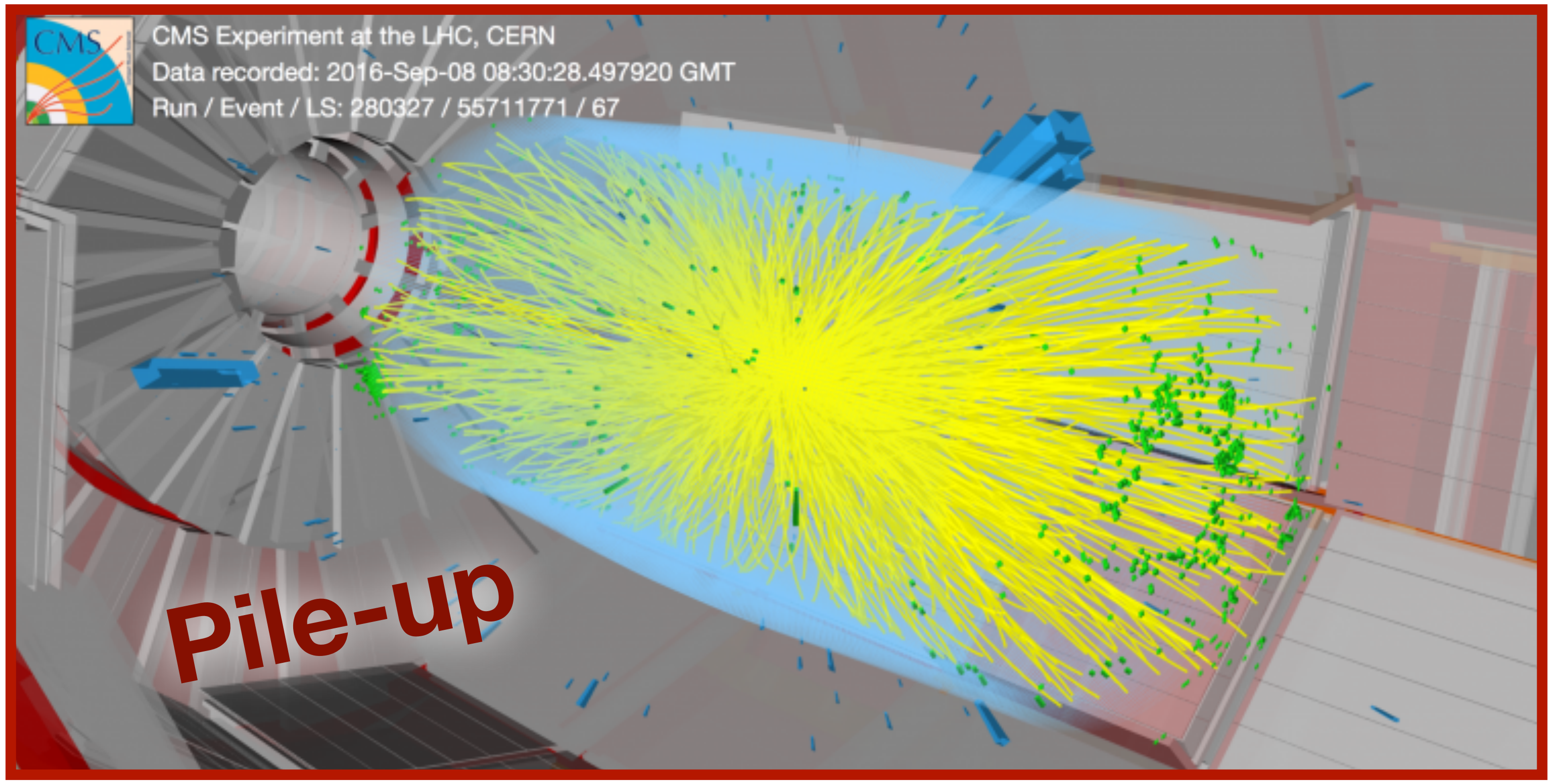
Graph Neural Network reconstruction in HGCAL

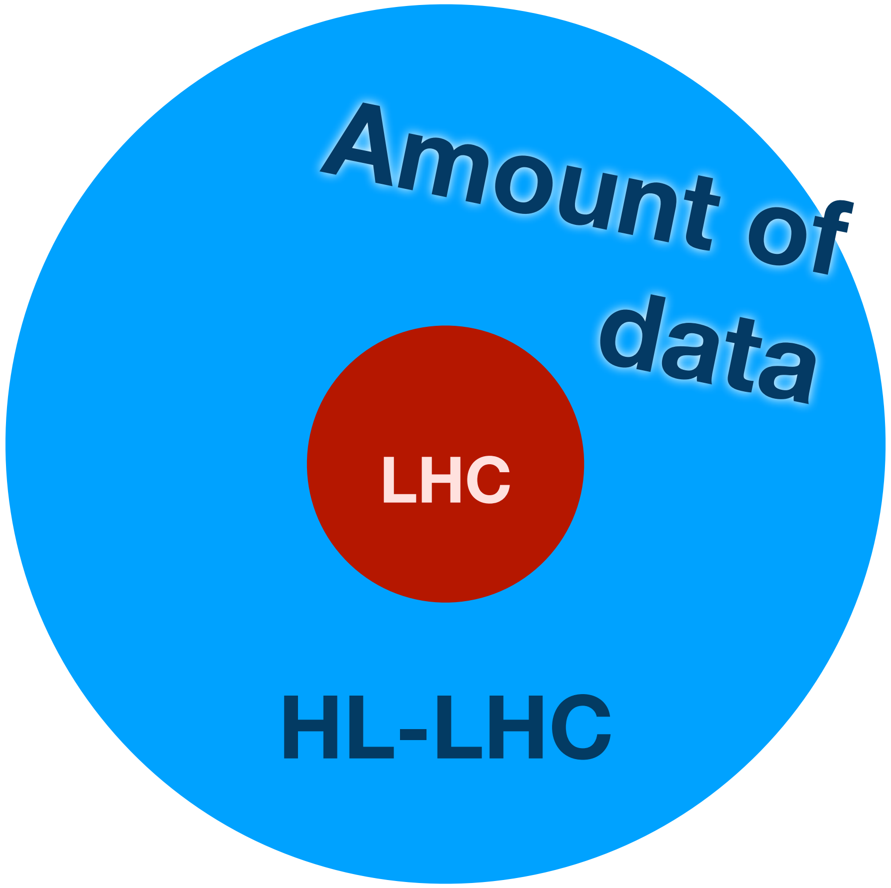
Thea Aarrestad, Sitong An, Gianluca Cerminara, Javier Duarte, Sergei Gleyzer, Dejan Golubovic, Shamik Gosh, Lindsey Gray, Phil Harris, Yutaro Iiyama, Jan Kieseler, **Thomas Klijnsma**, Kenneth Long, Jennifer Ngadiuba, Gerrit van Onsem, Joosep Pata, Kevin Pedro, Maurizio Pierini, Shah Rukh Qasim, Marcel Rieger, Sioni Summers, Mary Touranakou, Nhan Tran, Oleksander Viazlo, Kinga Wozniak

7 April 2020

Fermilab

Motivation

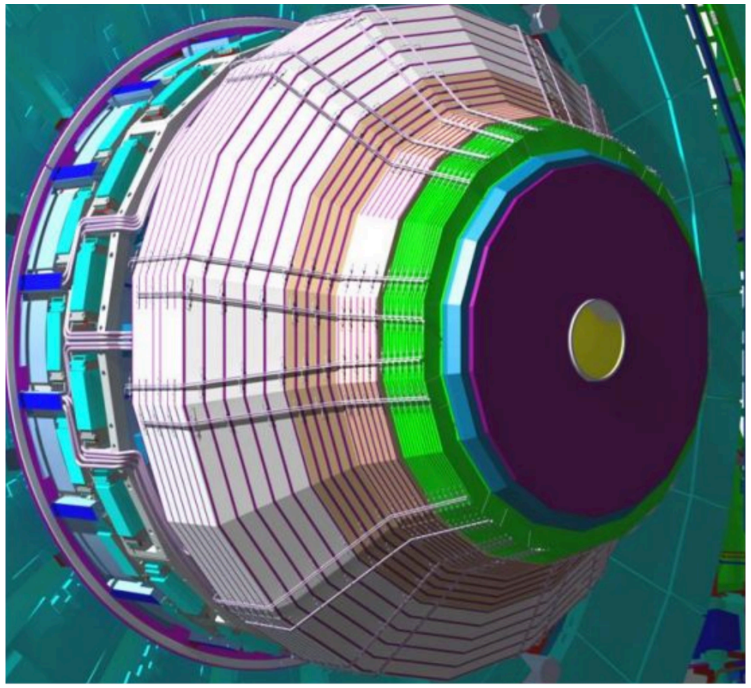


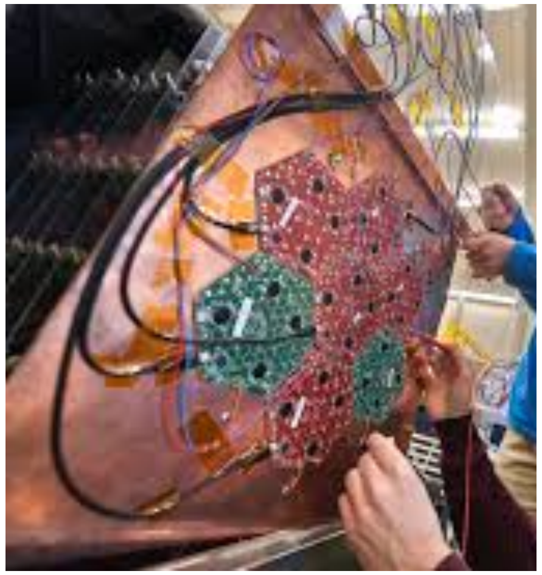


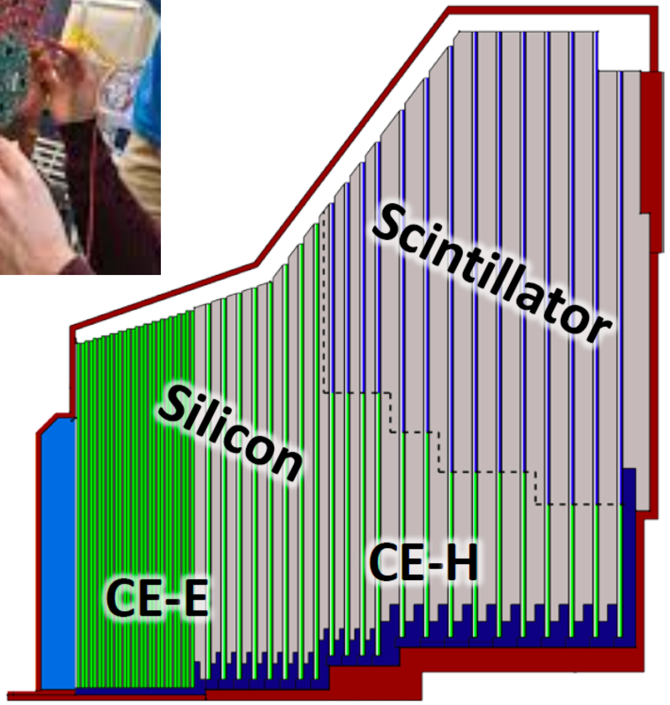
- Traditional algorithms scale **combinatorially**
  - **Explodes** going from 32 to **200** PU interactions
- A neural network could reconstruct in **constant time**

About HGCAL

- One event in HGCAL is a large **set of hits (x, y, z, E, t)** (i.e. feature space is 5D)
- Dimensionality + geometry not well suited for the CNNs in industry (rectangular, non-sparse, 3 colors)
    - Showers and tracks are pretty naturally represented by a graph -> **GNNs**





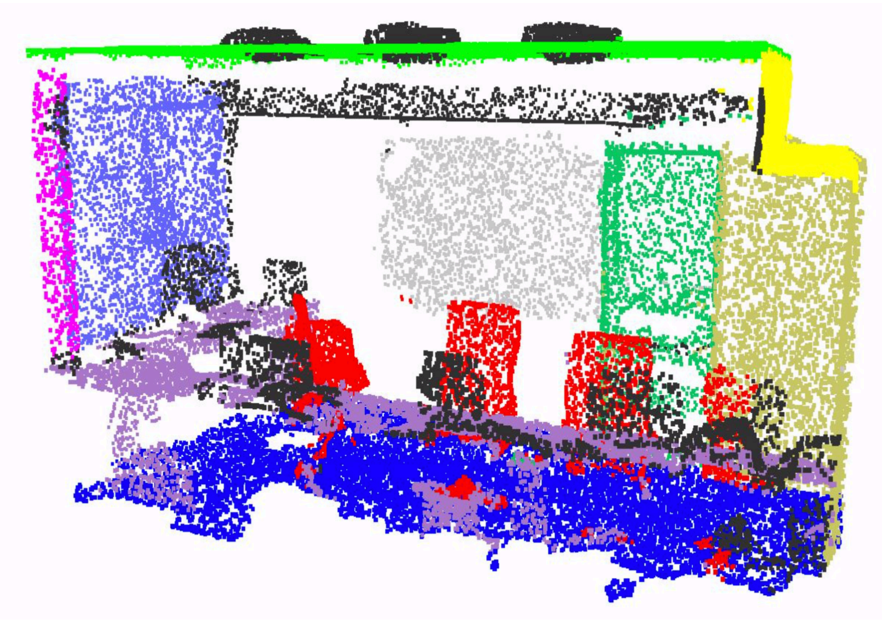


GNNs

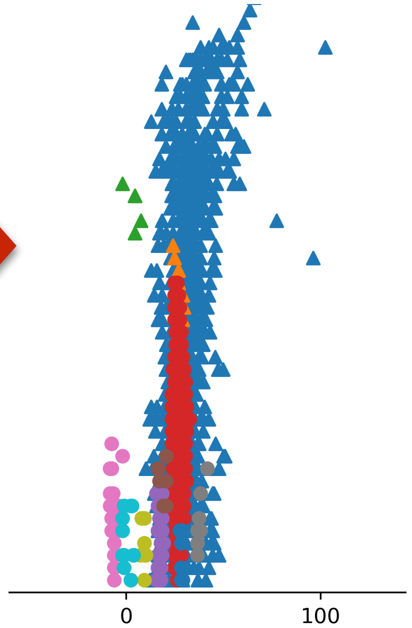
- **PointNet:** one of the pioneering deep neural network approach to work with point clouds

- Limited use of neighborhood information

Semantic segmentation



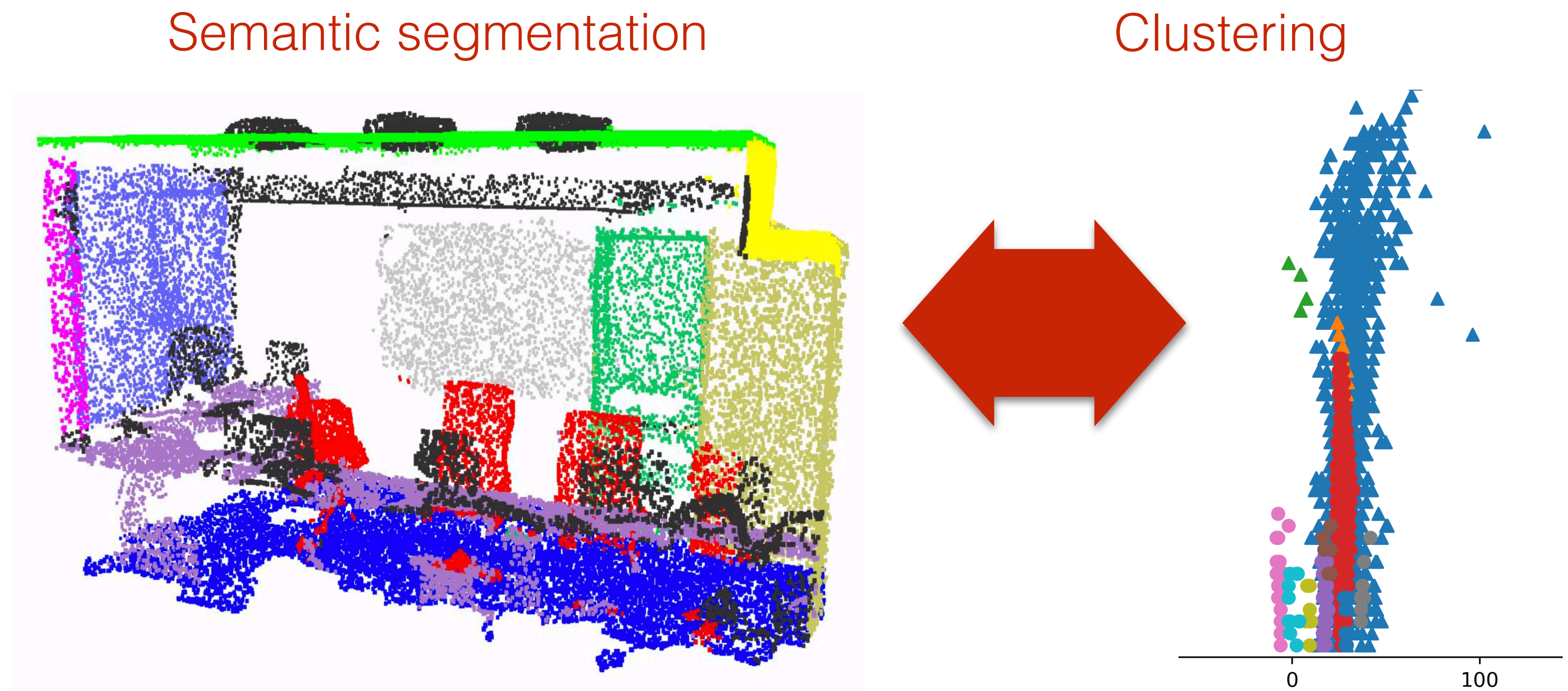
Clustering

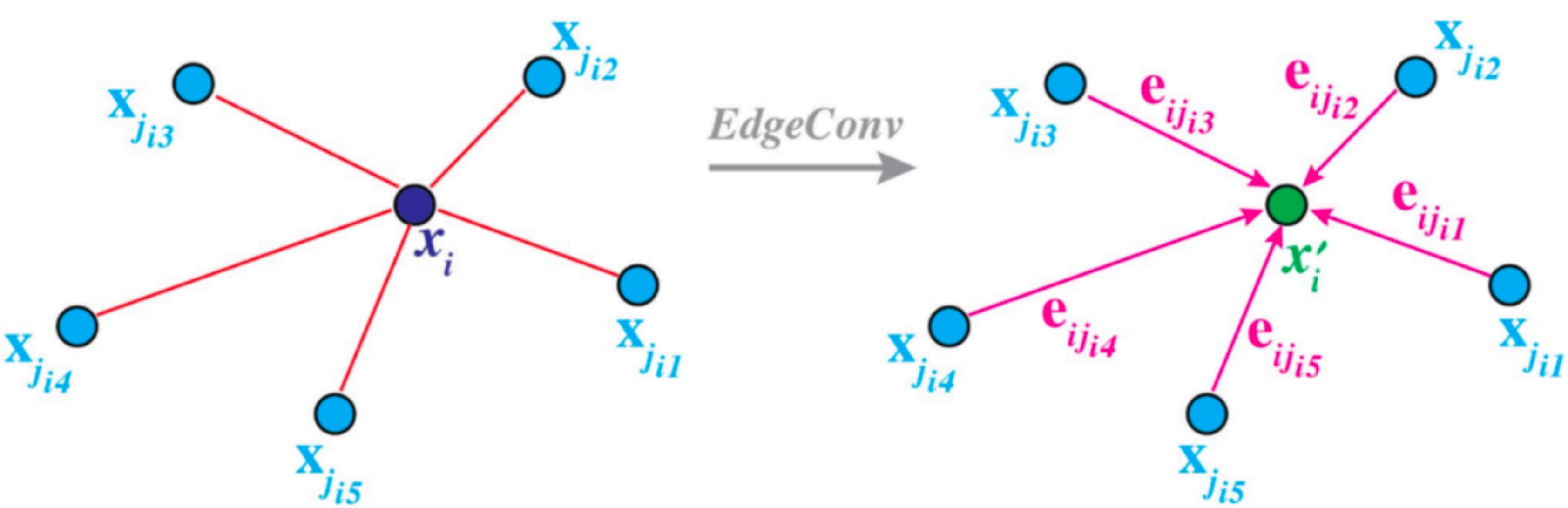


GNNs

- **PointNet:** one of the pioneering deep neural network approach to work with point clouds

