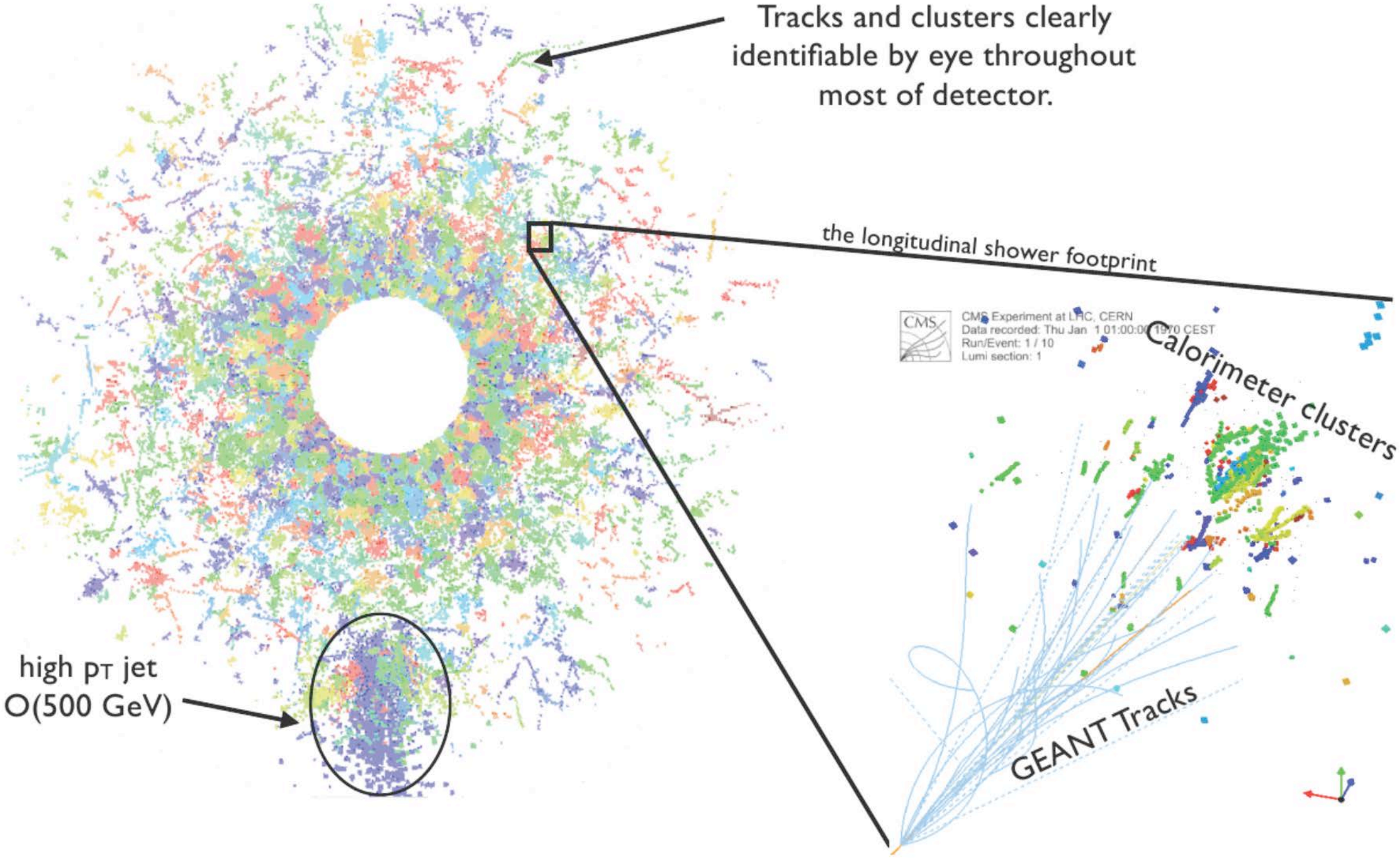
Graph nets and transformers: intro for HEP folks

Joosep Pata (joosep.pata@cern.ch)
KBFI, Tallinn, Estonia

Lecture at
3rd Terascale School of Machine Learning (Hamburg)
October 11, 2022



source: L. Gray

Data modalities

- Tabular: no particular relation between features of a sample
  - x = [$feat_1$, $feat_2$, ...]
- Image: features naturally embed into a 2D/3D Cartesian grid
  - x = Matrix(320,320,3)
- Set: each sample consists data points with features, no particular order
  - x = {p_1, p_2, p_3, ...}, p_n = [$feat_1$, $feat_2$, ...]
  - Set embaddable in 2D/3D: some features can be easily interpreted as x, y, z in a space

Data modalities

- Sequence: data points in a sample are naturally ordered (e.g in time)
  - $x = [p_1, p_2, p_3, ...]$, $p_n = [feat_1, feat_2, ...]$
- Heterogeneous set: data points in a sample have different features
  - $x = \{p_1, p_2, k_1, ...\}$, $p_n = [feat_1, feat_2]$, $k_n = [feat_3, feat_4, feat_5]$
- Graph: data points in a sample have meaningful, quantifiable, observable relations
  - $x = \{p_1, p_2, p_3, ...\}$, $A = \{A_{12}, A_{13}, A_{23}, ...\}$

Message Passing Networks

Operations on a graph

Jet tagging with a simple neural network

Interaction Networks

Graph Attention Networks

Graph Convolutional Networks

heterogeneous graphs

Dynamic Graphs: ParticleNet

Graph Structure Learning

normalization, regularization

Sparse graph models: GravNet and detector clustering

generative models

graph sampling

Scalable graph models: particle flow

transductive learning

very large graphs (>100000 nodes)

production use

Jet tagging

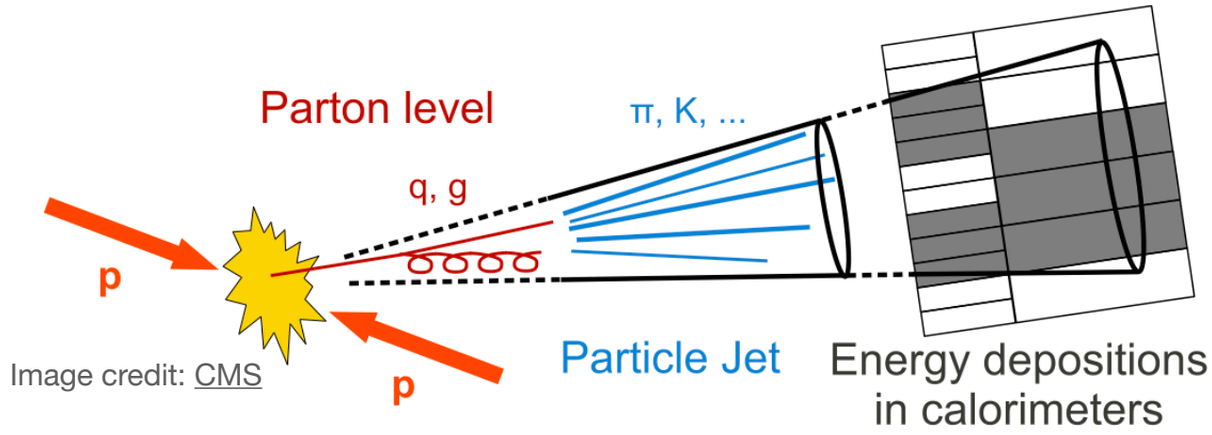
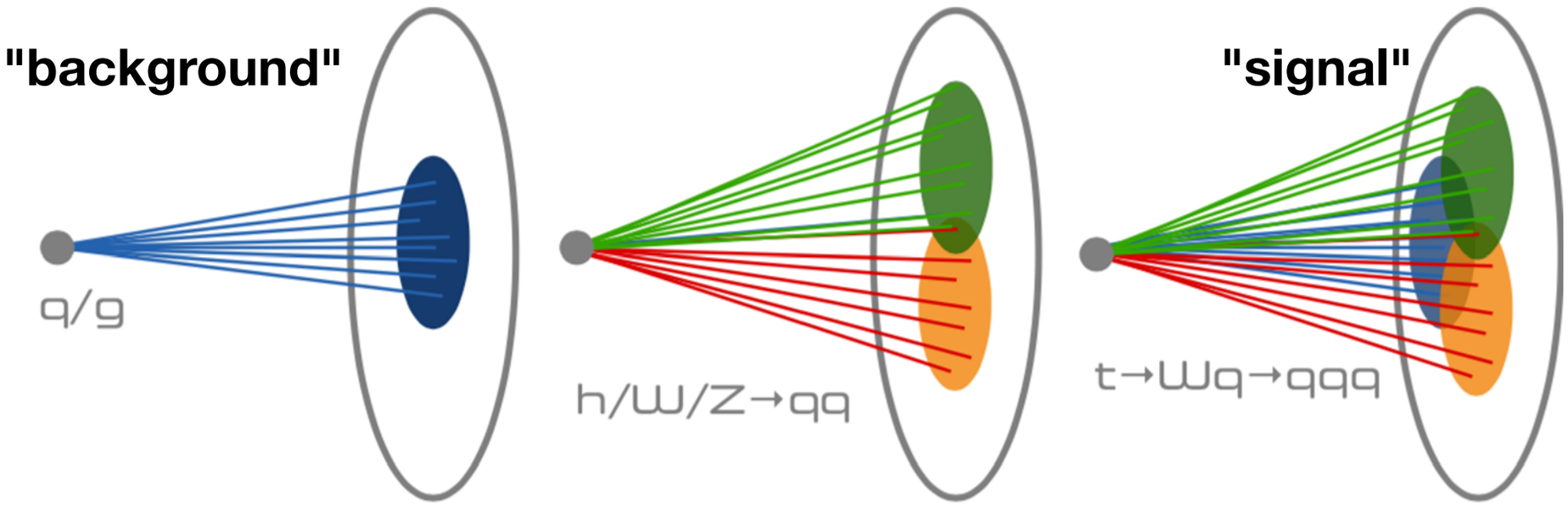


Image credit: CMS



jet constituents: set of particles
{..., (pT, η, ϕ, particle ID), ...}

→

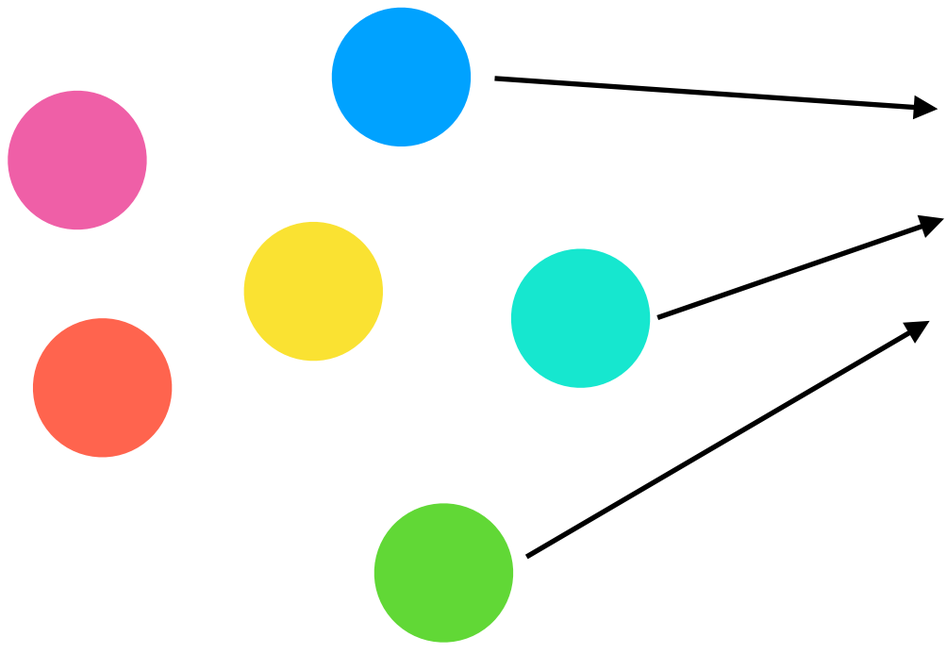
target:
jet originator particle ID

Moreno, E.A., Cerri, O., Duarte, J.M. *et al.* JEDI-net: a jet identification algorithm based on interaction networks. *Eur. Phys. J. C* **80,** 58 (2020). https://doi.org/10.1140/epjc/s10052-020-7608-4

Data representation

set of inputs with N constituents, M features

{..., (pT, η, ϕ, particle ID), ...}



jet constituents

feature matrix (N, M)

pT (GeV)	η	ϕ	particle ID
12.3	*1.2*	*0.5*	*pi+*
11.8	*1.24*	*0.45*	*K0*
10.4	*1.18*	*0.43*	*pi-*
9.8	*1.39*	*...*	*e-*
6.4	*...*	*...*	*...*
5.3	*...*	*...*	

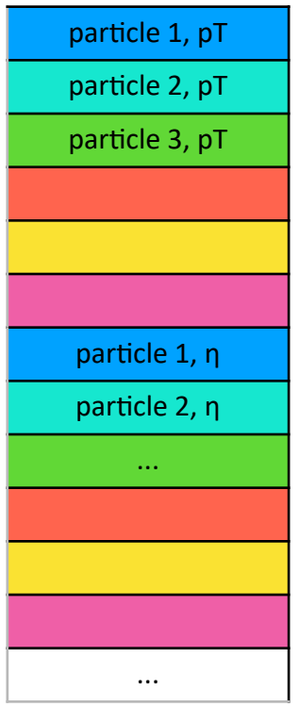
Set of feature vectors + ordering → feature matrix

