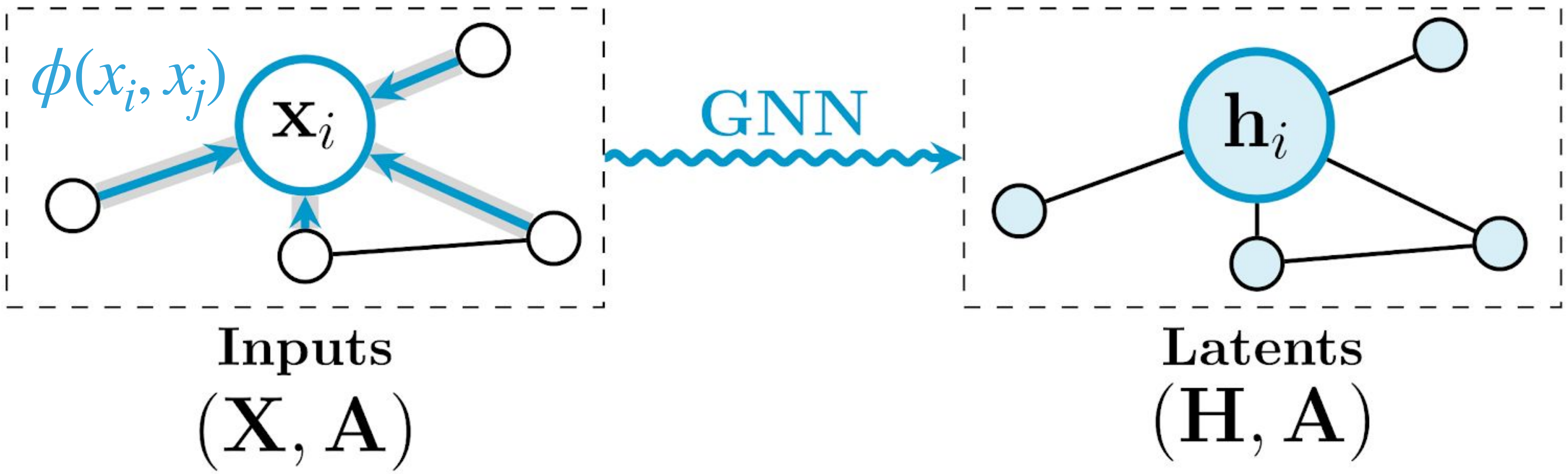
PHYS 139/239: Machine Learning in Physics

Lecture 12: More graph neural networks & transformers

Javier Duarte — February 16, 2023

Recap: Message passing

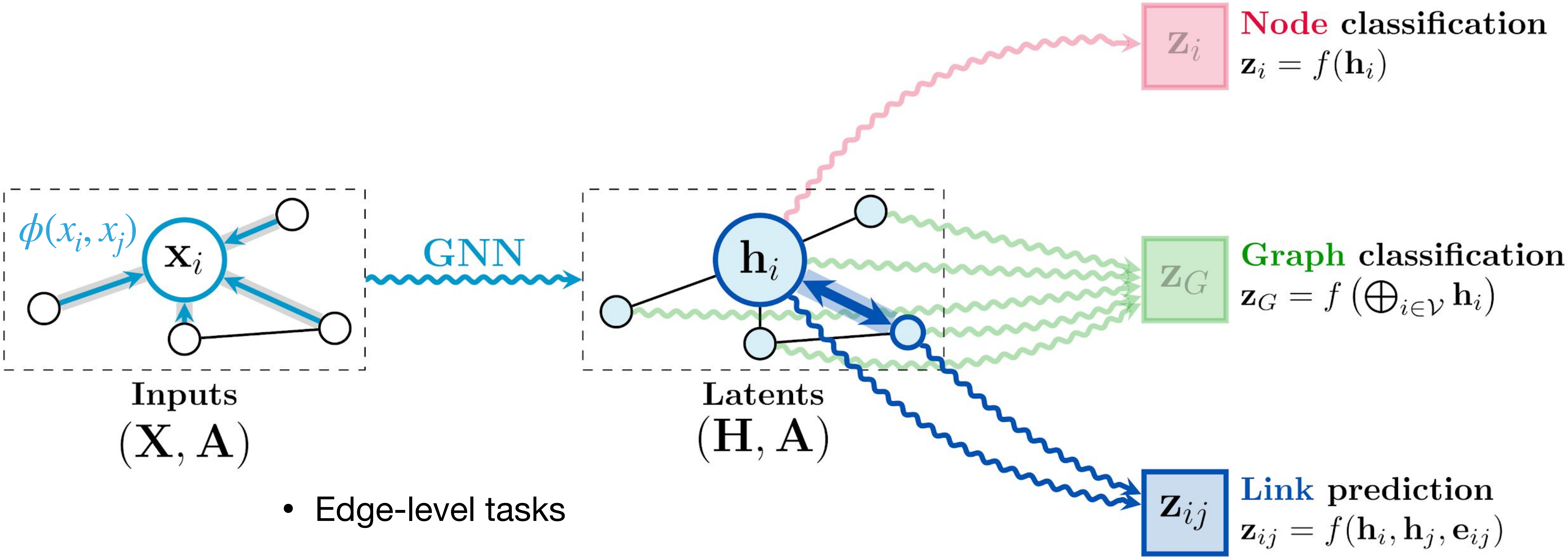


"message passing"

Recap: GNN tasks

Source: https://youtu.be/uF53xsT7mjc

- Node-level tasks
  - Identify "pileup" particles
- Graph-level tasks
  - Jet tagging

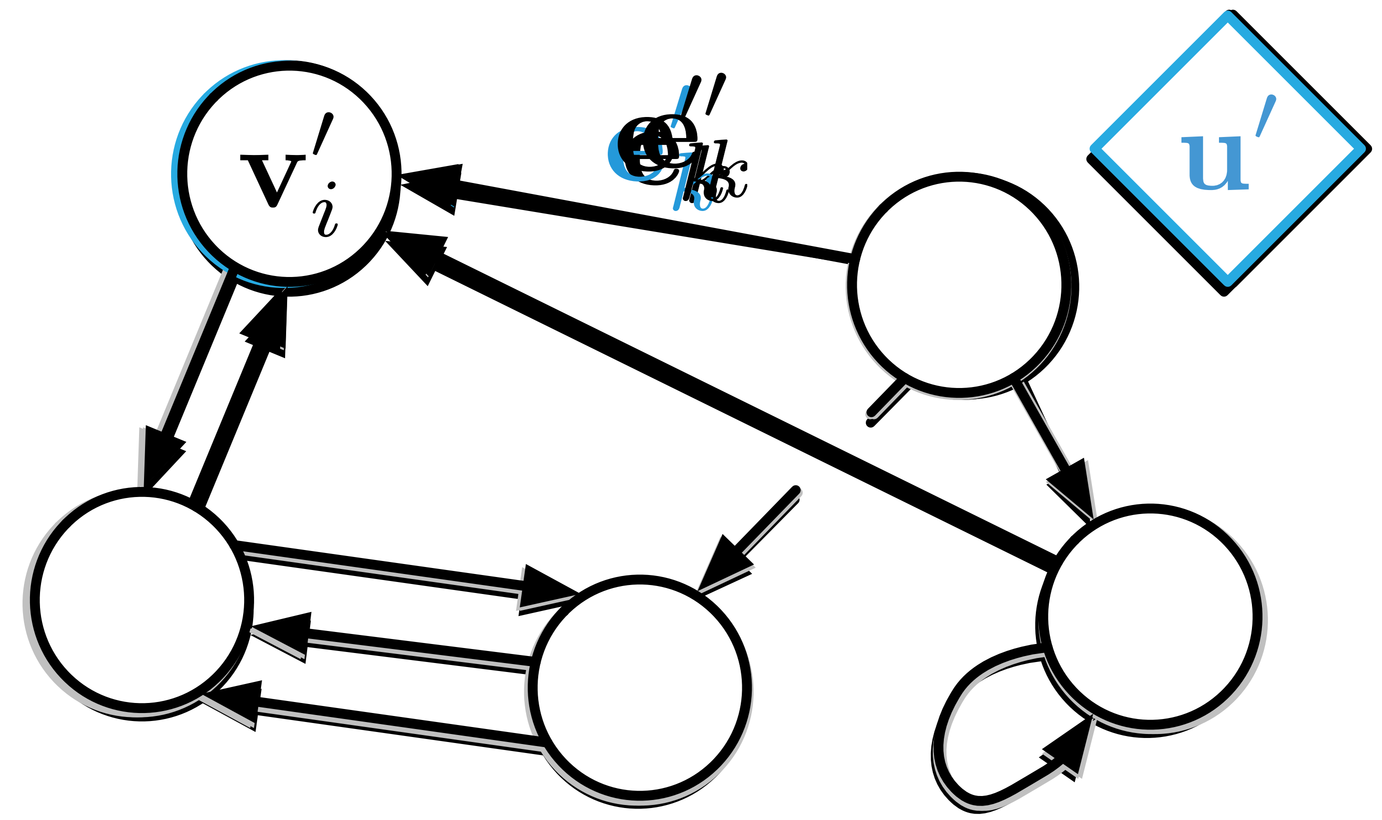


- Edge-level tasks
  - Identify good track candidates

Recap: GNNs*

*One framework for GNNs: arXiv:1806.01261

- GNNs are graph-to-graph mapping (in this case holding structure fixed)
- Inference divided into three parts: edge block, node block, global block