- Limited use of neighborhood information



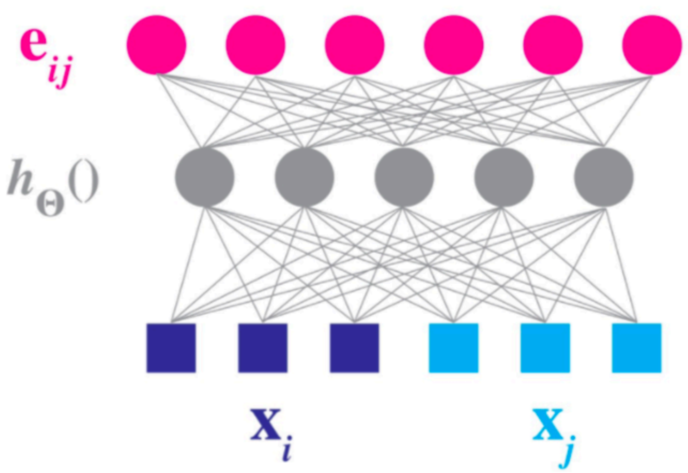


- **EdgeConv:** Update x_i → x_i' by using **edge features**

- i.e. learned features of the edges that connects x_i with its neighbors
- Still independent of ordering of points, but uses **local geometry**
- '**Convolutional**' as the operation is applied point by point to obtain **x**'

Dynamic Graph Convolutional NN

[1801.07829]



$$\mathbf{x}'_i = \underset{j:(i,j)\in\mathcal{E}}{\square} h_\Theta(\mathbf{x}_i, \mathbf{x}_j)$$

$h_\Theta(\mathbf{x}_i, \mathbf{x}_j) = h_\Theta(\mathbf{x}_i)$ No neighborhood info (only global)

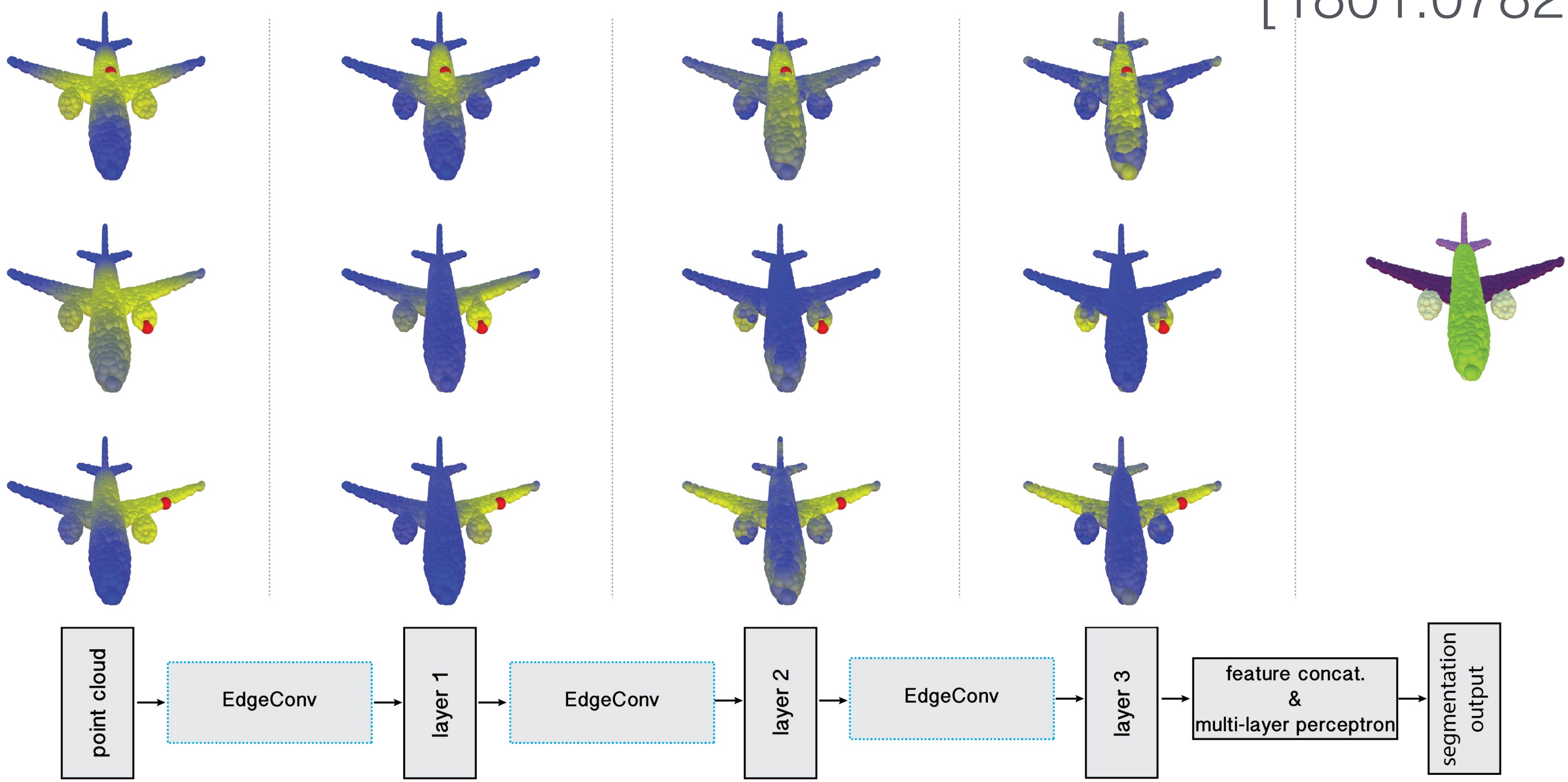
$h_\Theta(\mathbf{x}_i, \mathbf{x}_j) = h_\Theta(\mathbf{x}_j - \mathbf{x}_i)$ Only local information

$h_\Theta(\mathbf{x}_i, \mathbf{x}_j) = \bar{h}_\Theta(\mathbf{x}_i, \mathbf{x}_j - \mathbf{x}_i)$ Combination of both

- **Dynamic:** Redo kNN after every update
    - The connectivity matrix changes after every update

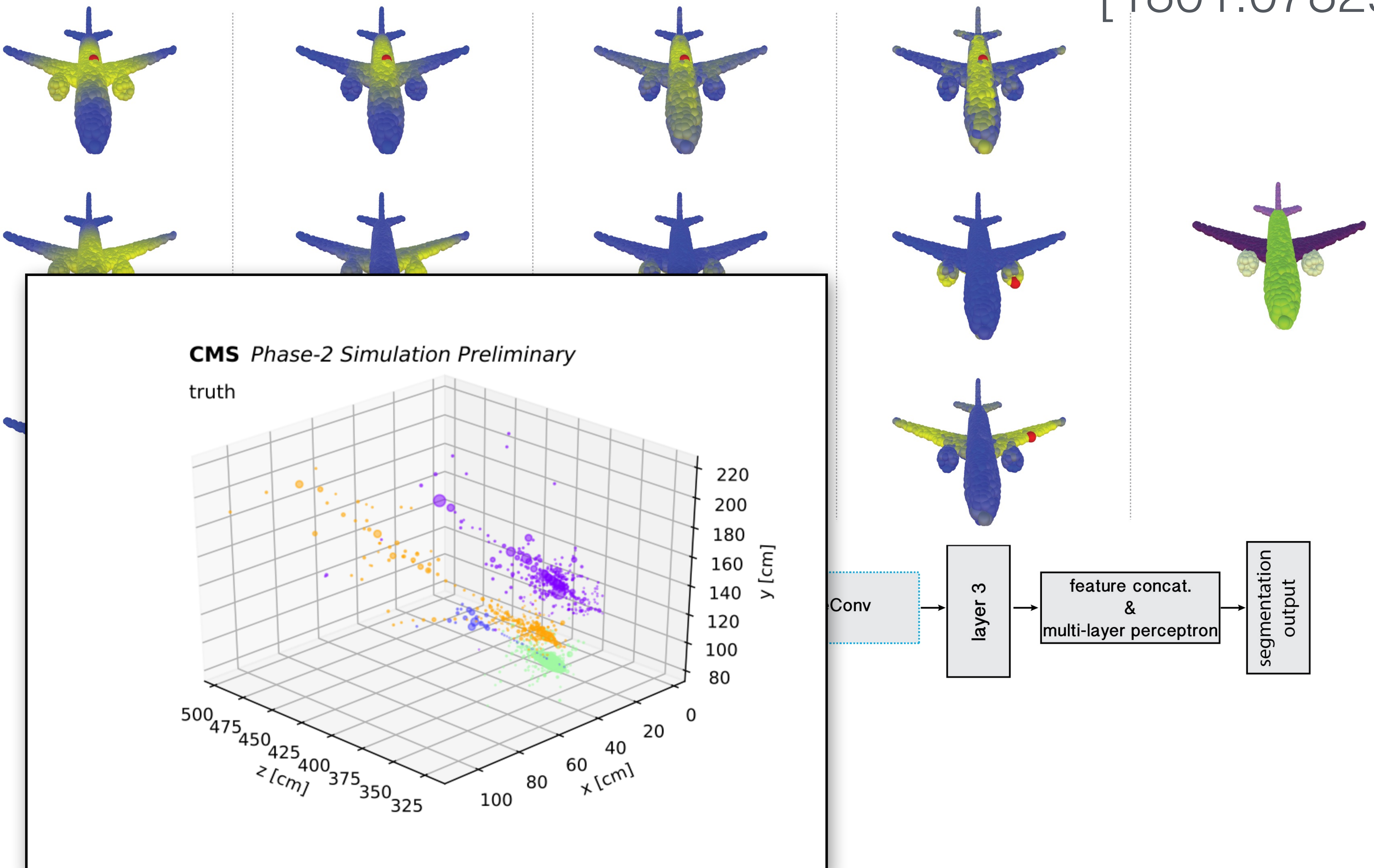
Dynamic Graph Convolutional NN

[1801.07829]

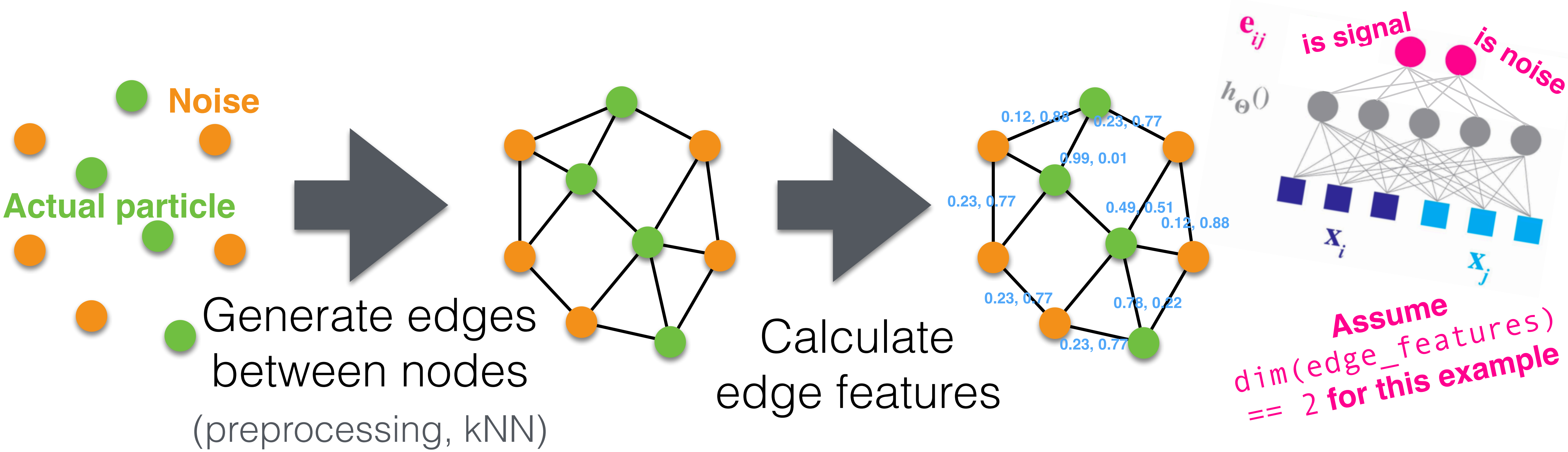


Dynamic Graph Convolutional NN

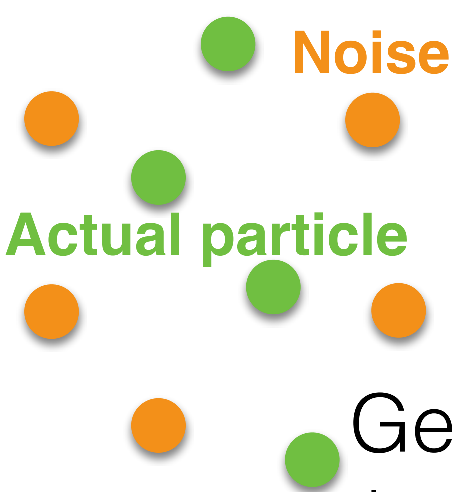
[1801.07829]



Graphical example

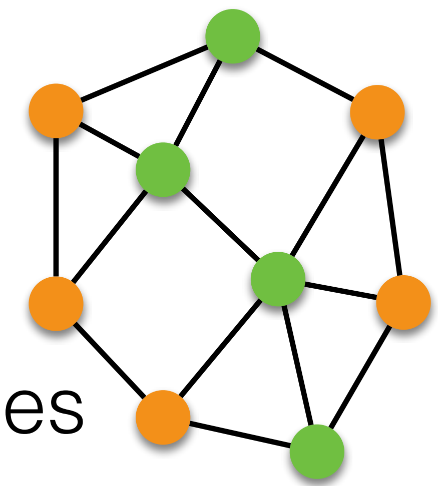


Graphical example

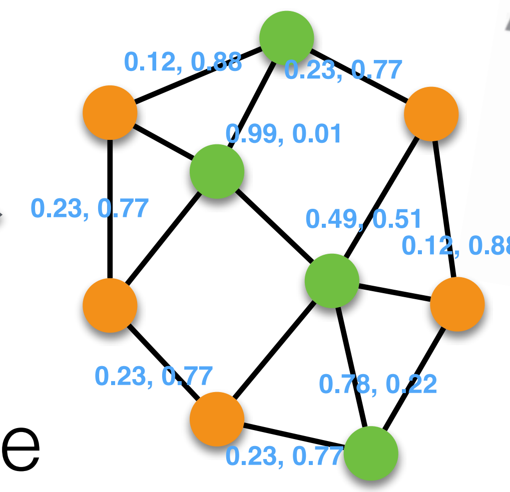


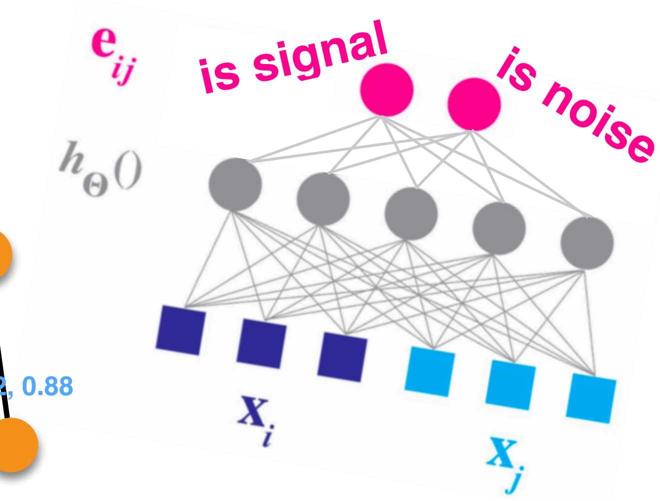
Generate edges between nodes

(preprocessing, kNN)

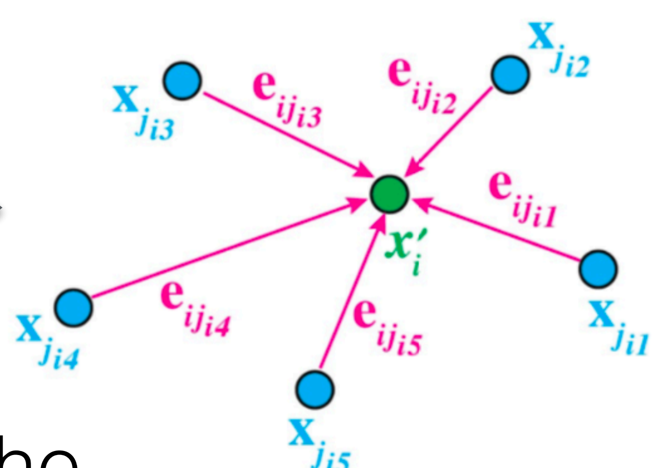


Calculate edge features





Assume `dim(edge_features) == 2` **for this example**

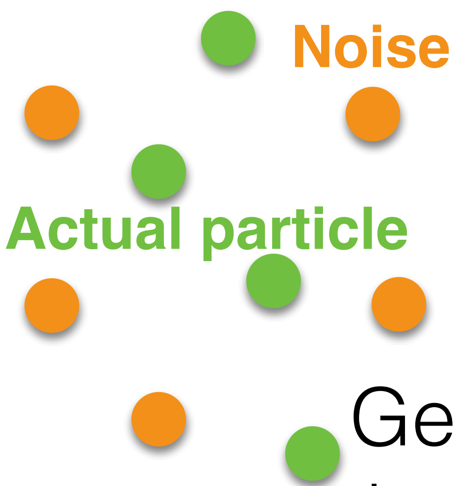


Update the node features using the edge features

'Message-passing' takes place

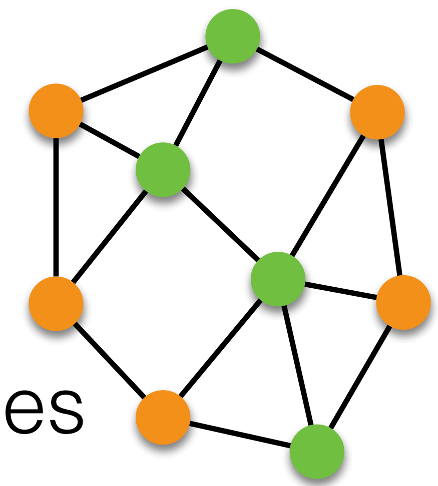
Important information from a neighbor spreads via xi to another neighbor in the next iteration

Graphical example

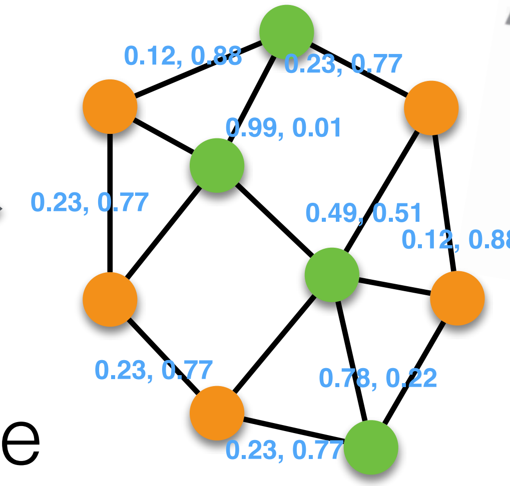


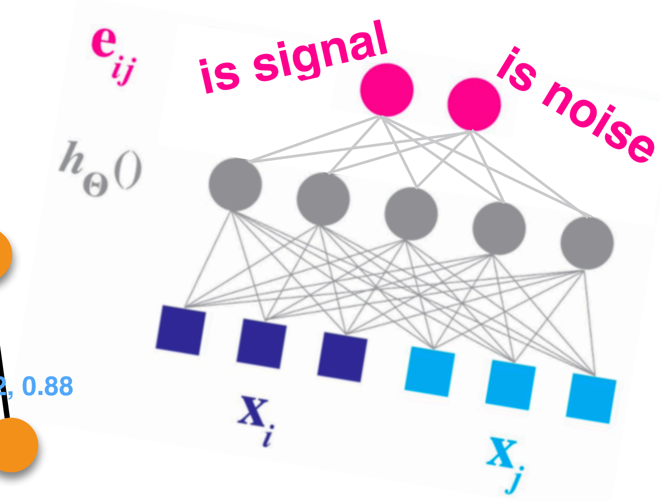
Generate edges between nodes

(preprocessing, kNN)