Simple neural network

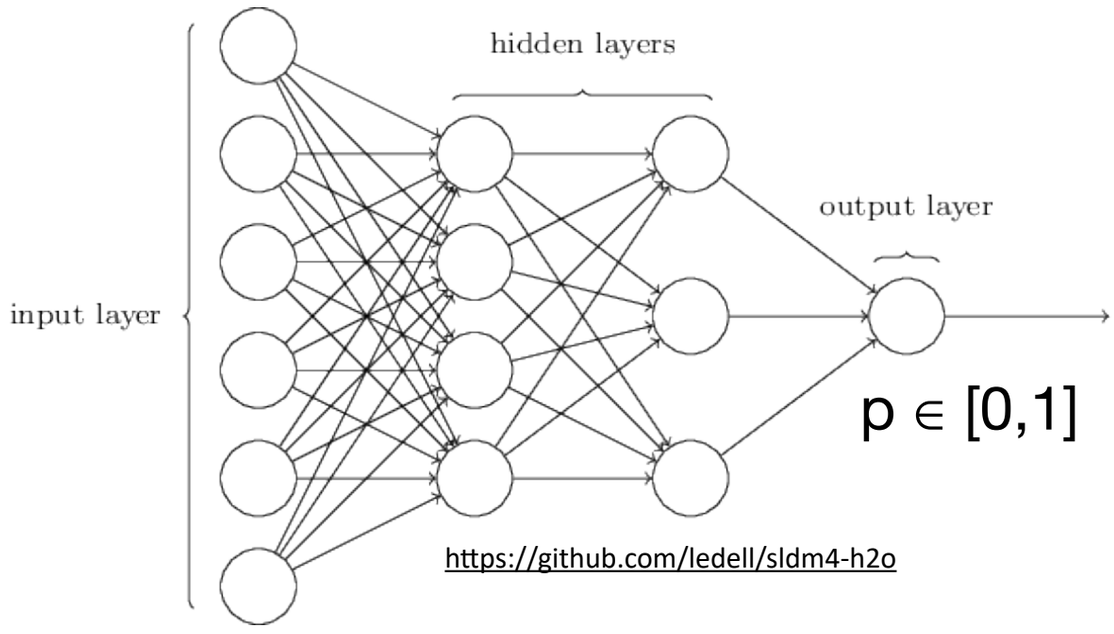
feature matrix (N, M)

pT (GeV)	η	ϕ	particle ID
12.3	1.2	0.5	pi+
11.8	1.24	0.45	K0
10.4	1.18	0.43	pi-
9.8	1.39	...	e-
6.4	...	...	...
5.3	...	...	

flat NxM feature vector



map feature vector to an output



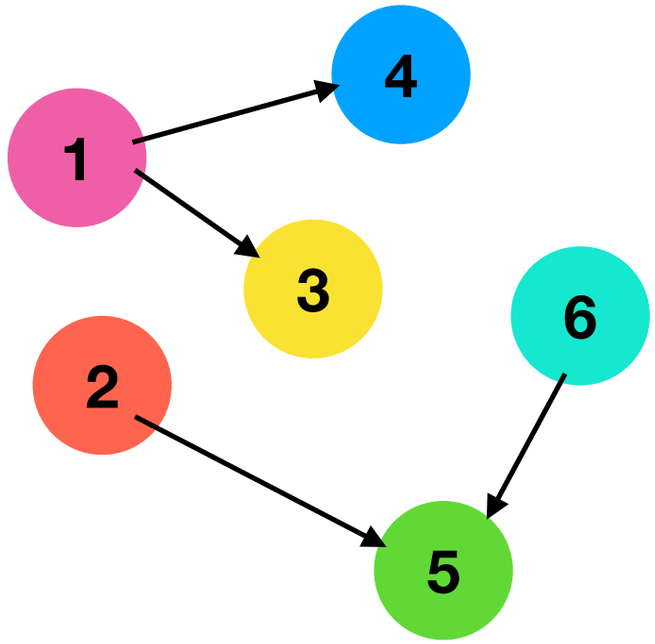
Order: The ordering is important! A feedforward network trained with e.g. pT-descending ordering would not necessarily work with pT-ascending. Which ordering is optimal?

Representation: What if for each jet you want to classify, the number of constituents N varies? Need to make all feature matrices the same size (e.g. with 0 padding).

Structure: All-to-all connectivity. Every constituent in the input layer can affect every other constituent in the next layer.

Graph structure

graph = set of nodes/vertices/elements + edges between them



Edges represented as a index pairs
edges = [(1,4), (1,3), (2,5), (6,5)]

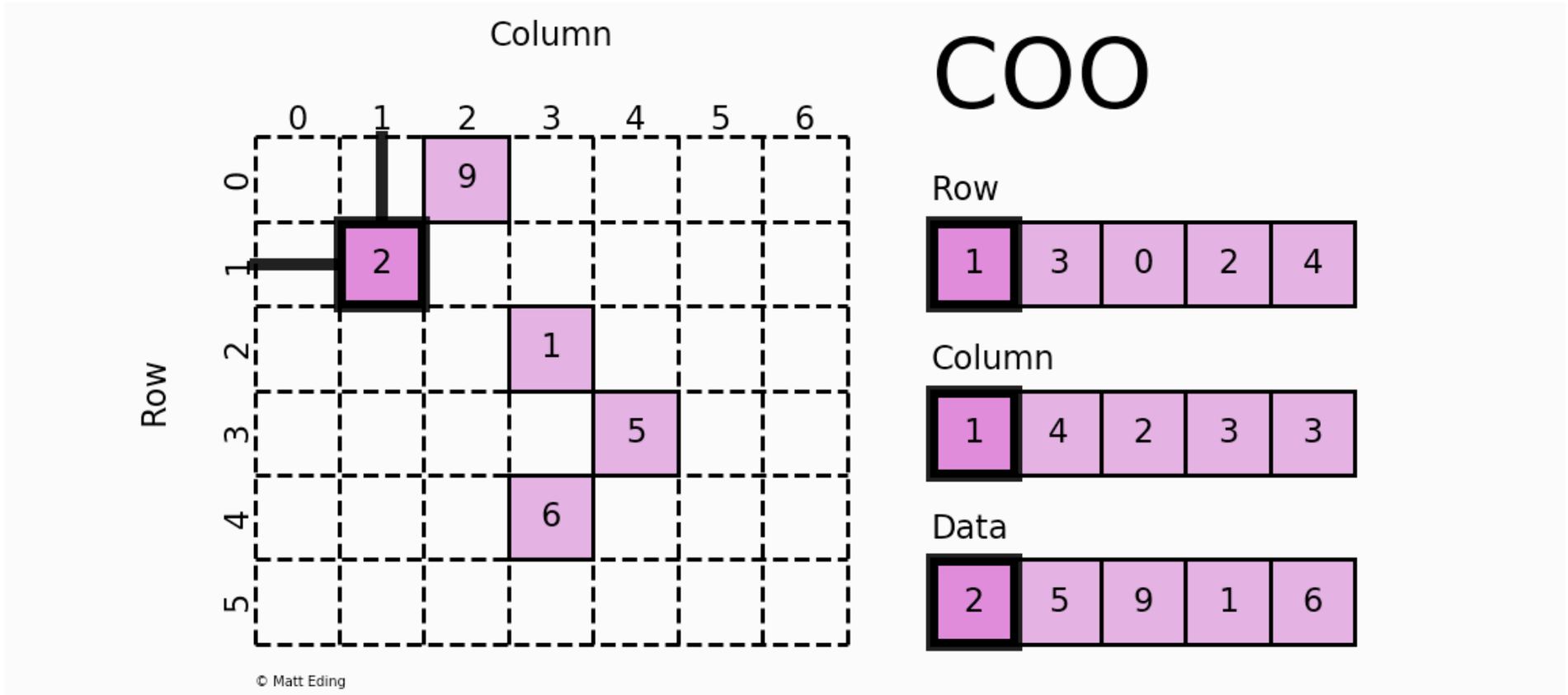
Or as an NxN adjacency matrix

$$A_{ij} = \begin{bmatrix} 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

Where do we get this graph structure?

1. All-to-all connections, in case of small input sets.
2. From physics priors: connect "nearby" elements in advance
3. Optimize as a part of the learning process (Graph Structure Learning)

Sparse tensors

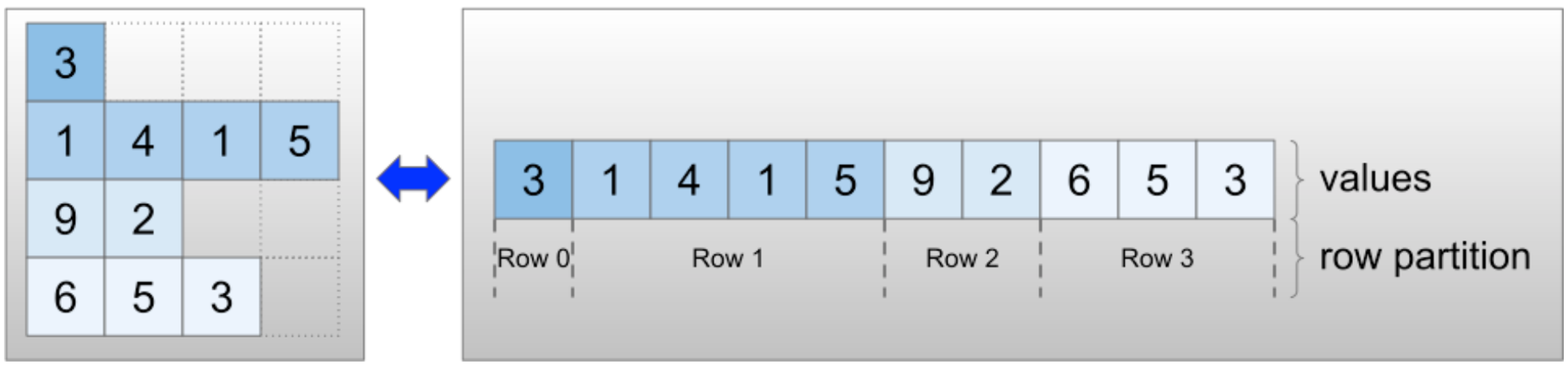


- Not always possible to store NxN adjacency matrix (e.g. if N > few thousands)

- Sparse graph adjacency matrices are typically represented in COO sparse format in DL libraries

- Backprop only on nonzero data values, not on row/column indices

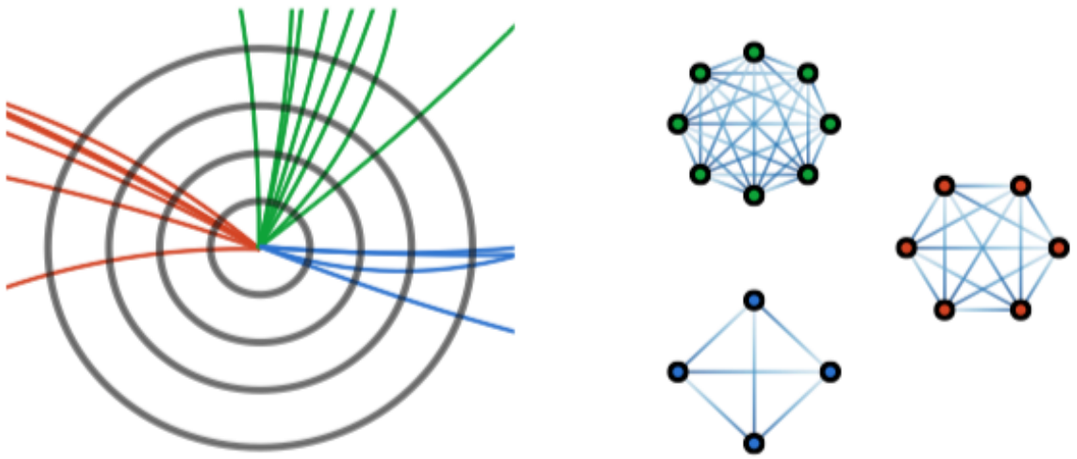
Ragged/jagged tensor



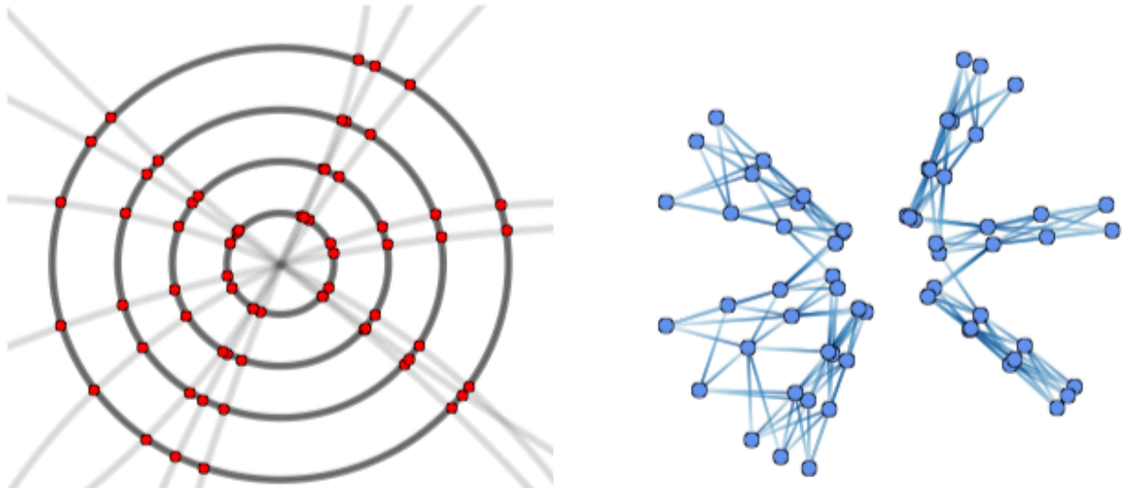
- Memory-efficient way to represent sequences of different length in a single batch
- Used to avoid zero-padding & masking when doing batch processing
- GPUs generally like inputs to be of the same length: memory / runtime tradeoff

Example graph structures

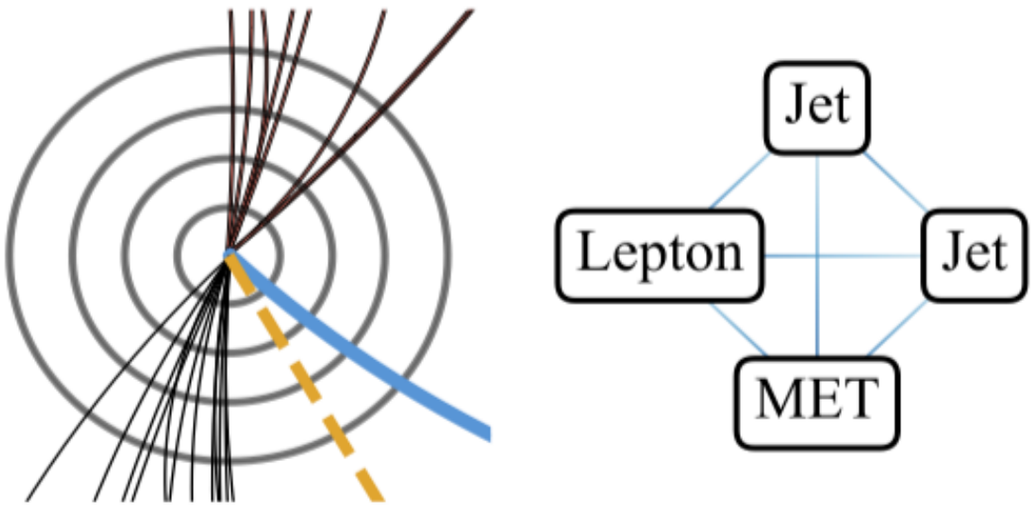
Jet constituents (all-to-all)



