

# Accelerating GNNs on FPGAs for Particle Track Reconstruction using OpenCL and hls4ml

**Aneesh Heintz**, Vesal Razavimaleki, Javier Duarte, Gage DeZoort, Isobel Ojalvo, Savannah Jennifer Thais, Markus Julian Atkinson, Mark Neubauer, Lindsey Gray

12-02-2020



Cornell University

UC San Diego



PRINCETON  
UNIVERSITY



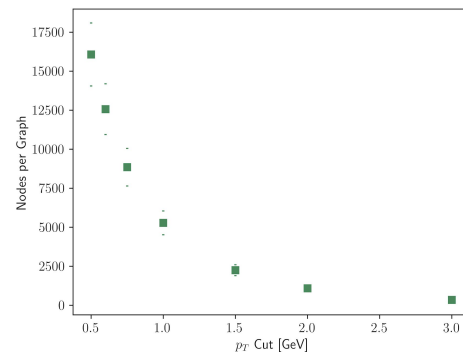
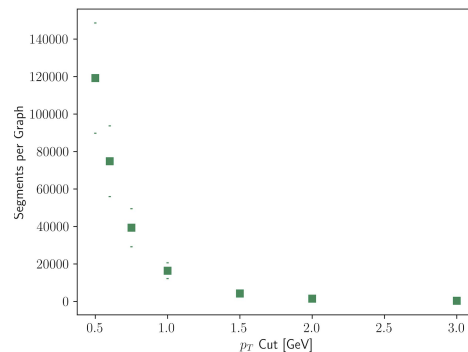
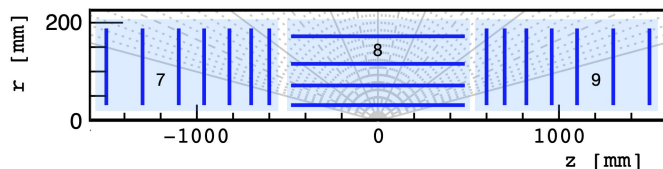
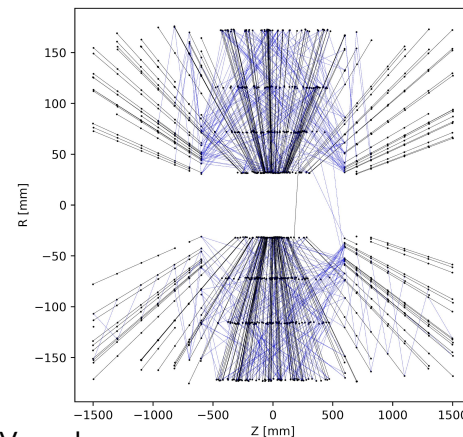
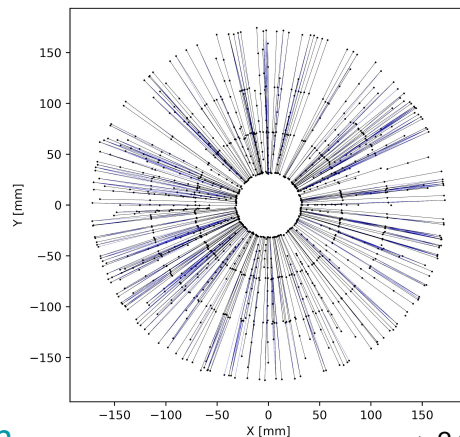
UNIVERSITY OF  
**ILLINOIS**  
URBANA-CHAMPAIGN