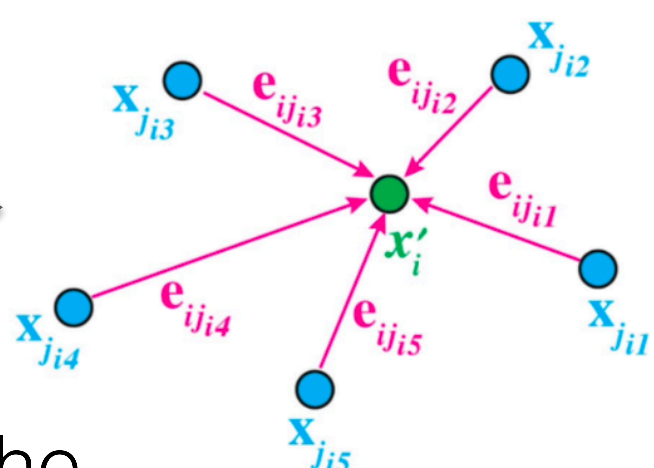


Calculate edge features



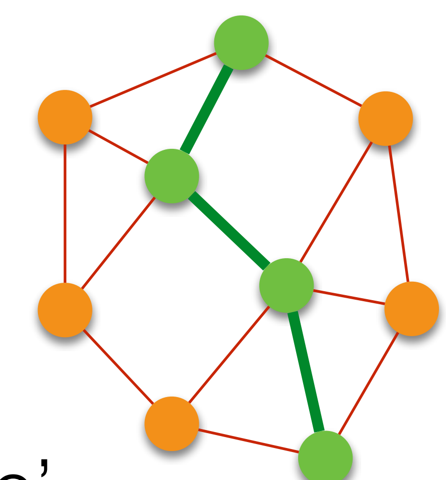


Assume `dim(edge_features) == 2` **for this example**



Update the node features using the edge features

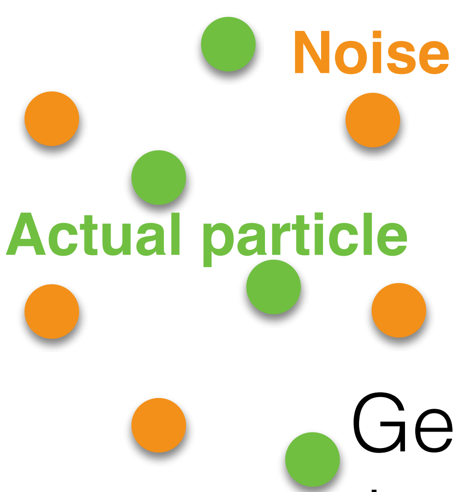
(repeat)



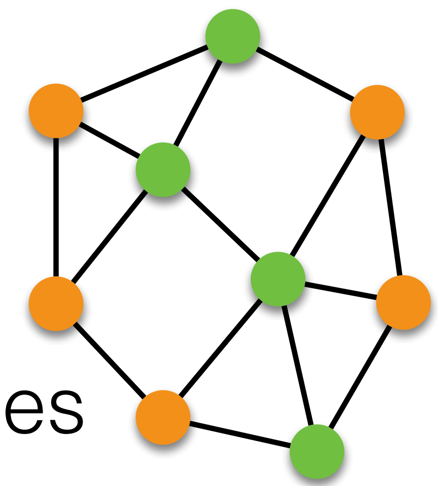
Classify into 'true' edges vs 'false' edges

(ML, 2 edge features)

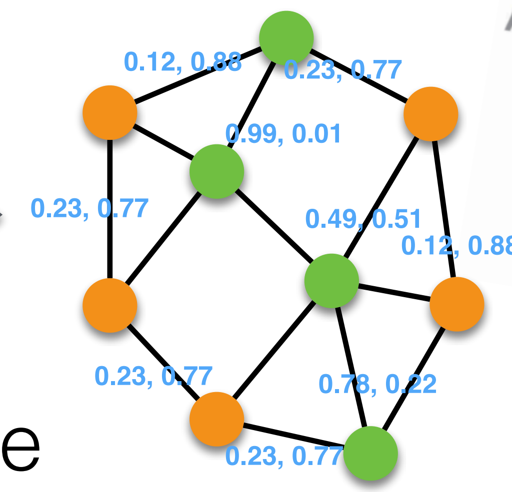
Graphical example

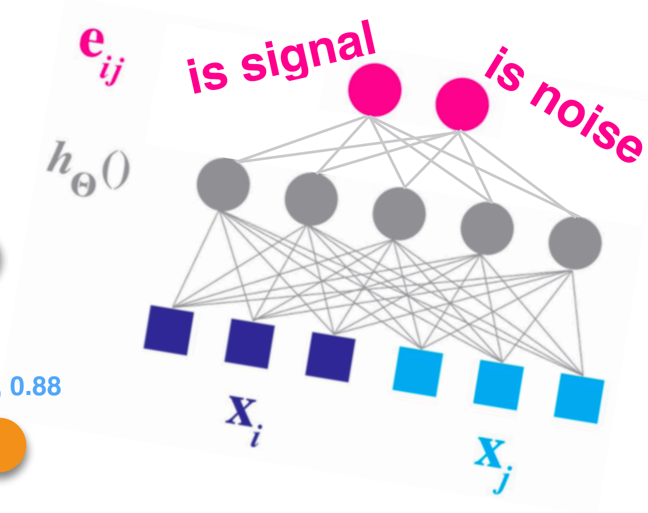


Generate edges between nodes
(preprocessing, kNN)



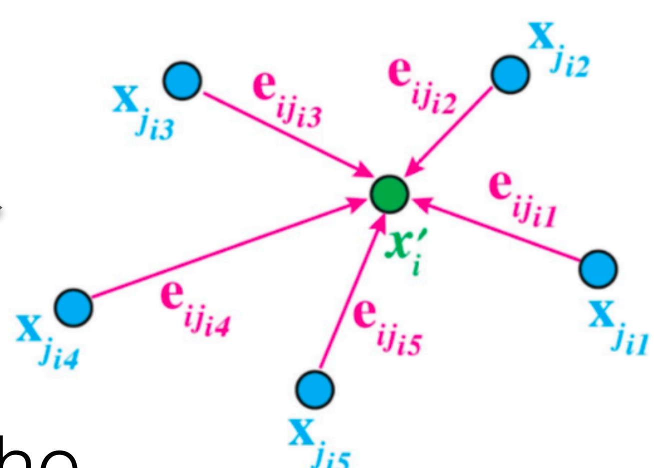
Calculate edge features





Assume `dim(edge_features) == 2` **for this example**

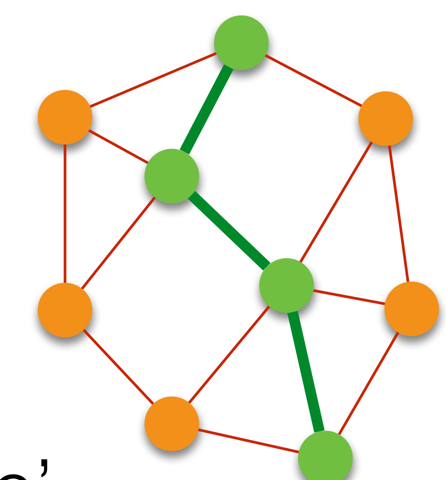
Update the node features using the edge features



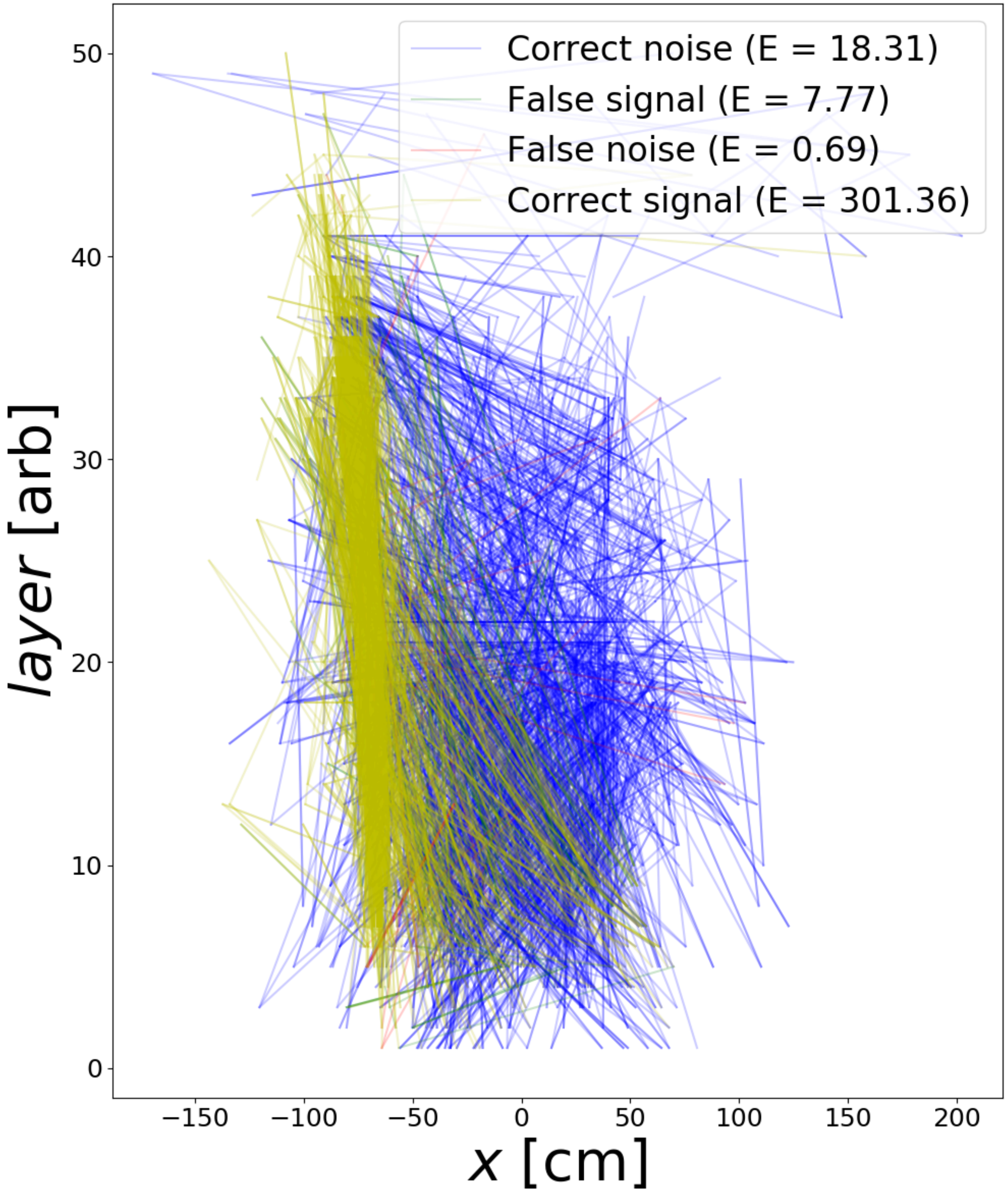
DGCNN would *re-kNN* at this point

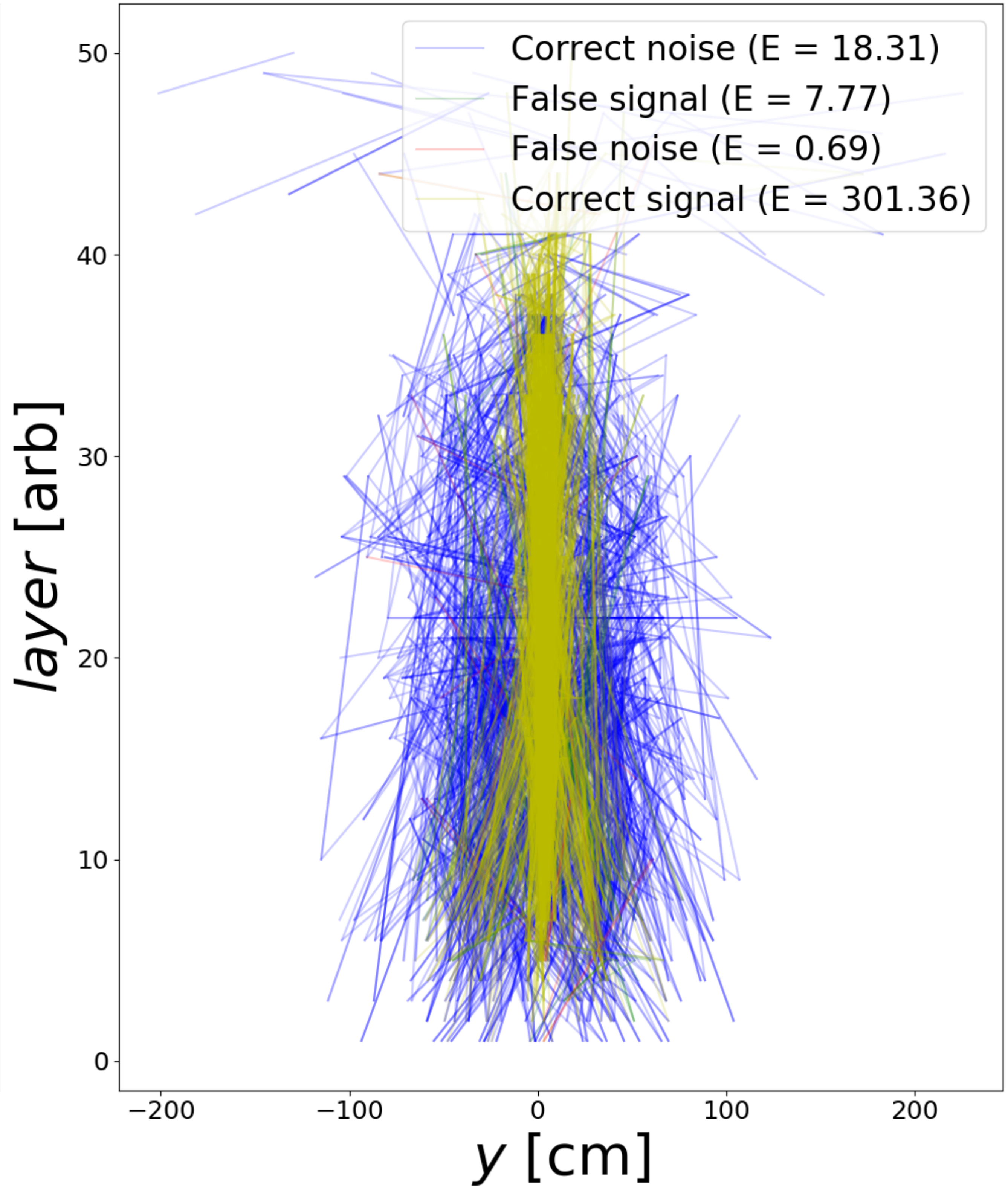
(repeat)

Classify into 'true' edges vs 'false' edges
(ML, 2 edge features)



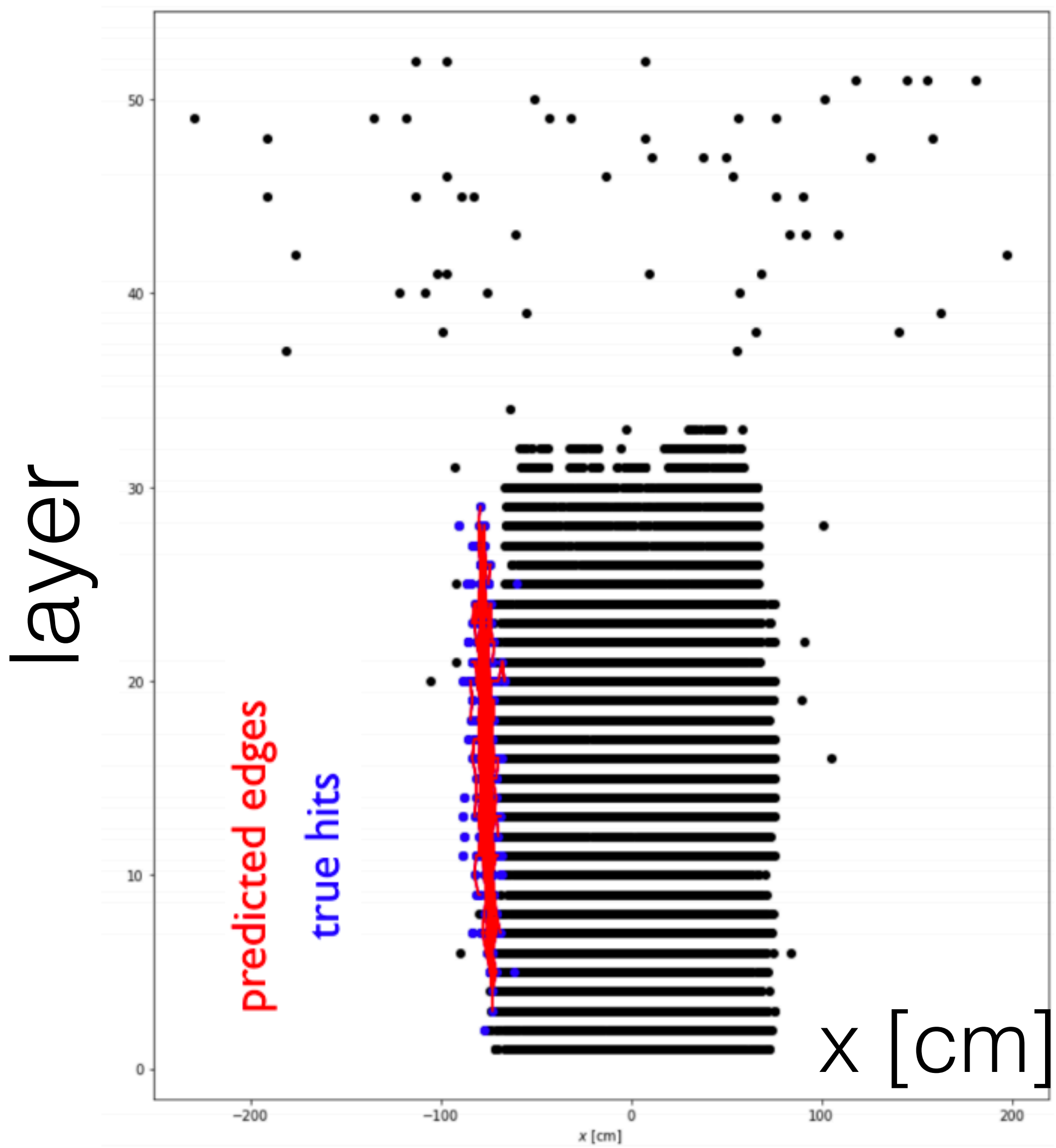
Proof of concept: Single pion

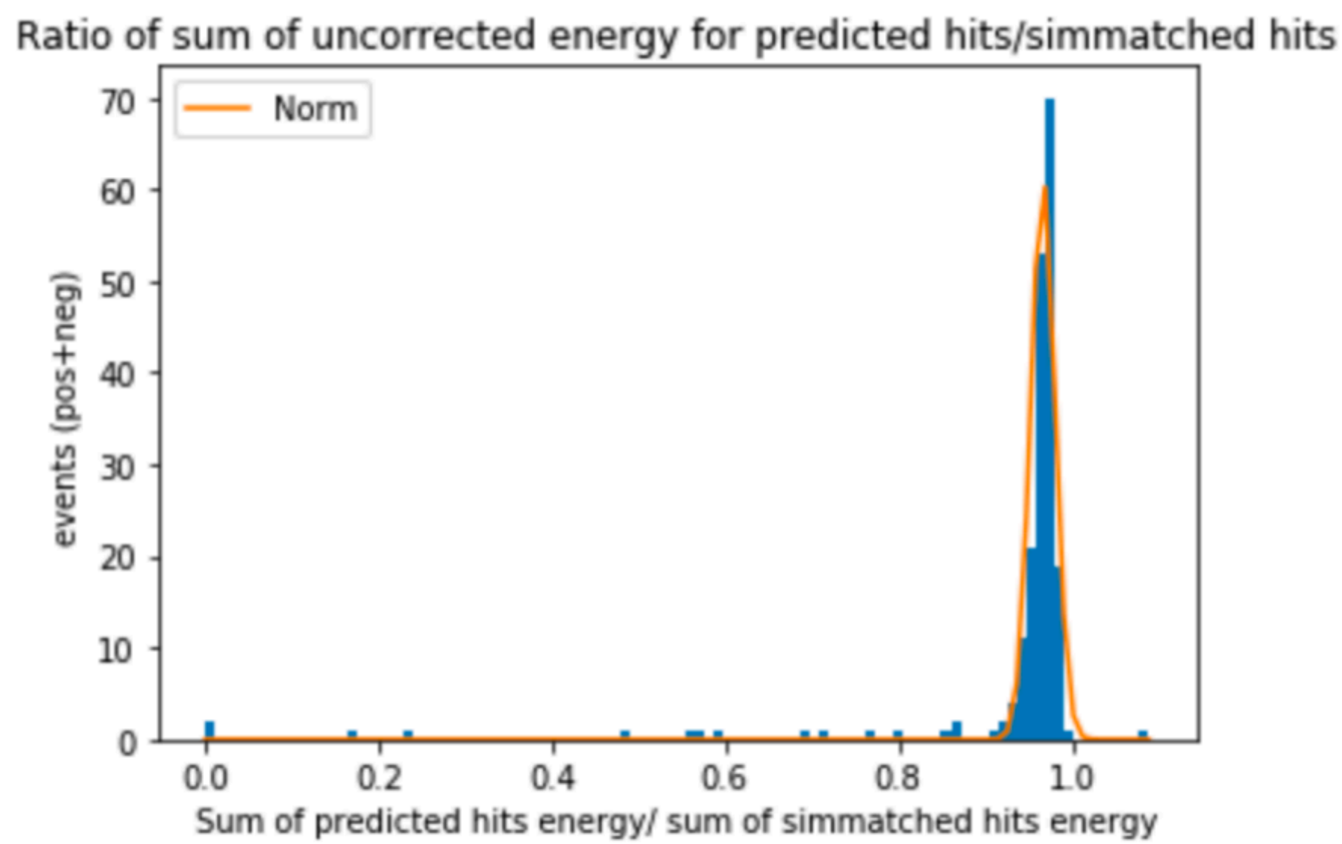




Proof of concept: Single photon

GNNs seem to do well for single particles!





