST 19 P06029

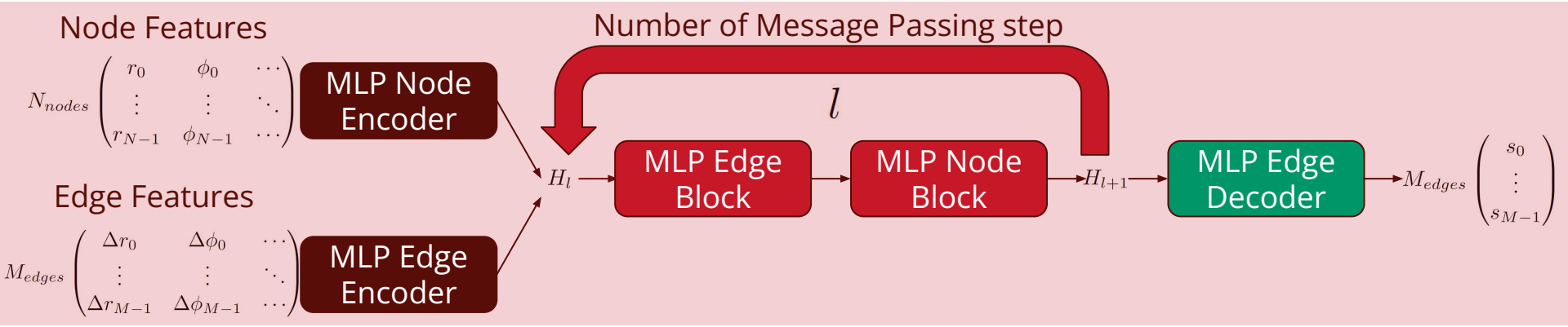
Importance of Tracking in the Trigger @ HL-LHC



ATLAS-TDR-029

Edge Labeling with Graph Neural Network

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



Input Graph → Encoders → Interaction Network → Decoder → Edge Scores

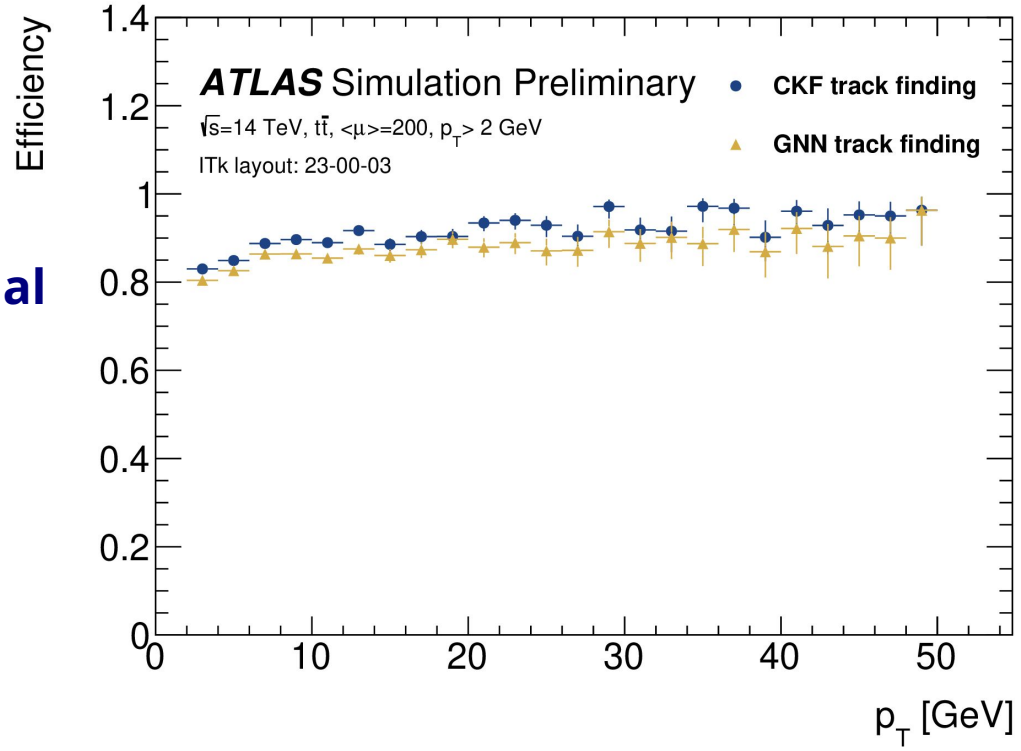
Embeds graph input features into a D-dimensional latent space