**Fermilab**

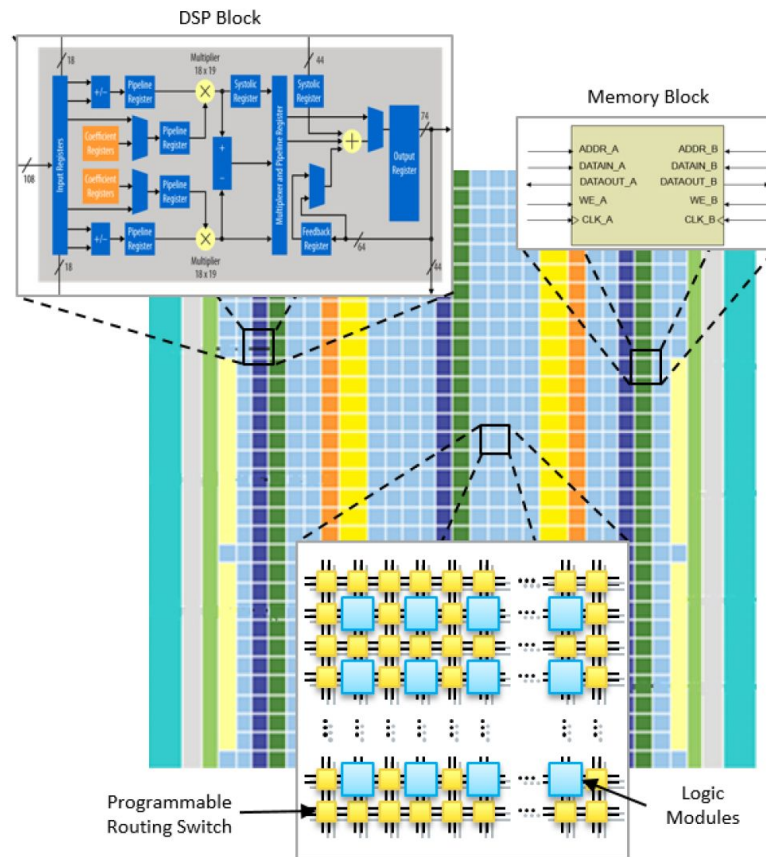
# TrackML dataset for track reconstruction

- Track reconstruction is cast as a GNN “edge classification” problem
- Interest in using GNNs in FPGA-based trigger or co-processors
  - Requires FPGA-specific implementation
- Input data represented as a graph
  - Nodes → hits
  - Edges → linear approx. of particle track

Graph size vs.  $p_T$  in the pixel barrel & endcaps $p_T > 2$  GeV graph

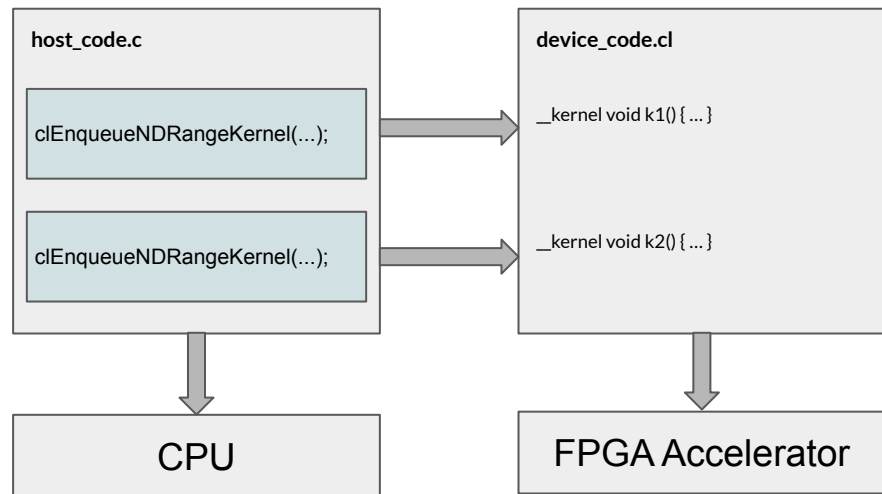
# FPGAs

- Re-configurable Integrated circuits
  - Consists of several small computational units
- Integrates combinations of
  - Lookup tables (LUTs)
  - Registers
  - On-chip memories (global and local)
  - Arithmetic hardware
- Advantages of FPGAs
  - Support wide, heterogenous and unique parallel implementations
  - Lower latency & more energy efficient than GPUs



# OpenCL

- Open source C-based interface for parallel computing (on CPUs, GPUs & FPGAs) using task and data-based parallelism.
- Targets CPU+FPGA co-processing system
  - Intel Xeon (CPU, host)
  - Intel Arria 10 (FPGA)
- Adaptable to changes in network architecture



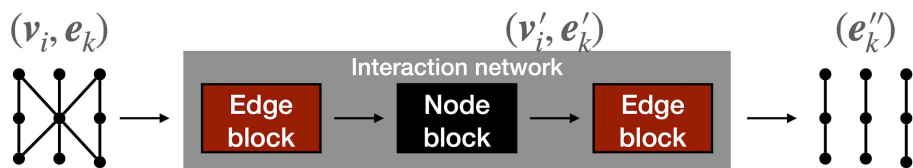
Adapted from Intel FPGA SDK for OpenCL Pro Edition: Programming Guide