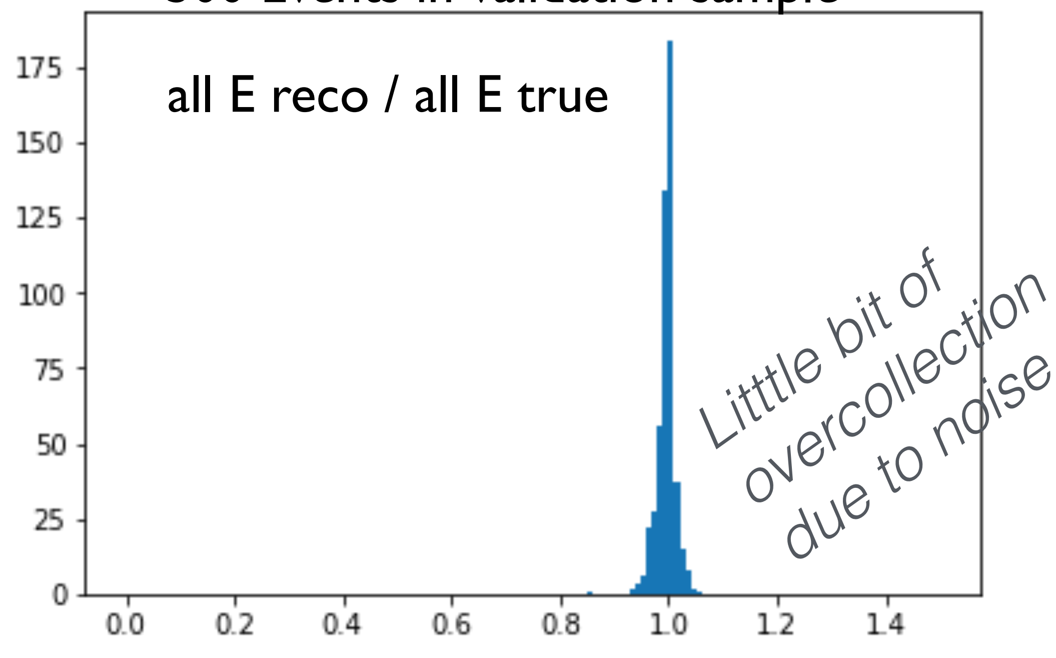
Gaussian fit mean: 0.9664587 , std dev: 0.014030526

- The real challenge: **Many-particle events**

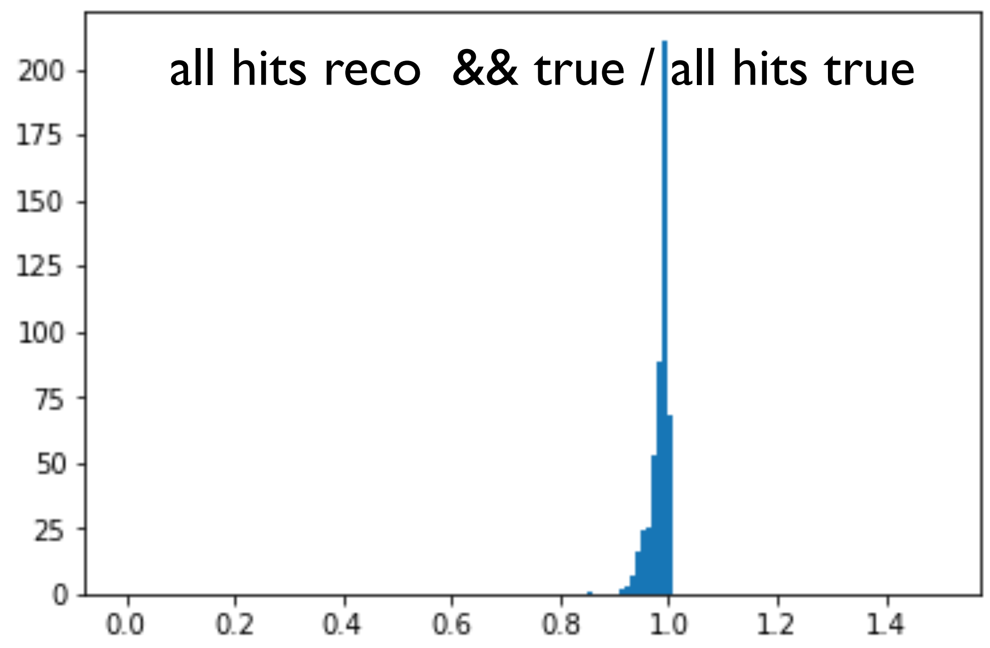
Proof of concept: tau decays

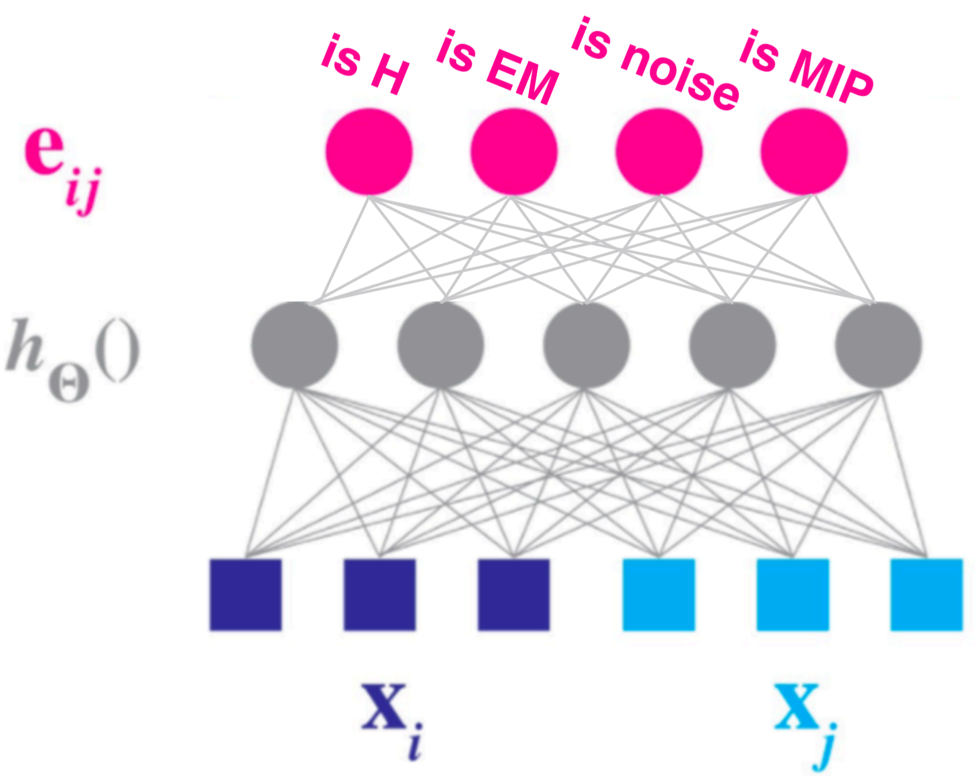
- Most recent development: **clustering** the output of the GNN into **multiple different-type particles** seems to do a good job

500 Events in validation sample



E_sumhits_pred / E_sumhits_true

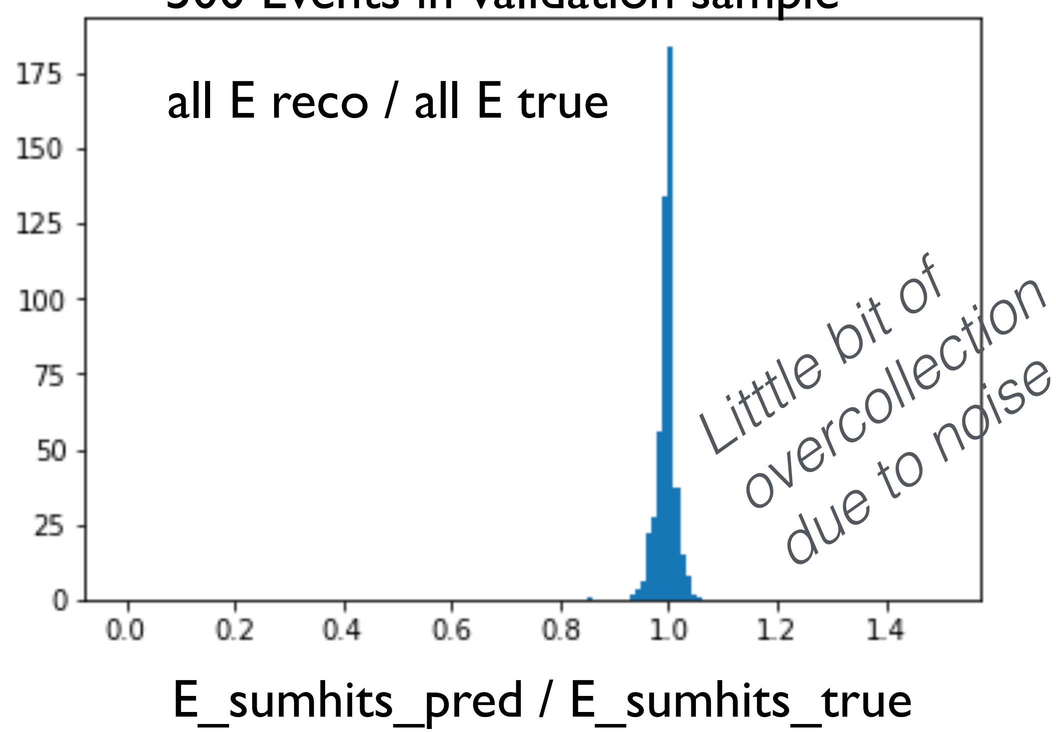


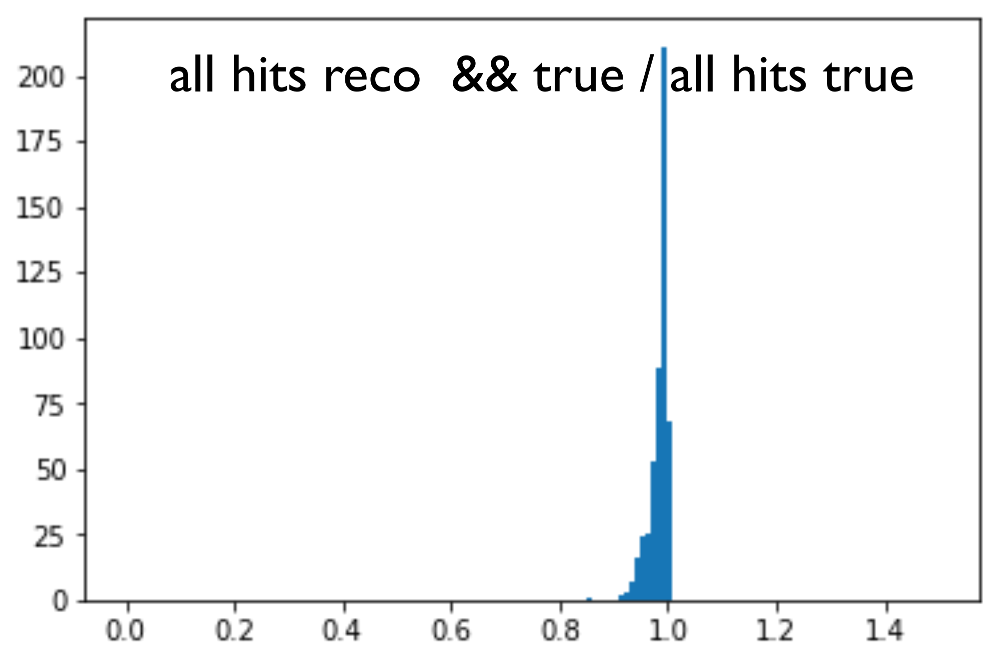


Proof of concept: tau decays

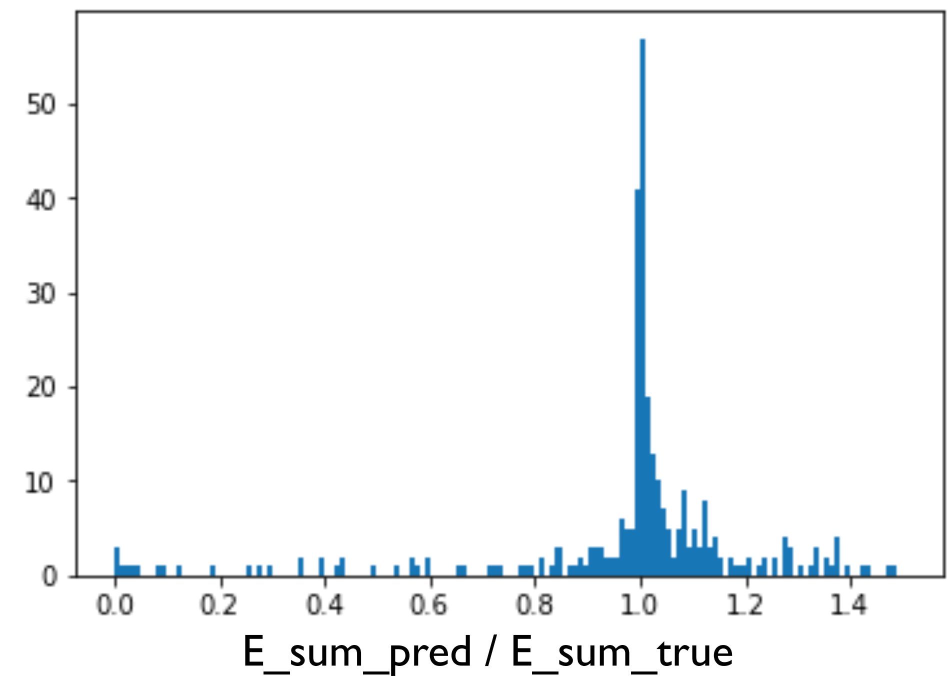
- Most recent development: **clustering** the output of the GNN into **multiple different-type particles** seems to do a good job

500 Events in validation sample

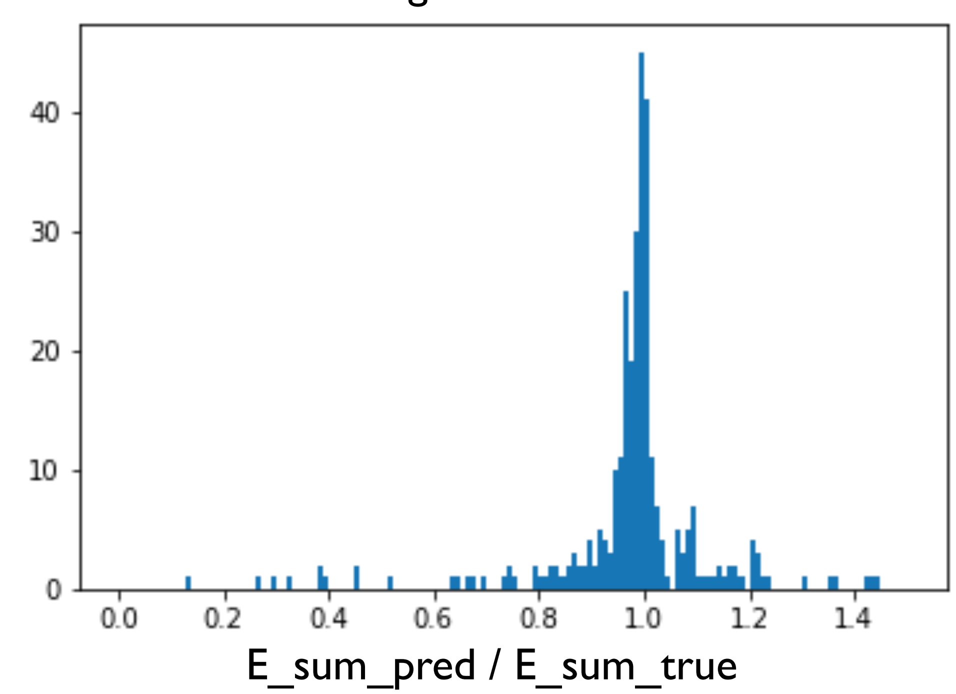




Categorized as Hadron



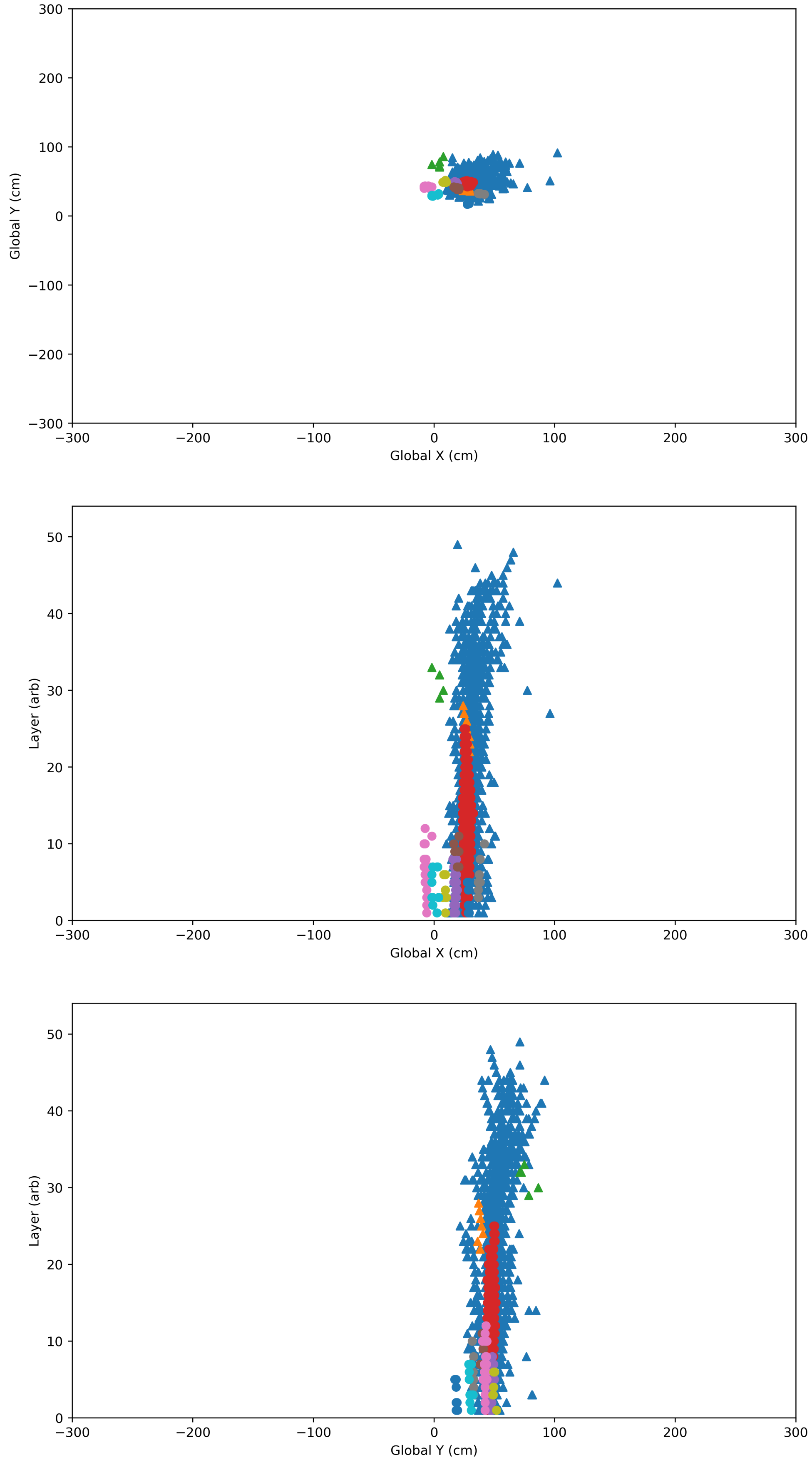
Categorized as EM



'Bare' energy prediction (hadronic vs em) looks promising

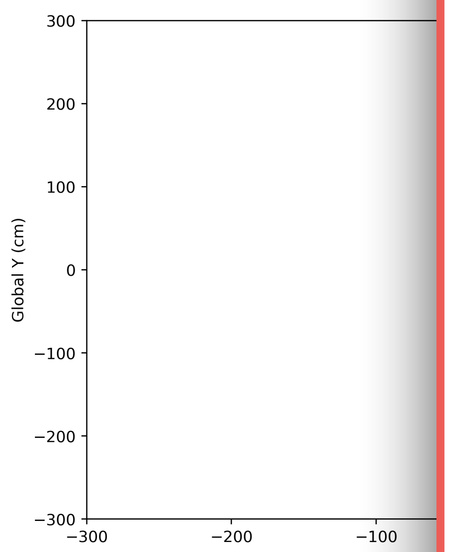
To be regressed...

