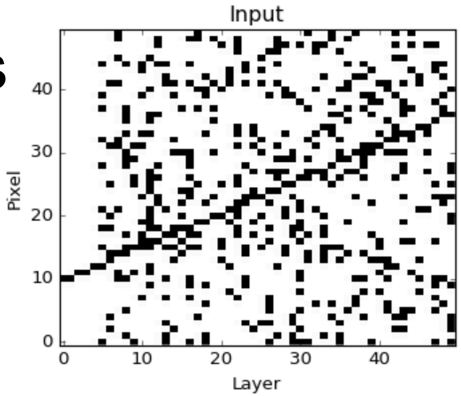
Outlook on GNNs for HEP tracking

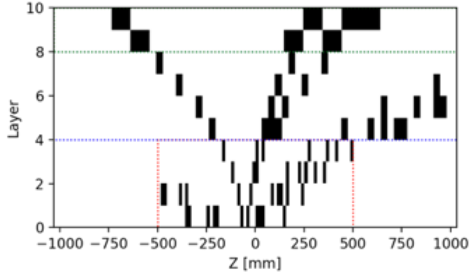
Steve Farrell, Xiangyang Ju
NERSC, LBNL

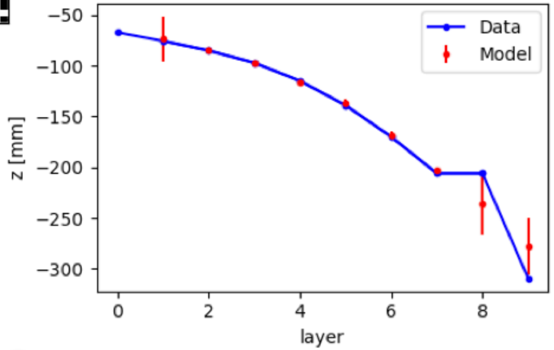
Exa.TrX kickoff, 2019-06-04

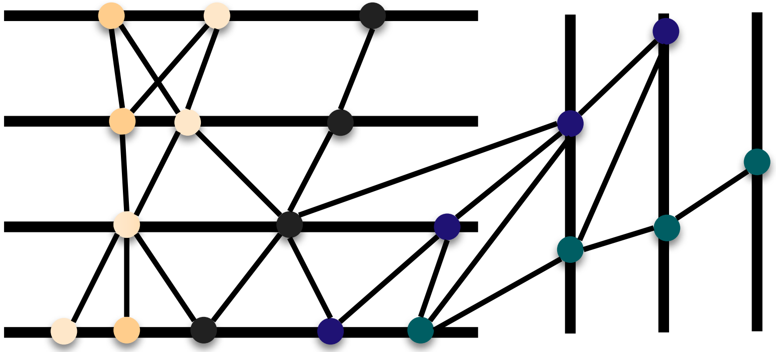
Thoughts on representations

- **We tried a bunch of things in HEP.TrkX**
  - Images of simple toy data
  - Detector images
  - Hit sequences
  - Hit graphs
- **Regardless of the specific model architecture or application, I believe we've found the right *representation* of HEP tracking data.**