Learns geometric patterns of tracks

Transforms the latent features of each edge into a classification score

Results of GNN tracking for ATLAS ITk

- GNN track finding $\rightarrow \chi^2$ fit to extract track parameters
- **Comparison** against ATLAS default tracking: **Combinatorial Kalman Filter** (CKF)
- **GNN efficiency close to CKF**
- **Comparable resolution** on impact parameters



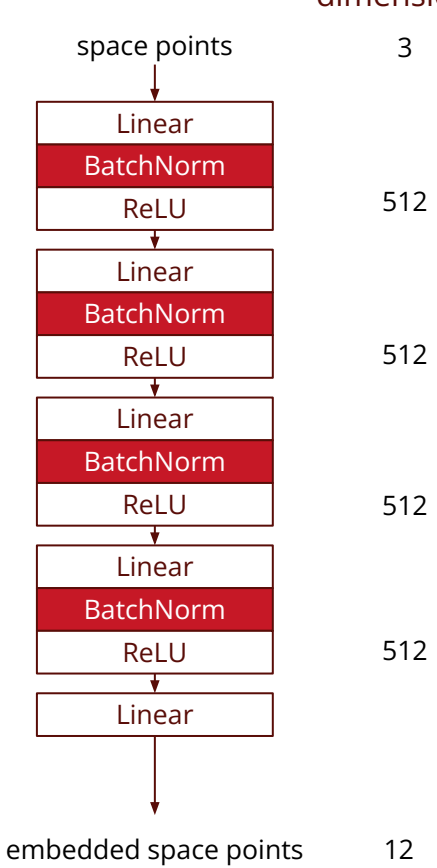
https://atlas.web.cern.ch/Atlas/GROUPS/PHYSICS/PLOTS/IDTR-2023-06/
https://cds.cern.ch/record/2882507

Metric Learning: Model Compression

- Application of **Quantization** and **Pruning**:
  - Single **largest MLP** in whole pipeline
  - GNN also heavily relies on MLPs
- Trained on ITk Geant4 simulation of $t\bar{t}$, $\langle\mu\rangle$ = 200
- Target particles
  - Primaries except electrons
  - $p_T > 1$ GeV
  - ≥ 3 space points
- BatchNorm instead of LayerNorm for FPGA study
- Performance metric

  Purity of constructed graphs evaluated

  @ 98 % edge-wise graph construction efficiency

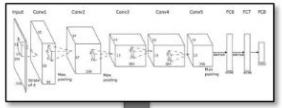


~ 800k Parameters

Workflow for this study

[Public]

FINN Framework: From DNN to FPGA Deployment



Brevitas
Training in PyTorch
Algorithmic optimizations

- Quantization-aware training for DNNs
- Library of common quantized layers
- Includes pre-trained examples

other quantizers
(QKeras, HAWQ)

QONNX

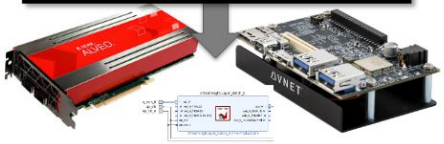
- Quantized NN exchange format + toolkit

FINN Compiler
QNN-to-accelerator

- Perform optimizations
- Assemble parameterized HLS/RTL modules
- Generate a DNN hardware IP

Deployment
Verification, integration...

- Run RTL testbenches to simulate IP
- System-level integration in Vivado IPI
- Rapid prototyping with PYNQ