$\mathbf{e}'_k$: message computed for edge k connecting nodes r_k, s_k

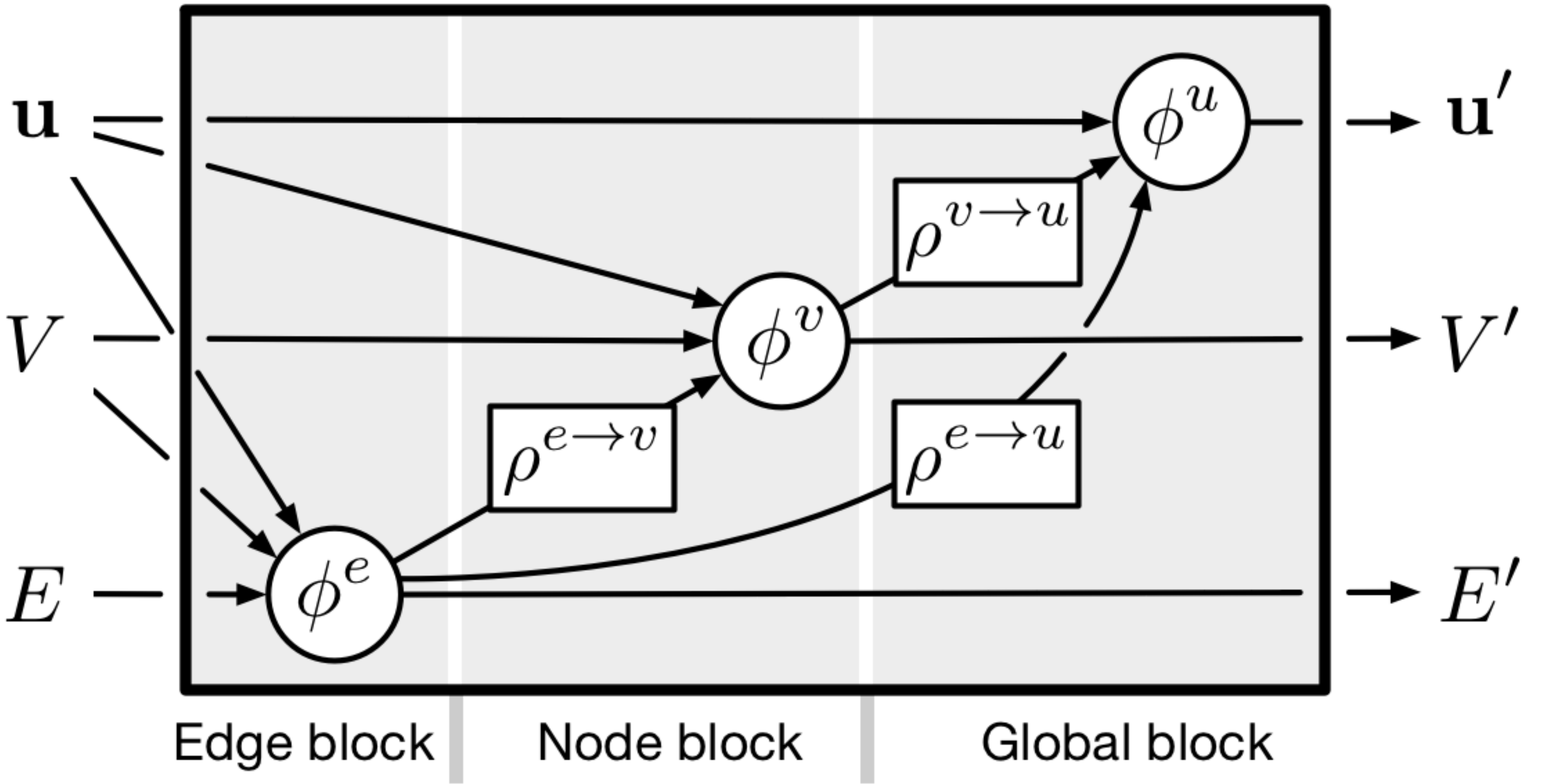
$\mathbf{v}'_i$: node feature update based on aggregated messages and previous features

$\mathbf{u}'$: global feature update based on aggregated, updated node and edge features

$$\mathbf{e}'_k = \phi^e(\mathbf{e}_k, \mathbf{v}_{r_k}, \mathbf{v}_{s_k}, \mathbf{u}) \qquad \bar{\mathbf{e}}'_i = \rho^{e\rightarrow v}(E'_i)$$

$$\mathbf{v}'_i = \phi^v\left(\bar{\mathbf{e}}'_i, \mathbf{v}_i, \mathbf{u}\right) \qquad \bar{\mathbf{e}}' = \rho^{e\rightarrow u}(E')$$

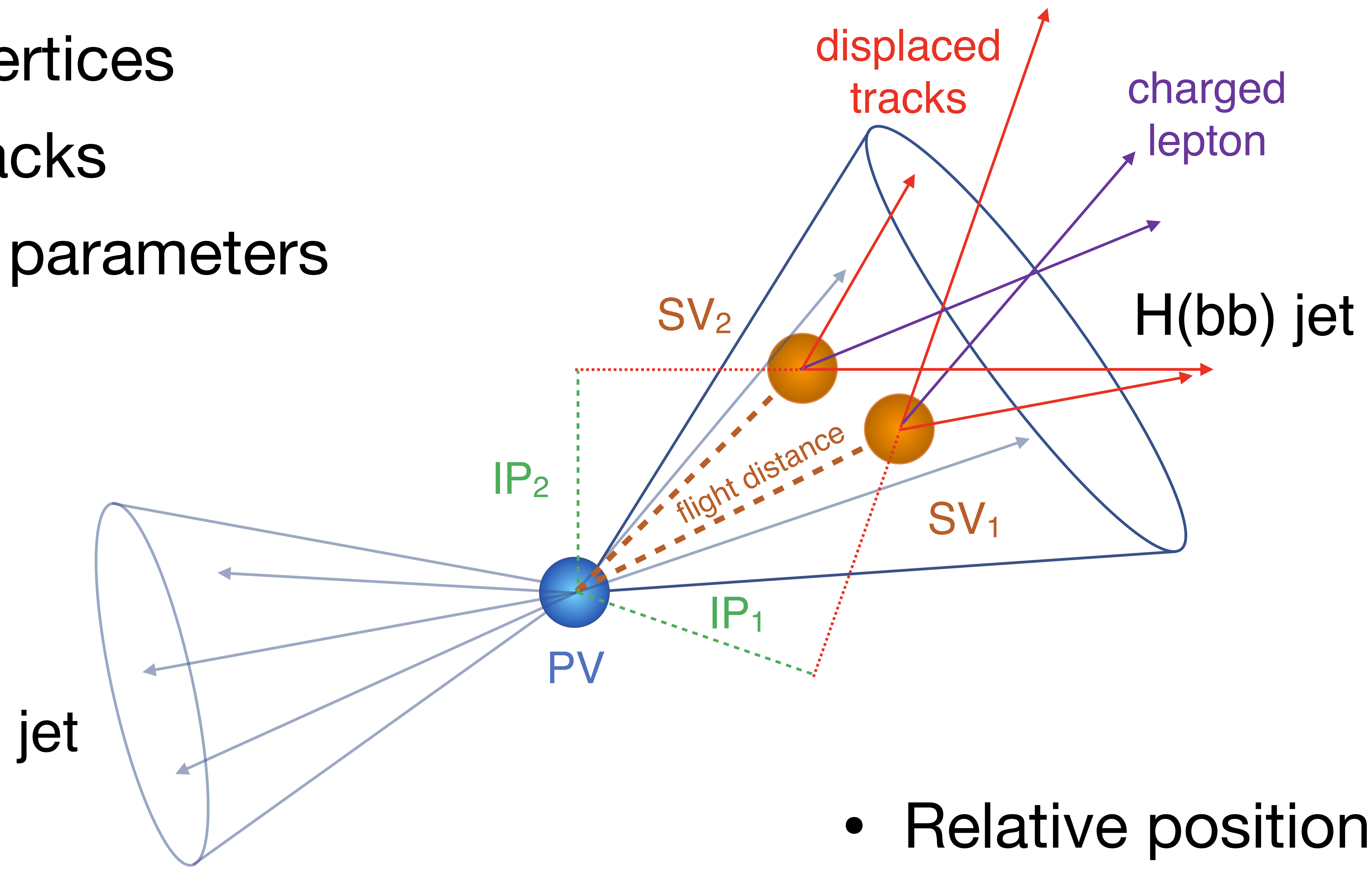
$$\mathbf{u}' = \phi^u(\bar{\mathbf{e}}', \bar{\mathbf{v}}', \mathbf{u}) \qquad \bar{\mathbf{v}}' = \rho^{v\rightarrow u}(V')$$



Basics of H(bb) tagging

b hadrons have long lifetimes:
travel O(mm) before decay!