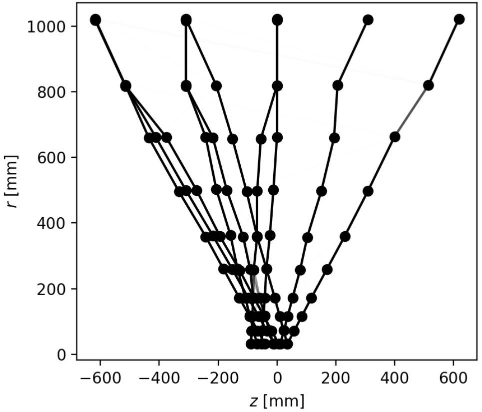


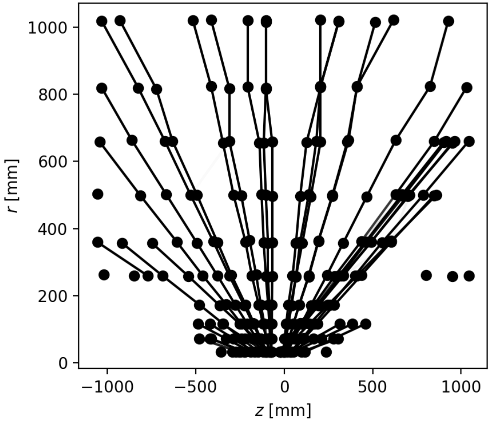


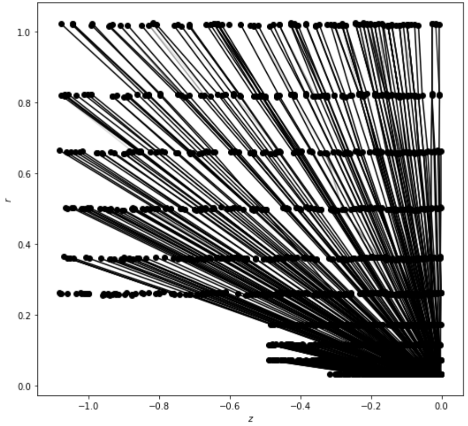


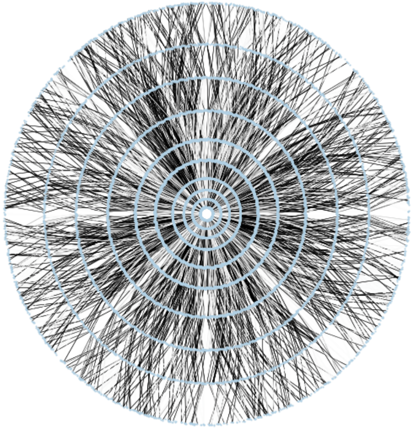
Increasing complexity

- The basic GNN approach seems to be holding up with fairly major upgrades in data complexity
- Steps still in progress
    - Whole detector, including endcaps
    - Robust handling of missing hits, double hits, shared hits









What's missing?

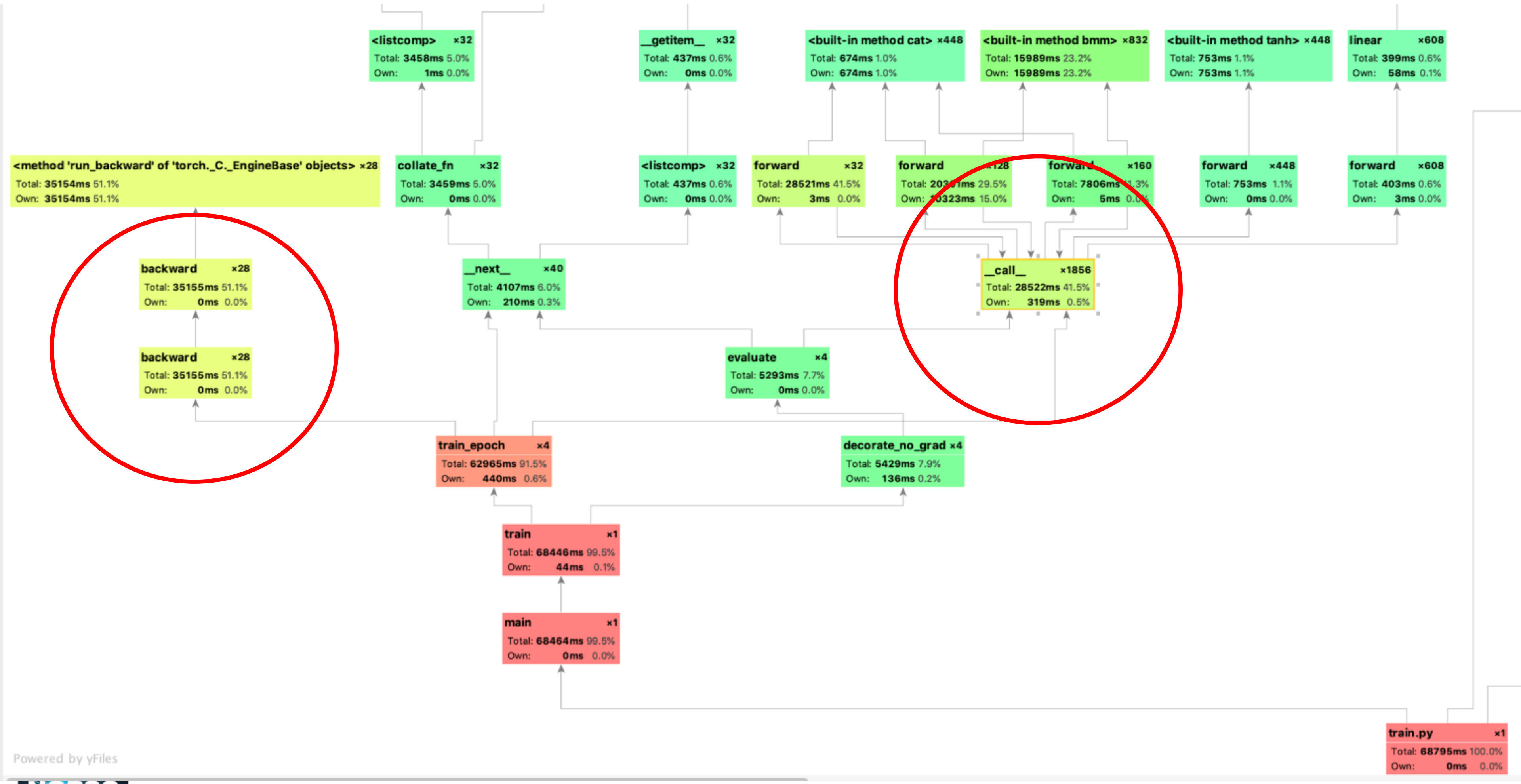
- **Post-processing**
  - Need a robust method to build final candidate tracks
    - Able to stitch together pieces of tracks
- **Baseline methods results**
  - E.g., Kalman Filter tracking on the TrackML dataset