All open source & actively maintained

69

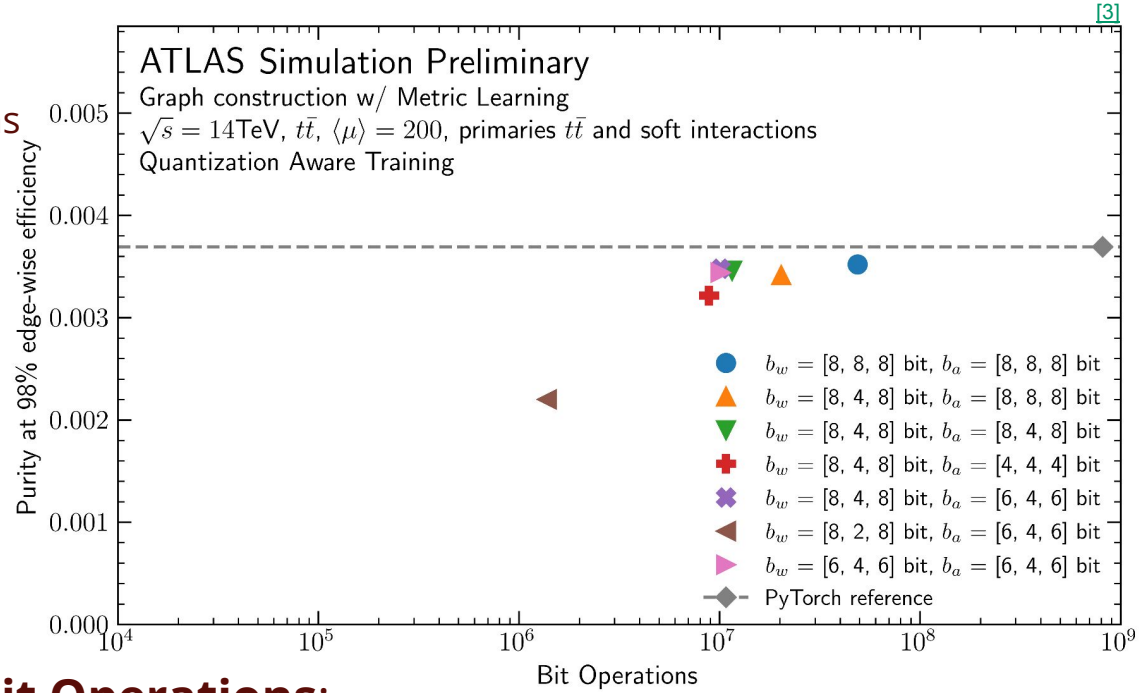
AMD
together we advance_

https://indico.cern.ch/event/1283970/contributions/5537711/

Quantization Aware Training Results

- Fully quantized
  - Linear layers: weights & bias
  - ReLU activations
  - Input data
- Heterogeneous quant.
  - Bit widths b_i= [x,y,z] bit
    - b_w: weights
    - b_a: activations
    - x: first layer
    - y: hidden layers
    - z: last layer

[3]



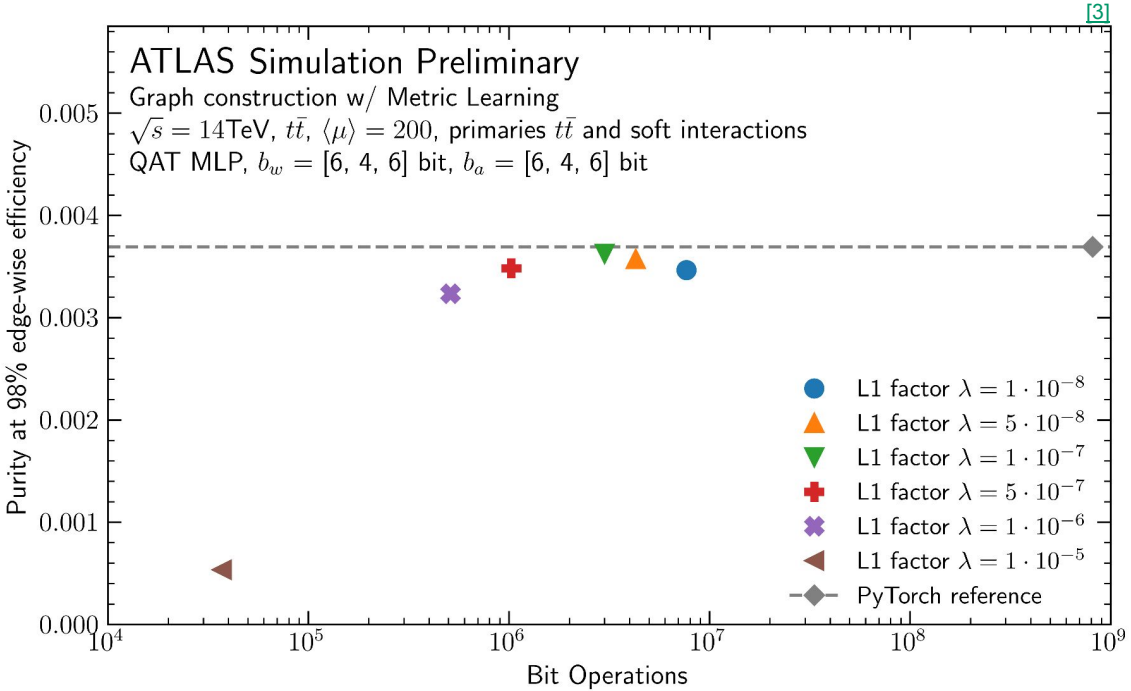
- Model size evaluated in **Bit Operations**:

  BOP $\propto$ $(1 - f_p)\, b_a\, b_w$, f_p: pruning fraction (or sparsity)

L1 Regularisation Results

- L1 loss weighting factor λ highly **model dependent**
- A well chosen weight can even **enhance accuracy**
- **Creates sparsity** → reflects in bit operations for quantized networks

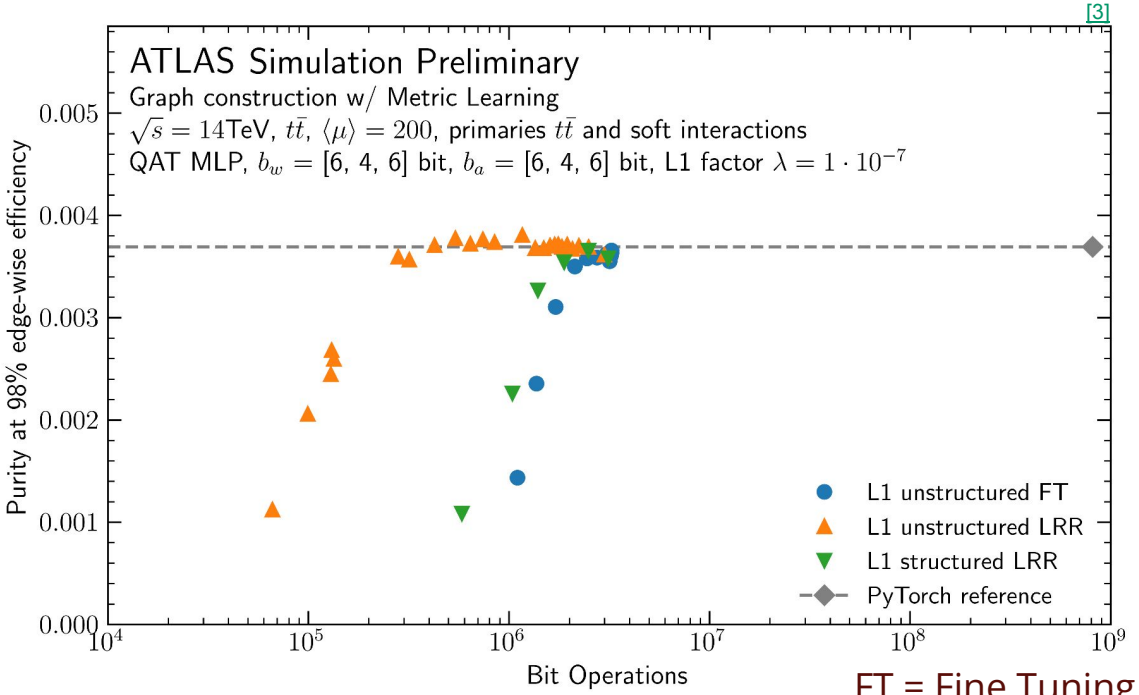
[3]



Iterative Pruning Results

- Best performance for every pruning step versus model size
- **Unstructured pruning:**
  - Fine tuning allows for about 83 % sparsity
  - **Learning Rate Rewinding** allows for **> 98 %** sparsity
- Structured pruning may require different regularisation

[3]