- Handles:
  - secondary vertices
  - displaced tracks
  - large impact parameters
  - soft leptons



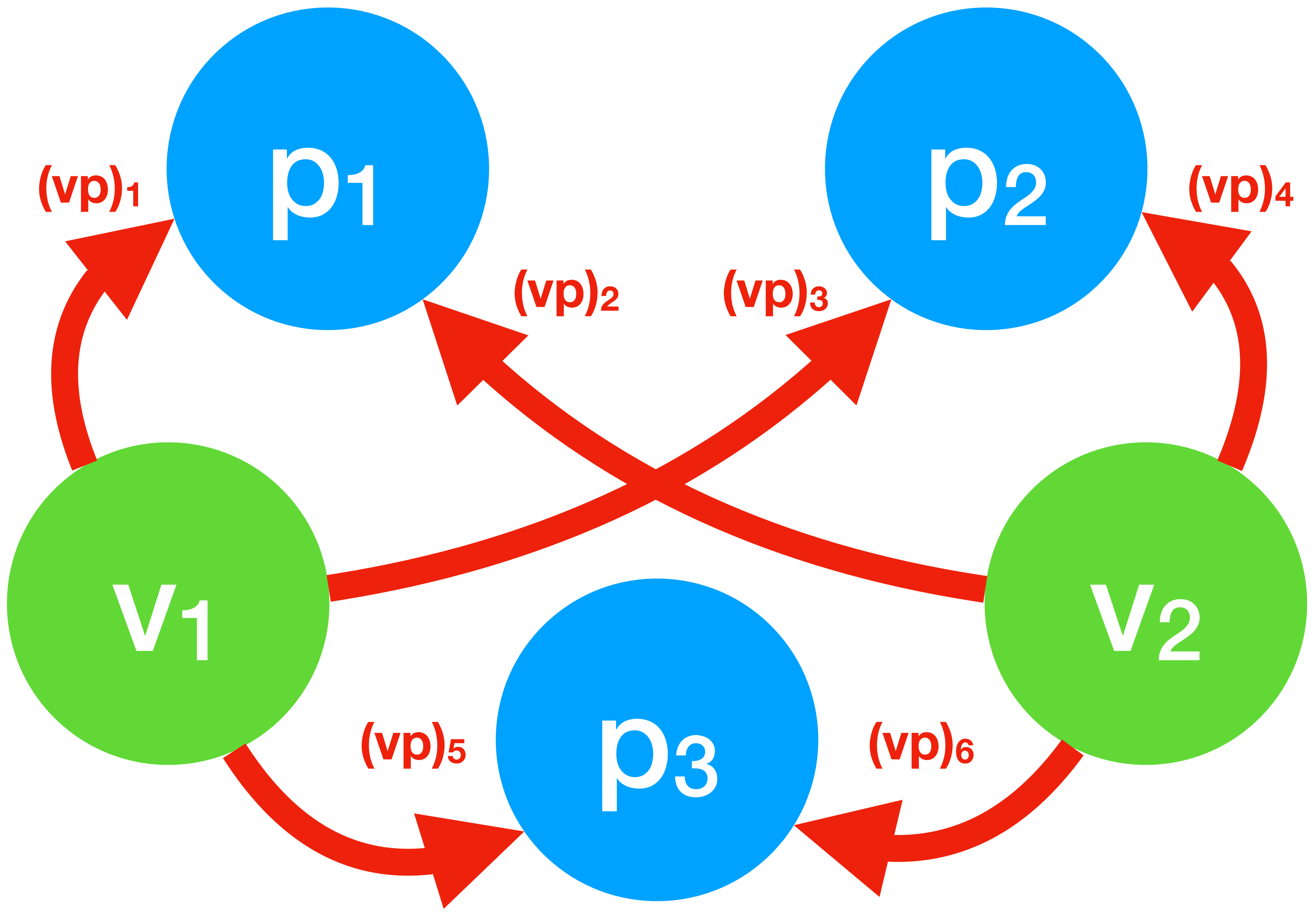
- Relative positions of SVs

Particles and vertices: two graphs

arXiv:1909.12285

$$p_i = [p_T^{\mathrm{rel}}, \phi^{\mathrm{rel}}, \eta^{\mathrm{rel}}, \ldots, d_{3D}, \mathrm{cov}(p_T, p_T), \ldots]$$

- Particles (i.e. tracks) and vertices are two separate inputs with different feature vectors (*heterogenous graph*)
- GNNs typically consider a *homogenous graph* (e.g. particle-particle graph)
- Vertex-particle graph can also be considered
- Combined GNN can consider both by constructing two separate graphs



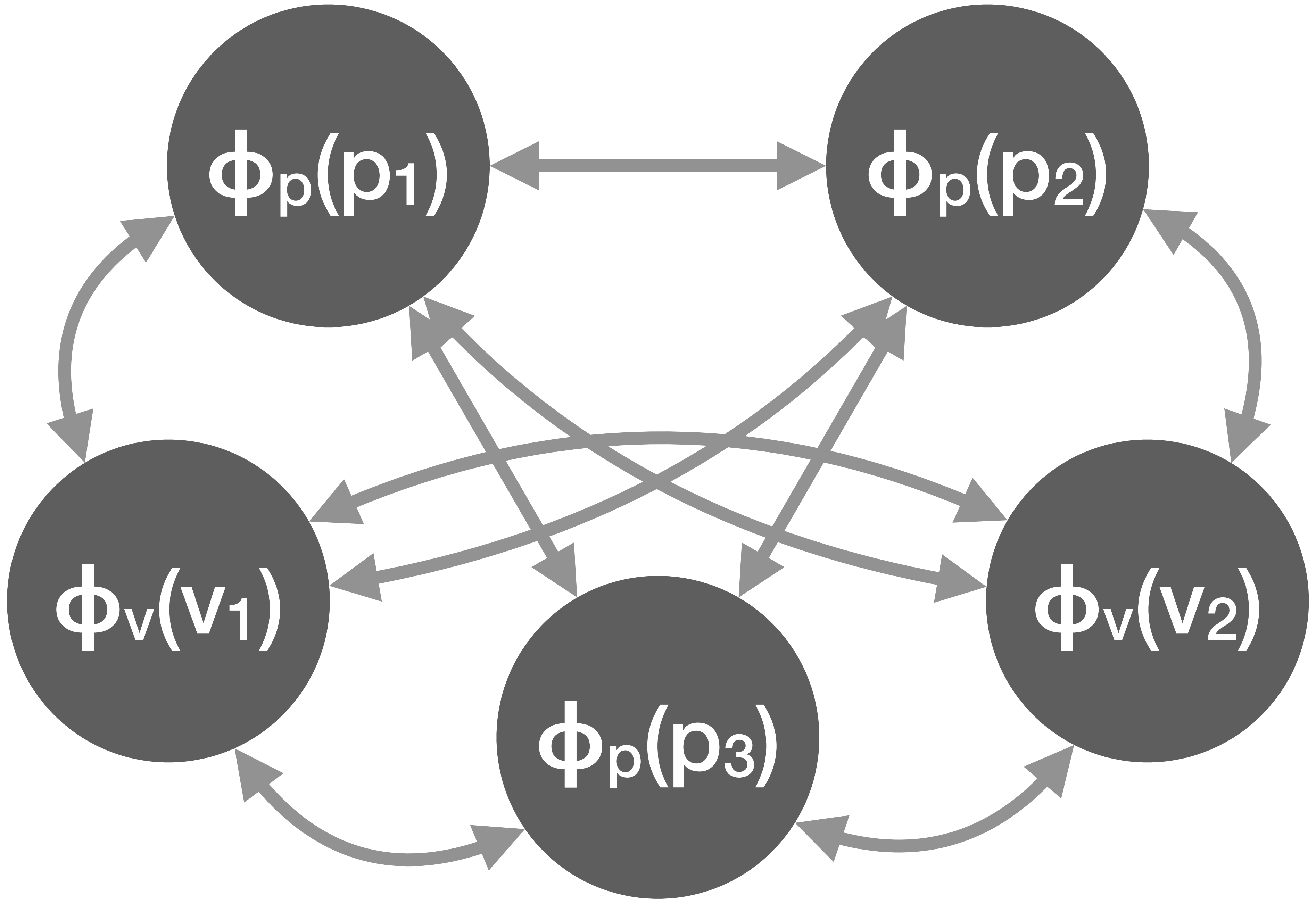
$$v_i = [p_T^{\mathrm{rel}}, \phi^{\mathrm{rel}}, \eta^{\mathrm{rel}}, \ldots, n_{\mathrm{tracks}}, \cos\theta_{\mathrm{PV}}, \ldots]$$