HPC scaling work with Cray BDC

- **Through the Cray Big Data Center collaboration, we're engaging with folks from Cray and LBNL's CRD to push on HPC scaling of GNNs (for tracking)**
  - Strong interest in scaling GNNs in PyTorch
- **Computational challenges**
  - Graphs with sparse connectivity => need sparse op support
  - Variable sized graphs => need to handle load imbalance at scale
  - Large scale training of GNNs => not much experience/intuition

Single node optimizations

- Saliya Ekanayake started with my original dense representation model (dense adjacency matrices)
  - Did some profiling
  - Implemented faithful translation to sparse representations and using some functionality from the pytorch-geometric package
  - Huge cost from the dense representation
- Later, I re-implemented the original model in native pytorch-geometric representation and operations (scatter_add)
  - Saliya is comparing these



<listcomp> ×32
Total: 3458ms 5.0%
Own: 1ms 0.0%
__getitem__ ×32
Total: 437ms 0.6%
Own: 0ms 0.0%
<built-in method cat> ×448
Total: 674ms 1.0%
Own: 674ms 1.0%
<built-in method bmm> ×832
Total: 15989ms 23.2%
Own: 15989ms 23.2%
<built-in method tanh> ×448
Total: 753ms 1.1%
Own: 753ms 1.1%
linear ×608
Total: 399ms 0.6%
Own: 58ms 0.1%
<method 'run_backward' of 'torch._C._EngineBase' objects> ×28
Total: 35154ms 51.1%
Own: 35154ms 51.1%
collate_fn ×32
Total: 3459ms 5.0%
Own: 0ms 0.0%
<listcomp> ×32
Total: 437ms 0.6%
Own: 0ms 0.0%
forward ×32
Total: 28521ms 41.5%
Own: 3ms 0.0%
forward
29.5%
15.0%
forward ×160
Total: 7806ms
Own: 5ms
forward ×448
Total: 753ms 1.1%
Own: 0ms 0.0%
forward ×608
Total: 403ms 0.6%
Own: 3ms 0.0%
backward ×28
Total: 35155ms 51.1%
Own: 0ms 0.0%
__next__ ×40
Total: 4107ms 6.0%
Own: 210ms 0.3%
__call__ ×1856
Total: 28522ms 41.5%
Own: 319ms 0.5%
backward ×28
Total: 35155ms 51.1%
Own: 0ms 0.0%
evaluate ×4
Total: 5293ms 7.7%
Own: 0ms 0.0%
train_epoch ×4
Total: 62965ms 91.5%
Own: 440ms 0.6%
decorate_no_grad ×4
Total: 5429ms 7.9%
Own: 136ms 0.2%
train ×1
Total: 68446ms 99.5%
Own: 44ms 0.1%
main ×1
Total: 68464ms 99.5%
Own: 0ms 0.0%
train.py ×1
Total: 68795ms 100.0%
Own: 0ms 0.0%
Powered by yFiles