# OpenCL Network

## ● Architecture: Interaction Network

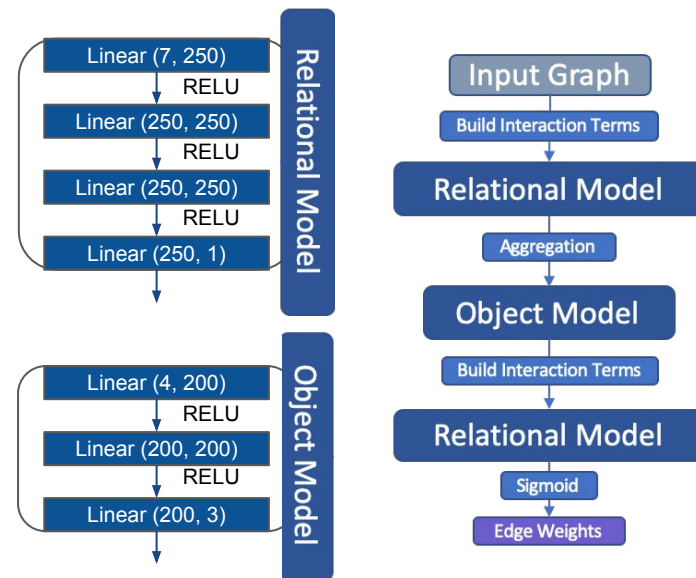
[[arXiv:1612.00222](https://arxiv.org/abs/1612.00222)]

- Example of a message-passing NN
- Applies separate object and relational models to update node and edge embeddings
- Can be successfully applied to edge classification



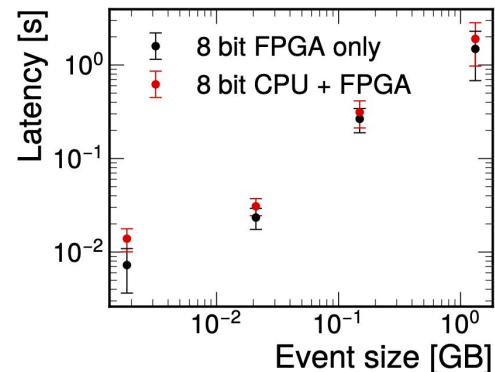
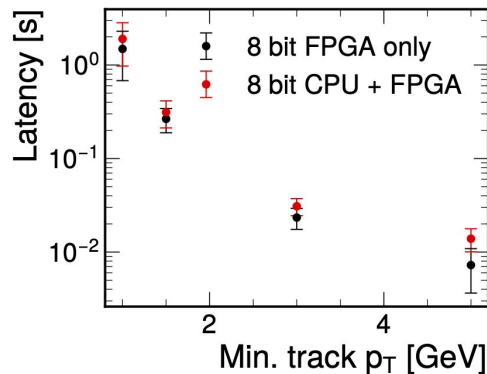
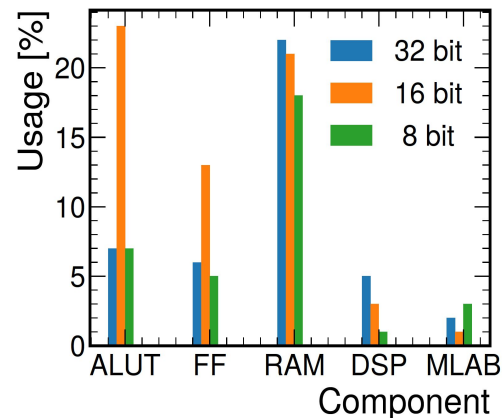
$$e'_k = \phi_2^e(e_k, v_{r_k}, v_{s_k}) \quad v'_i = \phi_2^v(\bar{e}'_i, v_i) \quad e''_k = \phi_2^e(e'_k, v'_{r_k}, v'_{s_k})$$

