Towards Achieving Real-time GNN Inference

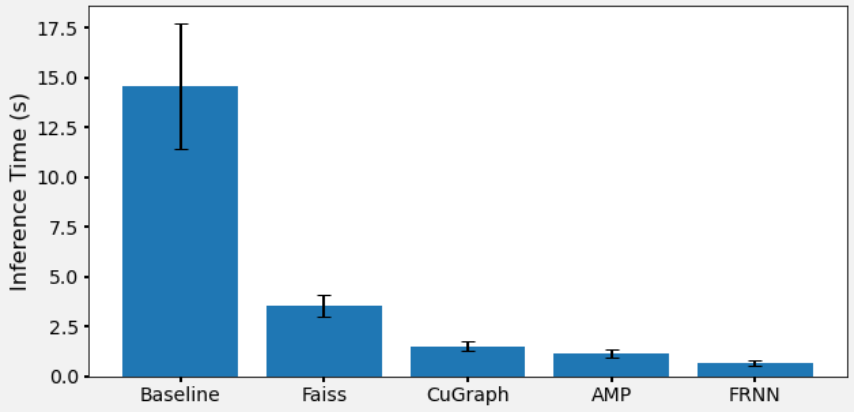
Alina Lazar on behalf of the Exa.TrkX Collaboration
alazar@ysu.edu
Youngstown State University

Accelerating the Exa.TrkX Inference on GPU

	Baseline Imp. (s)	Faiss	cuGraph	AMP	FRNN
Data Loading	0.0022 ± 0.0003	0.0021 ± 0.0003	0.0023 ± 0.0003	0.0022 ± 0.0003	0.0022 ± 0.0003
Embedding	0.02 ± 0.003	0.02 ± 0.002	0.02 ± 0.002	0.0067 ± 0.0007	0.0067 ± 0.0007
Build Edge	11.52 ± 2.64	0.54 ± 0.07	0.53 ± 0.07	0.53 ± 0.07	0.04 ± 0.01
Filtering	0.67 ± 0.15	0.67 ± 0.15	0.67 ± 0.15	0.37 ± 0.08	0.37 ± 0.08
GNN	0.17 ± 0.03	0.17 ± 0.03	0.17 ± 0.03	0.17 ± 0.03	0.17 ± 0.03
Labeling	2.16 ± 0.3	2.14 ± 0.3	0.11 ± 0.01	0.09 ± 0.008	0.09 ± 0.008
Total Time	14.57 ± 3.14	3.56 ± 0.55	1.53 ± 0.26	1.17 ± 0.18	0.7 ± 0.13

1. Baseline radius_graph → faiss KNN
2. DBSCAN clustering → cuGraph connected components
3. Full precision → mixed precision
4. Faiss knn → FRNN radius graph