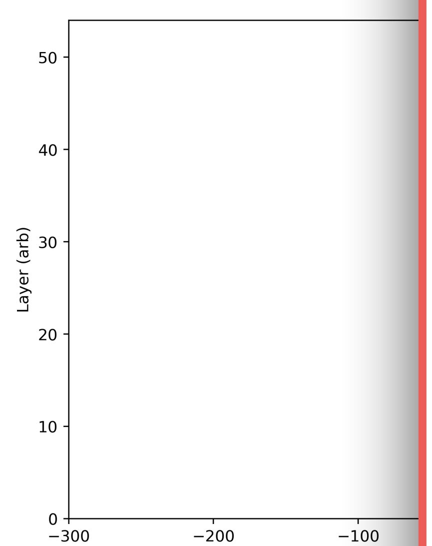
Proof of concept: tau decays

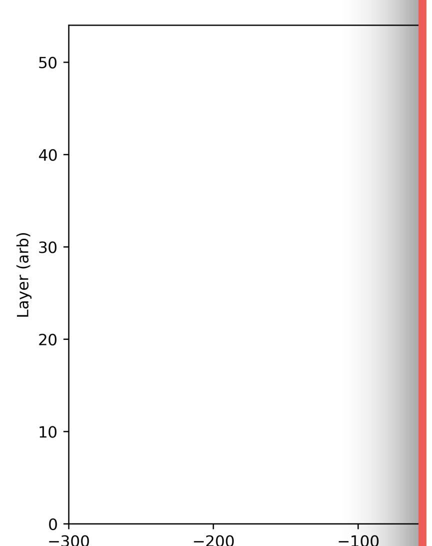


- Example event display:
  Clear particle-like clusters are constructed
- Clusters are separated by EM ●, Hadronic ▲, MIP ✚, and noise (not plotted)
- Work in progress:
  - Pileup
  - Need better 'truth' definition (data prep)
  - Integration into CMSSW (longterm solution probably PyTorch in CMSSW)
  - Hardware acceleration
- Many problems in common with the PF effort

Proof








```python
class EdgeNetWithCategories(nn.Module):
    def __init__(self, input_dim=3, hidden_dim=8, output_dim=4, n_iters=1, aggr='add',
                 norm=torch.tensor([1./500., 1./500., 1./54., 1/25., 1./1000.])):
        super(EdgeNetWithCategories, self).__init__()

        self.datanorm = nn.Parameter(norm)

        start_width = 2 * (hidden_dim + input_dim)
        middle_width = (3 * hidden_dim + 2*input_dim) // 2

        self.n_iters = n_iters

        self.inputnet =  nn.Sequential(
            nn.Linear(input_dim, 2*hidden_dim),
            nn.Tanh(),
            nn.Linear(2*hidden_dim, 2*hidden_dim),
            nn.Tanh(),
            nn.Linear(2*hidden_dim, hidden_dim),
            nn.Tanh(),
        )

        self.edgenetwork = nn.Sequential(nn.Linear(2*n_iters*hidden_dim, 2*hidden_dim),
                                         nn.ELU(),
                                         nn.Linear(2*hidden_dim, 2*hidden_dim),
                                         nn.ELU(),
                                         nn.Linear(2*hidden_dim, output_dim),
                                         nn.LogSoftmax(dim=-1),
        )

        for i in range(n_iters):
            convnn = nn.Sequential(nn.Linear(start_width, middle_width),
                                   nn.ELU(),
                                   #nn.Dropout(p=0.5, inplace=False),
                                   nn.Linear(middle_width, hidden_dim),
                                   nn.ELU()
                                  )
            setattr(self, 'nodenetwork%d' % i, EdgeConv(nn=convnn, aggr=aggr))

    def forward(self, data):
        row,col = data.edge_index
        x_norm = self.datanorm * data.x
        H = self.inputnet(x_norm)
        H = getattr(self,'nodenetwork0')(torch.cat([H, x_norm], dim=-1), data.edge_index)
        H_cat = H
        for i in range(1,self.n_iters):
            H = getattr(self,'nodenetwork%d' % i)(torch.cat([H, x_norm], dim=-1), data.edge_index)
            H_cat = torch.cat([H, H_cat], dim=-1)
        return self.edgenetwork(torch.cat([H_cat[row],H_cat[col]],dim=-1)).squeeze(-1)
```

See github

re constructed

M ●, Hadronic

tted)

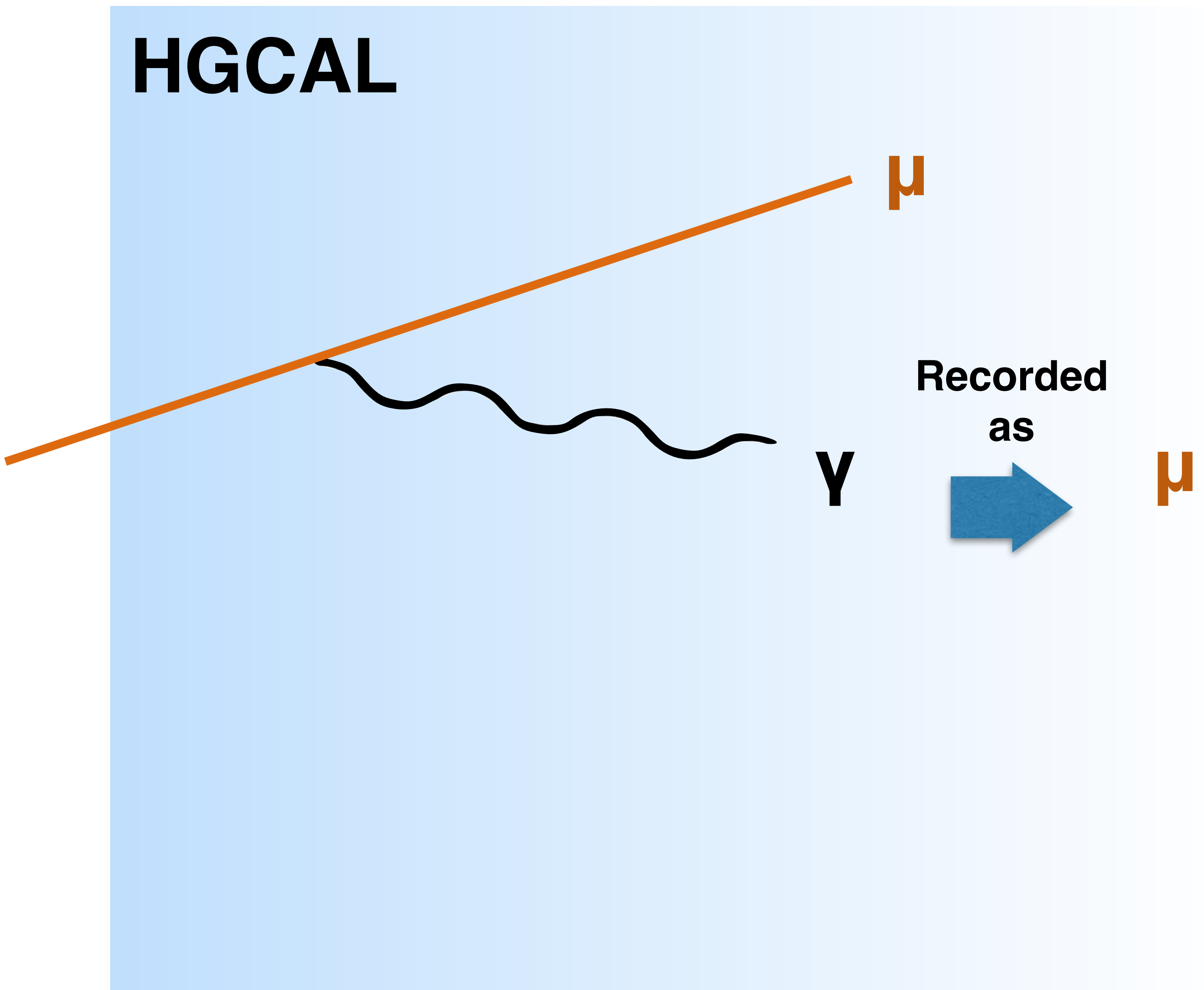
on (data prep)

longterm solution

with the PF

Intermezzo: The 'history' issue

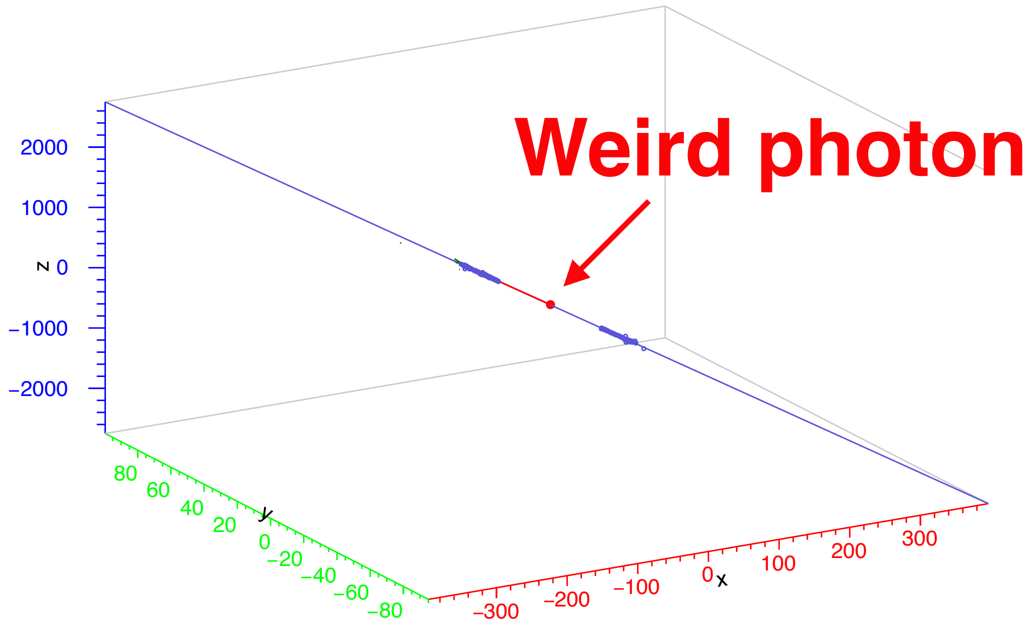
- Secondary particles are recorded as if they are part of their parent



- This leads to showers that *look* a lot like like their type (e.g. photon), but are truth-tagged as their parent-type (e.g. muon)

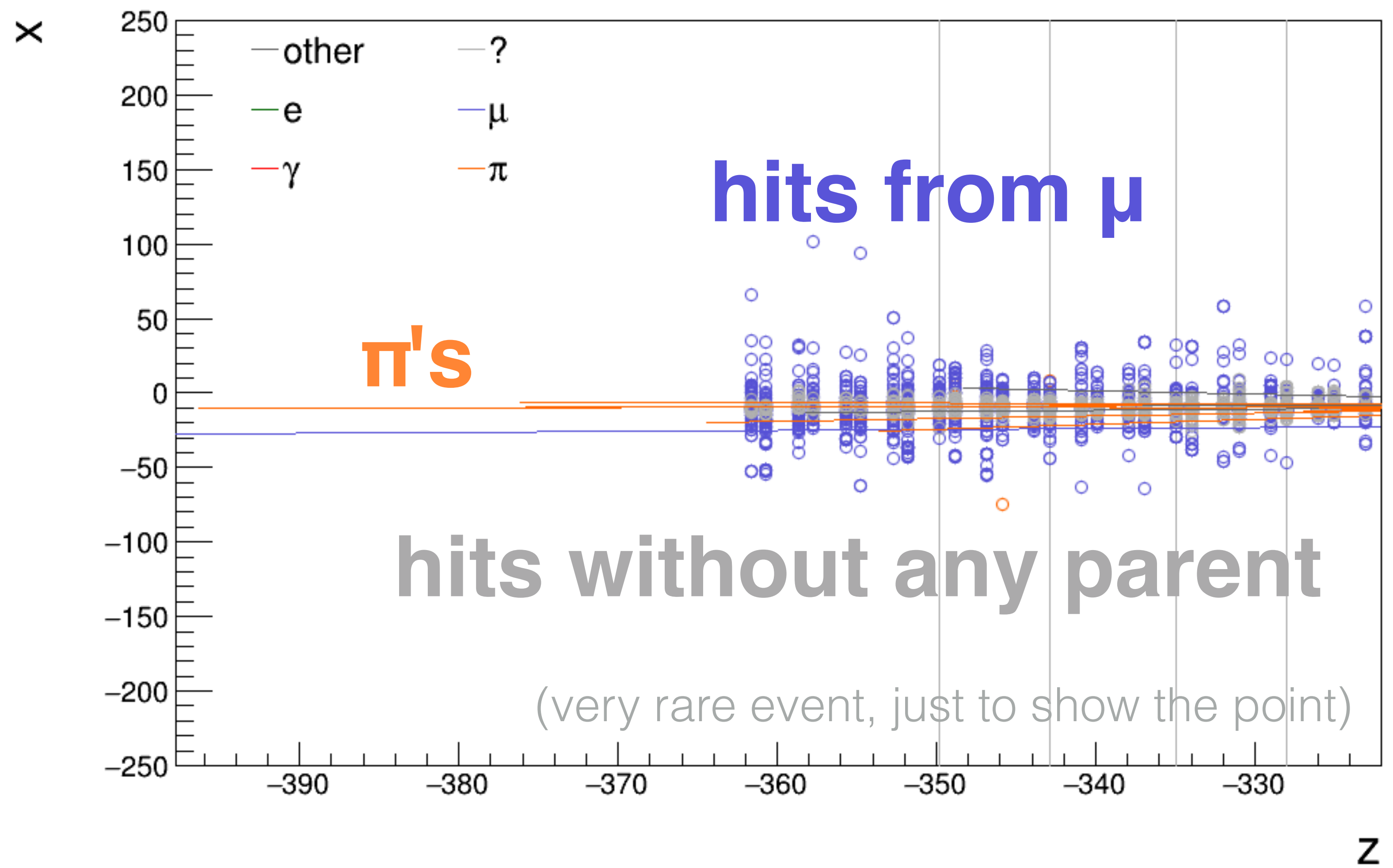
- This throws off any deep learning

Work in progress, bugs to solve



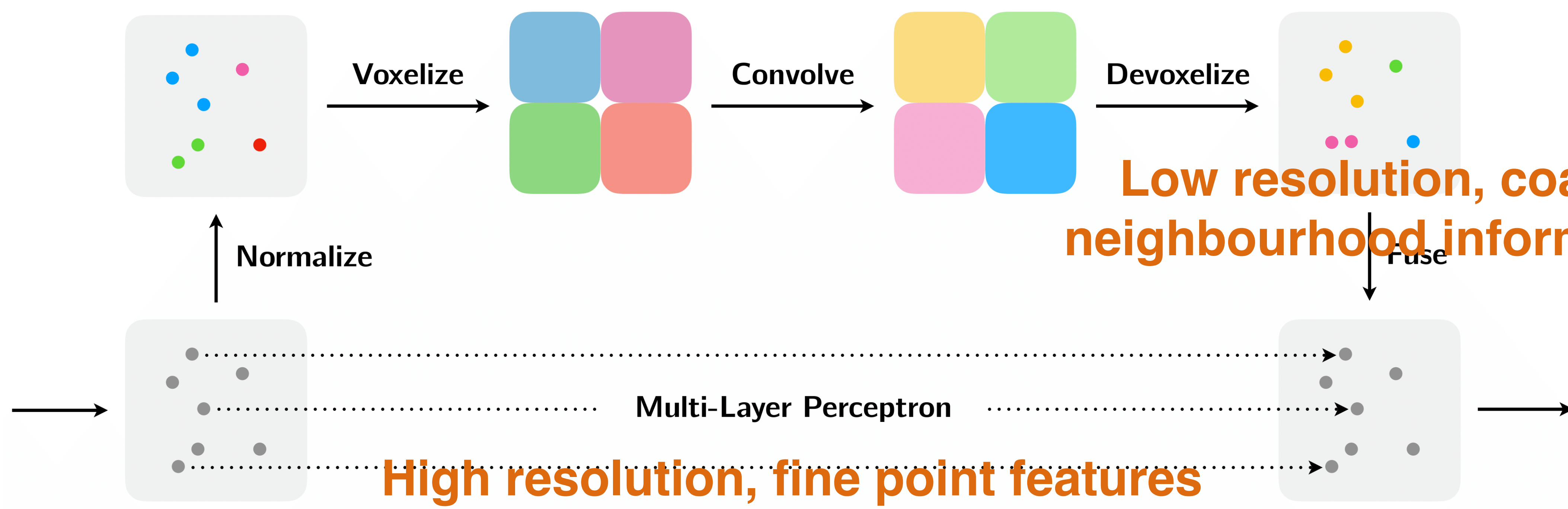
- Presence of a buggy photon

- Still incorrectly labeled hits
- Planning on getting involved with this software ourselves