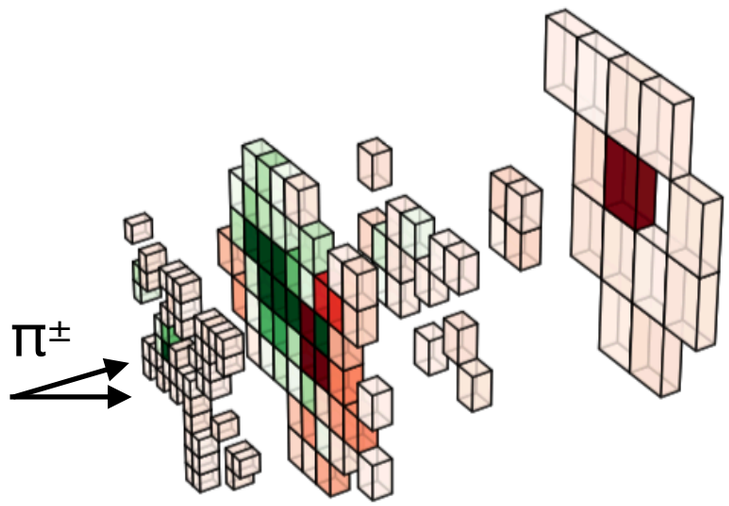
Particle tracking (neighborhood)

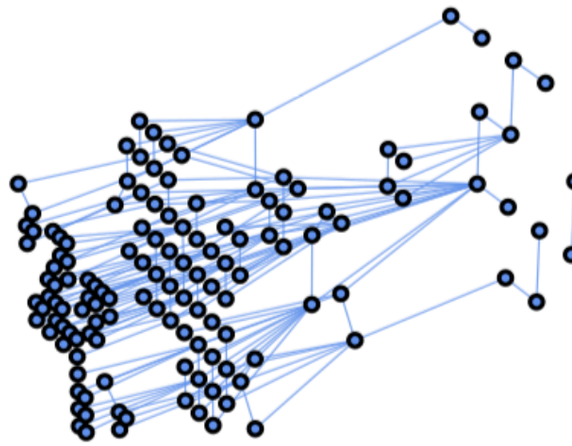


event constituents (all-to-all)



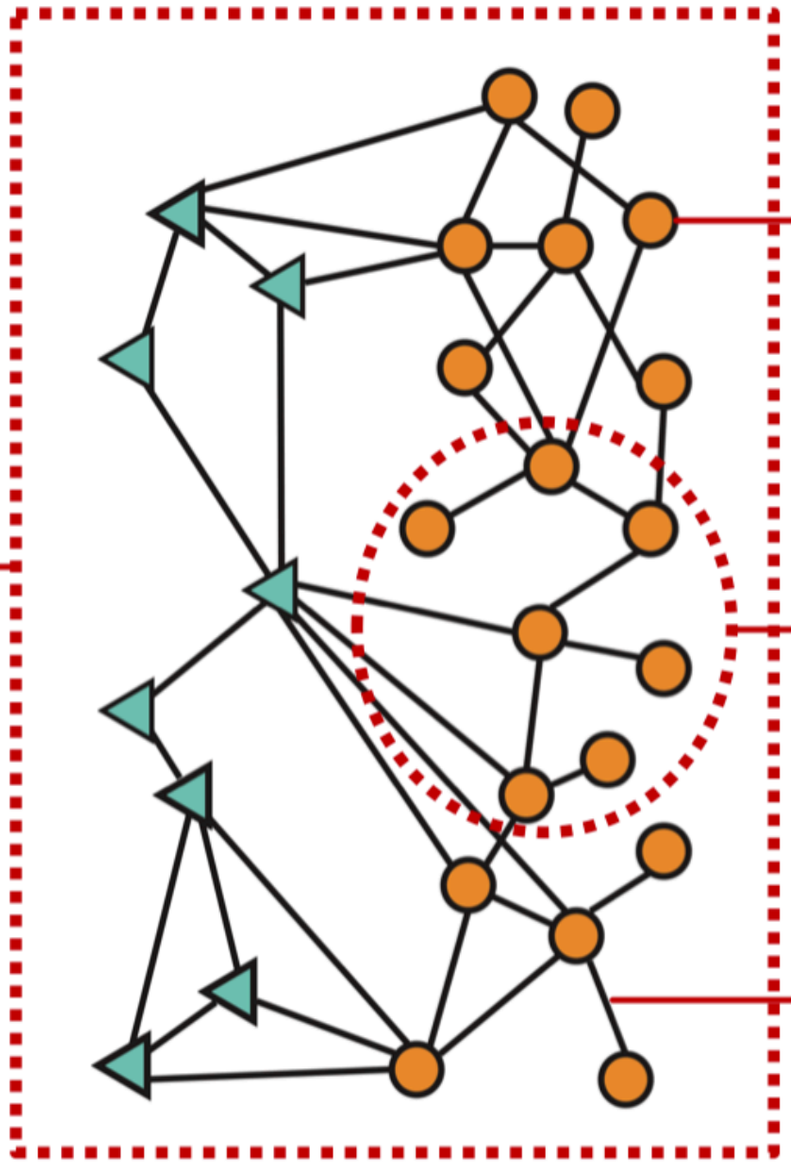
Multilayer calorimeter hits (neighborhood)





Graph Neural Networks in Particle Physics, Jonathan Shlomi, Peter Battaglia, Jean-Roch Vlimant, 2007.13681, 10.1088/2632-2153/abbf9a

Graph problems



Node level

Particle Flow reconstruction, detector hit segmentation

Graph-level prediction, Graph generation

Jet tagging, event tagging

Community (subgraph) level

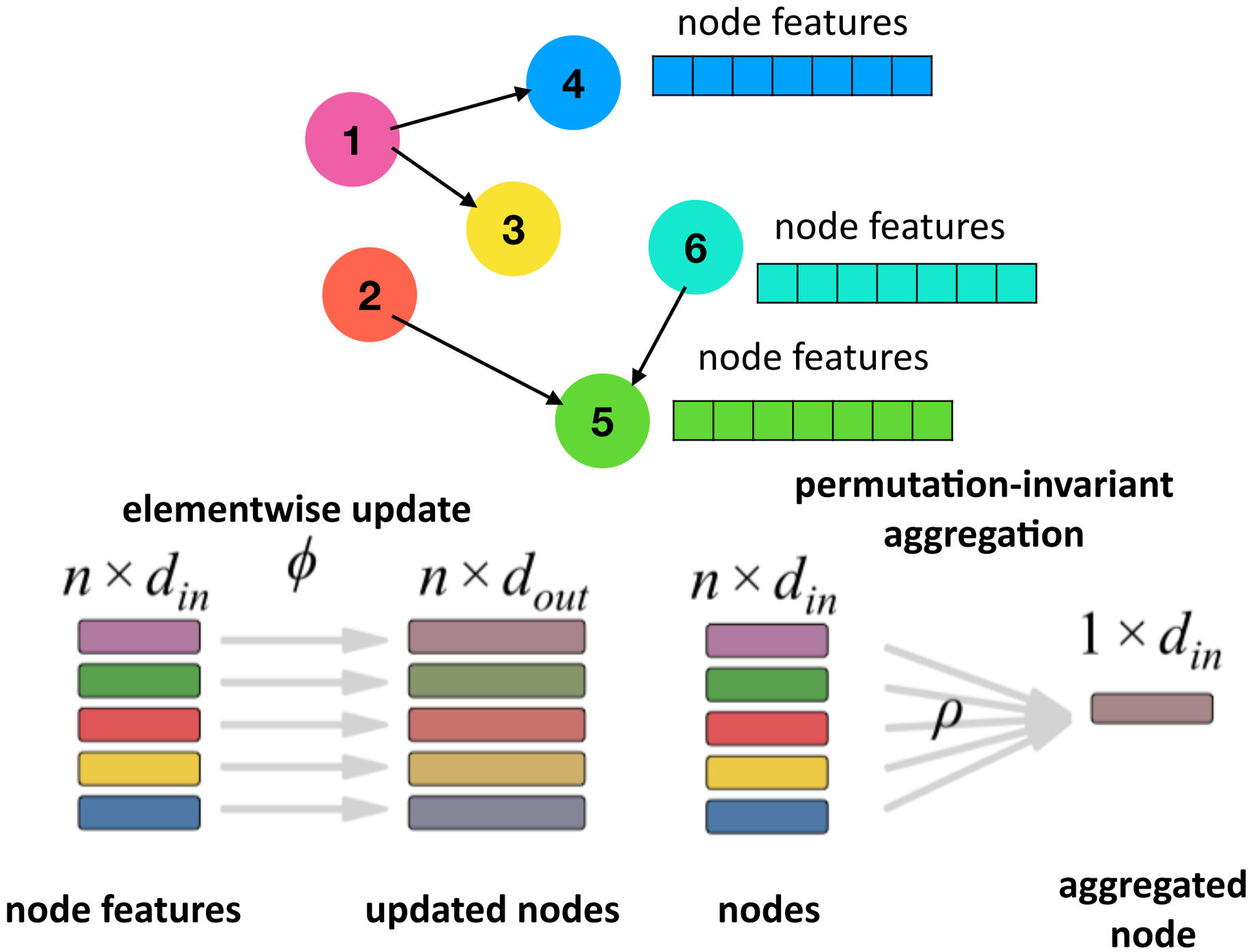
detector hit clustering

Edge-level

Track reconstruction

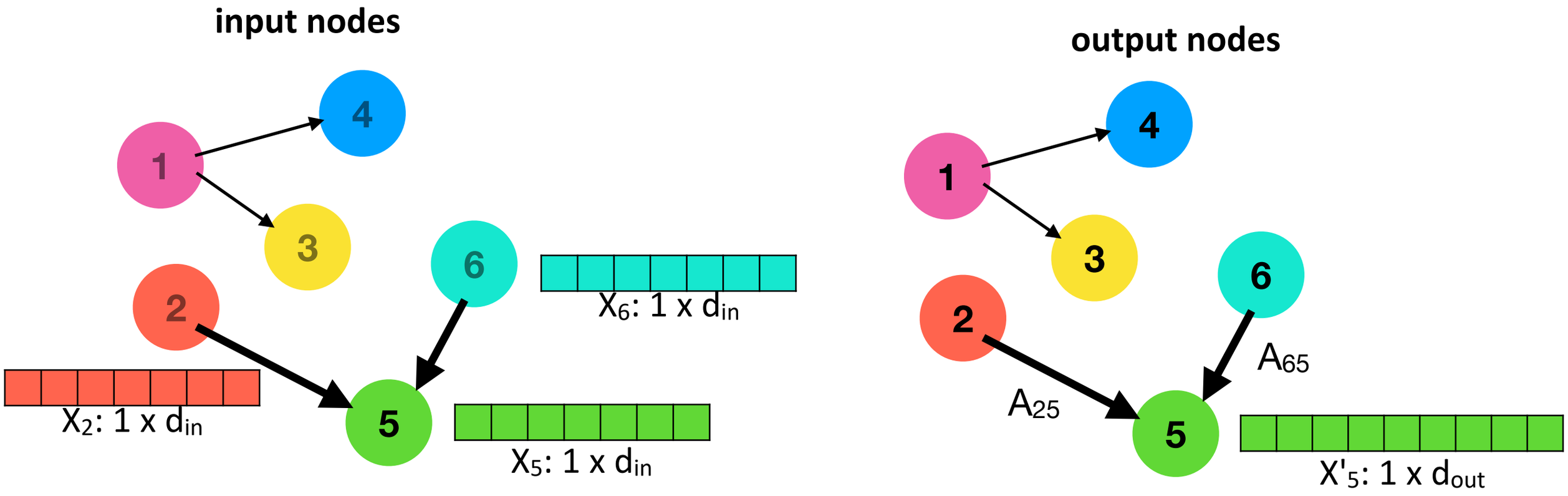
J. Leskovec et al [2021]

Operations on a graph



Graph Neural Networks in Particle Physics, Jonathan Shlomi, Peter Battaglia, Jean-Roch Vlimant, 2007.13681, 10.1088/2632-2153/abbf9a

Graph Convolutional Network (GCN)



trainable weight matrix **W: d_{in} x d_{out}**

update rule: **$X'_i \rightarrow ReLU[A_{ji} \cdot (X_j \cdot W)]$**

e.g. $X'_5 = ReLU[A_{65}(X_6 \cdot W) + A_{25}(X_2 \cdot W)]$

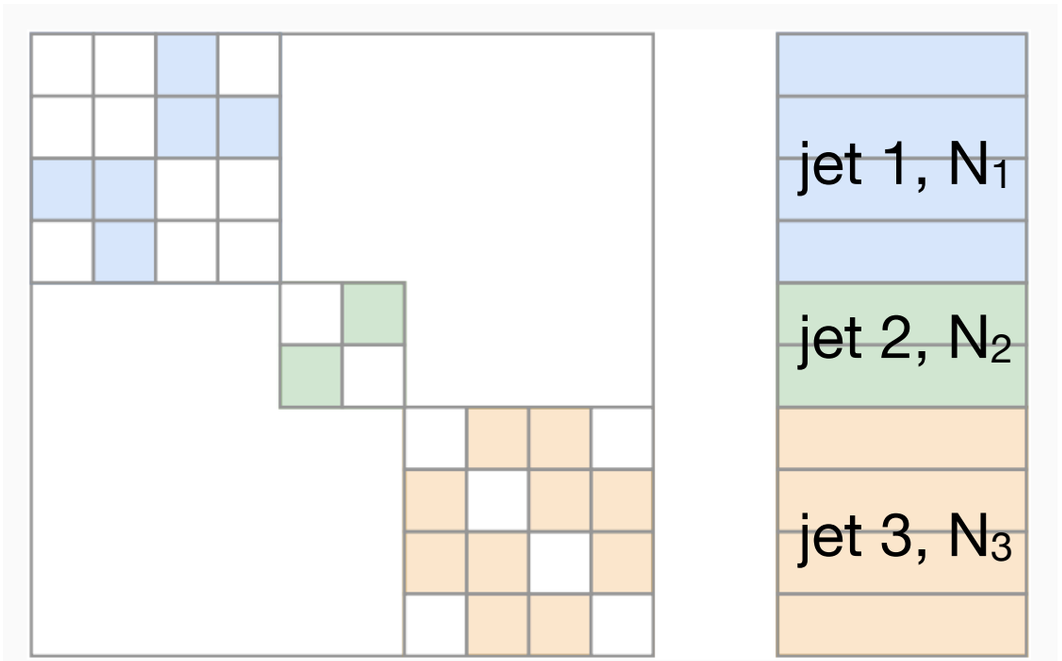
Computational modes

Suppose we have a dataset of jets we want to classify, each jet having N_i constituents.
NN training often requires batching the data to average gradient updates.