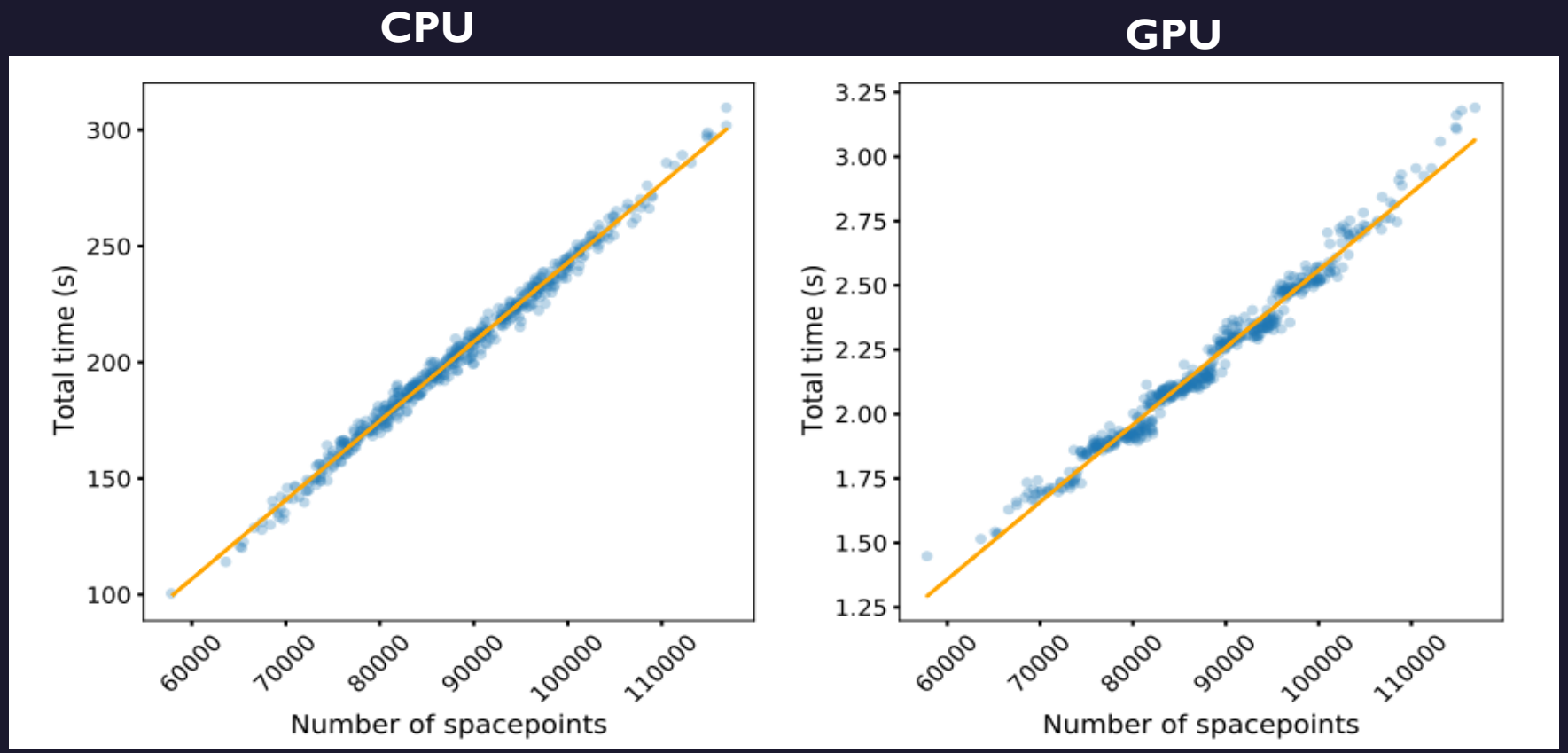
FRNN + mixed precision + cuGraph: 0.7 ± 0.13 sec →



arXiv preprint https://arxiv.org/pdf/2202.06929.pdf

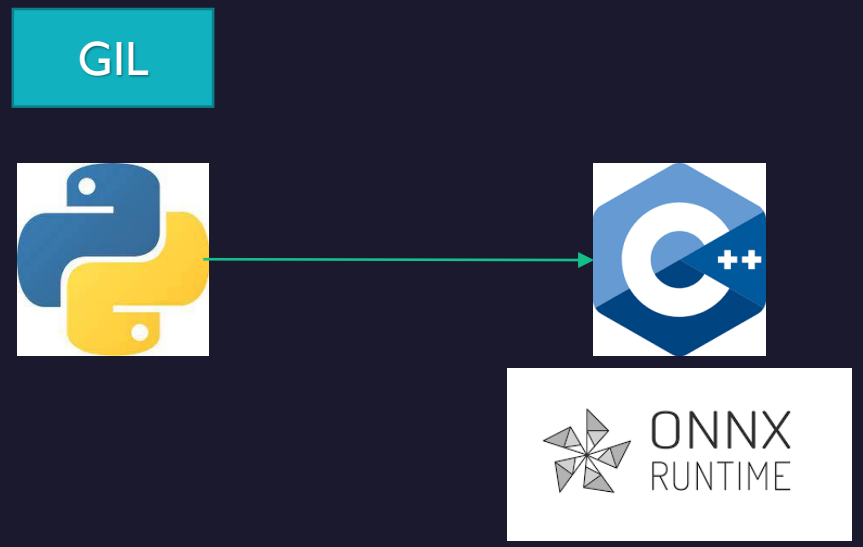
Exa.TrkX Track Reconstruction - Scaling

Computing performance scales linearly with the number of input space points.