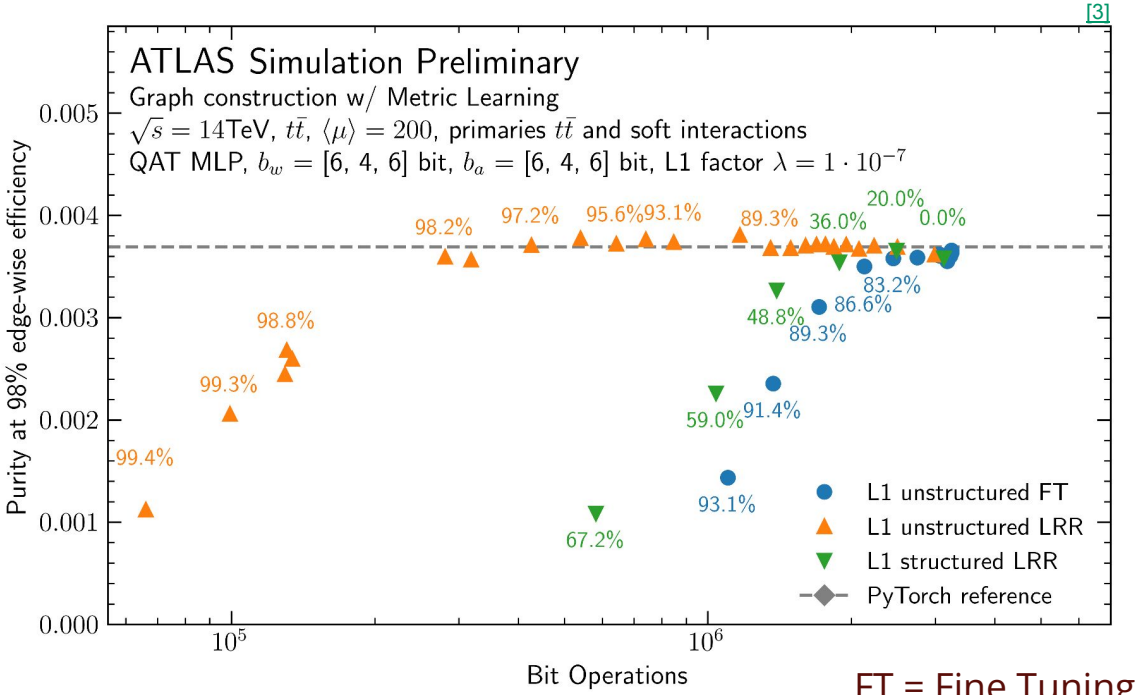
FT = Fine Tuning

LRR = Learning Rate Rewinding

Iterative Pruning Results

- Best performance for every pruning step versus model size
- **Unstructured pruning:**
  - Fine tuning allows for about 83 % sparsity
  - **Learning Rate Rewinding** allows for **> 98 %** sparsity
- Structured pruning may require different regularisation

[3]

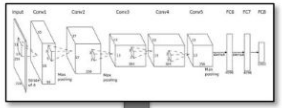


FT = Fine Tuning
LRR = Learning Rate Rewinding
sparsity in %

Workflow for this study

[Public]

FINN Framework: From DNN to FPGA Deployment



Brevitas
Training in PyTorch
Algorithmic optimizations

- Quantization-aware training for DNNs
- Library of common quantized layers
- Includes pre-trained examples

other quantizers
(QKeras, HAWQ)

QONNX

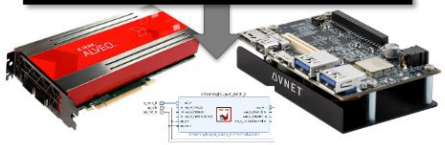
- Quantized NN exchange format + toolkit

FINN Compiler
QNN-to-accelerator

- Perform optimizations
- Assemble parameterized HLS/RTL modules
- Generate a DNN hardware IP

Deployment
Verification, integration...

- Run RTL testbenches to simulate IP
- System-level integration in Vivado IPI
- Rapid prototyping with PYNQ