Disjoint

features: $(N_1 + N_2 + ...) \times D$

adjacency: $(N_1 + N_2 + ...)^2$

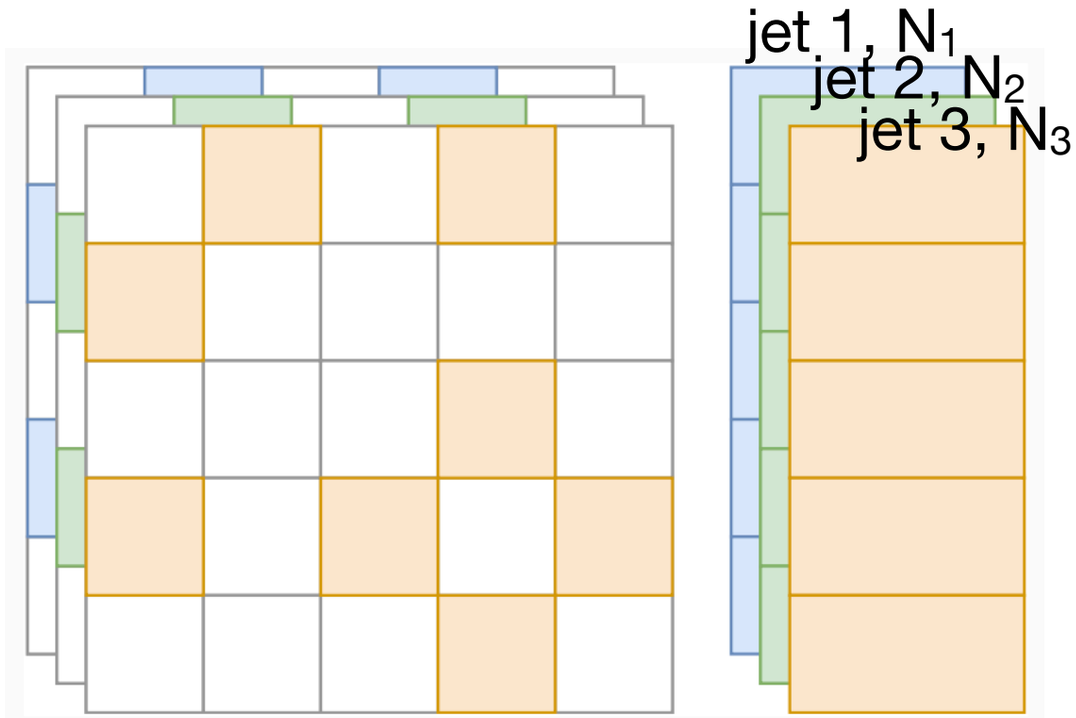


Adjacency is typically sparse.
Suitable for large inputs (N > 1000).
Typically requires on-the-fly computation (e.g. pytorch).

Batched

features: B x N x D

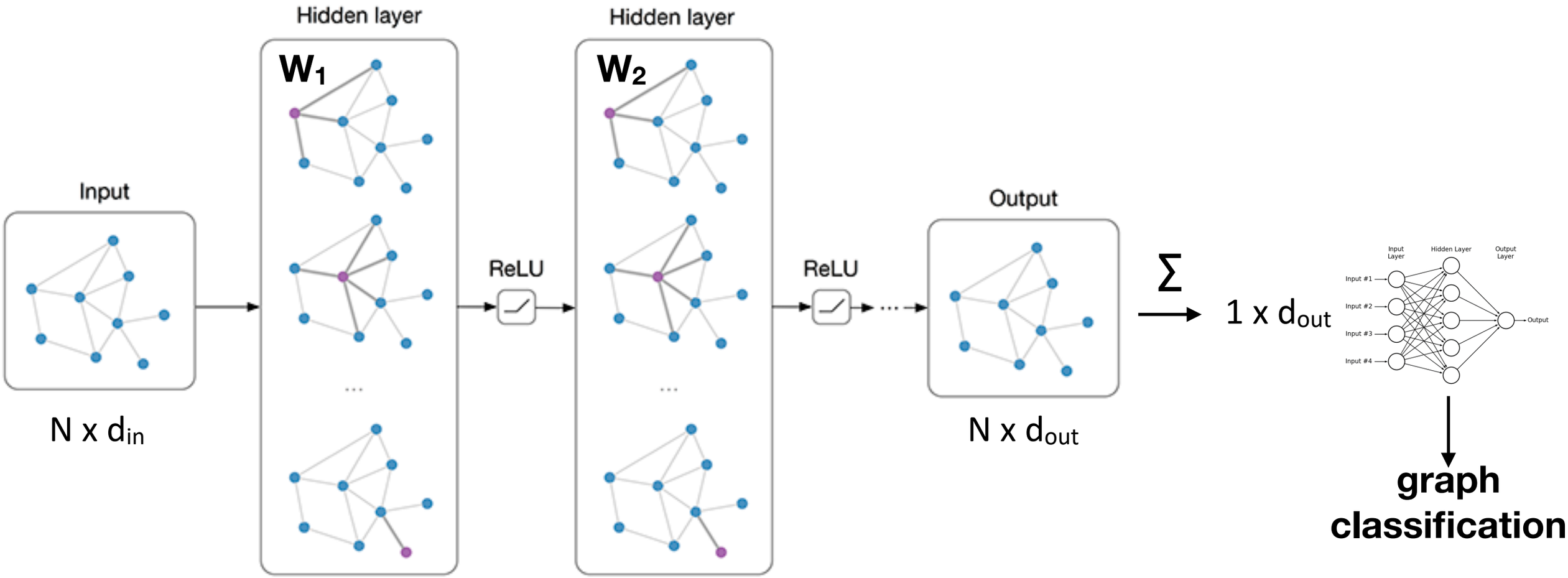
adjacencies: B x N x N



Adjacency is typically dense.
Graphs may be zero-padded / masked to size N.
Suitable for small inputs (<1000) and static computational graphs (e.g. tensorflow).

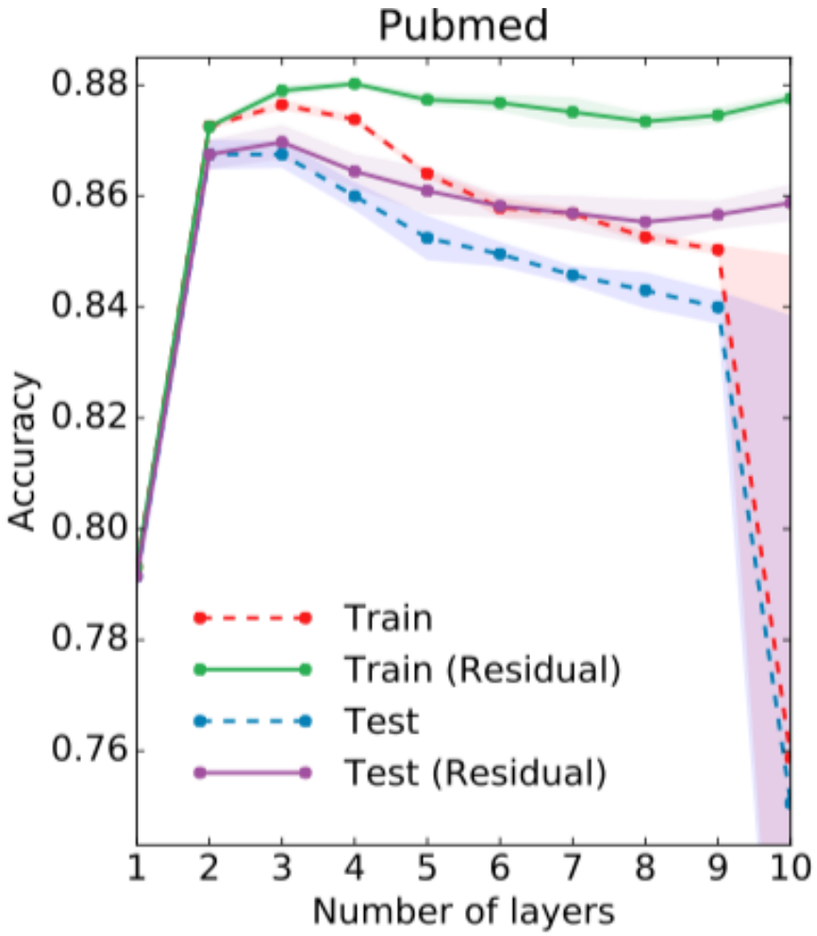
https://graphneural.network/data-modes/

GCN properties



- A trainable weight matrix W_i (d_{in} x d_{out}) in layer *i* shared across all nodes
- The input and output is a graph. The node features are transformed, the graph structure does not change.
- The GCN is permutation-invariant: it does not matter in which order the set of nodes is formatted as a matrix for computations, due to the permutation-invariant aggregation funcion
- A very nice overview can be found from Kipf & Welling: https://tkipf.github.io/graph-convolutional-networks/

Node smoothing



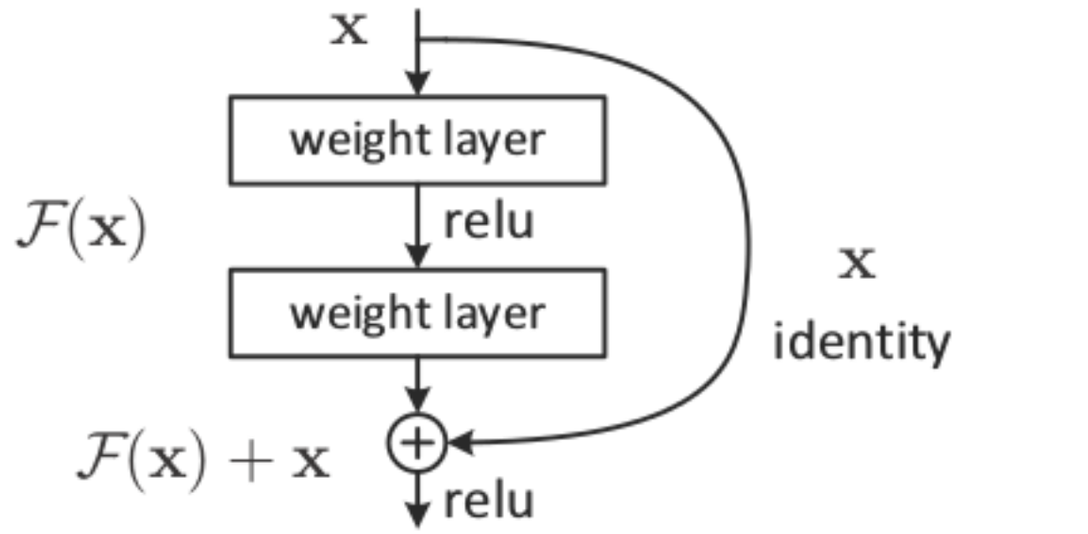


Figure 2. Residual learning: a building block.

Deep GCN without skip connections → oversmoothing, performance drops

Kipf, Thomas N., and Max Welling. "Semi-supervised classification with graph convolutional networks." *arXiv preprint arXiv:1609.02907* (2016).

Message passing

- Different types of graph-related algorithms can be formulated in the **message passing** language
- Nodes pass messages to their neighbors
- Aggregate the messages and update the node state

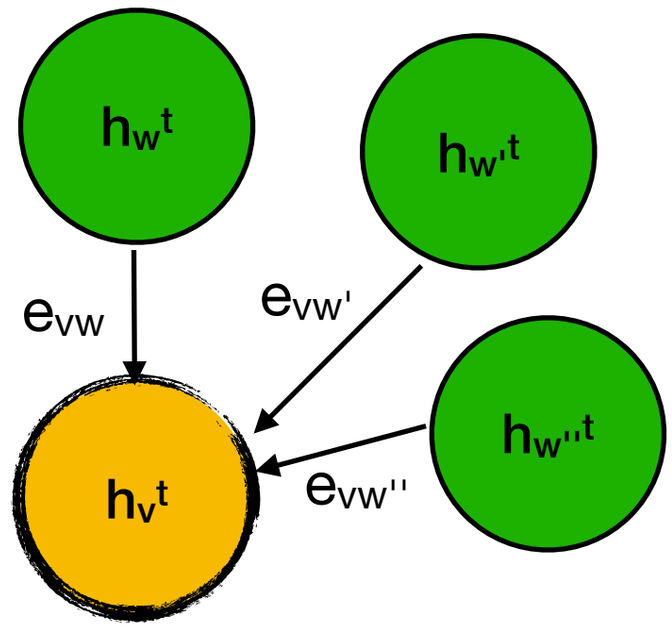
learnable message function

edge features

message $$m_v^{t+1} = \sum_{w \in N(v)} M_t(h_v^t, h_w^t, e_{vw})$$

node features $$h_v^{t+1} = U_t(h_v^t, m_v^{t+1})$$

learnable update function



Gilmer, Justin, et al. "Neural message passing for quantum chemistry." *International Conference on Machine Learning*. PMLR, 2017.