Ju, X., Murnane, D., Calafiura, P., Choma, N., Conlon, S., Farrell, S., Xu, Y., Spiropulu, M., Vlimant, J.R., Aurisano, A. and Hewes, J., 2021. Performance of a geometric deep learning pipeline for HL-LHC particle tracking. *The European Physical Journal C*, *81*(10), pp.1-14. https://link.springer.com/article/10.1140/epjc/s10052-021-09675-8

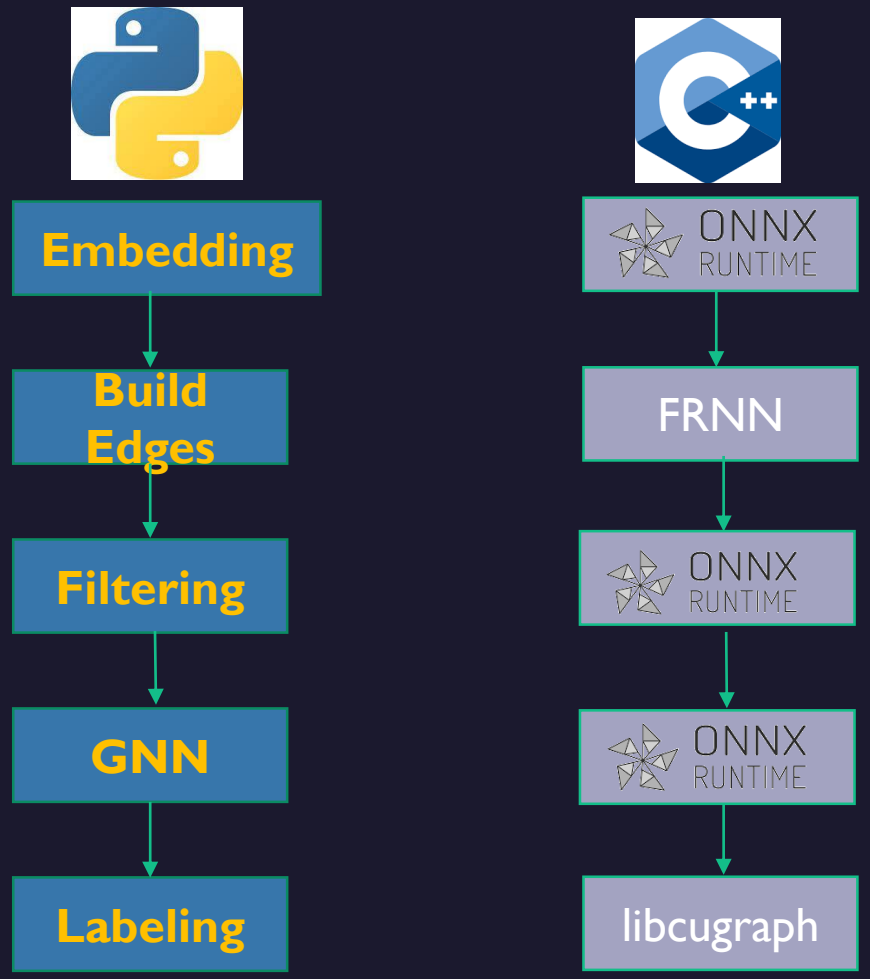
Towards Particle Tracking in Production



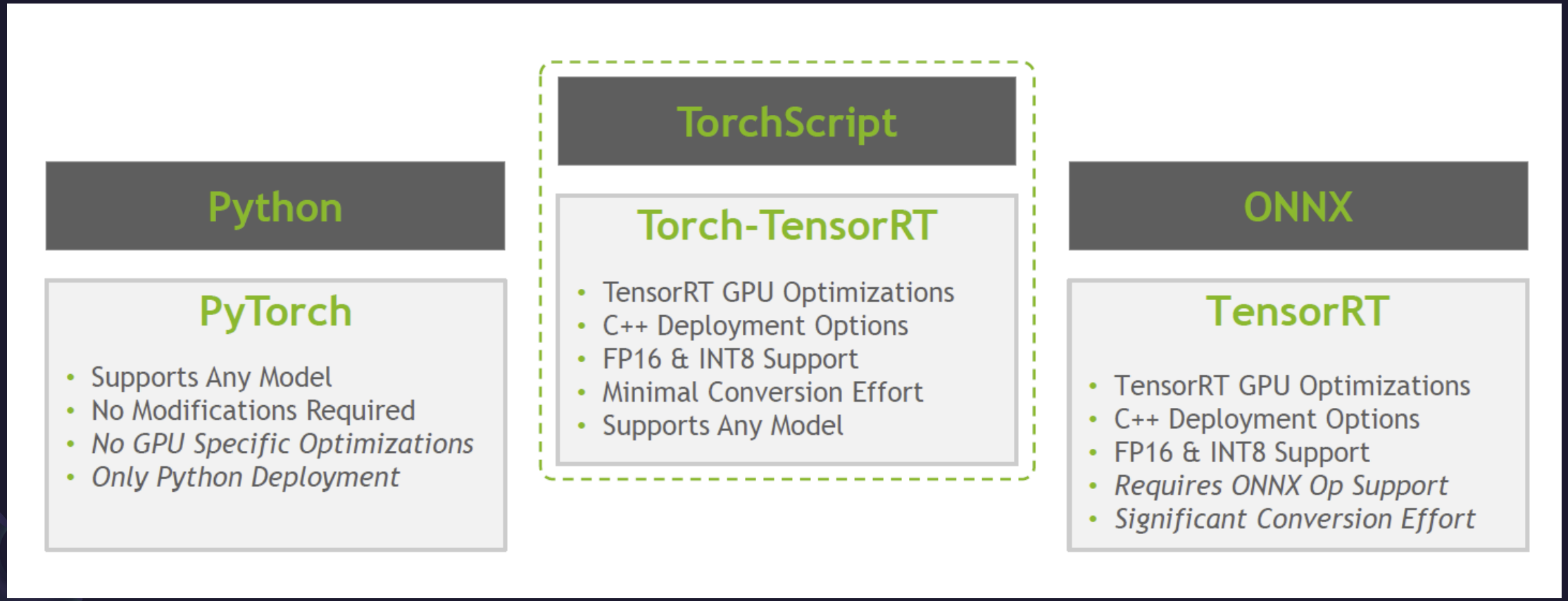
- We need a mechanism to integrate the **Exa.TrkX pipeline** with C++-based event reconstruction workflows (ACTS).
- Deep learning inference runs predominantly on the **GPUs**.
- Python's threading model is limited by the Global Interpreter Lock (**GIL**), slowing down throughput.
- By converting the pipeline to **C++**, we can overcome Python threading drawbacks.

Python to C++

- Build Edges uses FRNN written in CUDA and libtorch
- Track Labeling uses cuGraph from RapidsAI
- Convert embedding, filtering, and GNN to ONNX models; Use OnnxRuntime to run them
- **Problem:** The physics performance of GNN ONNX is compromised because the **scatter_add** operator does not work correctly
- **Solution:** Convert embedding, filtering, and GNN to TorchScript models



Torch-TensorRT



GNN Inference Accelerators

- PyTorch models can be ran with AMP (automated mixed precision)
- Convert the GNN PyTorch Geometric models to TorchScript (Jit)
- Convert models (TensorFlow and PyTorch) to ONNX and run them with half-precision
- **scatter_add – not implemented**