forward() -- time in μs

```
Total time: 28.7951 s
File: /Users/esaliya/sali/git/github/esaliya/python/heptrkx-gnn-tracking/models/gnn.py
Function: forward at line 78

Line #      Hits         Time  Per Hit   % Time  Line Contents
==============================================================
    78                                           @profile
    79                                           def forward(self, inputs):
    80                                               """Apply forward pass of the model"""
    81        32        480.0     15.0      0.0      X, Ri, Ro = inputs
    82                                               # Apply input network to get hidden representation
    83        32      83301.0   2603.2      0.3      H = self.input_network(X)
    84                                               # Shortcut connect the inputs onto the hidden representation
    85        32      13714.0    428.6      0.0      H = torch.cat([H, X], dim=-1)
    86                                               # Loop over iterations of edge and node networks
    87       160        268.0      1.7      0.0      for i in range(self.n_iters):
    88                                                   # Apply edge network
    89       128    6265327.0  48947.9     21.8          e = self.edge_network(H, Ri, Ro)
    90                                                   # Apply node network
    91       128   20835340.0 162776.1     72.4          H = self.node_network(H, e, Ri, Ro)
    92                                                   # Shortcut connect the ...
    93       128      31330.0    244.8      0.1          H = torch.cat([H, X], dim=-1)
    94                                               # Apply final edge network
    95        32    1565309.0  48915.9      5.4      return self.edge_network(H, Ri, Ro)
```

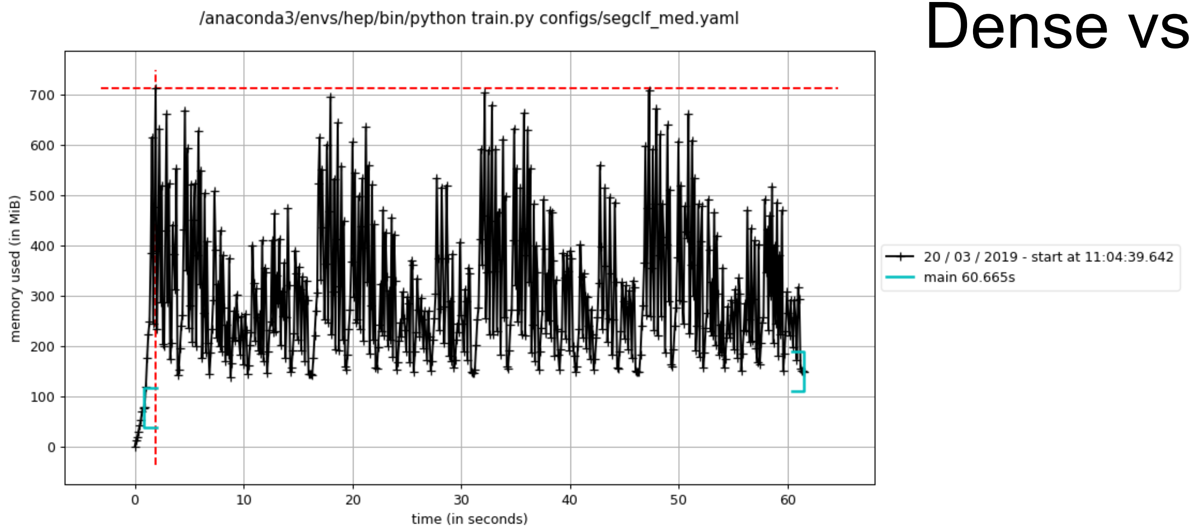
Node Network forward() -- time in μs

```
Total time: 20.5506 s
File: /Users/esaliya/sali/git/github/esaliya/python/heptrkx-gnn-tracking/models/gnn.py
Function: forward at line 49

Line #      Hits         Time  Per Hit   % Time  Line Contents
==============================================================
    49                                           @profile
    50                                           def forward(self, X, e, Ri, Ro):
    51       128    2549745.0  19919.9     12.4      bo = torch.bmm(Ro.transpose(1, 2), X)
    52       128    2588509.0  20222.7     12.6      bi = torch.bmm(Ri.transpose(1, 2), X)
    53       128    5036099.0  39344.5     24.5      Rwo = Ro * e[:,None]
    54       128    5280113.0  41250.9     25.7      Rwi = Ri * e[:,None]
    55       128    2318774.0  18115.4     11.3      mi = torch.bmm(Rwi, bo)
    56       128    2370559.0  18520.0     11.5      mo = torch.bmm(Rwo, bi)
    57       128      89165.0    696.6      0.4      M = torch.cat([mi, mo, X], dim=2)
    58       128     317685.0   2481.9      1.5      return self.network(M)
```