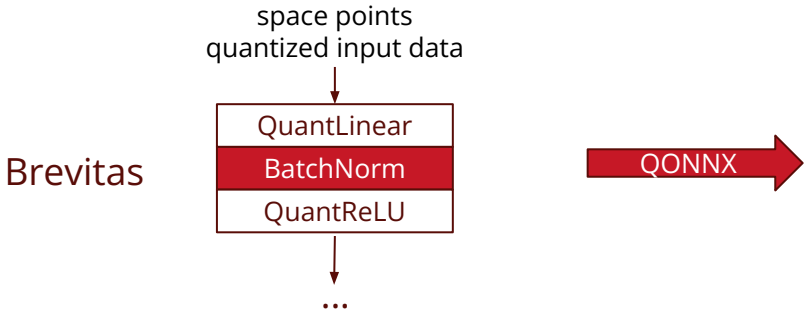
All open source & actively maintained

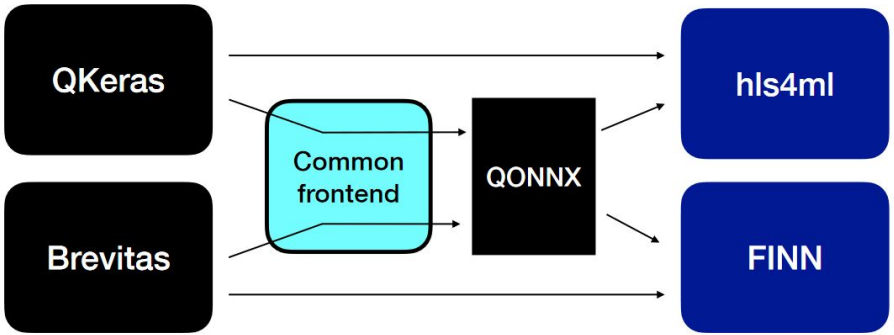
AMD
together we advance_

69

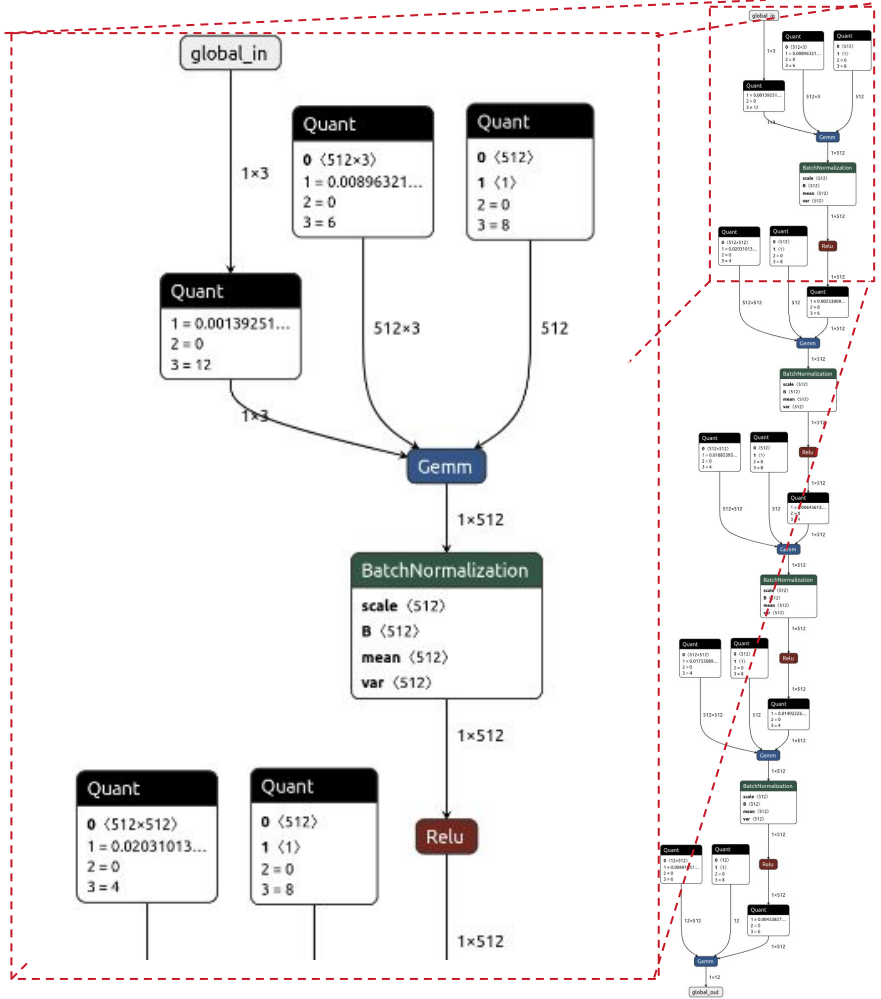
https://indico.cern.ch/event/1283970/contributions/5537711/

QONNX model export





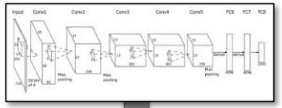
https://arxiv.org/abs/2206.11791



Workflow for this study

[Public]