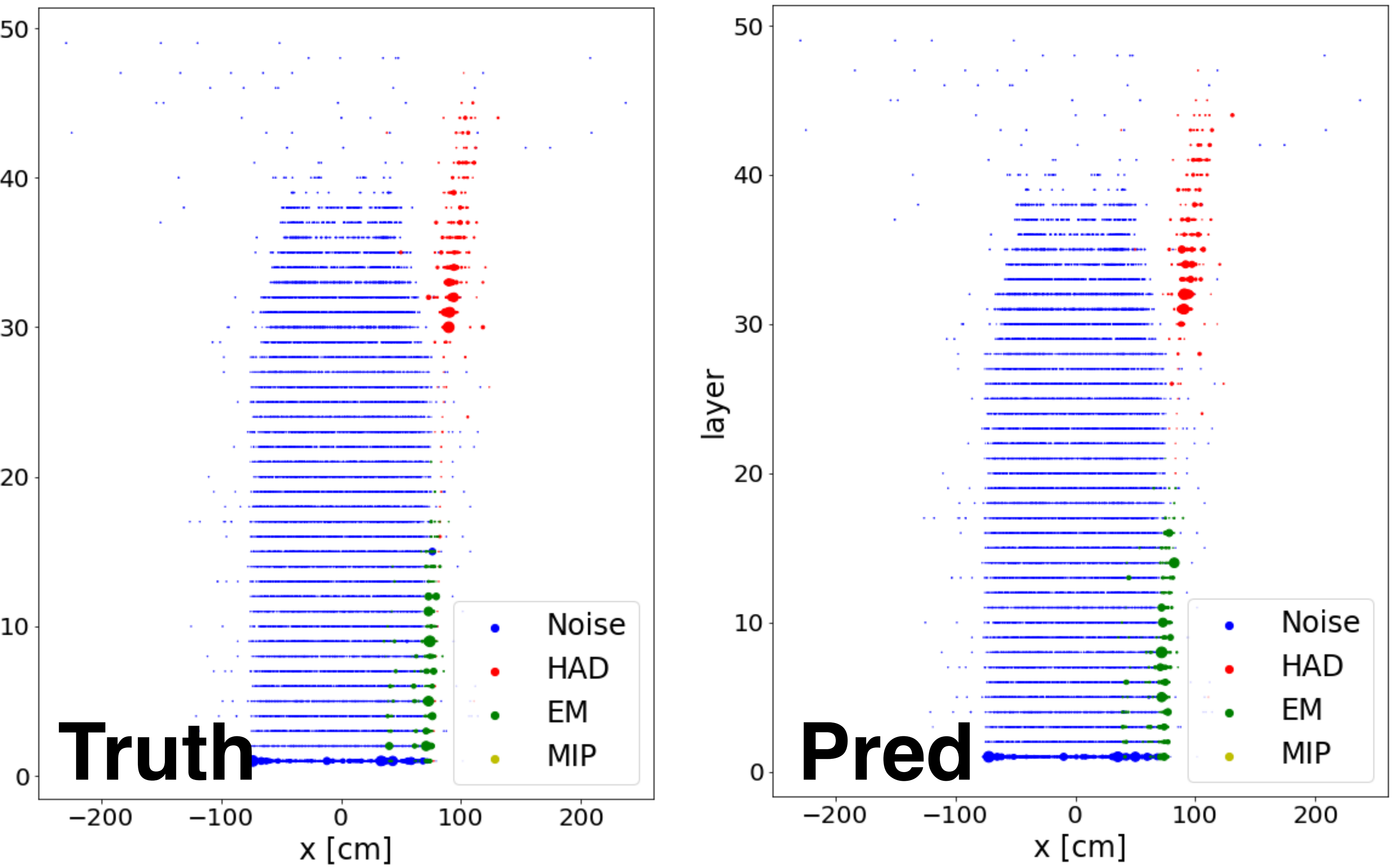


Point-Voxel CNN

- Use **fine-grained point information** in a small dense layer, **and coarse-grained voxelization** in convolutional layers
    - Mitigate effects from poor memory locality (pure point cloud) and huge memory consumption (pure voxelization)
- In principle a **point-cloud network, not graph based**
    - Working with the authors to get to an edge-classification network



Point-Voxel CNN

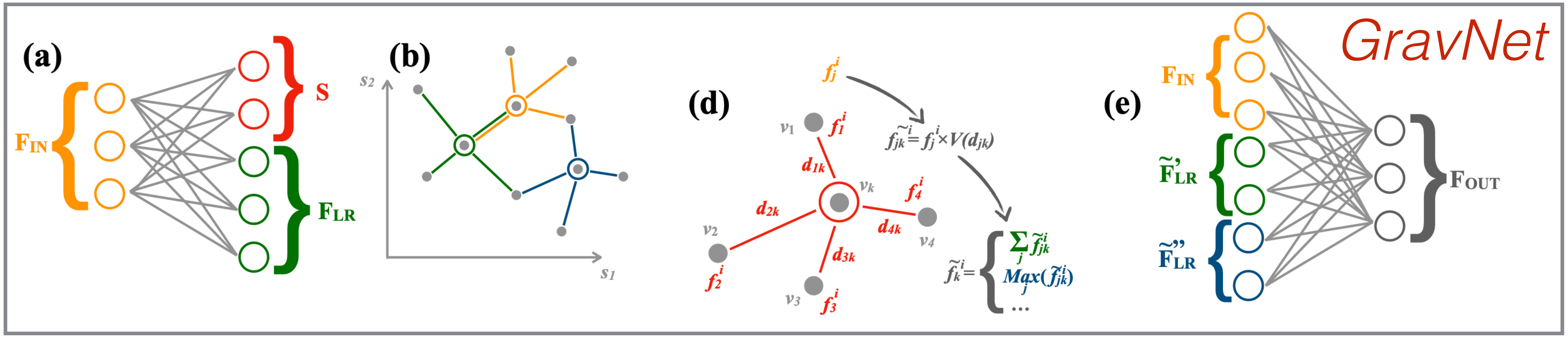


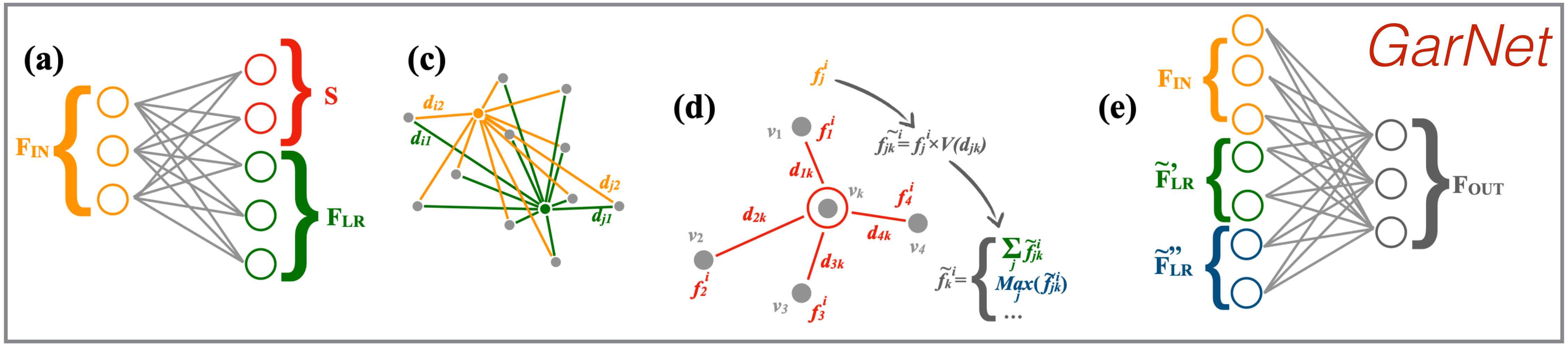
Confusion matix				
	Noise	**HAD**	**EM**	**MIP**
Noise	0.9984	0.0268	0.0275	0.0164
HAD	0.0009	0.8004	0.1487	0.0680
EM	0.0007	0.1715	0.8233	0.0089
MIP	0.0001	0.0014	0.0005	0.9068

- Does pretty good job out of the box (arch optimized for classifying office furniture)
- Needs further study into
  - Instance segmentation
  - Inference speed
  - Using 'better' PVCNN called SPVNAS

GravNet/GarNet

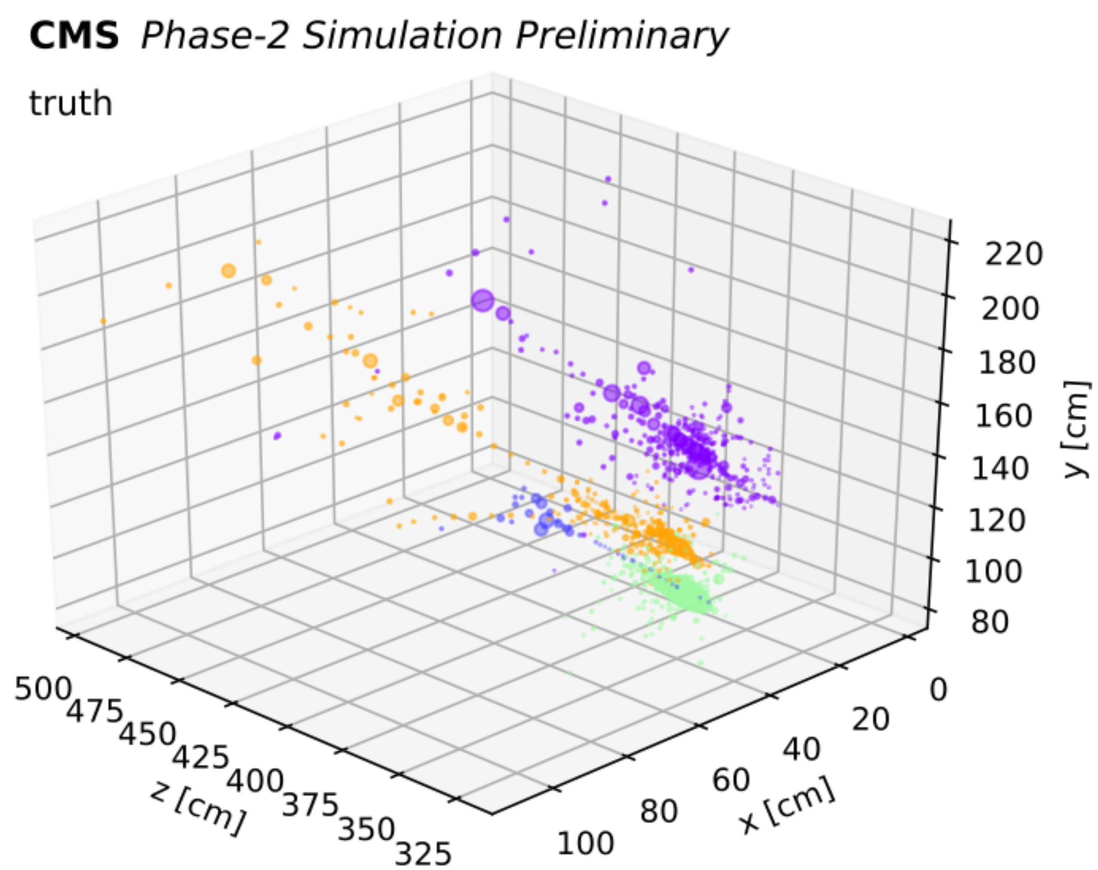
- **DGCNN** uses *large* amount of memory and keeping *inference time* under control is a challenge
- **GravNet/GarNet** greatly reduces computational needs
  - Split coordinate and feature space

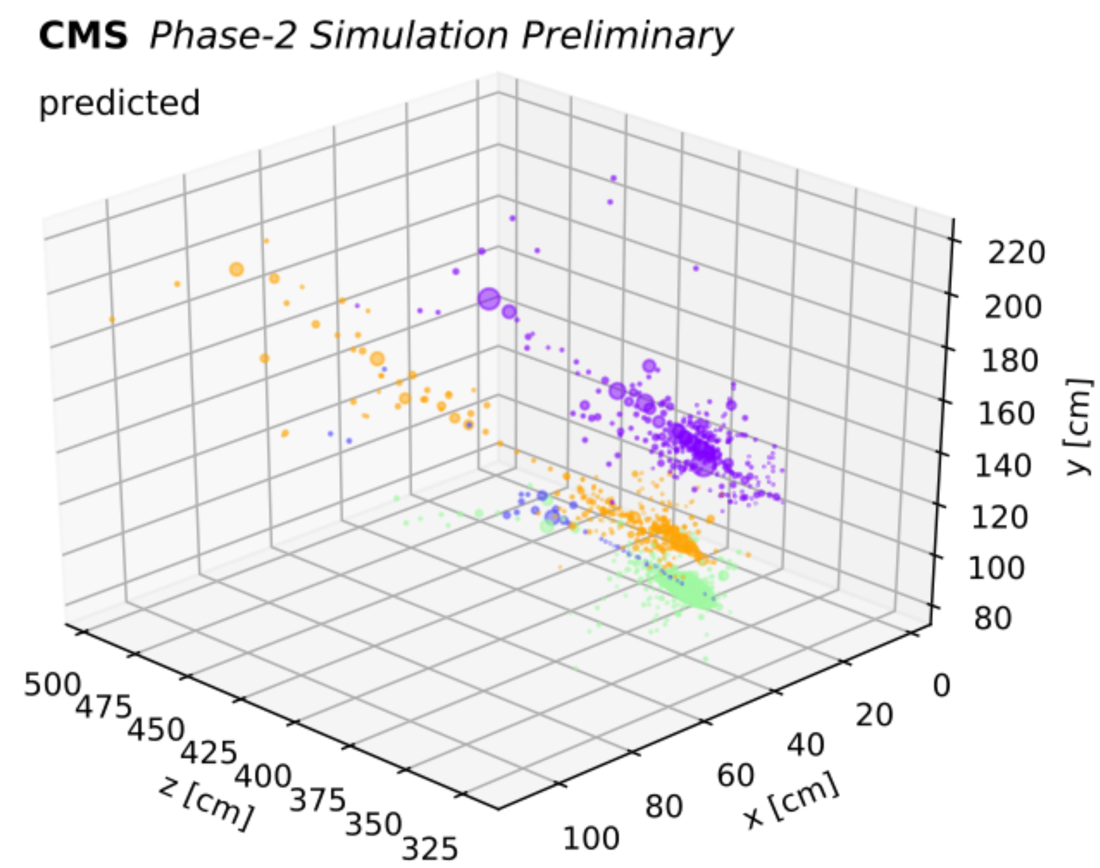


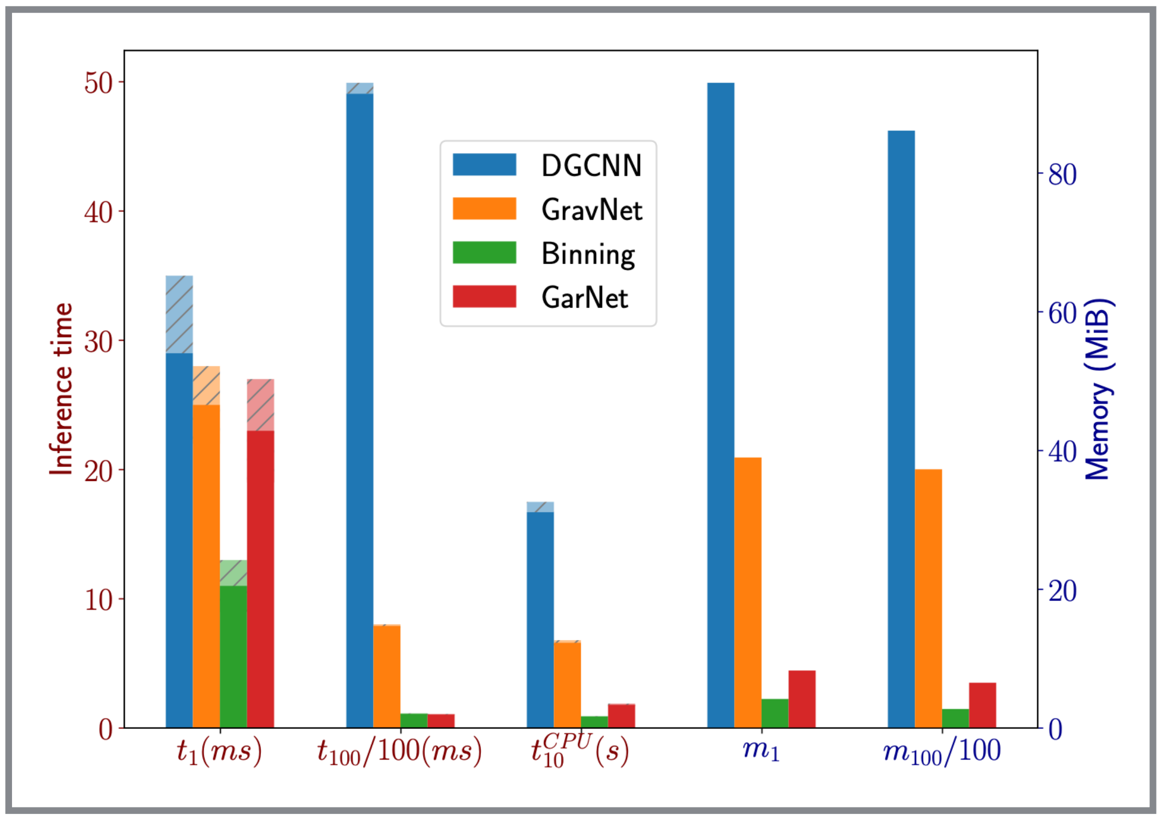


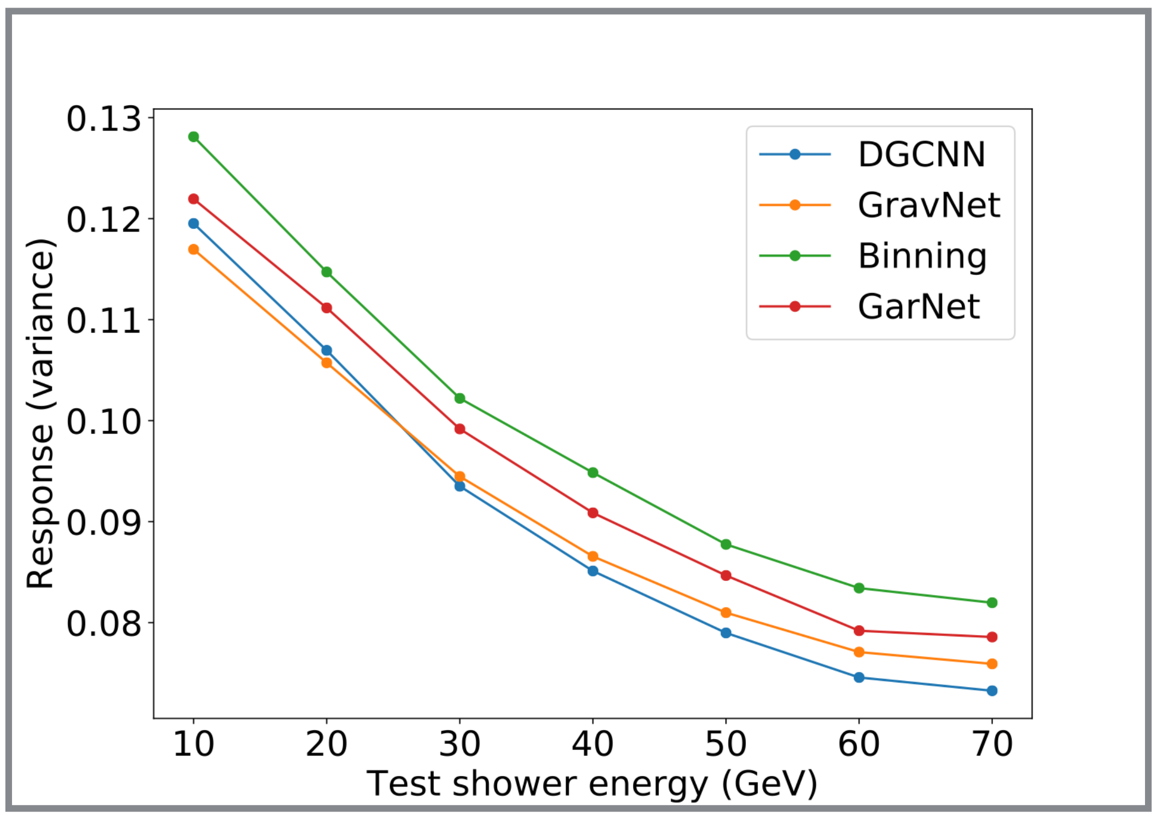
GravNet/GarNet

[1902.07987]



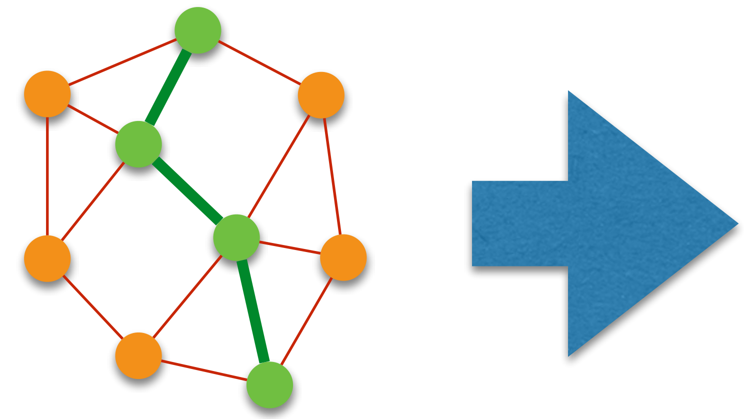






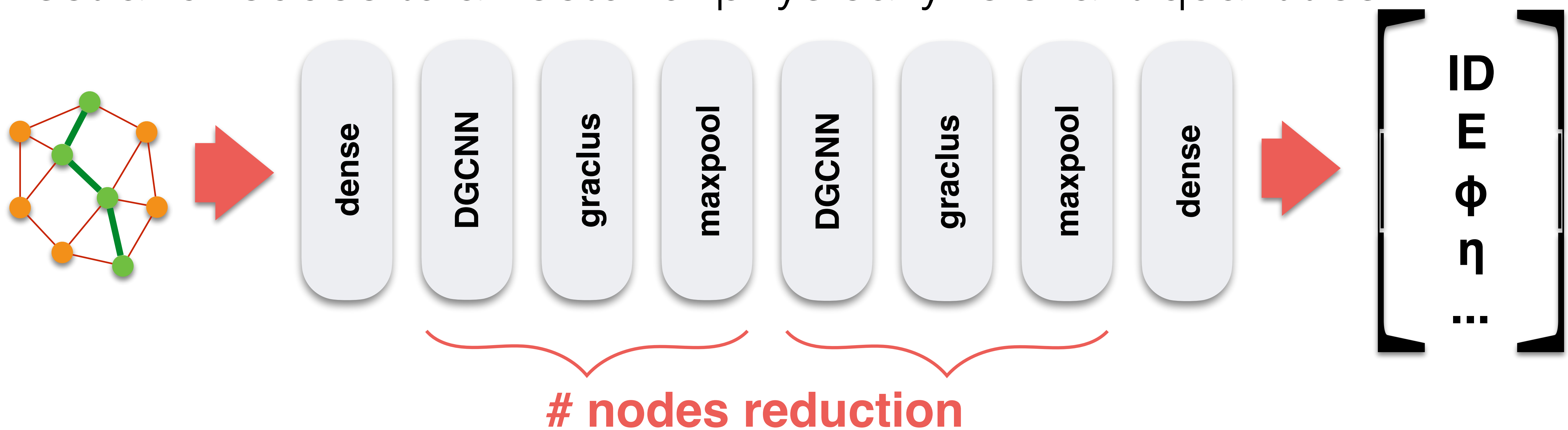
Regression

- In the end, want to get back meaningful properties of cluster



ID, energy, 4-mom, ...