GCN as message passing

Incoming message on a node	$m_v^{t+1} = \sum_{w \in N(v)} M_t(h_v^t, h_w^t)$
Message function	$M_t(h_v^t, h_w^t) \propto A_{vw} h_w^t$
Node update rule	$h_v^{t+1} = U_v^t(h_v^t, m_v^{t+1})$
Node update function	$U_v^t(h_v^t, m_v^{t+1}) = \mathbf{ReLU}(W^t m_v^{t+1})$

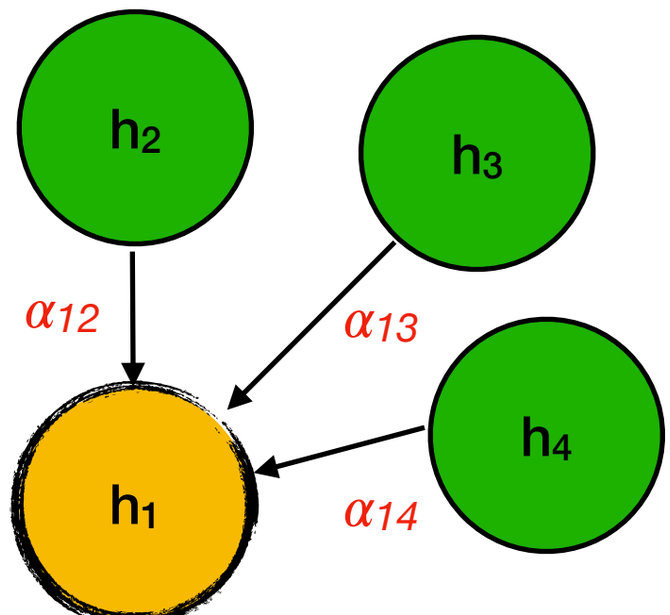
Gilmer, Justin, et al. "Neural message passing for quantum chemistry." *International Conference on Machine Learning*. PMLR, 2017.

Graph Attention (GAT)

Compute an attention coefficient α_{ij} between pairs of connected nodes.

Trainable attention vector **a**, feature weight vector **W.**

$$\alpha_{ij} = \frac{\exp\left(\text{LeakyReLU}\left(\vec{\mathbf{a}}^T[\mathbf{W}\vec{h}_i \| \mathbf{W}\vec{h}_j]\right)\right)}{\sum_{k\in\mathcal{N}_i}\exp\left(\text{LeakyReLU}\left(\vec{\mathbf{a}}^T[\mathbf{W}\vec{h}_i \| \mathbf{W}\vec{h}_k]\right)\right)}$$



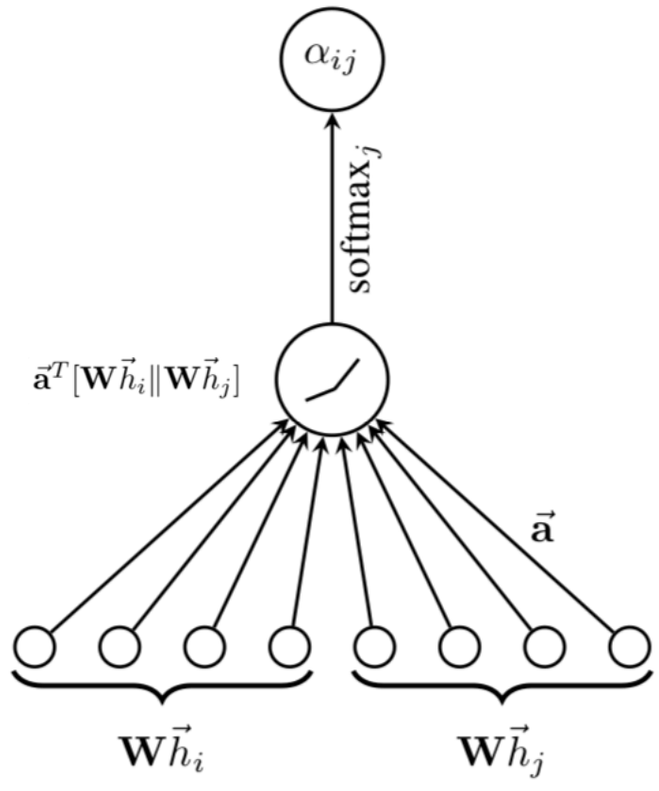
Update the node feature vector based on nearby attention coefficients.

$$\vec{h}'_i = \sigma\left(\sum_{j\in\mathcal{N}_i}\alpha_{ij}\mathbf{W}\vec{h}_j\right)$$

Inputs are graphs: N x d_{in}

Outputs are graphs: N x d_{out}

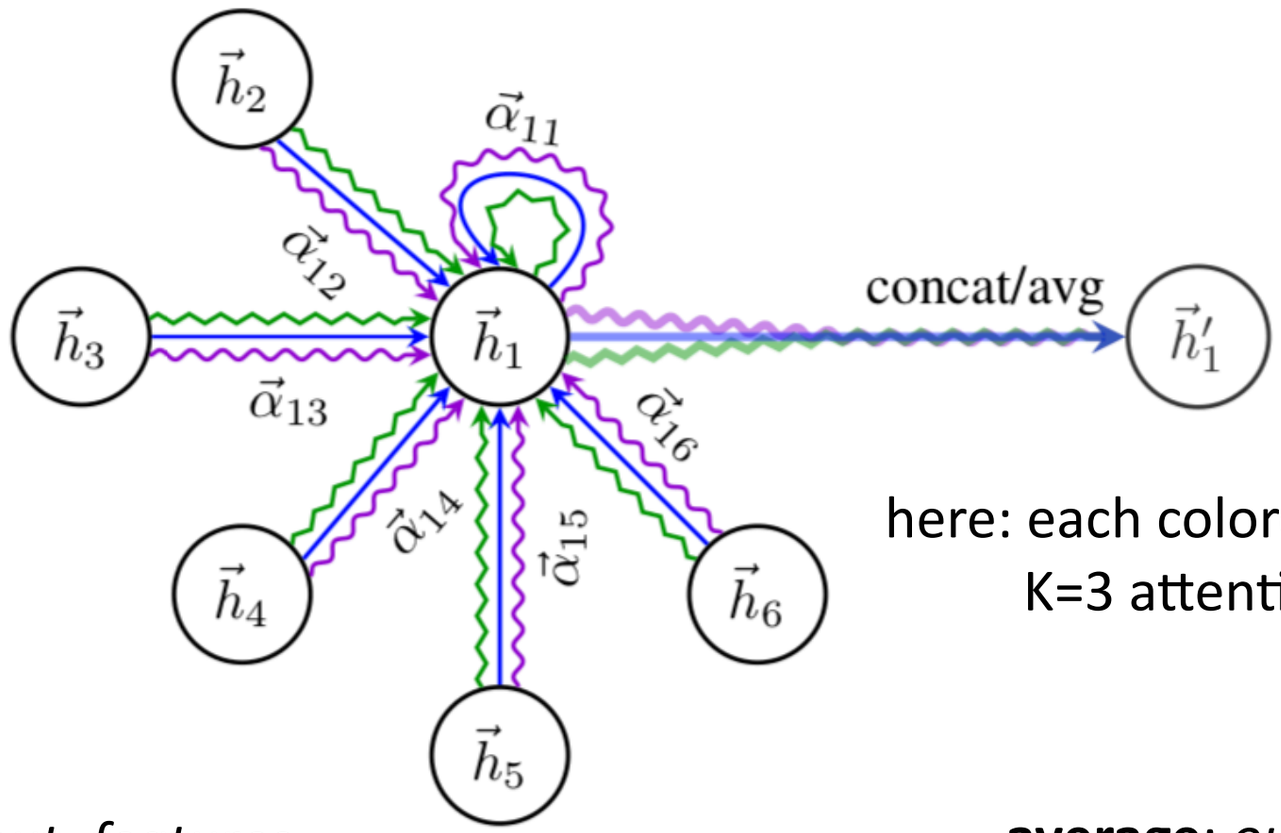
Attention vector **a** can be interpreted as feature-to-feature association.



Veličković, Petar, et al. "Graph attention networks." *arXiv preprint arXiv:1710.10903* (2017).

Multi-head GAT

Instead of a single attention coefficient α_{ij} per a node pair, compute K independent values α_{ij}^k.



here: each colored line is one of K=3 attention heads.

concatenate: $K \times$ *out_features*

$$\vec{h}'_i = \Big\Vert_{k=1}^{K} \sigma\left(\sum_{j \in \mathcal{N}_i} \alpha_{ij}^k \mathbf{W}^k \vec{h}_j\right)$$

average: *out_features*

$$\vec{h}'_i = \sigma\left(\frac{1}{K}\sum_{k=1}^{K}\sum_{j \in \mathcal{N}_i} \alpha_{ij}^k \mathbf{W}^k \vec{h}_j\right)$$

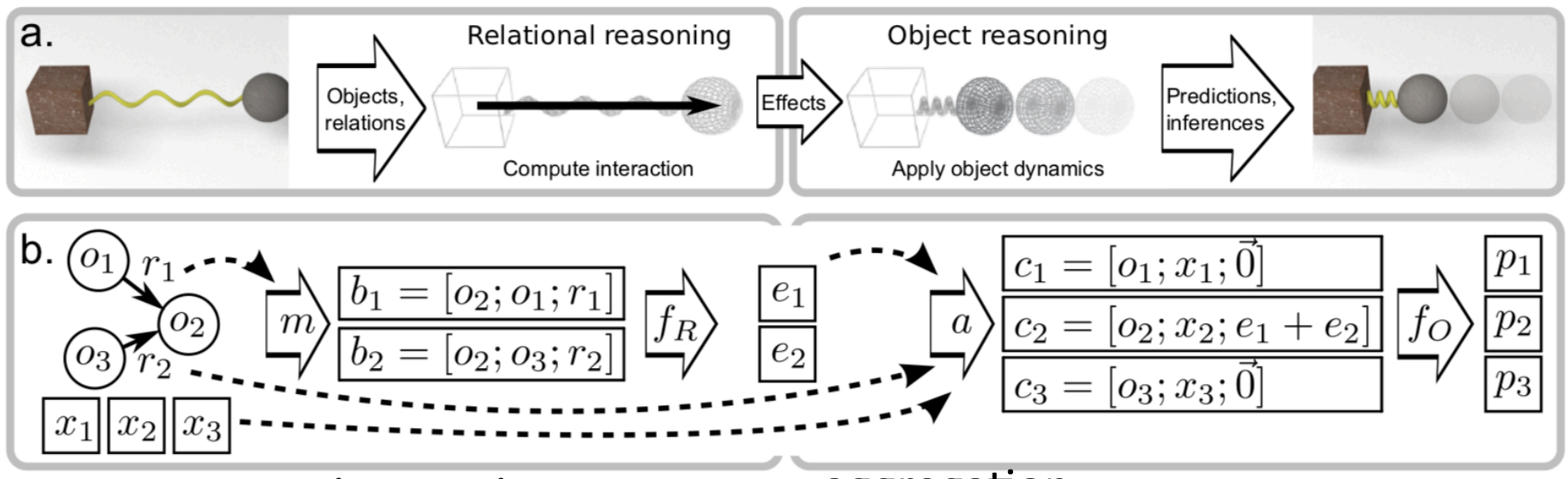
Veličković, Petar, et al. "Graph attention networks." *arXiv preprint arXiv:1710.10903* (2017).

Interaction network (IN)

In the Interaction Network (2016), the message function M_t and the node update function U_t are given as generic neural networks operating on concatenated node and edge inputs.

objects o_i,
relations r_i

externals x_i



updated objects p_i

interaction terms

effects

aggregation

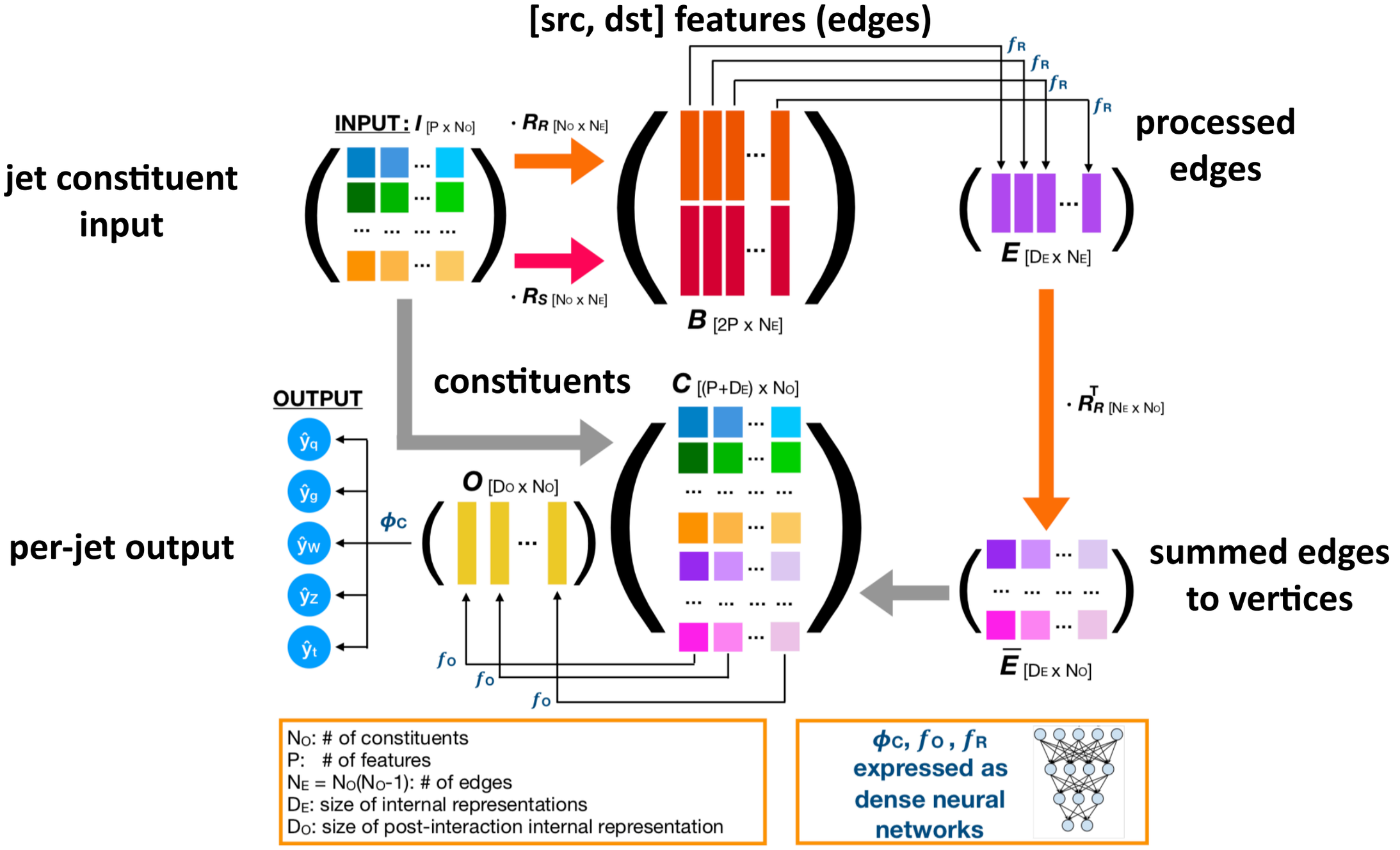
object model f_O

$$f_R(b_1) = \mathbf{MLP}([o_2; o_1; r_1]) = e_1$$

$$f_O(c_2) = \mathbf{MLP}([o_2; x_2; e_1 + e_2]) = p_2$$

Battaglia, Peter W., et al. "Interaction networks for learning about objects, relations and physics." *arXiv preprint arXiv:1612.00222* (2016).