$$\bar{e}'_i = \rho^{e \rightarrow v}(E_i)$$



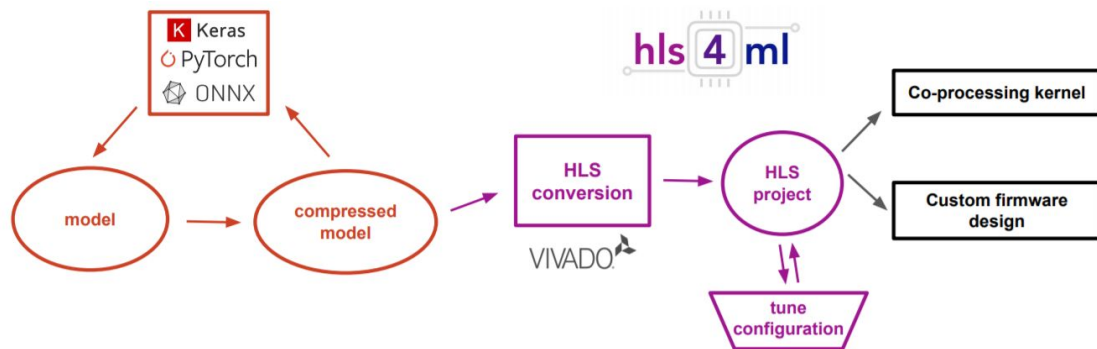
## OpenCL Implementation Results

- Kernel: NDRange loc. + reg. Tiling
- Lower bit precision allows for more on-device data (lower  $p_T$  cuts)
- Runtime results presented for minimum  $p_T > 5$  GeV
- Largest Bottlenecks
  - Matrix multiplication
  - Overhead due to enqueueing NDRange kernels
  - $p_T > 0.75$  GeV cut event graphs were too large for the resources available



# hls4ml

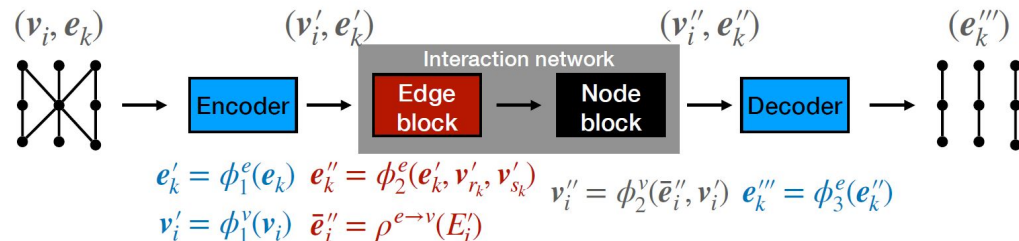
- User-friendly open-source tool to build and optimize deep learning (DL) models for FPGAs
- Reads in models trained with standard DL libraries
- Utilizes Xilinx HLS software
- Includes pre-implemented NN building blocks (layers, activation functions, binary NN ...)



<https://github.com/fastmachinelearning/hls4ml>

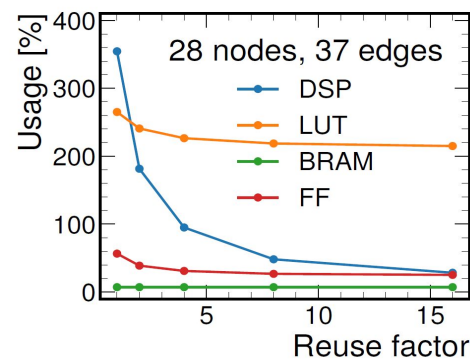
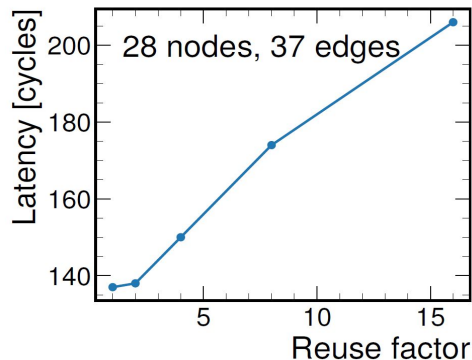
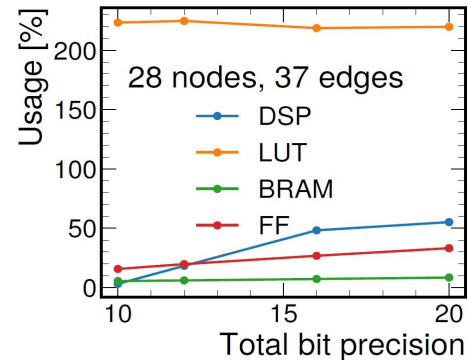
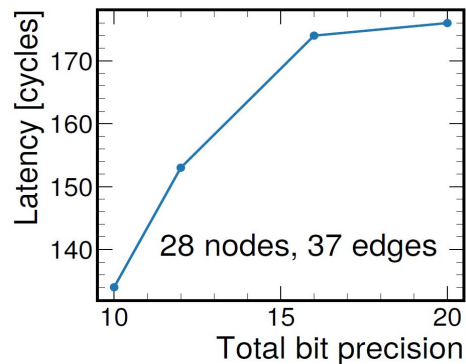
## hls4ml Network