GNN Applications Real-time Processing

Low

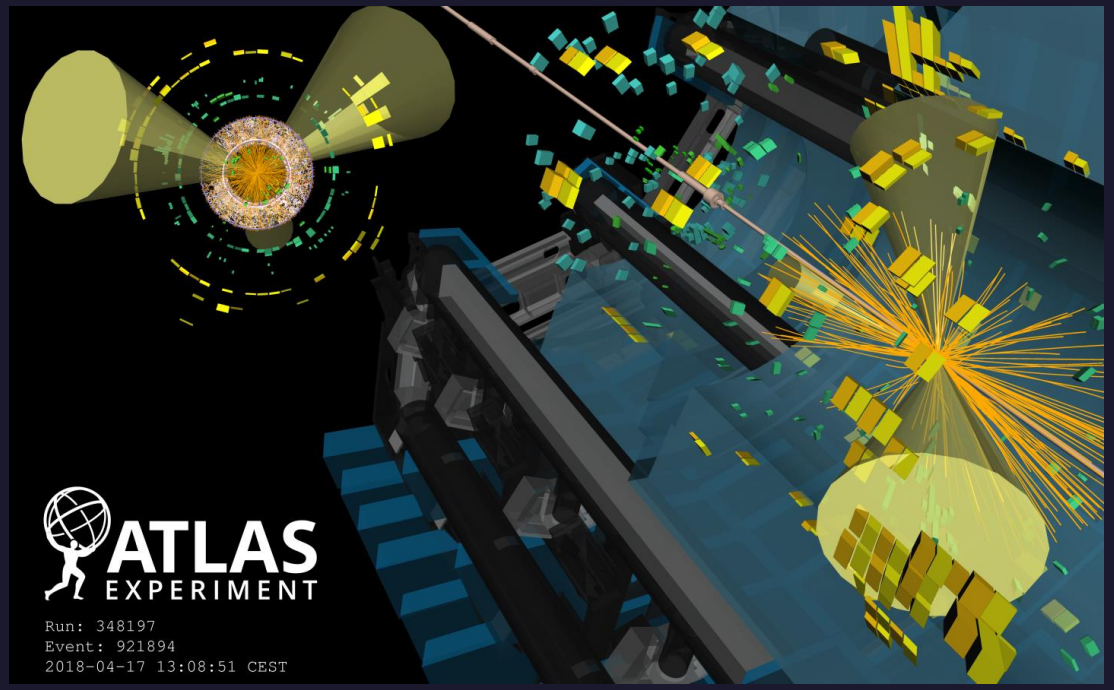
- Code analysis
- Automated HW/SW co-design
- Scene graph understanding
- Smart EDA tools

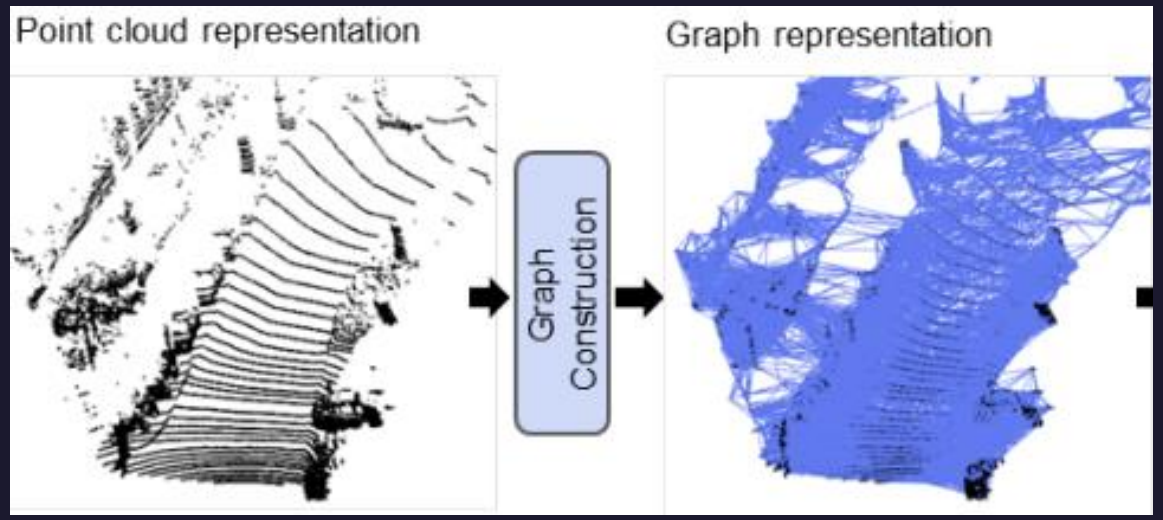
Medium

- Transportation and traffic forecasting
- Social network analysis
- Recommender system
- Molecule generation and drug discovery
- Health records modeling

High

- LiDAR and point cloud data for autonomous driving
- High energy physics
- Network intrusion detection