IN for jet tagging



Moreno, E.A., Cerri, O., Duarte, J.M. *et al.* JEDI-net: a jet identification algorithm based on interaction networks. *Eur. Phys. J. C* **80,** 58 (2020). https://doi.org/10.1140/epjc/s10052-020-7608-4

IN for particle tracking

Fully connected: 1000 nodes -> 500k edges, not feasible!

Set up an initial sparse hit graph based on node proximity.

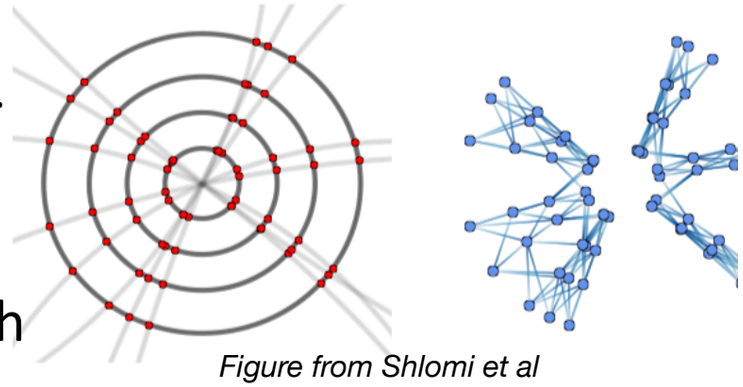
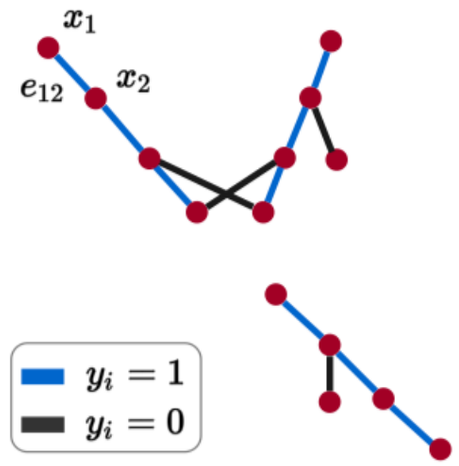


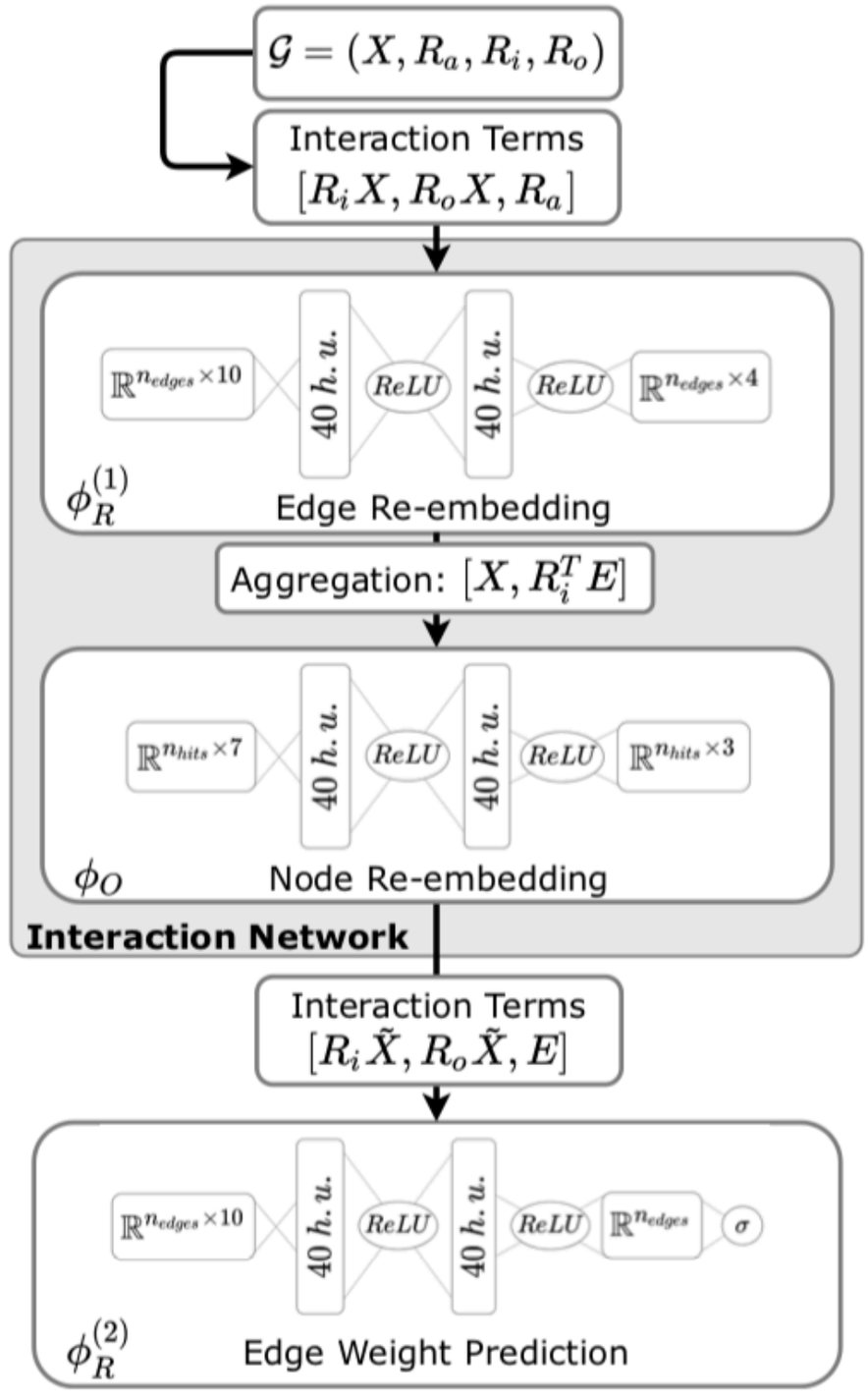
Figure from Shlomi et al

Classify possible edges as true/false based on actual track information, predict edge weight

X: node features (nodes × 3)
R_a: edge features (edges × 4)
R_i, R_o: incoming/outgoing edge matrix
$R_{i,o}X$: incoming/outgoing nodes (edges x 3)



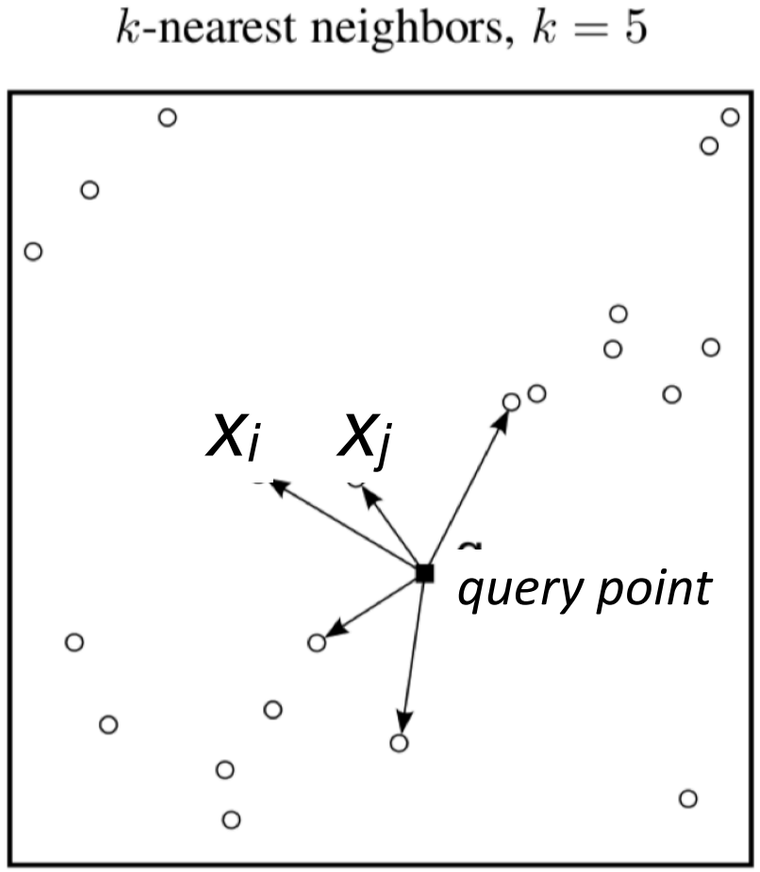
Hitgraph View
DeZoort et al

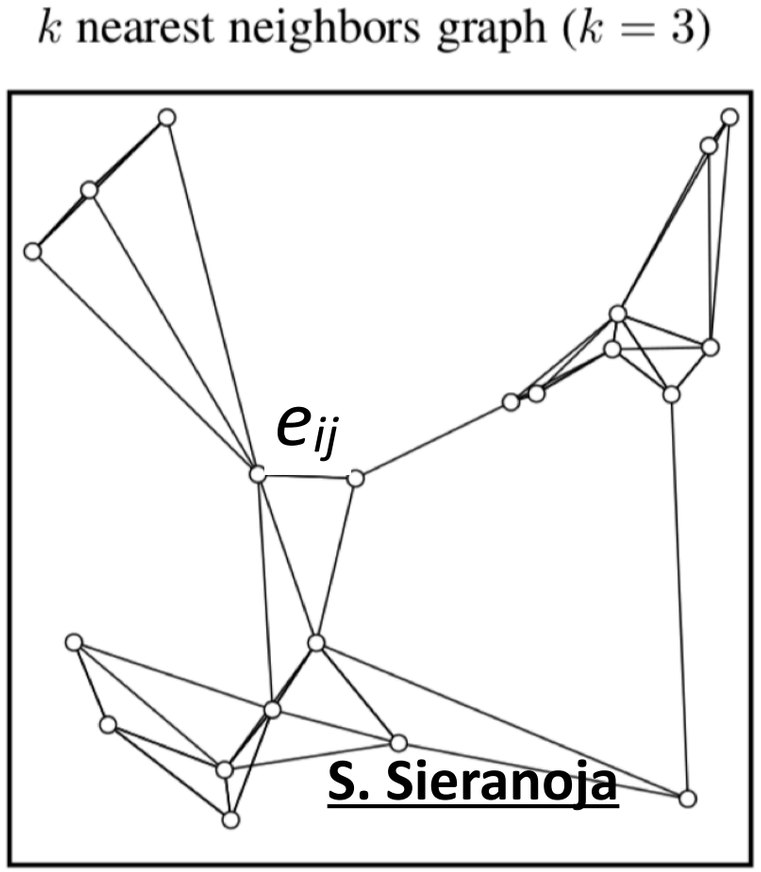


DeZoort, Gage, et al. "Charged particle tracking via edge-classifying interaction networks." *arXiv preprint arXiv:2103.16701* (2021).