Particles and vertices: two graphs

arXiv:1909.12285

$$p_i = [p_{\mathrm{T}}^{\mathrm{rel}}, \phi^{\mathrm{rel}}, \eta^{\mathrm{rel}}, \ldots, d_{3\mathrm{D}}, \mathrm{cov}(p_{\mathrm{T}}, p_{\mathrm{T}}), \ldots]$$

- Alternatively, after embedding feature vectors in a common *latent space* (via a NN), *combined graph* can be constructed

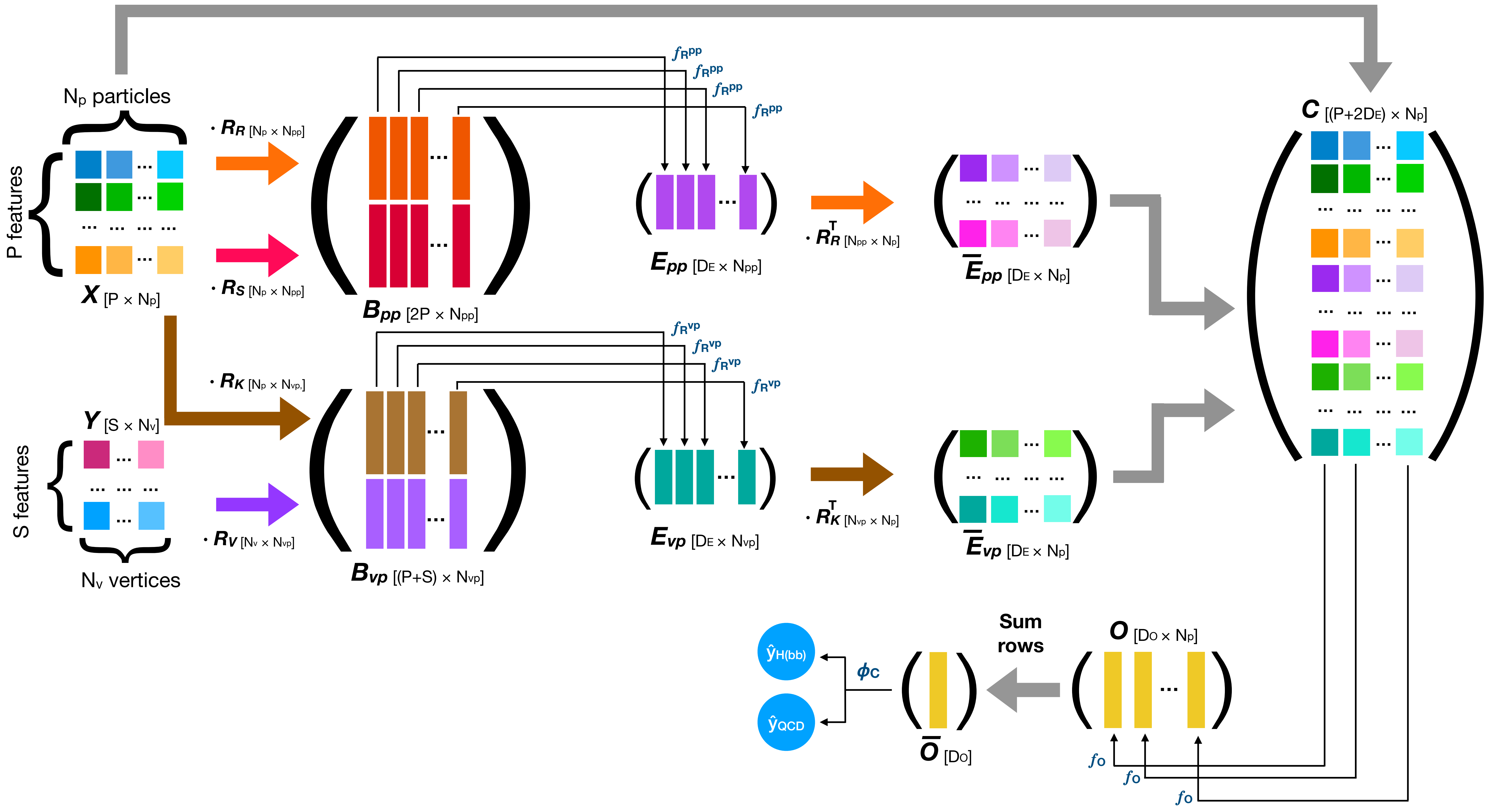


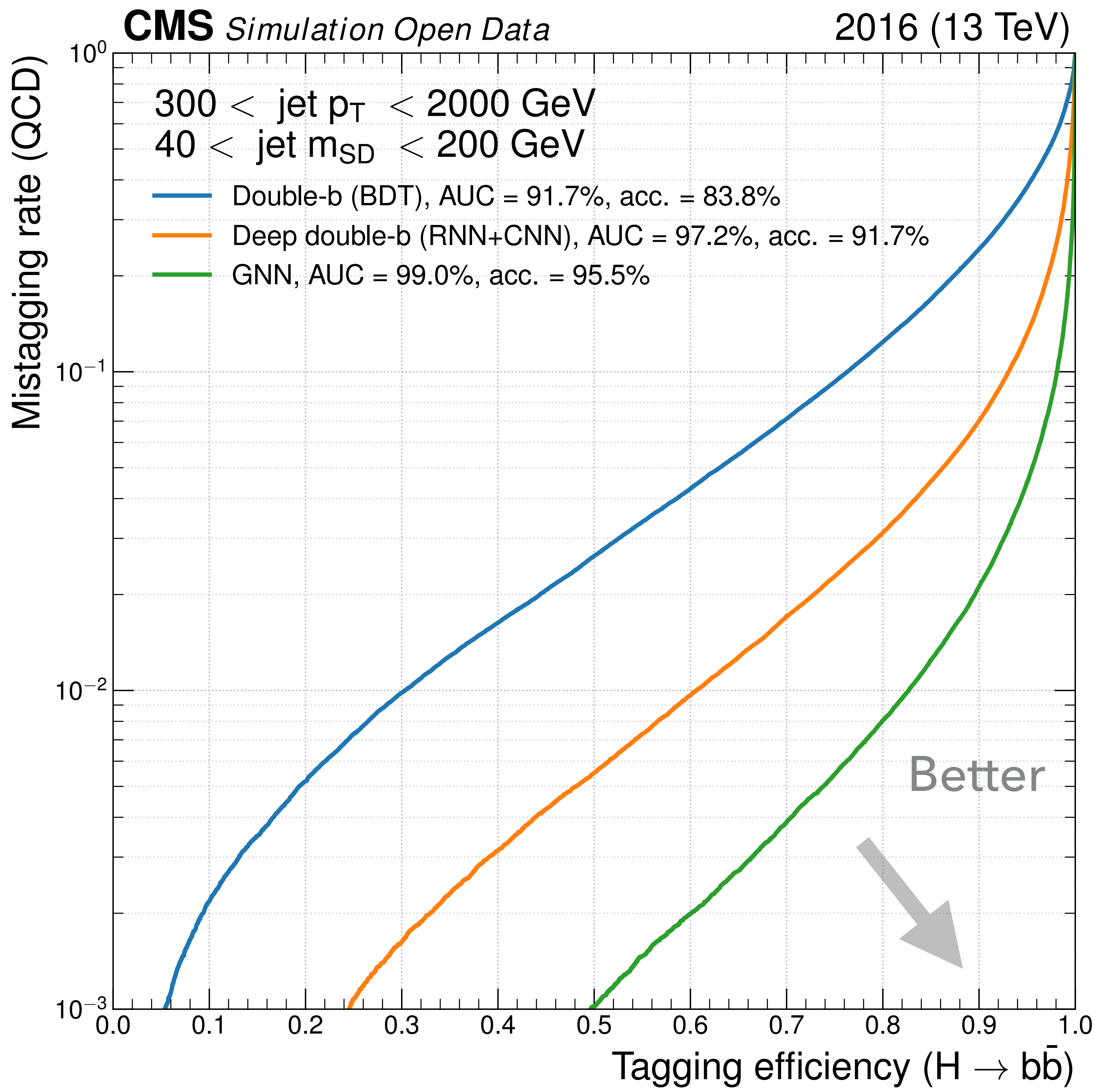
$$v_i = [p_{\mathrm{T}}^{\mathrm{rel}}, \phi^{\mathrm{rel}}, \eta^{\mathrm{rel}}, \ldots, n_{\mathrm{tracks}}, \cos\theta_{\mathrm{PV}}, \ldots]$$

GNN model and performance

arXiv:1909.12285

- Edge convolutions for particle-particle and particle-vertex connections update particle features; summed particle features used to predict H(bb) or QCD prob.
- GNN improves on previous methods