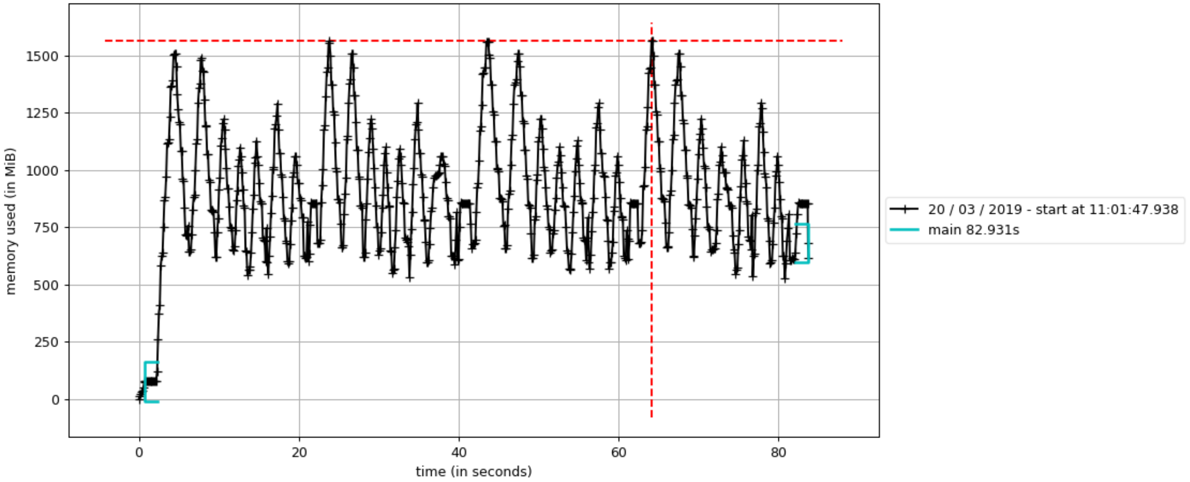
Edge Network forward() -- time in μs

```
Total time: 7.78296 s
File: /Users/esaliya/sali/git/github/esaliya/python/heptrkx-gnn-tracking/models/gnn.py
Function: forward at line 24

Line #      Hits         Time  Per Hit   % Time  Line Contents
==============================================================
    24                                           @profile
    25                                           def forward(self, X, Ri, Ro):
    26                                               # Select the features of the associated nodes
    27       160    3169923.0  19812.0     40.7      bo = torch.bmm(Ro.transpose(1, 2), X)
    28       160    3249926.0  20312.0     41.8      bi = torch.bmm(Ri.transpose(1, 2), X)
    29       160     540897.0   3380.6      6.9      B = torch.cat([bo, bi], dim=2)
    30                                               # Apply the network to each edge
    31       160     822219.0   5138.9     10.6      return self.network(B).squeeze(-1)
```