Dynamic graph with kNN

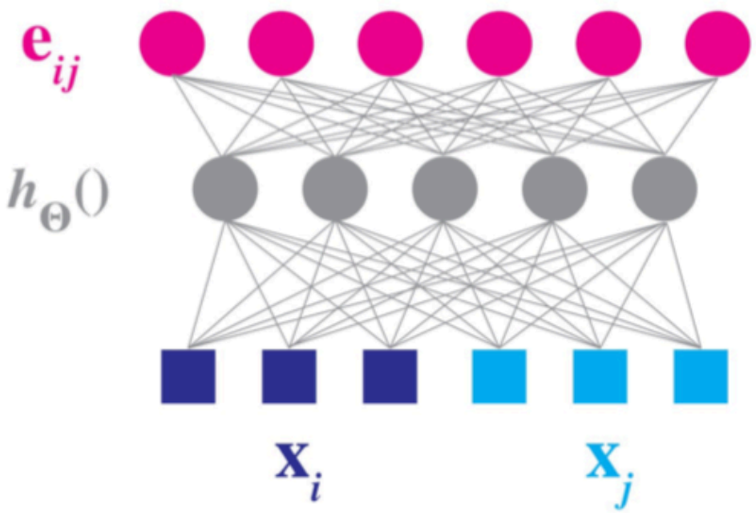
- In the previous examples with GCN, GAT and IN, the graph was static and defined/known in advance
- Often, the graph structure may not be known in advance, or may be inaccurate
- Construct dynamically: point cloud $\{x_i\}$ → for each point x_i, find k closest neighbors $\{x_j\}$, edges $\{e_{ij}\}$





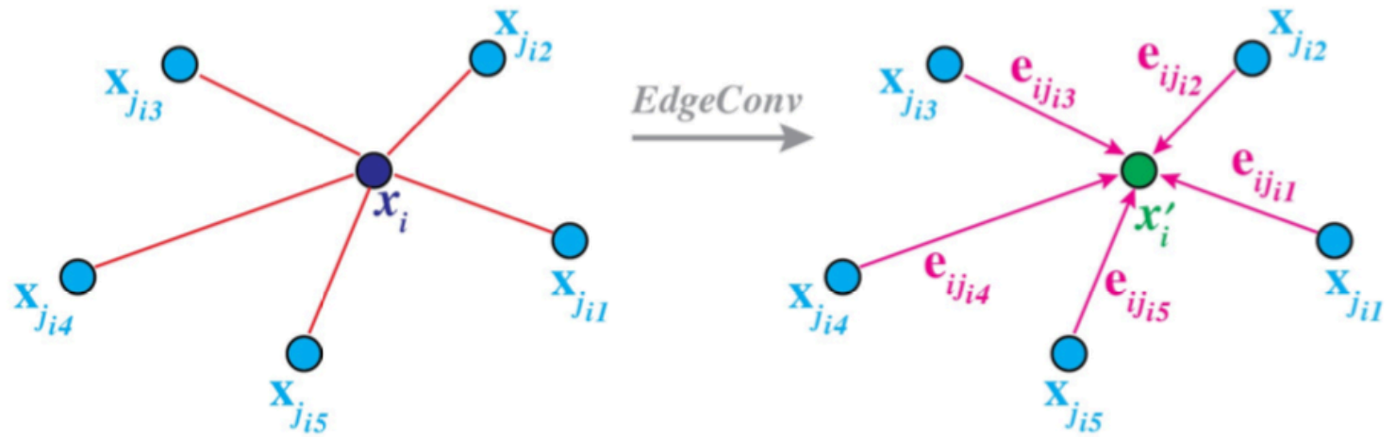
Dynamic graph CNN (DGCNN)

Construct neighbor graph: for each point x_i, find k closest neighbors $\{x_j\}$, edges $\{e_{ij}\}$



construct an edge feature using a learnable function

$$\boldsymbol{h_\Theta}(\boldsymbol{x}_i, \boldsymbol{x}_{i_j}) = \bar{\boldsymbol{h}}_{\boldsymbol{\Theta}}(\boldsymbol{x}_i, \boldsymbol{x}_{i_j} - \boldsymbol{x}_i),$$



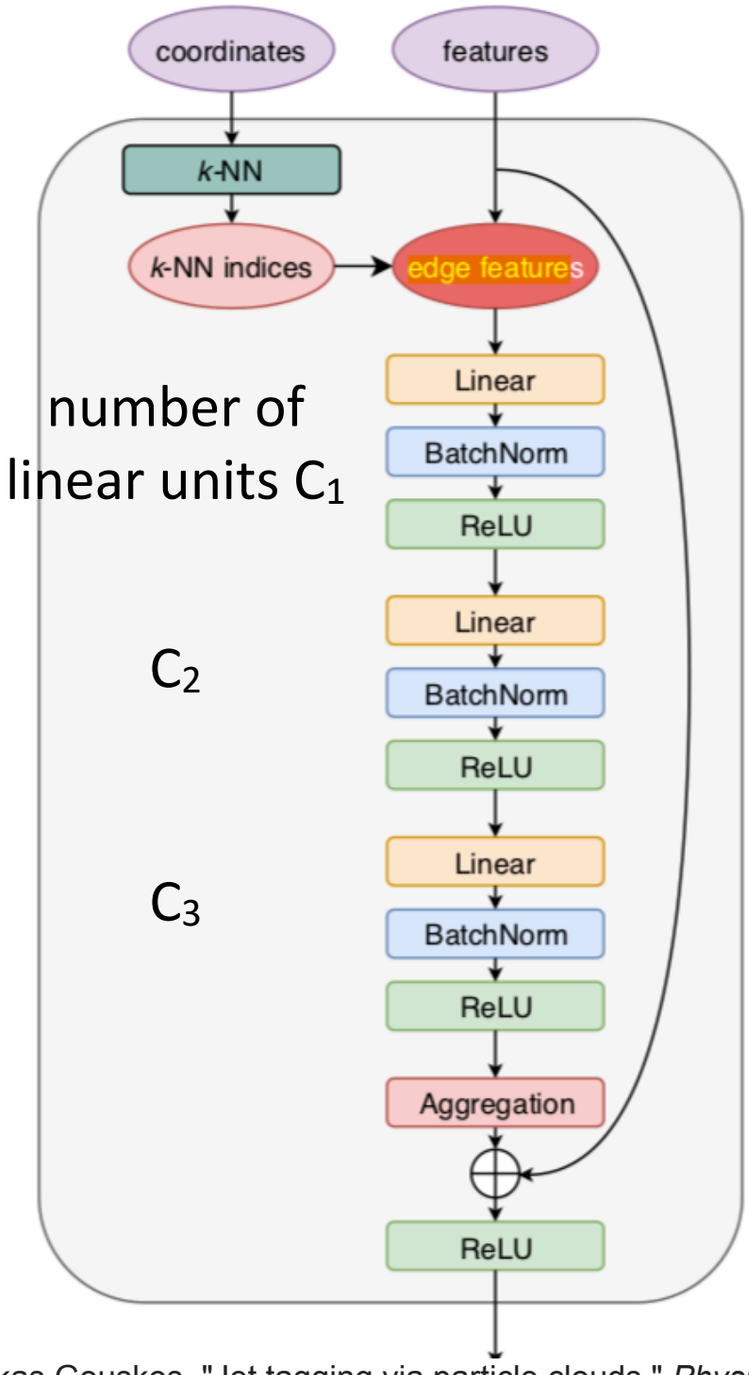
Compute the new point features x_i' using an aggregation over the edges

$$\boldsymbol{x}_i' = \square_{j=1}^{k} \boldsymbol{h_\Theta}(\boldsymbol{x}_i, \boldsymbol{x}_{i_j}),$$

Wang, Yue, et al. "Dynamic graph CNN for learning on point clouds." *Acm Transactions On Graphics (tog)* 38.5 (2019): 1-12.

ParticleNet: EdgeConv block

input coordinates (B, N, C)



input features: (B, N, F)

distance matrix: (B, N, N)

edge features: (B, N, K, 2*F)

$$\boldsymbol{h}_{\boldsymbol{\Theta}}(\boldsymbol{x}_i, \boldsymbol{x}_{i_j}) = \bar{\boldsymbol{h}}_{\boldsymbol{\Theta}}(\boldsymbol{x}_i, \boldsymbol{x}_{i_j} - \boldsymbol{x}_i),$$

edge convolutions

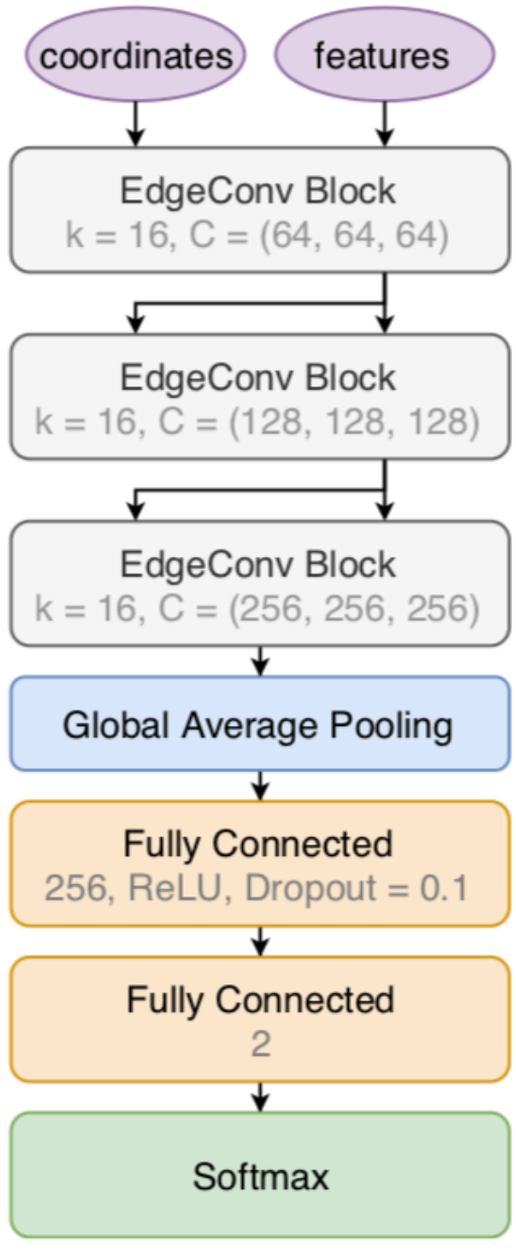
$$\boldsymbol{x}'_i = \square_{j=1}^{k} \boldsymbol{h}_{\boldsymbol{\Theta}}(\boldsymbol{x}_i, \boldsymbol{x}_{i_j}),$$

output features: (B, N, O)

Qu, Huilin, and Loukas Gouskos. "Jet tagging via particle clouds." *Physical Review D* 101.5 (2020): 056019.

ParticleNet full model

- Up to 100 highest-pT constituents of each jet
- relative η, φ coordinates wrt. the jet axis as coordinates
- Features are derived from 4-momentum (log transforms, ratios)
- Coordinates in subsequent layers are derived from previous layer outputs



(a) ParticleNet

Qu, Huilin, and Loukas Gouskos. "Jet tagging via particle clouds." *Physical Review D* 101.5 (2020): 056019.

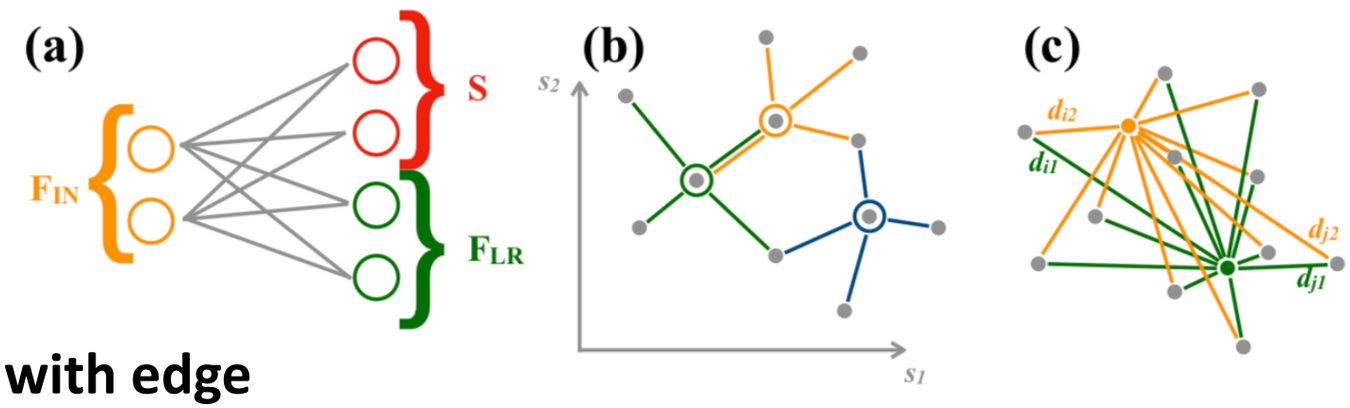
GravNet/GarNet

- Full, dense NxN distance matrix can be too large to store for N>few hundred, kNN can be expensive in a high-dimensional input space

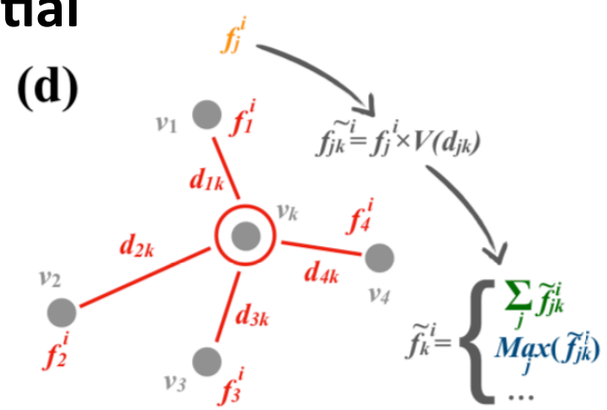
- In case low latency, low memory consumption is desirable, optimize by using a sparse adjacency matrix, separating spatial components and feature components

GravNet: full kNN graph on all nodes

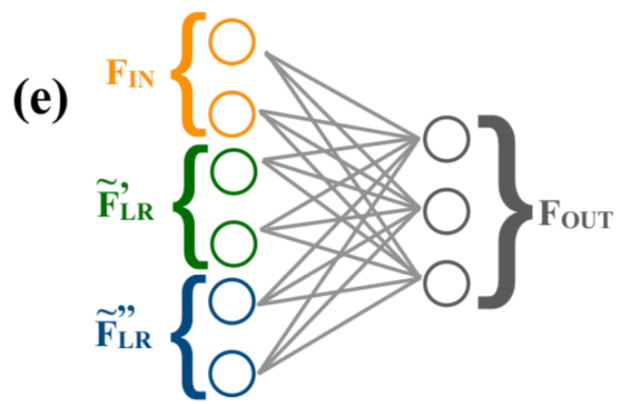
GarNet: choose a fixed number of aggregators



transform nodes with edge info + potential



aggregate



compute node output

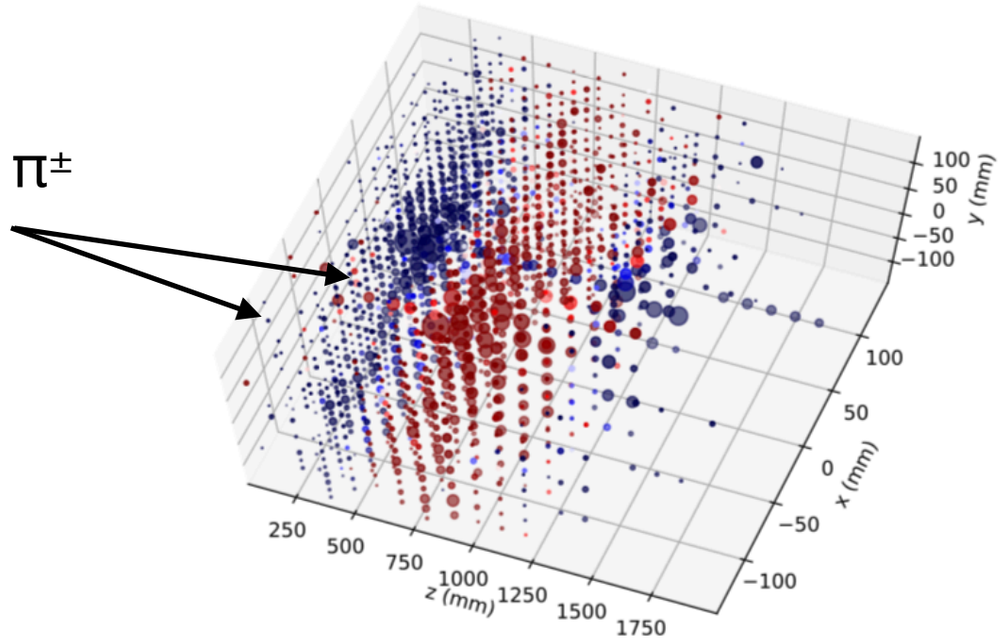
Qasim, Shah Rukh, et al. "Learning representations of irregular particle-detector geometry with distance-weighted graph networks." *The European Physical Journal C* 79.7 (2019): 1-11.