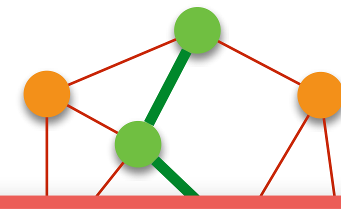
- '**Dynamic Reduction Network**' capable of taking an unordered set and reduce to a vector of physically relevant quantities



- Because of *EdgeConv*, **learns** how to use **organization and weighting** of input data to regress to physics
- Gets 99.55% test accuracy on MNIST (#19-21 on leaderboard)

Regression

- In the end, want to get



ID, energy, 4-

```python
def forward(self, data):
    data.x = self.datanorm * data.x
    data.x = self.inputnet(data.x)

    data.edge_index = to_undirected(knn_graph(data.x, self.k, data.batch, loop=False, flow=self.edgeconv1.flow))
    data.x = self.edgeconv1(data.x, data.edge_index)

    weight = normalized_cut_2d(data.edge_index, data.x)
    cluster = graclus(data.edge_index, weight, data.x.size(0))
    data.edge_attr = None
    data = max_pool(cluster, data)

    data.edge_index = to_undirected(knn_graph(data.x, self.k, data.batch, loop=False, flow=self.edgeconv2.flow))
    data.x = self.edgeconv2(data.x, data.edge_index)

    weight = normalized_cut_2d(data.edge_index, data.x)
    cluster = graclus(data.edge_index, weight, data.x.size(0))
    x, batch = max_pool_x(cluster, data.x, data.batch)

    x = global_max_pool(x, batch)

    return self.output(x).squeeze(-1)
```

See github

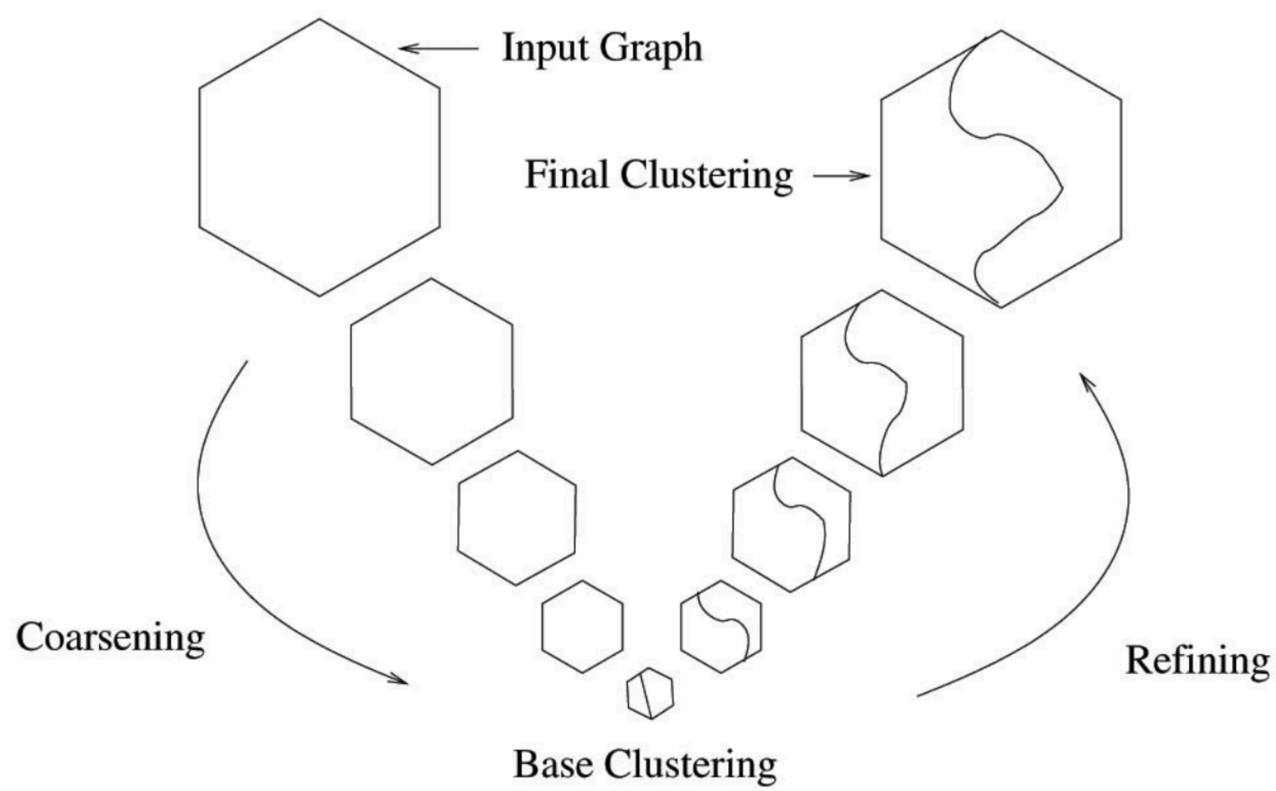
- Gets 99.55% test accuracy on MNIST (#19-21 on leaderboard)

What's "graclus"????

- Greedy, popularity based graph clustering algorithm
  - A node is more popular if it has more and close-by neighbors
  - Greedy means that the most popular nodes seed clusters and accumulate neighbors into themselves

Basic idea:

fuse points together by most neighbours



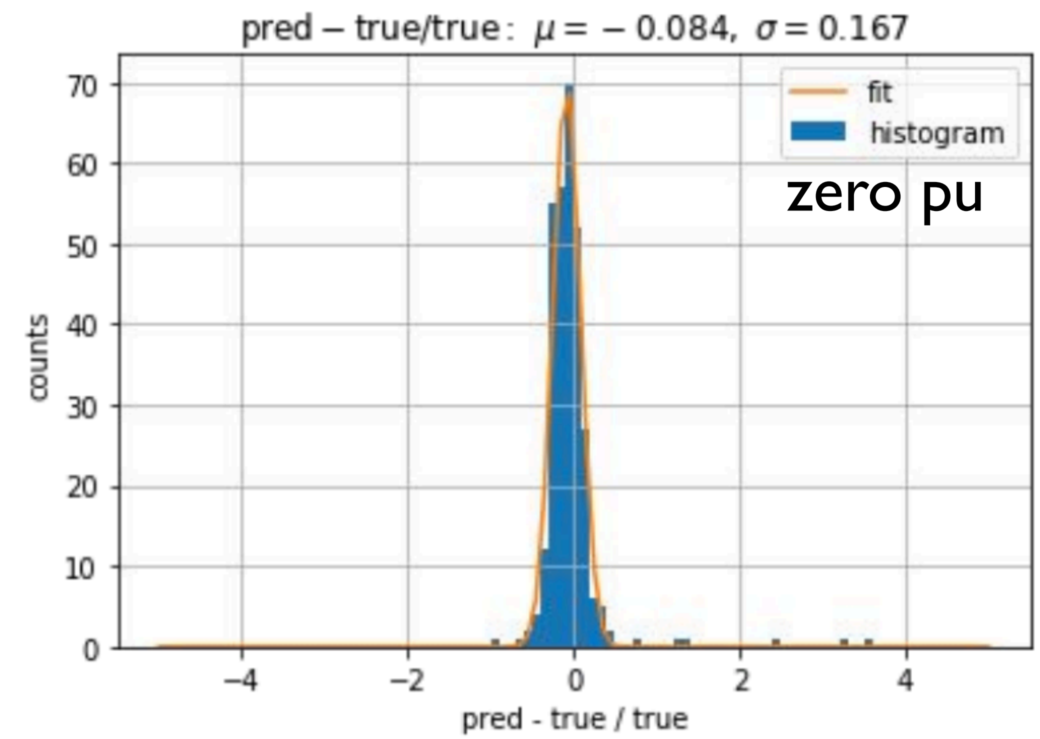
split points back apart, assign cluster by "popularity"

https://doi.org/10.1109/TPAMI.2007.1115

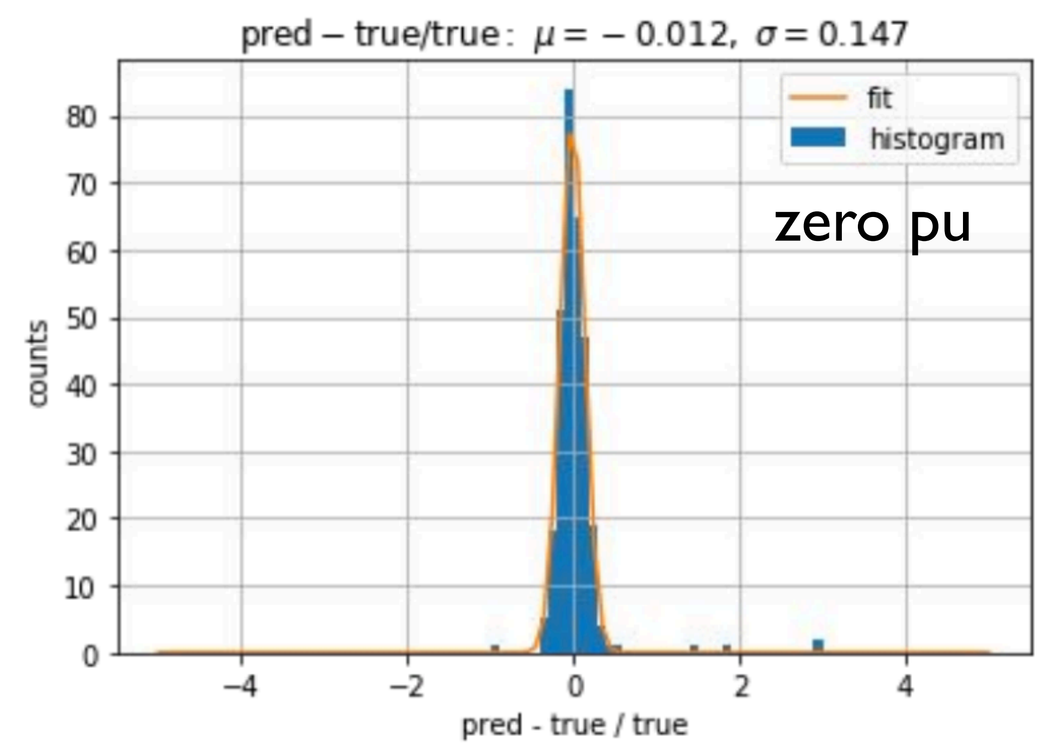
Hadrons - Qualitative Results

- Network learns quickly
  - 1 epoch ~3 minutes
  - loss settled by epoch 20
  - no signs of overtraining
- Clear improvement in first epochs
  - avg. scale centers
  - width and outliers reduced

Epoch 1

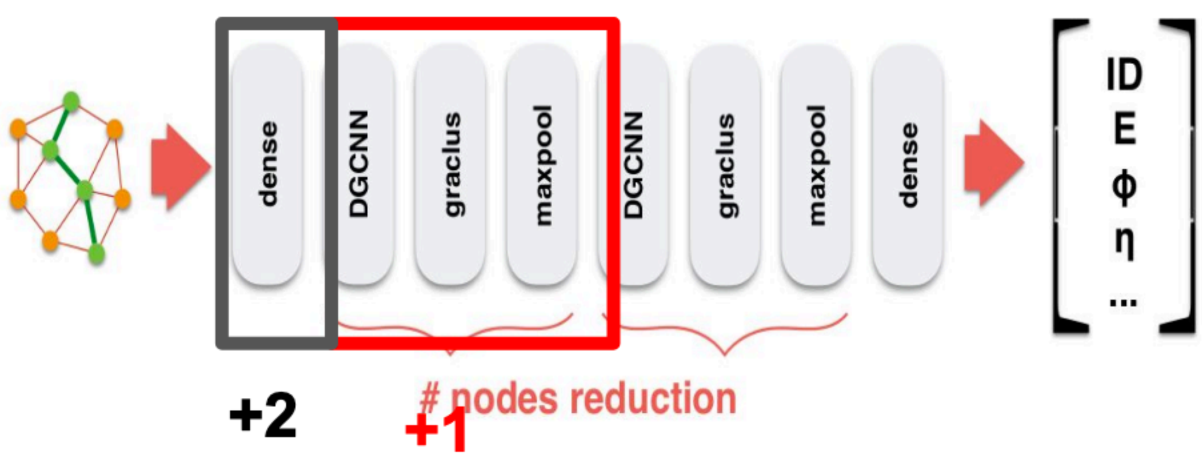


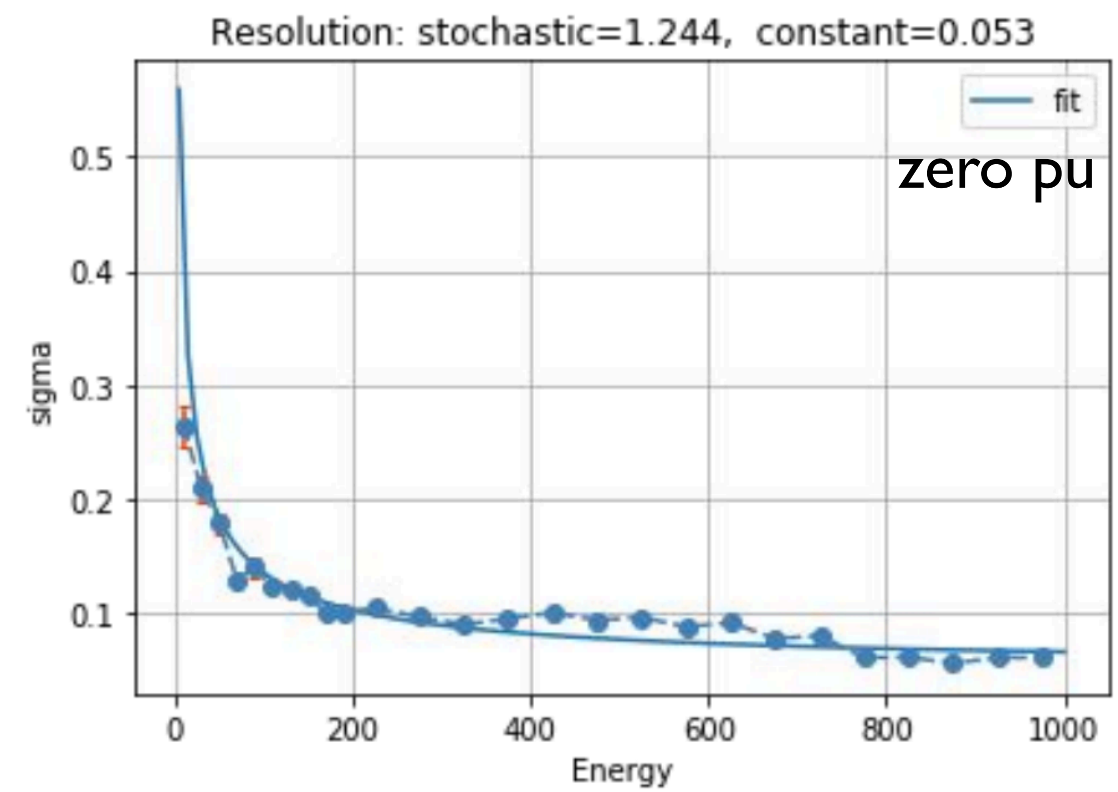
Epoch 9

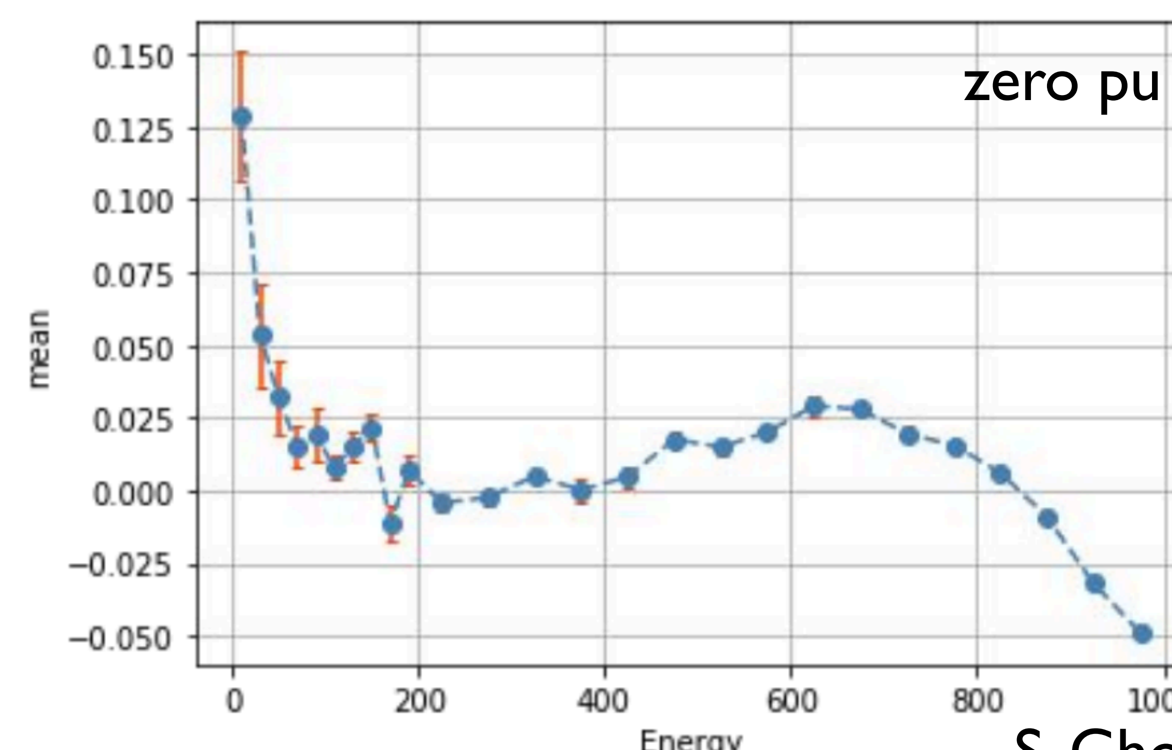


Hadrons - Best Model Variant So Far

- More ability to encode fluctuations of low energy showers
  - 124% stochastic term
  - 5.3% constant term
    - to improve
  - Scale rather flat (for hadrons)
- Good enough for now
  - next step: flatten scale