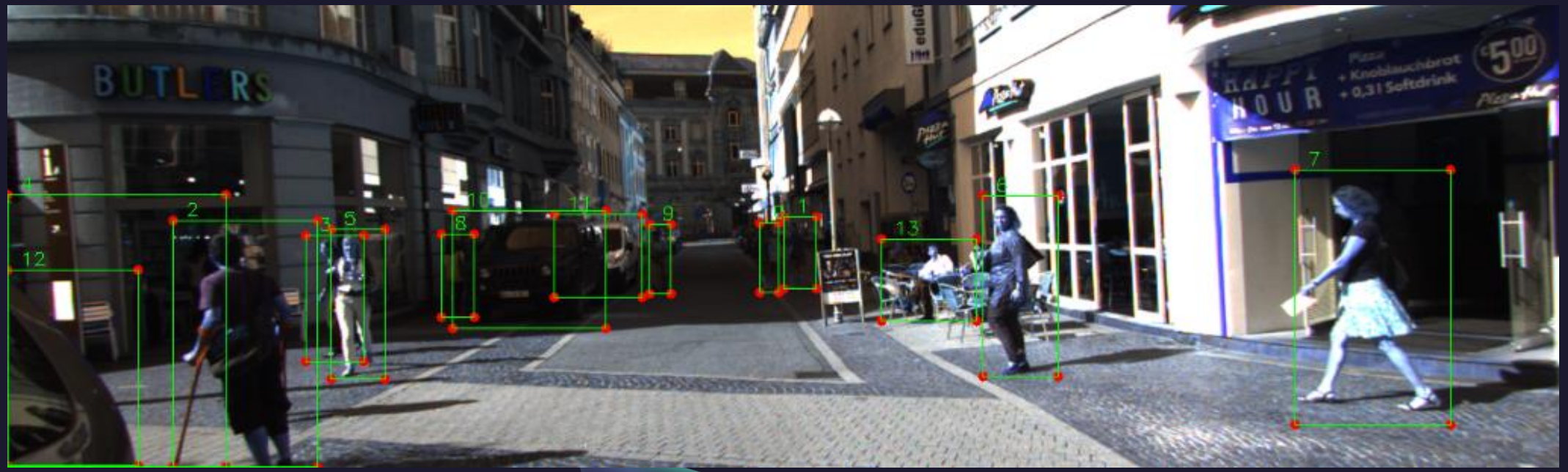
GNN Applications Real-time Processing

High

- LIDAR and point cloud data for autonomous driving
- High energy physics
- Network intrusion detection

3D Object Detection

Detection methods	Modality	Server GPU speed (ms)
F-PointNet [46]	R+L[a]	170
AVOD-FPN [28]		100
UberATG-MMF [33]		80
Fast Point R-CNN [9]	L	65
STD [60]		80
SECOND [59]		50
Point-GNN [53]		643
Ours		**18**



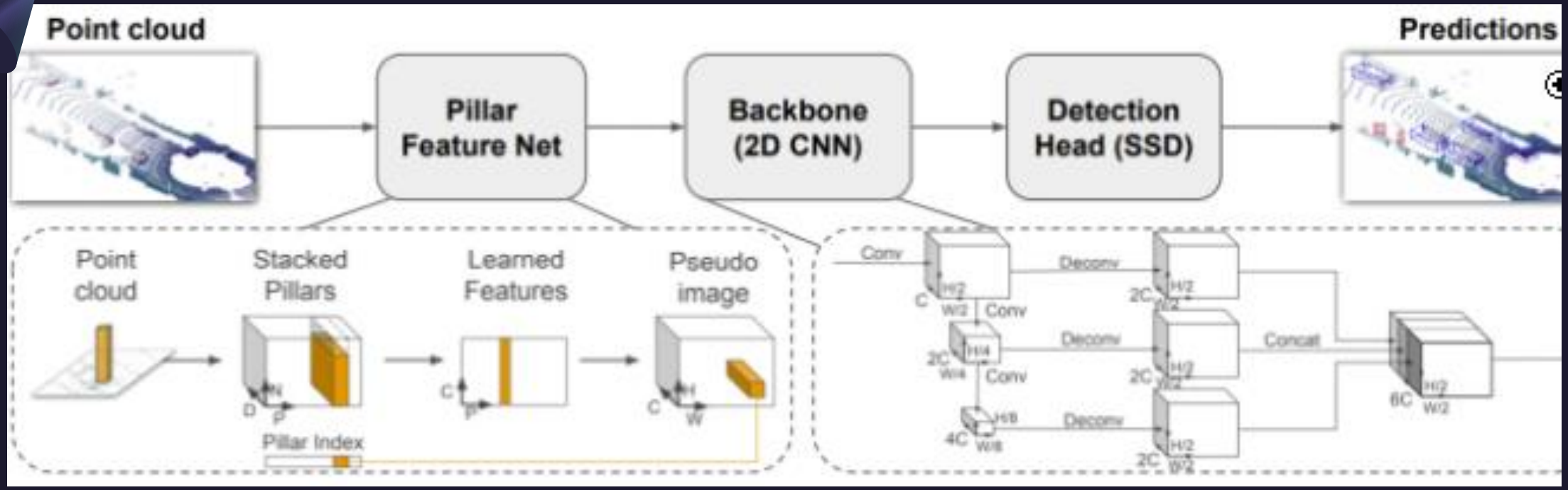
Zhao, P., Niu, W., Yuan, G., Cai, Y., Sung, H. H., Liu, S., ... & Lin, X. (2020). Achieving real-time lidar 3d object detection on a mobile device. arXiv preprint arXiv:2012.13801.