Hands-on exercise

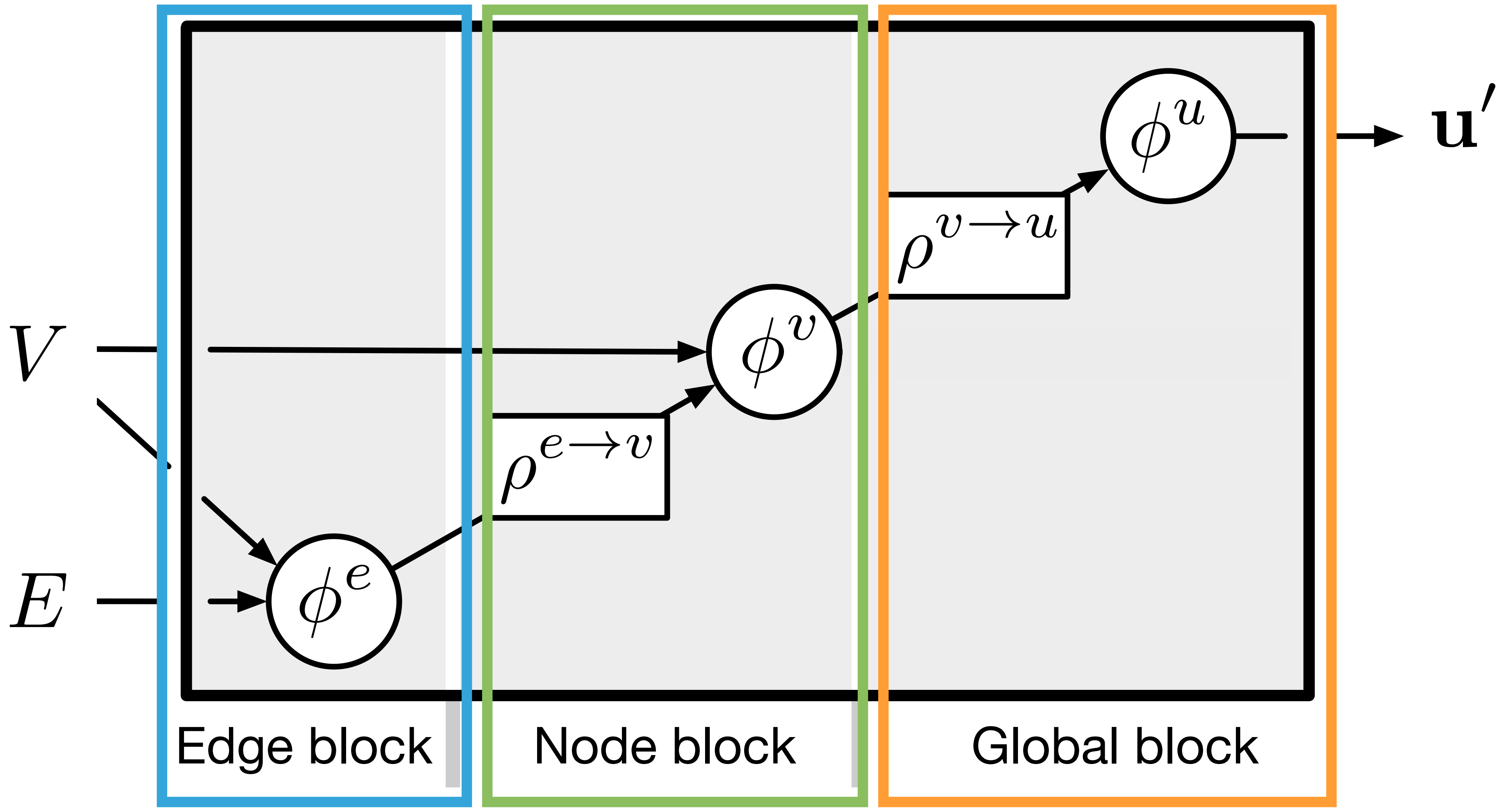
- GNN with PyTorch Geometric: https://jduarte.physics.ucsd.edu/phys139_239/06_Graph_Data_GNN.html

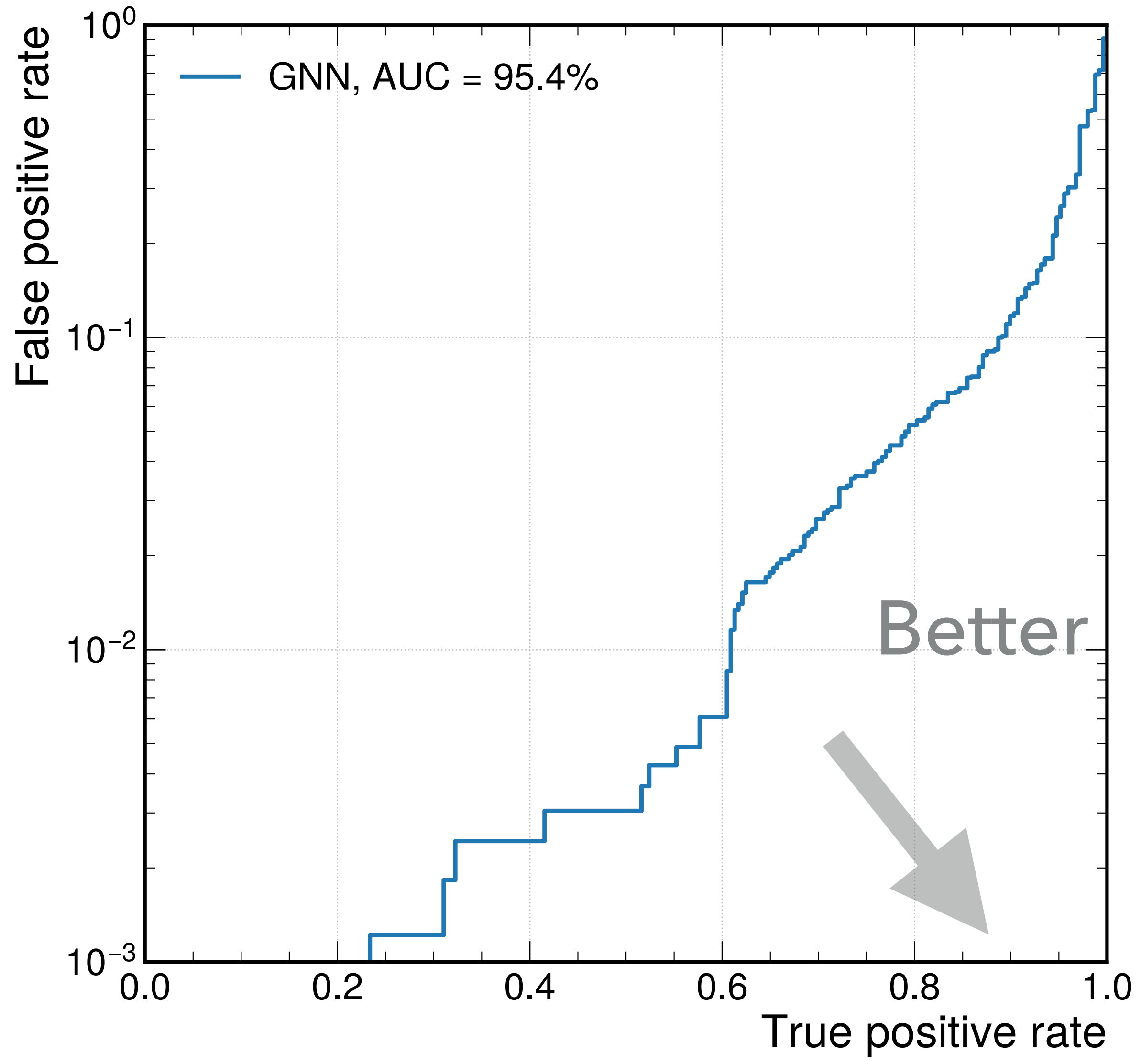
```python
class EdgeBlock(torch.nn.Module):
    def __init__(self):
        super(EdgeBlock, self).__init__()
        self.edge_mlp = Seq(Lin(inputs*2, hidden),
                            BatchNorm1d(hidden),
                            ReLU(),
                            Lin(hidden, hidden))

    def forward(self, src, dest, edge_attr, u, batch):
        out = torch.cat([src, dest], 1)
        return self.edge_mlp(out)
```

```python
class GlobalBlock(torch.nn.Module):
    def __init__(self):
        super(GlobalBlock, self).__init__()
        self.global_mlp = Seq(Lin(hidden, hidden),
                              BatchNorm1d(hidden),
                              ReLU(),
                              Lin(hidden, outputs))

    def forward(self, x, edge_index, edge_attr, u, batch):
        out = scatter_mean(x, batch, dim=0)
        return self.global_mlp(out)
```

```python
class NodeBlock(torch.nn.Module):
    def __init__(self):
        super(NodeBlock, self).__init__()
        self.node_mlp_1 = Seq(Lin(inputs+hidden, hidden),
                              BatchNorm1d(hidden),
                              ReLU(),
                              Lin(hidden, hidden))
        self.node_mlp_2 = Seq(Lin(inputs+hidden, hidden),
                              BatchNorm1d(hidden),
                              ReLU(),
                              Lin(hidden, hidden))

    def forward(self, x, edge_index, edge_attr, u, batch):
        row, col = edge_index
        out = torch.cat([x[row], edge_attr], dim=1)
        out = self.node_mlp_1(out)
        out = scatter_mean(out, col, dim=0, dim_size=x.size(0))
        out = torch.cat([x, out], dim=1)
        return self.node_mlp_2(out)
```

```python
class InteractionNetwork(torch.nn.Module):
    def __init__(self):
        super(InteractionNetwork, self).__init__()
        self.interactionnetwork = MetaLayer(EdgeBlock(), NodeBlock(), GlobalBlock())
        self.bn = BatchNorm1d(inputs)

    def forward(self, x, edge_index, batch):
        x = self.bn(x)
        x, edge_attr, u = self.interactionnetwork(x, edge_index, None, None, batch)
        return u
```