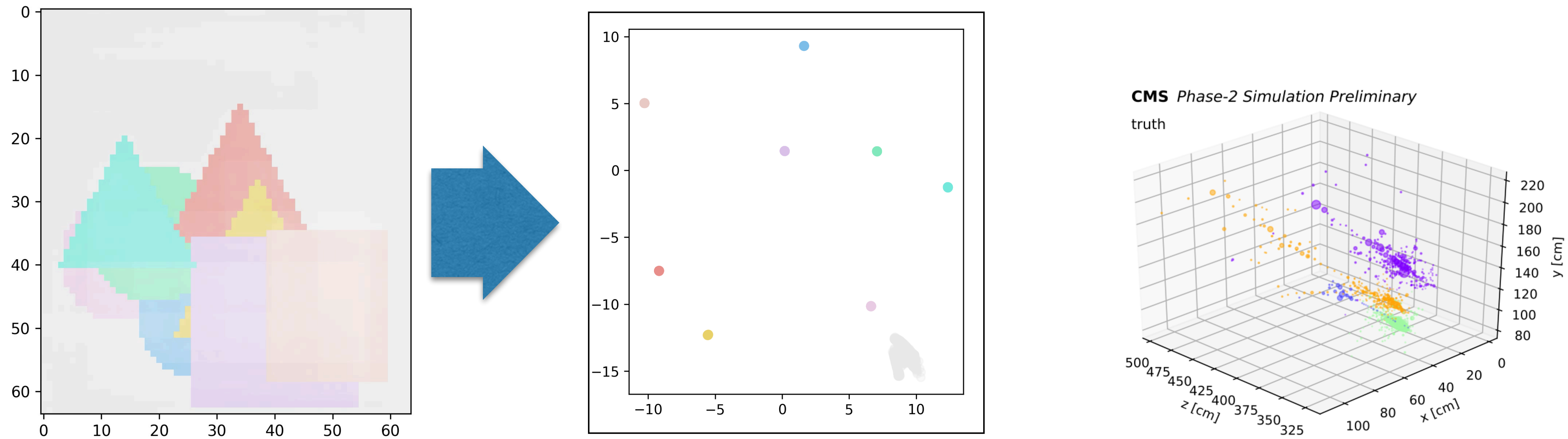




S. Ghosh

Object condensation [2002.03605]

- Method to perform multiple pattern recognition tasks in one operation
- **Input** is a set of related pixels/points/vertices/edges/...
  Output is a **number of objects** (e.g. number of particles in an event) each carrying their high-level object properties (e.g. their four-momenta)



- Eliminates need for bounding boxes (which don't work well for sparse objects)
- Method generalizes for point clouds, may use to do clustering + regression for HGCAL
- Results on this to be produced very soon

Conclusion

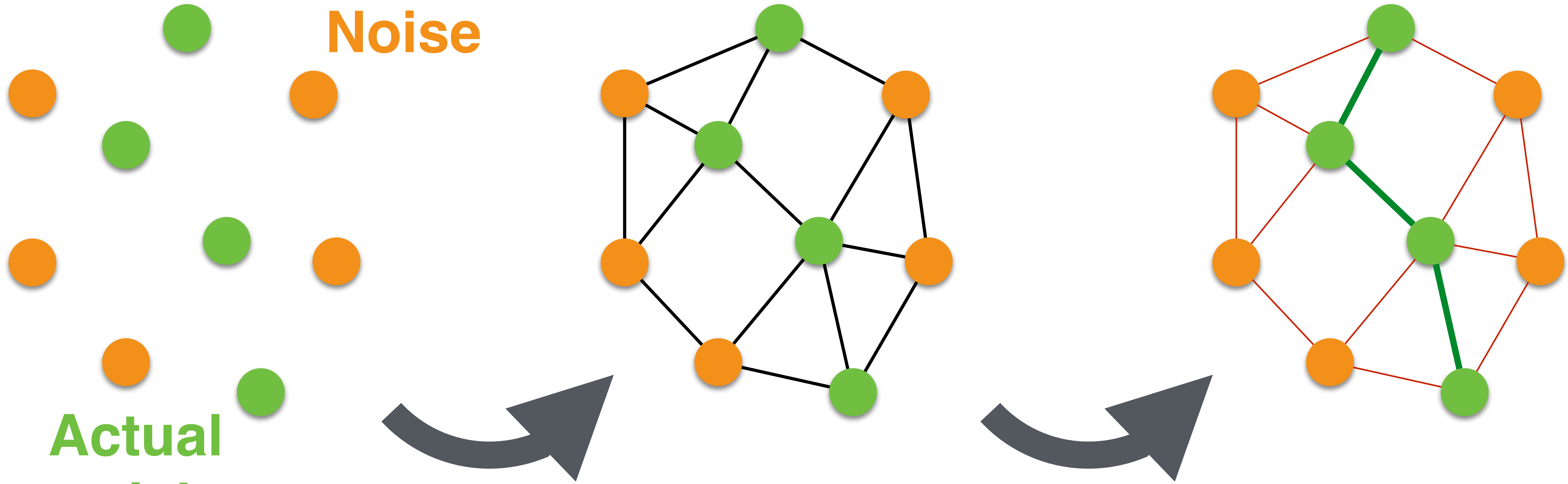
- Exciting early-stage results using GNNs for reconstruction in HGCAL
    - Collimated, multi-particle reconstruction within grasp
    - Multiple available methods, progressing on multiple fronts
- Other challenges:
    - Onwards to including pileup
    - Truth definition in dataset
    - Hardware acceleration
    - Integration into CMSSW

Backup

Graphical example

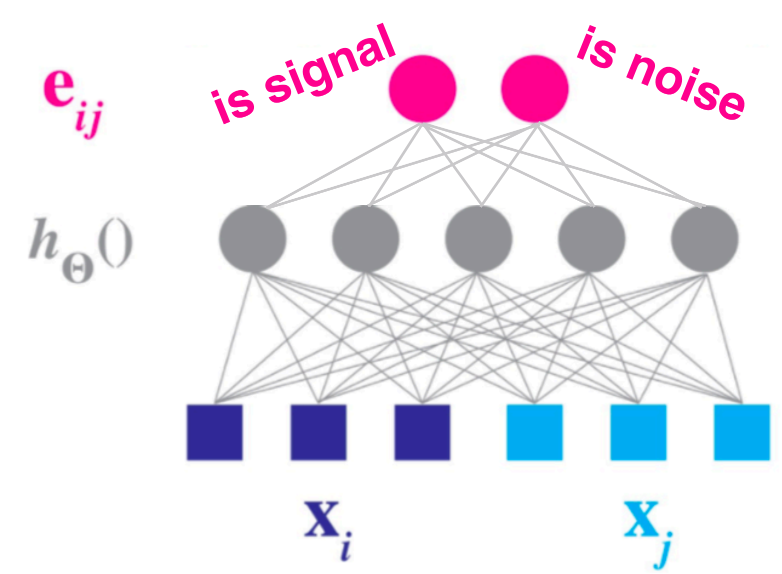
Noise

Actual particle



Generate edges between nodes
(preprocessing, kNN)

Classify into ‘true’ edges vs ‘false’ edges
(ML, 2 edge features)

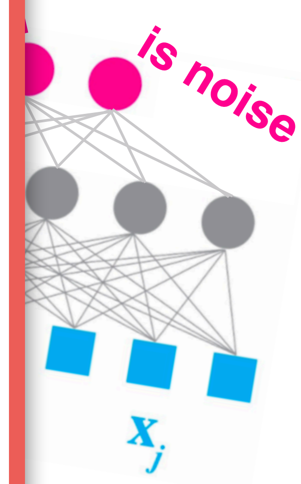


Graphical example

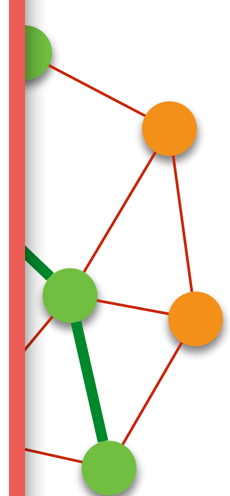
```python
class EdgeNet2(nn.Module):
    def __init__(self, input_dim=3, hidden_dim=8, output_dim=1, n_iters=1, aggr='add'):
        super(EdgeNet2, self).__init__()
        convnn = nn.Sequential(nn.Linear(2*(hidden_dim + input_dim), (3*hidden_dim + 2*input_dim) // 2),
                               nn.ReLU(),
                               nn.Dropout(),
                               nn.Linear((3*hidden_dim + 2*input_dim) // 2, hidden_dim),
                               nn.ReLU()
        )

        self.n_iters = n_iters

        self.inputnet =  nn.Sequential(
            nn.Linear(input_dim, hidden_dim),
            nn.BatchNorm1d(hidden_dim),
            nn.Tanh()
        )

        self.edgenetwork = nn.Sequential(nn.Linear(2*(n_iters*hidden_dim+input_dim), output_dim),
                                         nn.Sigmoid())

        self.nodenetwork = EdgeConv(nn=convnn,aggr=aggr)

    def forward(self, data):
        X = data.x
        H = self.inputnet(X)
        data.x = torch.cat([H, X], dim=-1)
        H_cat = X
        for i in range(self.n_iters):
            H = self.nodenetwork(data.x, data.edge_index)
            H_cat = torch.cat([H, H_cat], dim=-1)
            data.x = torch.cat([H, X], dim=-1)
        row,col = data.edge_index
        return self.edgenetwork(torch.cat([H_cat[row], H_cat[col]], dim=-1)).squeeze(-1)
```

See github

Actual par



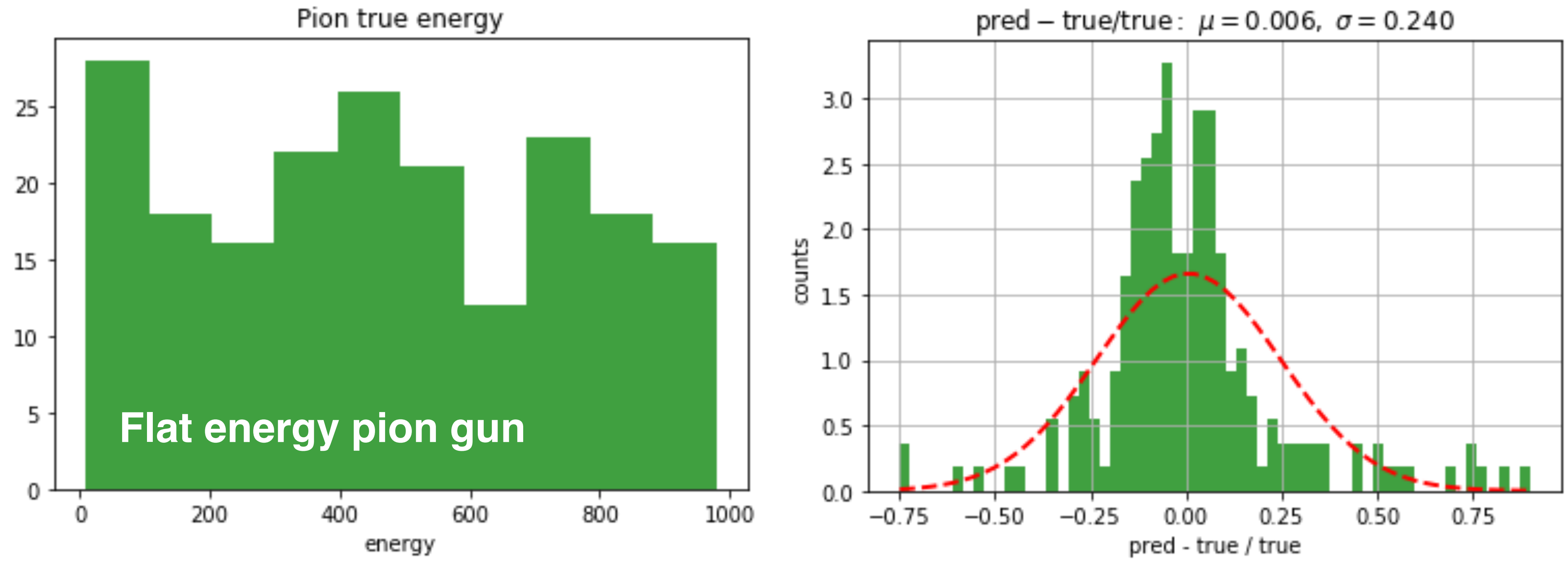
me
eatures)
is example



Updat
node fe
using
edge fe

Regression

- Starting to produce physics
  - First ever training of this regression in HEP on only **600** events, 200 testing events, and 14 epochs:

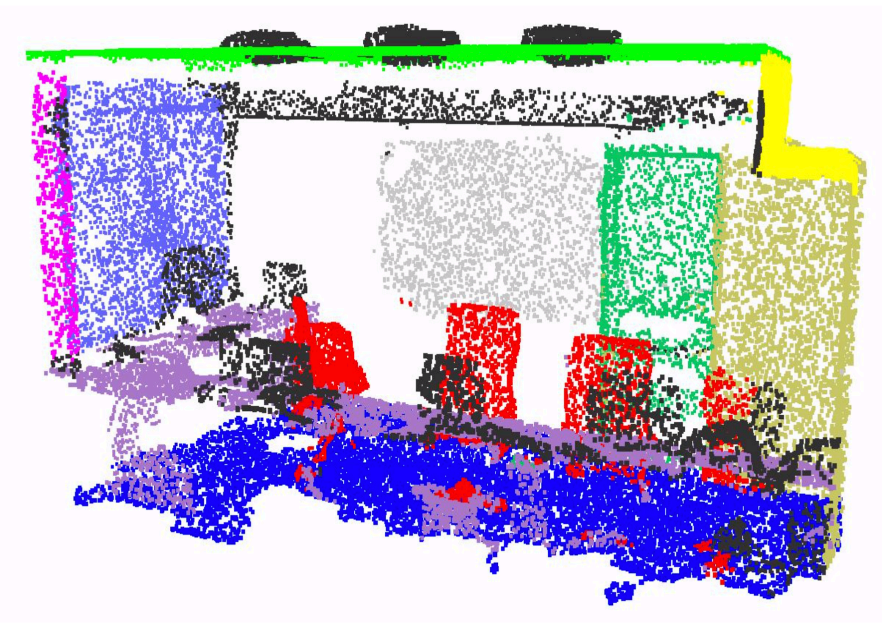


- Soon training with better statistics, but for the limited statistics already does a reasonable job
- Same regression method could be useful for other *unordered set* --> *regressed values* problems

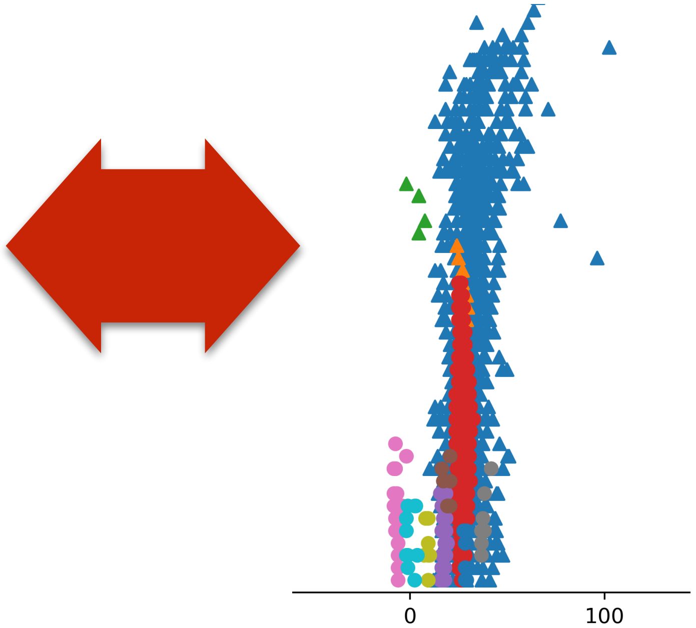
PointNet

[1612.00593]

Semantic segmentation



Clustering



- **PointNet:** one of the pioneering deep neural network approach to work with point clouds

- **Invariance under permutation of points:** Apply a *symmetrizing function* on the point cloud that yields the same output independent of order

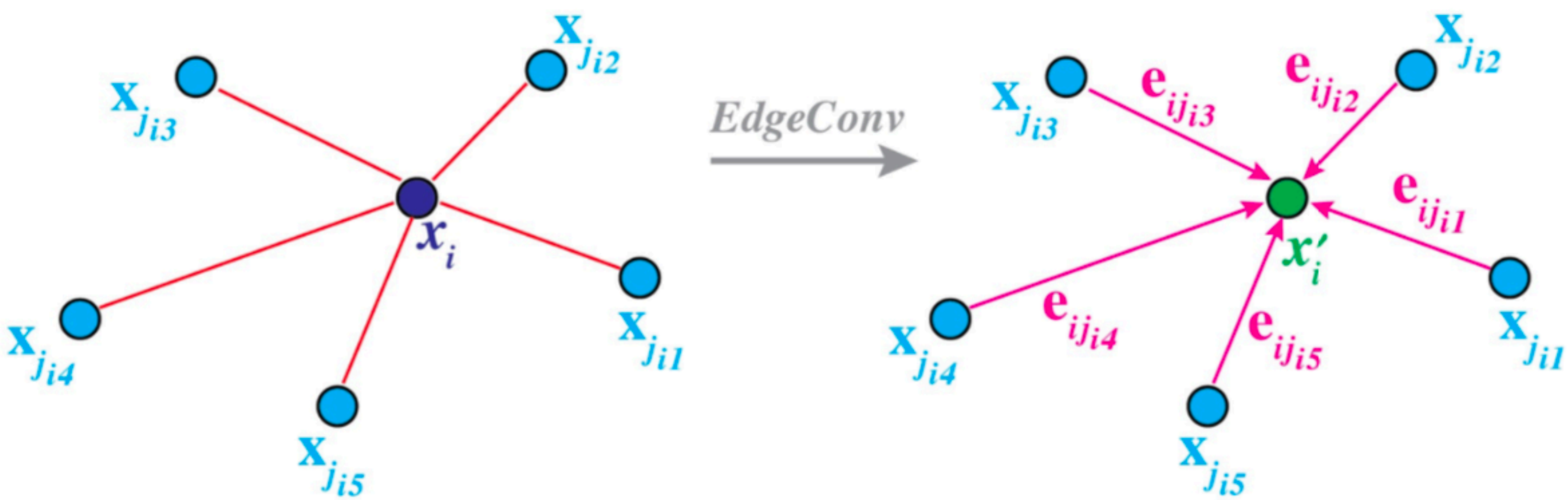
- **Invariance under rotation/translation**

- Need both **local** geometry and **global** semantics to do object classification

    - PointNet asks "what is so different about a point far away", but not "what is similar about points near me"

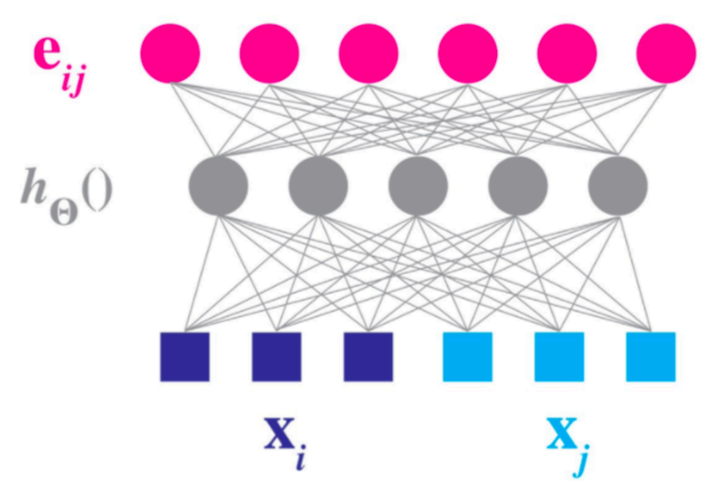
Edge Convolution

[1801.07829]



- Update $x_i \rightarrow x_i'$ by using **edge features**
  - i.e. learned features of the edges that connects x_i with its neighbors
  - Still independent of ordering of points, but uses **local geometry**
  - '**Convolutional**' as the operation is applied point by point to obtain **x**'
- Calculate edge features simply with e.g. a MLP which takes the node features as input

$$\mathbf{x}_i' = \mathop{\square}_{j:(i,j)\in\mathcal{E}} h_{\Theta}(\mathbf{x}_i, \mathbf{x}_j)$$



Can set dim of edge feature vector