3D Object Detection GNN Inference

- GNN-based architecture achieves the best accuracy
- The GNN inference takes between 8 and 36 times longer.

Detection methods	Modality	Server GPU speed (ms)	Car 3D detection			Car BEV detection		
			Easy	Moderate	Hard	Easy	Moderate	Hard
F-PointNet [46]	R+L[a]	170	82.13	69.22	60.78	90.58	84.73	75.12
AVOD-FPN [28]		100	82.77	71.94	66.31	90.64	84.37	80.04.
UberATG-MMF [33]		80	86.81	76.75	68.41	89.49	87.47	79.10
Fast Point R-CNN [9]	L	65	85.39	77.46	70.21	89.97	87.08	80.40
STD [60]		80	86.61	77.63	**76.06**	89.66	87.76	**86.89**
SECOND [59]		50	83.34	72.55	65.82	89.39	83.77	78.59
Point-GNN [53]		643	**87.25**	**78.34**	72.29	**92.04**	**88.20**	81.97
Ours		**18**	85.20	75.57	68.37	90.02	86.79	80.80

Zhao, P., Niu, W., Yuan, G., Cai, Y., Sung, H. H., Liu, S., ... & Lin, X. (2020). Achieving real-time lidar 3d object detection on a mobile device. arXiv preprint arXiv:2012.13801.



PointPillars Architecture

- First part is an MLP
- Second pard is a 2D CNN
- They are connected with a scatter_add

Get Ideas from 3D Object Detection

A case study of the deployment of PointPillars for LiDAR-based **3D object detection**.

They evaluate the runtime of the deployed DNN using two different libraries, **TensorRT (ONNX)** and **TorchScript (half precision)**.

They observe slight advantages of TensorRT for convolutional layers and TorchScript for fully connected layers.

The scatter operation is implemented with C++ functions using the CUDA library.

Stäcker, L., Fei, J., Heidenreich, P., Bonarens, F., Rambach, J., Stricker, D., & Stiller, C. (2021). Deployment of Deep Neural Networks for Object Detection on Edge AI Devices with Runtime Optimization. In Proceedings of the IEEE/CVF International Conference on Computer Vision (pp. 1015-1022).

ONNX and TensorRT scatter ops

- Both ONNX and TensorRT implementations of the **ScatterElements** and **ScatterND** don't work.
- ONNX opset 16 has the reduction implemented, but it doesn't work in ONNXRuntime.
- For Object Detection people use Nvidia's **graphsurgeon** to remove the **scatter_add** op out and replace it with the ONNX implementation or the C++ CUDA implementation.