Dense vs. sparse memory



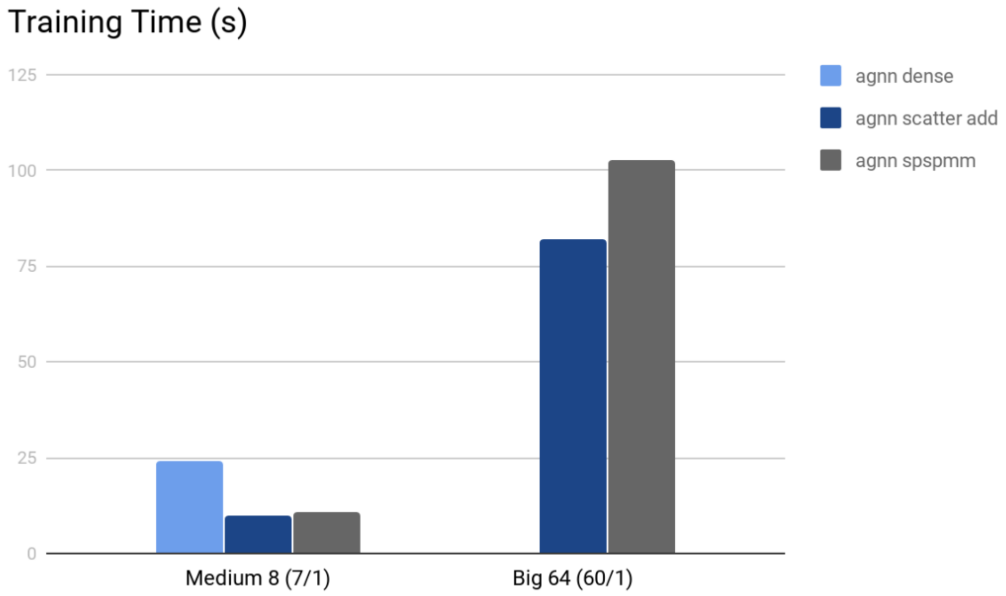
Sparse (150 - 700 MB)



Dense (600 - 1500 MB)

Training time

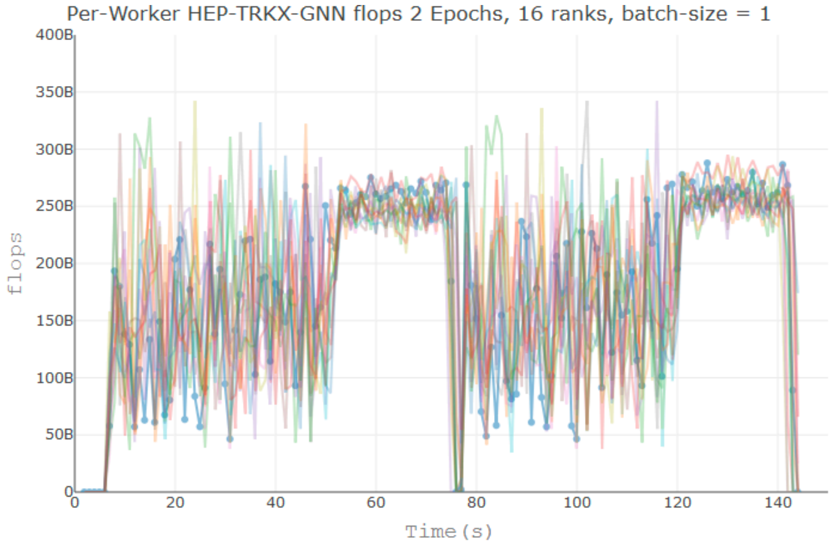
	Training Time (s)		
Dataset N (tr/val)	**agnn dense**	**agnn scatter add**	**agnn spspmm**
Med 8 (7/1)	24.0416	9.63175	10.6389
Big 64 (60/4)		81.9128	102.74



“Native” pytorch-geometric is fastest

Multi-node optimizations

- **Strong load imbalance in profiling =>**
- **Need some special handling of the data to balance the load**
  - E.g., grouping samples together on nodes to even it out
- **Other ongoing work**
  - Convergence at scale
  - I/O bottlenecks?



Work with Jacob Balma and Kristi Maschhoff from Cray

Summary

- We’ve made good progress, though there’s still some work to go
- External folks find our problem very interesting for a number of reasons
- Leveraging their expertise to optimize and scale these workflows should be beneficial