FINN Framework: From DNN to FPGA Deployment



Brevitas
Training in PyTorch
Algorithmic optimizations

- Quantization-aware training for DNNs
- Library of common quantized layers
- Includes pre-trained examples

other quantizers
(QKeras, HAWQ)

QONNX

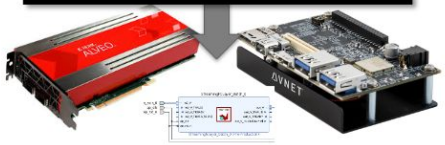
- Quantized NN exchange format + toolkit

FINN Compiler
QNN-to-accelerator

- Perform optimizations
- Assemble parameterized HLS/RTL modules
- Generate a DNN hardware IP

Deployment
Verification, integration...

- Run RTL testbenches to simulate IP
- System-level integration in Vivado IPI
- Rapid prototyping with PYNQ



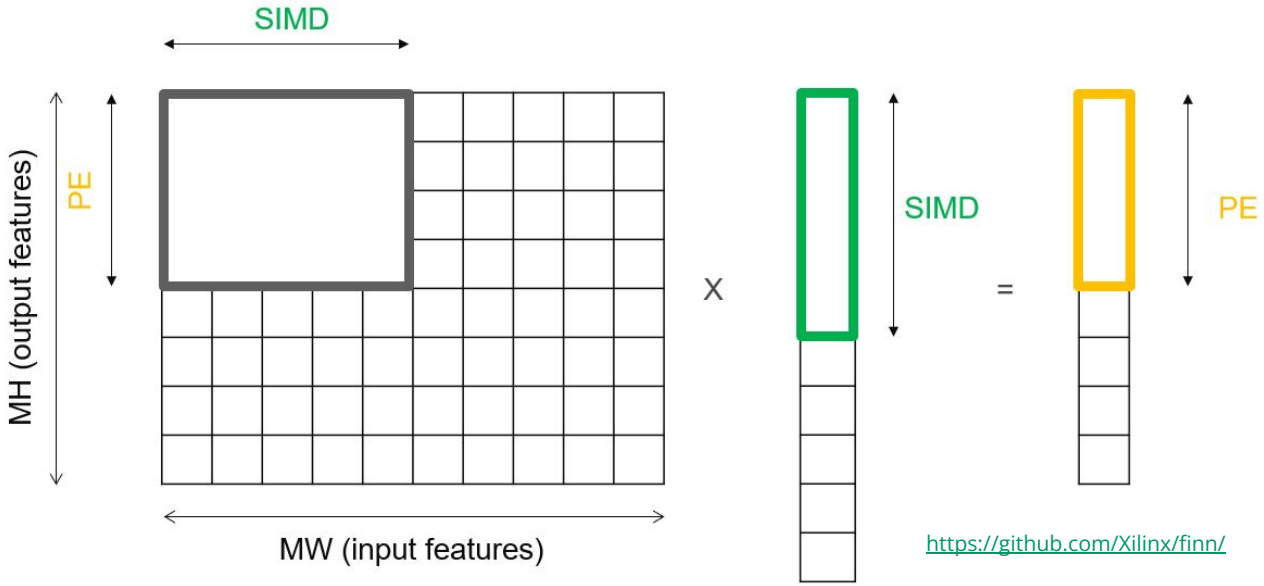
All open source & actively maintained

69