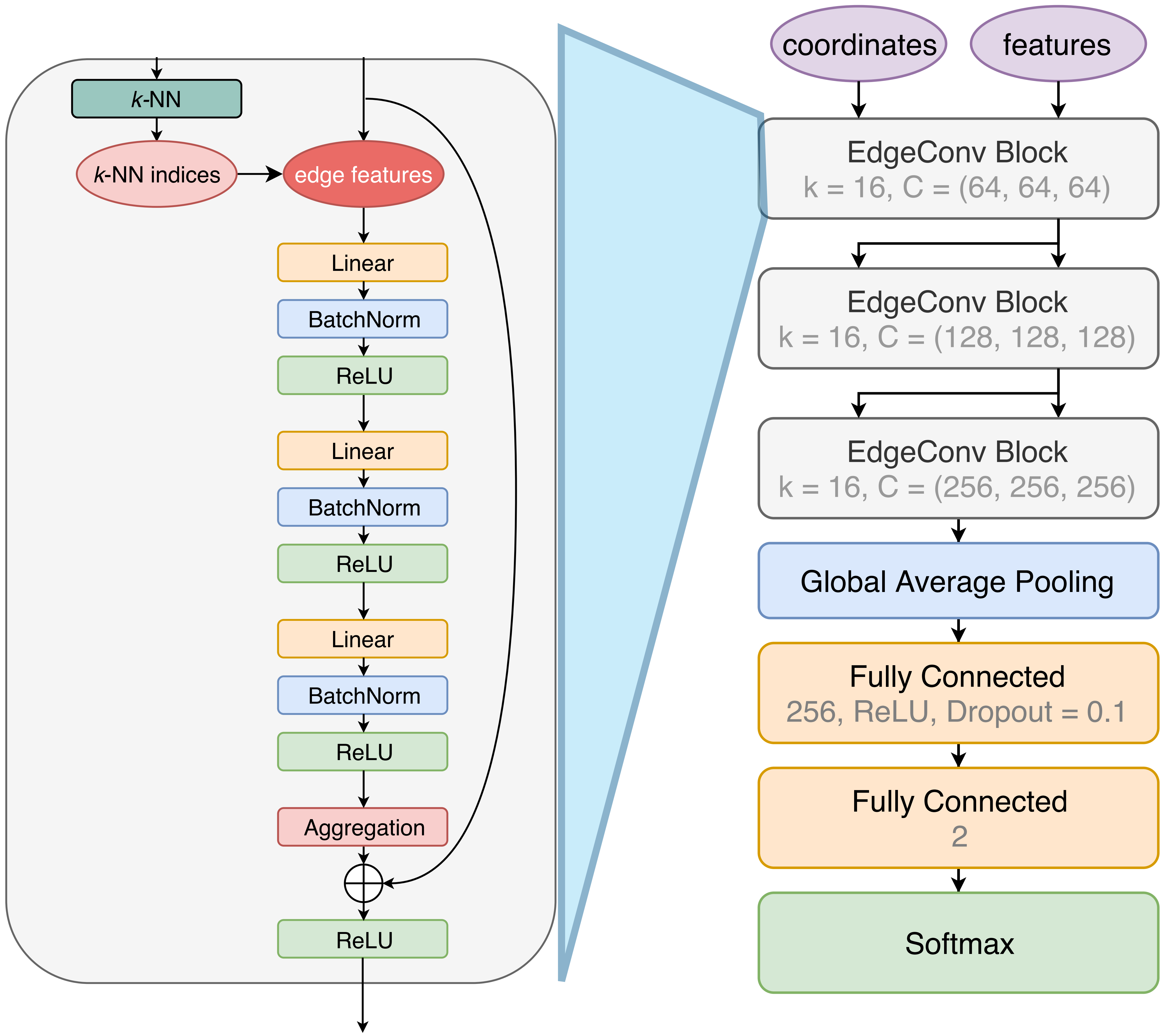


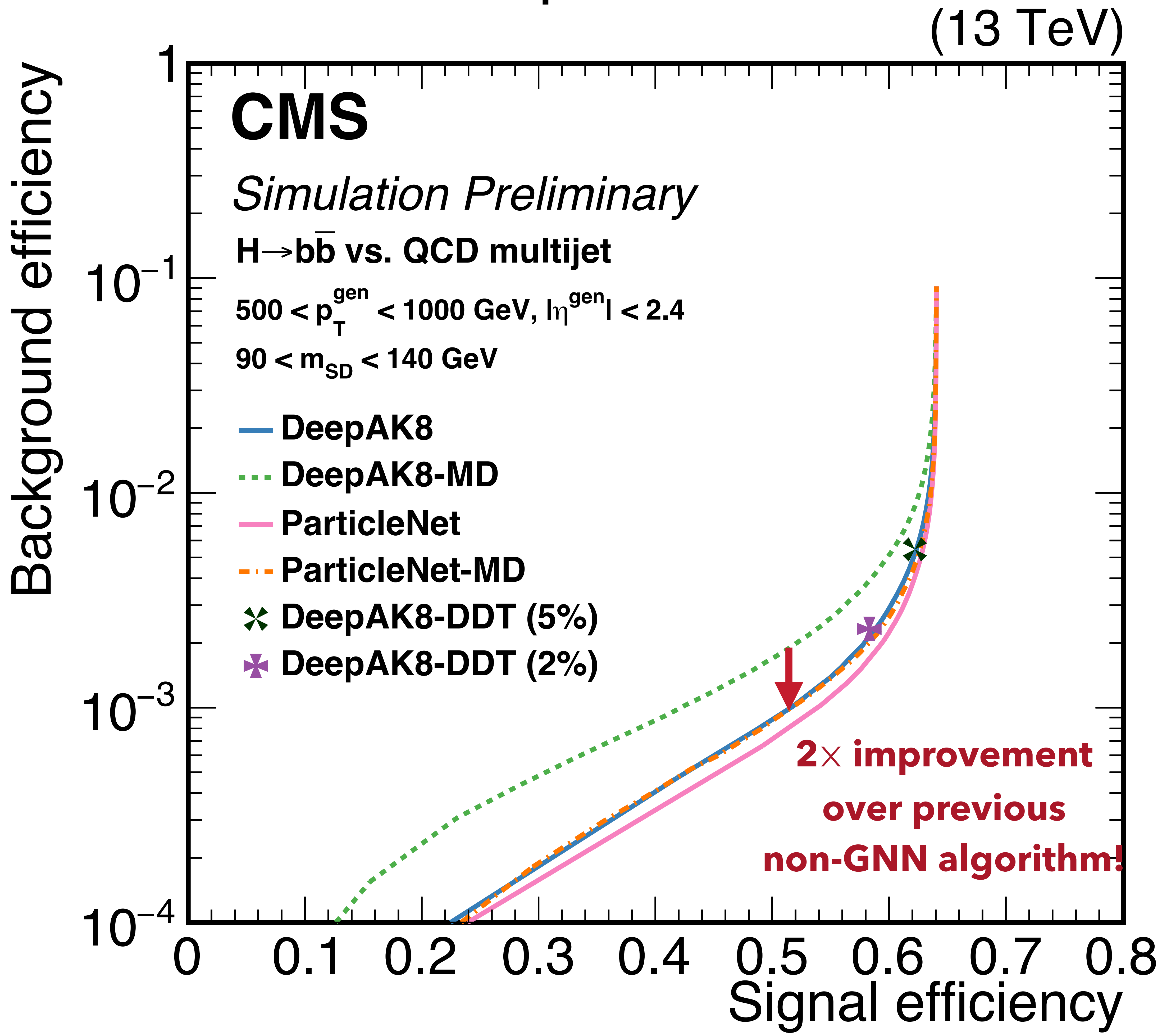


ParticleNet: DGCNN for jet tagging

arXiv:1902.08570
CMS-DP-2020-002

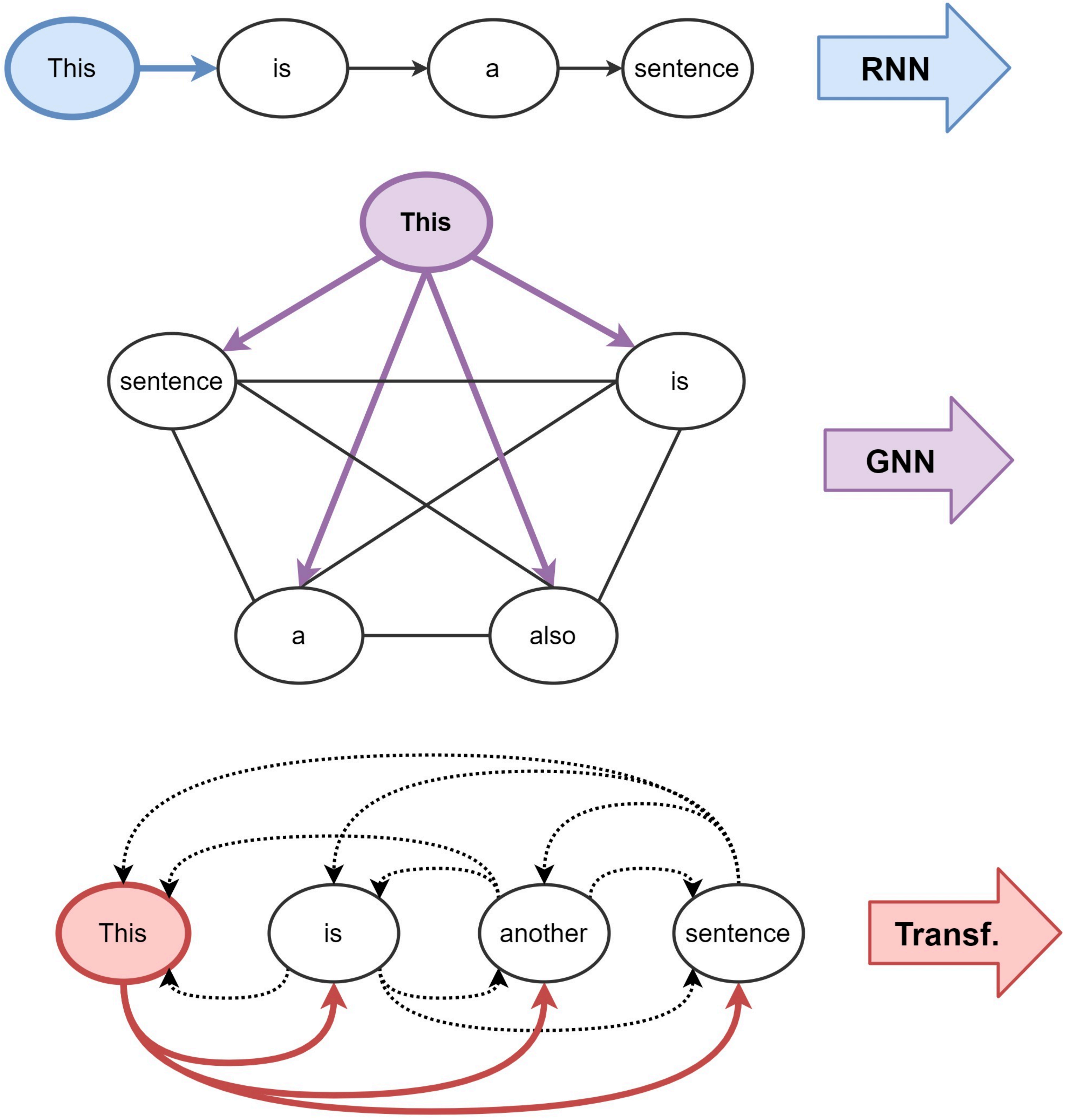
- ParticleNet, using "dynamic edge convolutions:" graph is constructed based on "closeness" in the latent space
- Identifies H(bb) with true positive rate of ~50% and false positive rate of 0.1%





Transformers

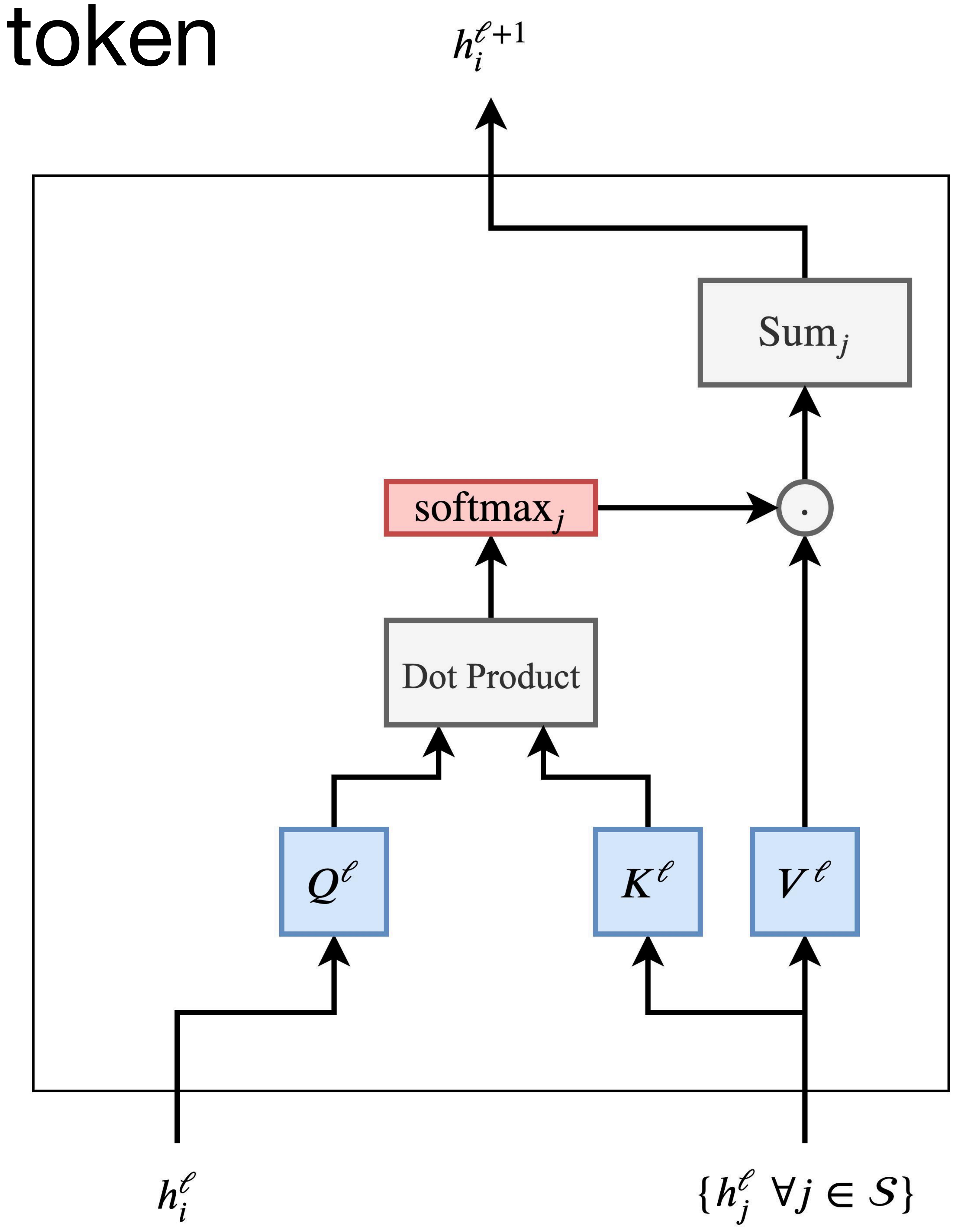
- Transformers are attention-based neural networks
- Generalization of an RNN
- Specific type of GNN (with a fully connected graph)



Attention

- Update the features of the ith token from the ℓth layer to the $(\ell + 1)$th layer
- Q, K, and V are learnable linear weights denoting the query, key, and value for the attention computation
- Attention mechanism is performed in parallel for each token to obtain their updated features in one shot
  - Faster than RNNs, which update features token-by-token

$$h_i^{\ell+1} = \text{Attention}\left(Q^\ell h_i^\ell, K^\ell h_j^\ell, V^\ell h_j^\ell\right)$$
$$= \sum_{j \in \mathcal{S}} \text{softmax}_j\left(Q^\ell h_i^\ell \cdot K^\ell h_j^\ell\right)\left(V^\ell h_j^\ell\right)$$