- Based on [[arXiv:2003.11603](https://arxiv.org/abs/2003.11603)]
- Encoder transforms node and edge features into an 8-dim latent space
- Edge block computes messages (updated edge features)
- Node block updates node features based on messages
- Decoder transforms edge features into edge classifier weights



## hls4ml Implementation Results

- Results are shown for 1/64 of a detector for  $p_T > 2$  GeV graphs -- Fixed graphs of 28 nodes, 37 edges (95th percentile of graph sizes)
- Latency  $< 1 \mu\text{s}$
- Bit precision scans show lower bit width results in smaller area, faster execution
- Reuse factor scans show trade-off between resource usage and latency



## Thank you! Questions?

### Summary

- We develop two FPGA implementations of GNNs for charged particle tracking
  - OpenCL: targets CPU-FPGA coprocessing and achieves a latency of 10 ms - 1 s depending on the  $p_T$  for full event graphs
  - hls4ml: targets both coprocessing and trigger and has a latency of 650 ns - 1  $\mu$ s (FPGA-only) for smaller, sectorized input graphs and smaller model
- Continued development may allow GNNs to be used in future computing workflows and the FPGA-based trigger at the LHC

### Acknowledgements

- This work was conducted with the support of the Institute for Research and Innovation in Software for High Energy Physics (IRIS-HEP)



Cornell University

UC San Diego



PRINCETON  
UNIVERSITY



UNIVERSITY OF  
**ILLINOIS**  
URBANA - CHAMPAIGN



**Fermilab**

# Backup Slides

---

**Aneesh Heintz**, Vesal Razavimaleki, Javier Duarte, Gage DeZoort, Isobel Ojalvo, Savannah Jennifer Thais, Markus Julian Atkinson, Mark Neubauer, Lindsey Gray