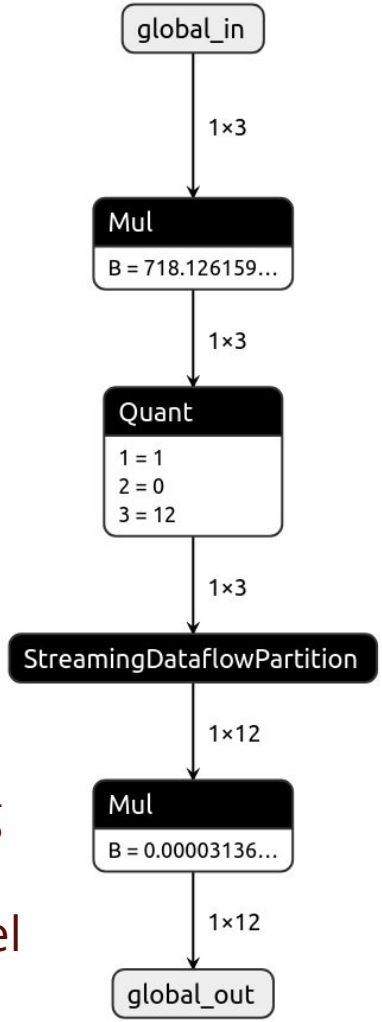
AMD
together we advance_

https://indico.cern.ch/event/1283970/contributions/5537711/

MLP Translation for FPGAs with FINN

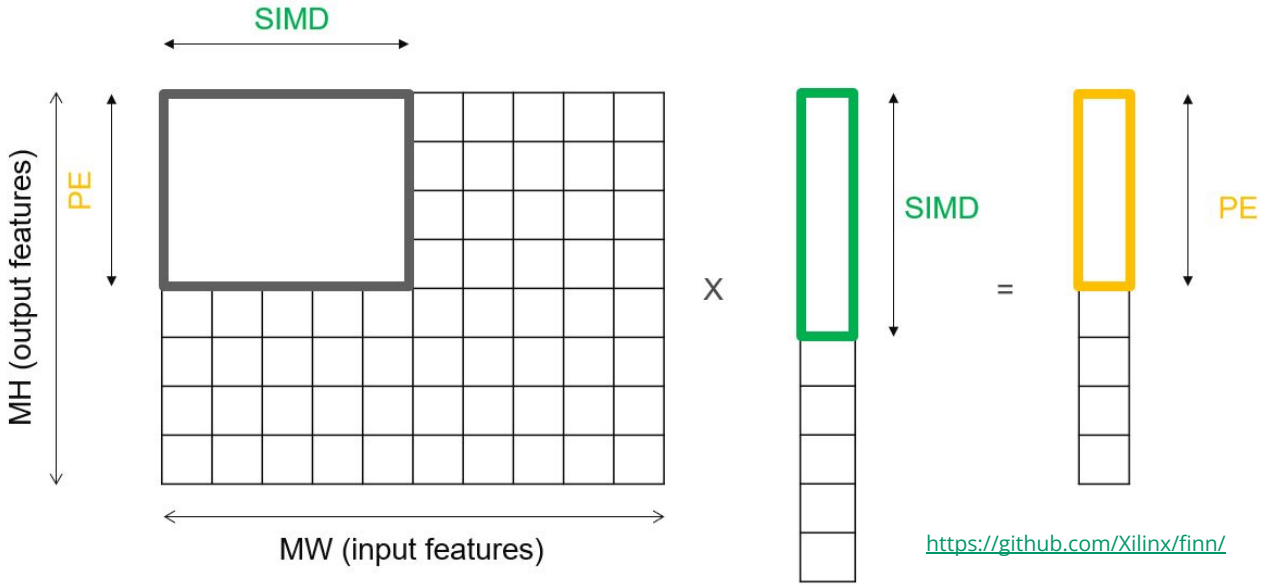


https://github.com/Xilinx/finn/

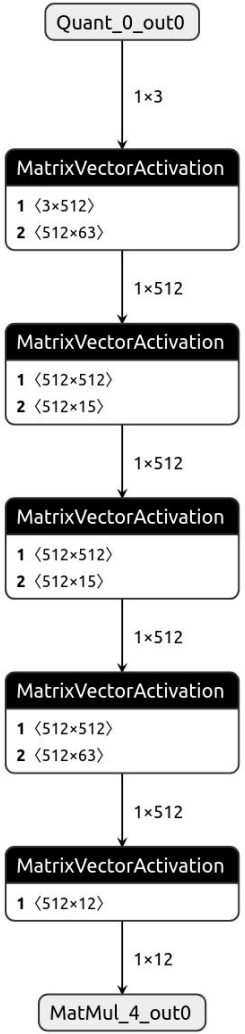


- We can **trade throughput vs. resource usage** by varying Processing Elements (PE) and Vectorization (SIMD)
- **Sparsity** can be exploited by **fully parallelizing** the model

MLP Translation for FPGAs with FINN



https://github.com/Xilinx/finn/



- We can **trade throughput vs. resource usage** by varying Processing Elements (PE) and Vectorization (SIMD)
- **Sparsity** can be exploited by **fully parallelizing** the model