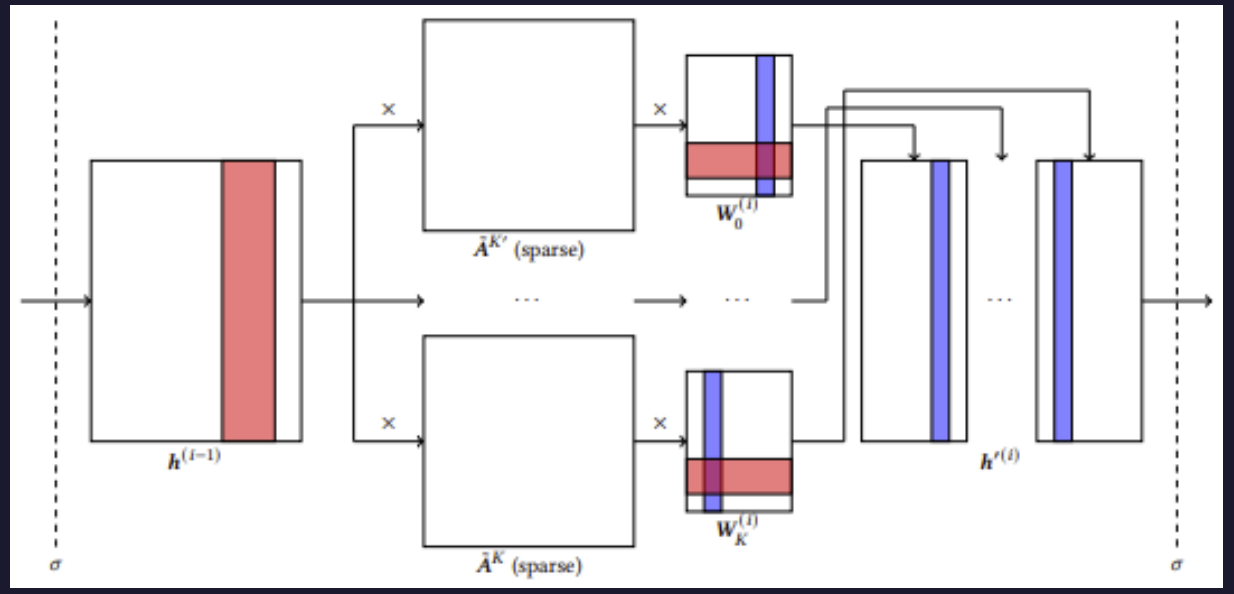
Other Inference Accelerators

- OLIve from ONNX - https://github.com/microsoft/OLive - is a Python package that automates the process of accelerating models with ONNX Runtime (ORT).
- Glow from PyTorch - https://github.com/pytorch/glow is a machine learning compiler and execution engine for hardware accelerators. It is designed to be used as a backend for high-level machine learning frameworks.

New Developments

- GNN Inference using Channel Pruning
- GenGNN Framework

GNN Inference using Channel Pruning



- This pruning framework uses a novel LASSO regression formulation for GNNs to identify feature dimensions (channels) that have high influence on the output activation.
- To further reduce the inference complexity, they effectively store and reuse hidden features of visited nodes, which significantly reduces the number of supporting nodes needed.
- They achieved an average of 3.27× speedup on GPU with little drop in performance.

Zhou, Hongkuan, Ajitesh Srivastava, Hanqing Zeng, Rajgopal Kannan, and Viktor Prasanna. "Accelerating Large Scale Real-Time GNN Inference using Channel Pruning." arXiv preprint arXiv:2105.04528 (2021).

GenGNN Framework

EXISTING WORK

- Most focus on Graph Convolution Network (GCN): A limited type
- Heavy pre-processing: Not suitable for real-time
- Most on application-specific integrated circuits (ASIC) via simulation: not end-to-end, far from practical

GENGNN FRAMEWORK

- A general framework for message passing
- A library for model specific components

GENGNN ADVANTAGES

- Support a wide range of GNNs: Generic
- No pre-processing required: Real-time oriented
- End-to-end open-source **FPGA** implementation

Abi-Karam, S., He, Y., Sarkar, R., Sathidevi, L., Qiao, Z., & Hao, C. (2022). GenGNN: A Generic FPGA Framework for Graph Neural Network Acceleration. arXiv preprint arXiv:2201.08475.).

Summary

- Research is focused on training and not inference
- Most accelerators work well for CNN and GCN
- There are some new approaches for speeding-up GNN inference

Thank You!