Multi-head attention

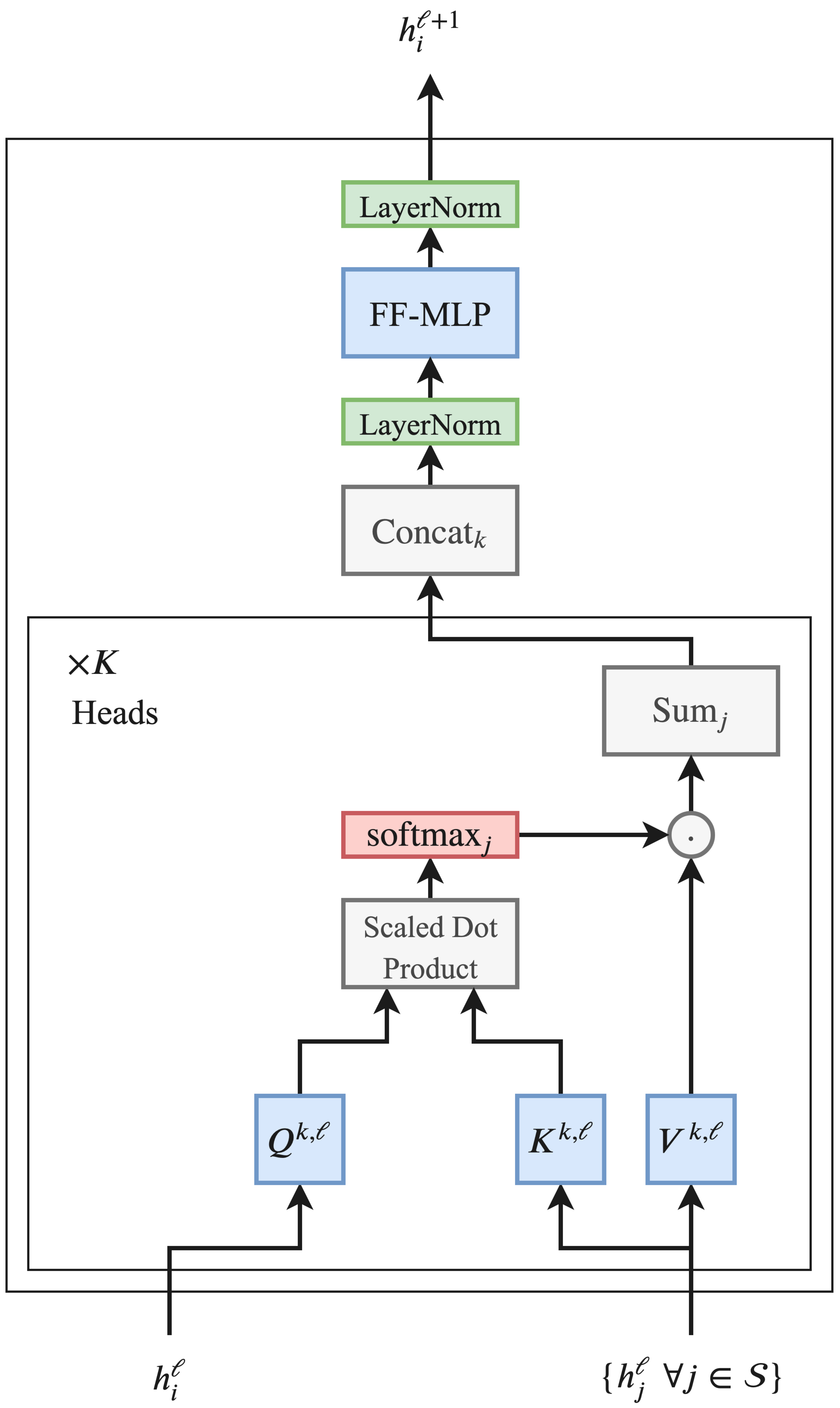
- Multiple heads allow the attention mechanism to to look at different transformations or aspects of the hidden features from the previous layer

- Layer normalization normalizes input across the features instead of normalizing input features across the batch dimension (as in batch normalization)

$$g_i^{\ell+1} = \text{Concat}\left(\text{head}_1, \ldots, \text{head}_K\right) O^{\ell},$$

$$\text{head}_k = \text{Attention}\left(Q^{k,\ell} h_i^{\ell}, K^{k,\ell} h_j^{\ell}, V^{k,\ell} h_j^{\ell}\right),$$

$$h_i^{\ell+1} = \text{LN}\left(\text{MLP}\left(\text{LN}\left(g_i^{\ell+1}\right)\right)\right)$$

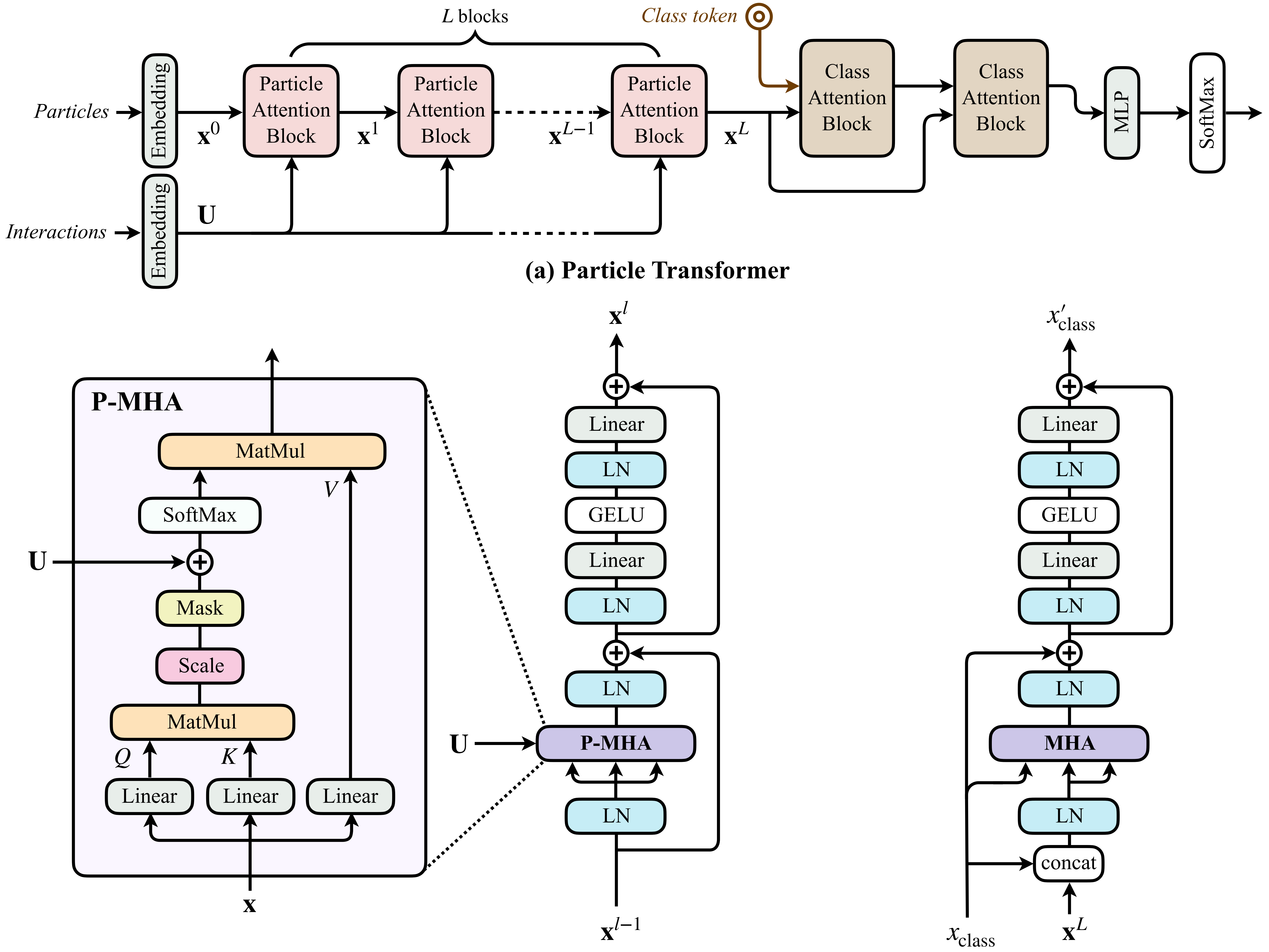


Particle transformer

arXiv:2202.03772

- Particle transformer (ParT) incorporates pairwise particle interactions in the attention mechanism and achieves higher tagging performance than a plain transformer and ParticleNet

	Accuracy	AUC	$\text{Rej}_{50\%}$	$\text{Rej}_{30\%}$
P-CNN	0.930	0.9803	201 ± 4	759 ± 24
PFN	—	0.9819	247 ± 3	888 ± 17
ParticleNet	0.940	0.9858	397 ± 7	1615 ± 93
JEDI-net (w/ $\sum O$)	0.930	0.9807	—	774.6
PCT	0.940	0.9855	392 ± 7	1533 ± 101
LGN	0.929	0.964	—	435 ± 95
rPCN	—	0.9845	364 ± 9	1642 ± 93
LorentzNet	0.942	0.9868	498 ± 18	2195 ± 173
ParT	0.940	0.9858	413 ± 16	1602 ± 81
ParticleNet-f.t.	0.942	0.9866	487 ± 9	1771 ± 80
ParT-f.t.	**0.944**	**0.9877**	$\mathbf{691 \pm 15}$	$\mathbf{2766 \pm 130}$



(a) Particle Transformer

(b) Particle Attention Block

(c) Class Attention Block

Next time

- Unsupervised learning for outlier detection