Cornell University

UC San Diego



PRINCETON  
UNIVERSITY



UNIVERSITY OF  
**ILLINOIS**  
URBANA - CHAMPAIGN



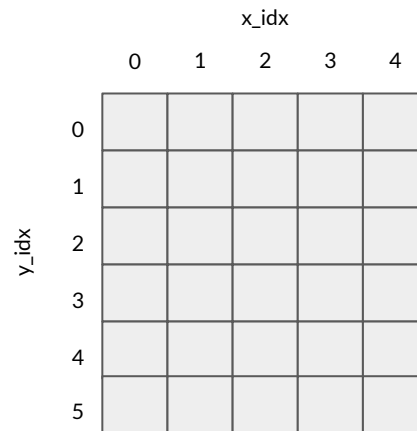
**Fermilab**

## OpenCL: NDRange Kernel

- Parallelize loop operations across work-items for simultaneous data processing
- Local and global work group dimensions explicitly specify data parallelism
  - Global Size
    - # of work-items to execute
    - Example in figure: 4 x 5
  - Local Size
    - # of work-items to group into a work group

```
#define global_idx(x_idx, y_idx, m) (x_idx * m + y_idx)

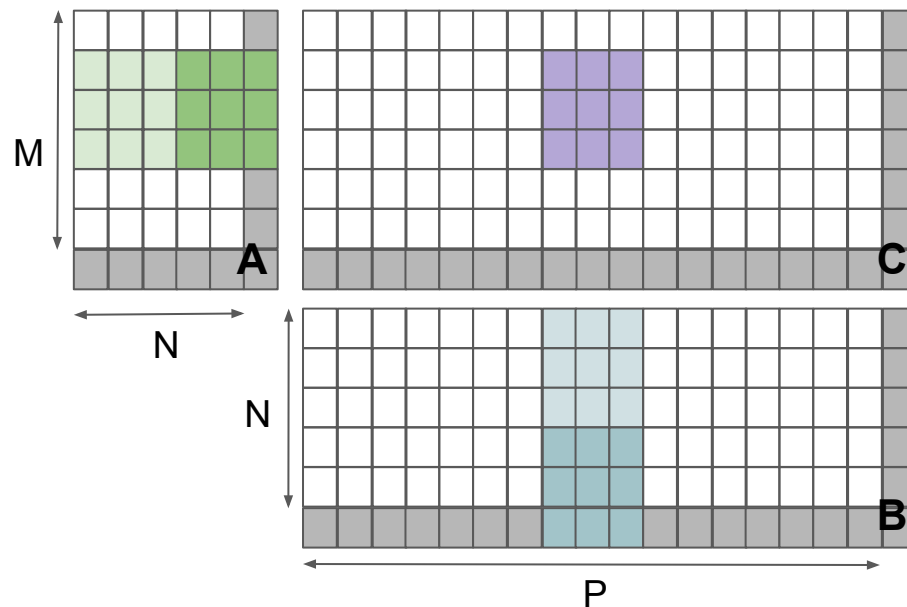
__attribute__((uses_global_work_offset(0)))
__kernel void transpose(__global float *a_t,
                       __global float *a,
                       ushort n,
                       ushort m)
{
    int x_idx = get_global_id(0);
    int y_idx = get_global_id(1);
    a_t[global_idx(y_idx, x_idx, n)] = a[global_idx(x_idx, y_idx, m)];
}
```



## OpenCL: NDRange Local Memory Tiling

- Tiling -- caches sub-blocks of matrices in on-chip local memory
- Reduces repetitive reading from FPGA globally shared memory
- Matrix Multiplication using local memory tiling works as fast as what Intel has written for their in-house FPGA acceleration library (Intel FAL).

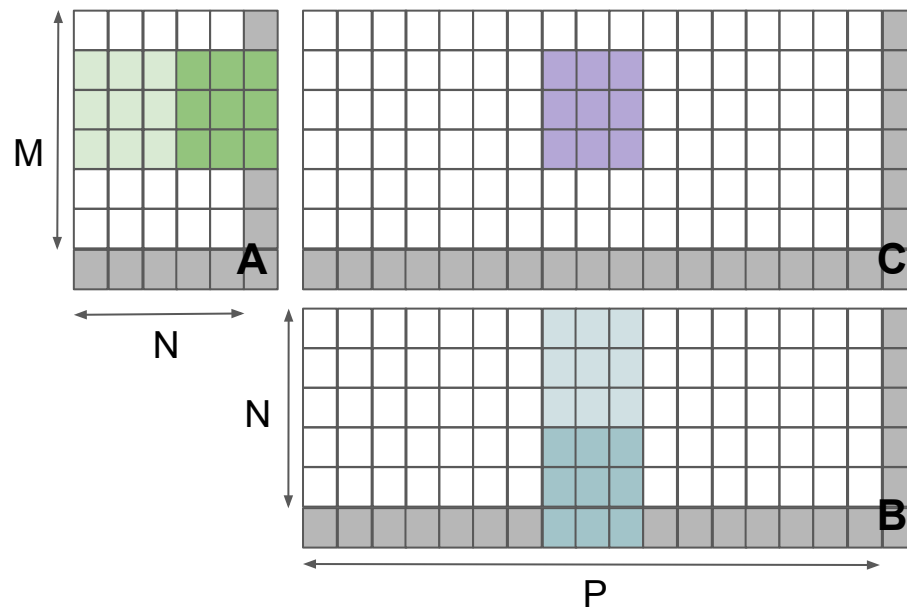
### Matrix Multiplication Example



## OpenCL: NDRange Register Blocking

- Increase the amount of work done per thread in each work-item
- Reduces repetitive reading from FPGA local memory
- Decreases global and local memory sizes
  - More device side resources used
  - Fewer host side resources used
- Adding register tiling on top of local memory tiling leads to even faster matrix multiplication execution

### Matrix Multiplication Example



## hls4ml Implementation Results: Performance

- “Receiver” and “sender” arrays allow dynamic indexing of node features
- “Zero-padding” sets max graph size (edges and nodes)
- Pipelining at edge/node-block-level with initiation interval = reuse factor
- Fixed-point precision bit width
  - Good performance at  $\langle 12,6 \rangle$
  - Small GNN can correctly classify track segments with AUC  $\sim 0.983$