Detector reconstruction

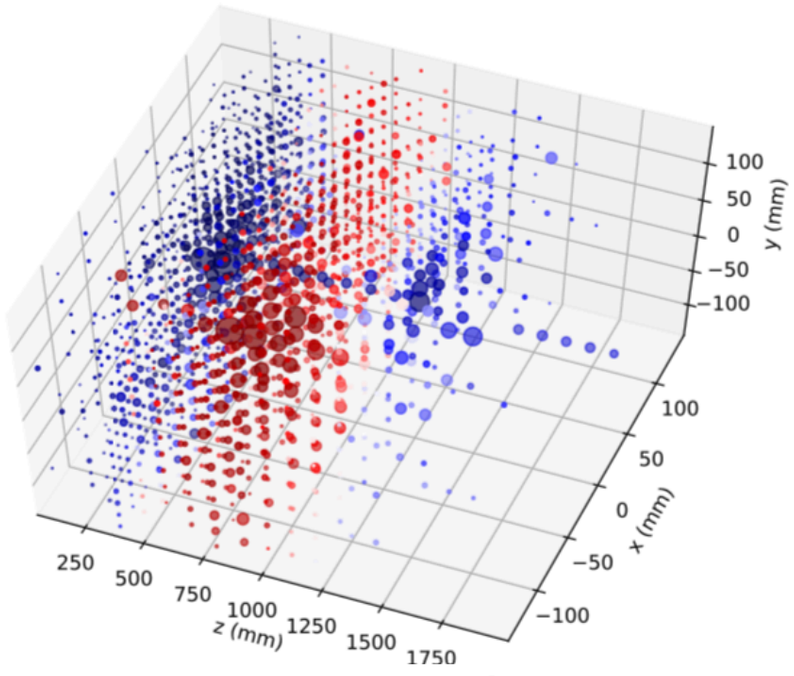
- kNN + sparse graph adjacency matrix: GravNet

- Cluster energy deposits from overlapping showers in a highly granular, layered tungsten detector simulation

- Predict the energy fraction of each sensor (I) belonging to each shower (K): p_{ik} vs t_{ik}



Two overlapping showers generated



reconstructed

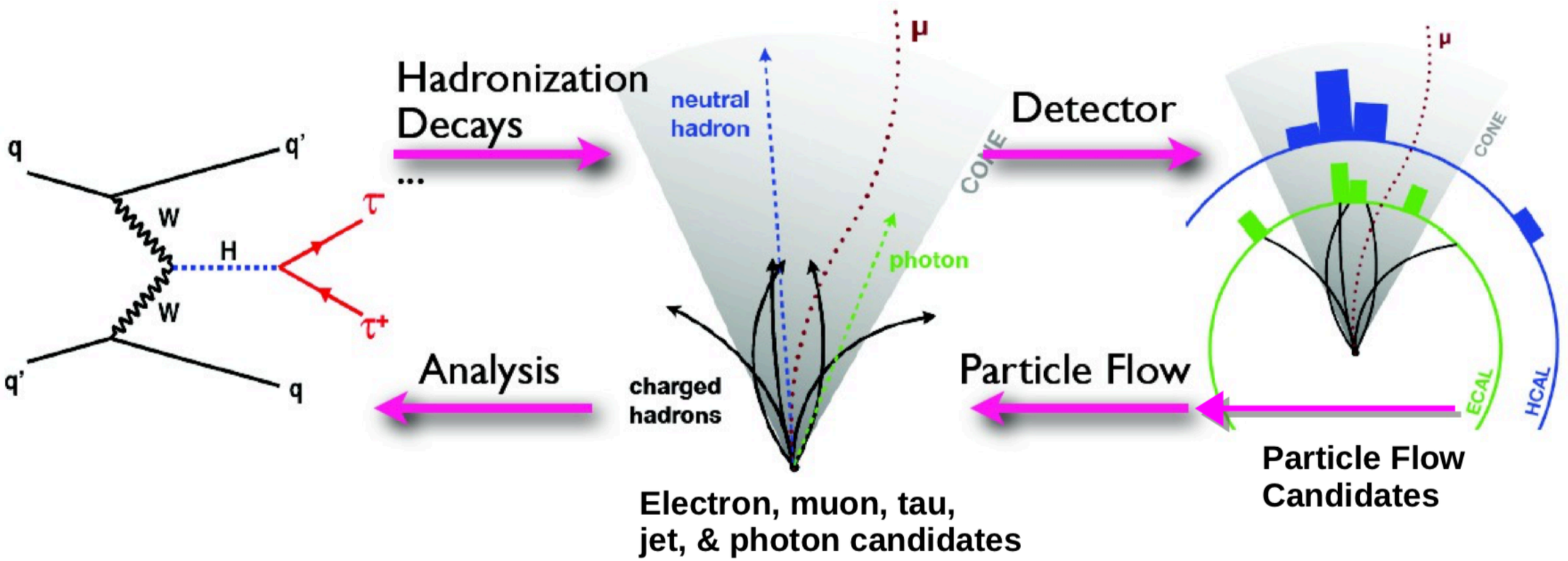
Qasim, Shah Rukh, et al. "Learning representations of irregular particle-detector geometry with distance-weighted graph networks." *The European Physical Journal C* 79.7 (2019): 1-11.

Particle Flow reconstruction

hard interaction

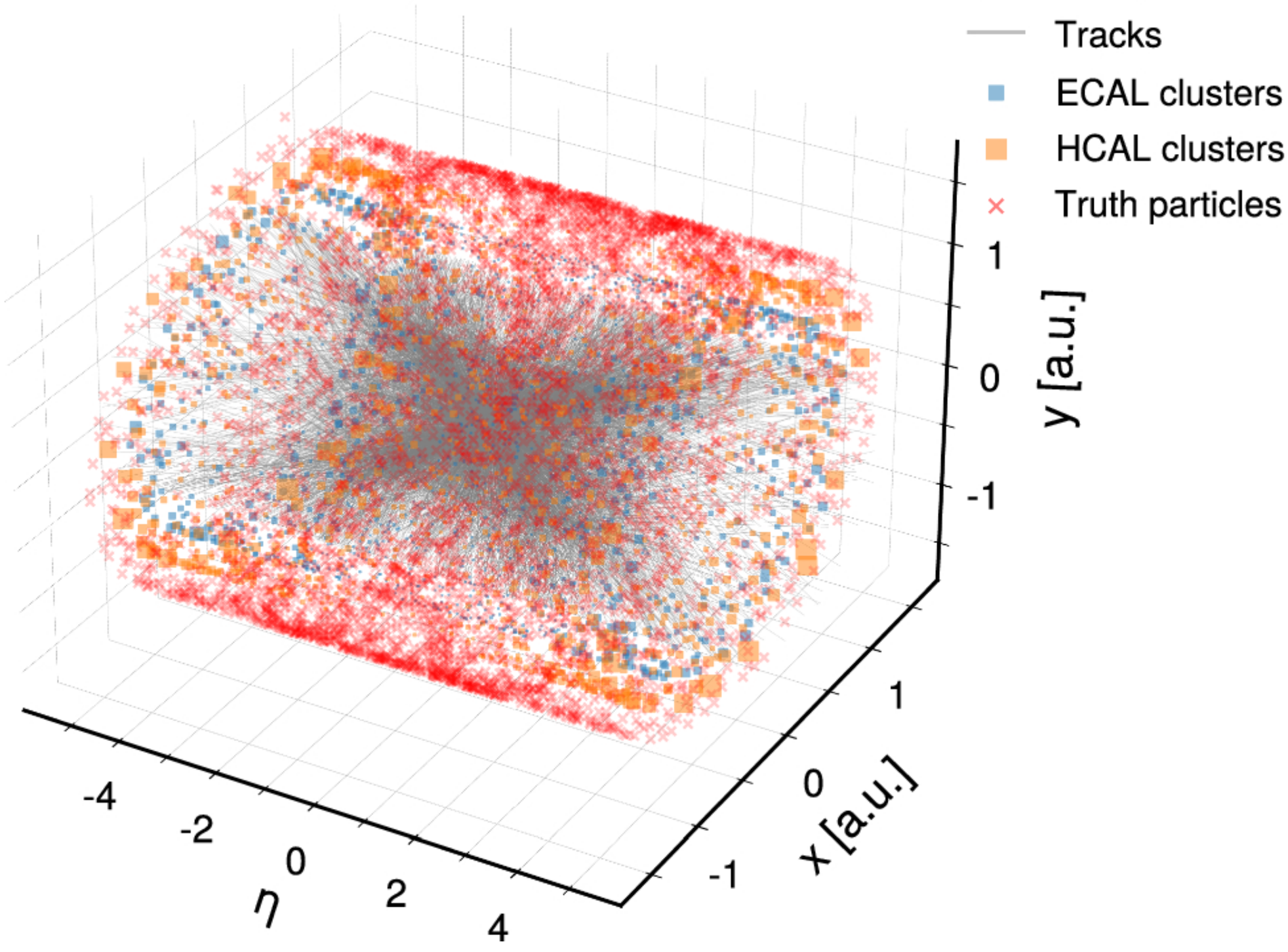
visible final state particles

visible detector hits



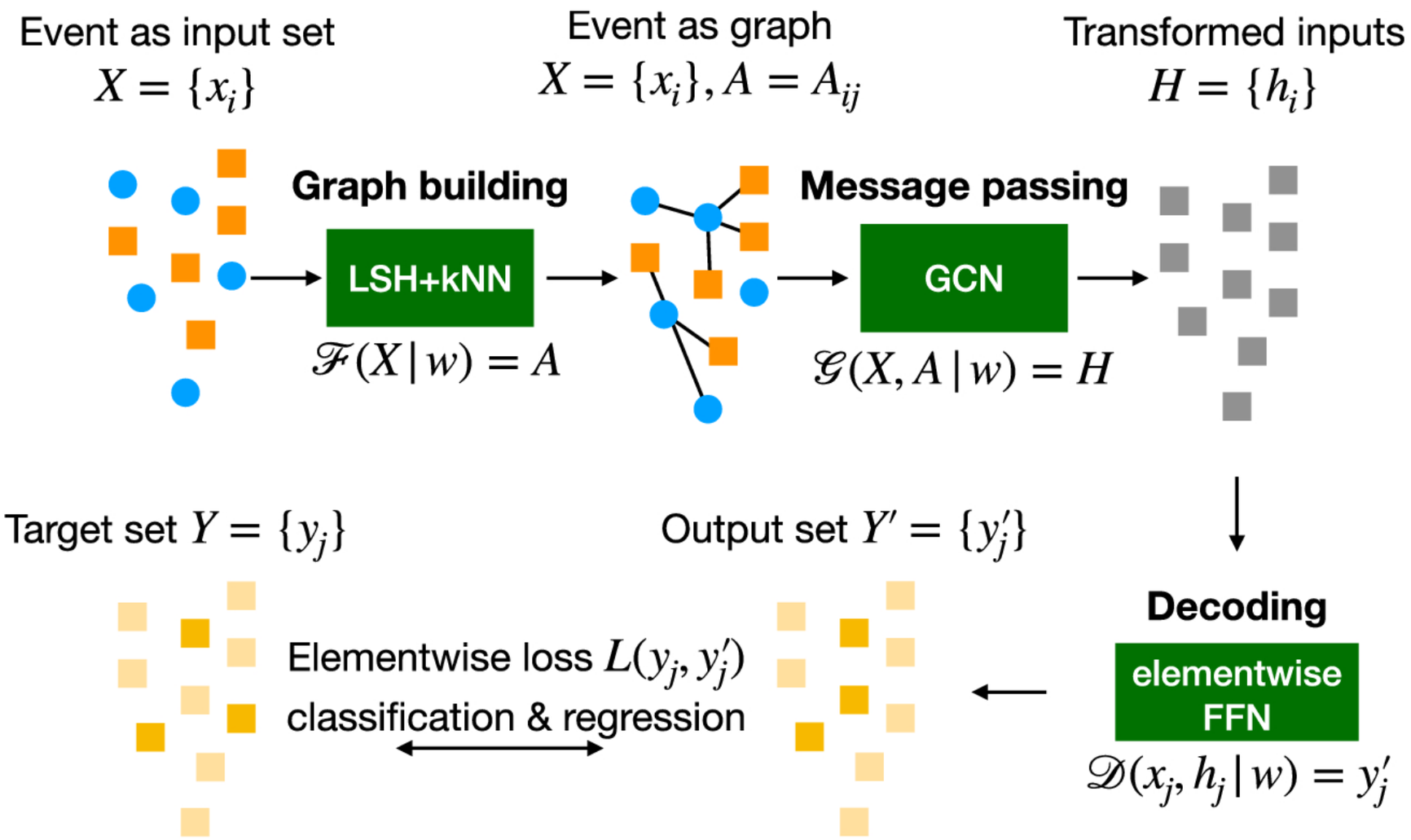
The Particle Flow algorithm combines elements across different detectors to a global particle-level representation of the collision.

Particle flow inputs and outputs in a single event



Pata, J., Duarte, J., Vlimant, JR. *et al.* MLPF: efficient machine-learned particle-flow reconstruction using graph neural networks. *Eur. Phys. J. C* **81,** 381 (2021)

GNNs for Particle Flow



$$x_i = [\text{type}, p_\text{T}, E_\text{ECAL}, E_\text{HCAL}, \eta, \phi, \eta_\text{outer}, \phi_\text{outer}, q, \ldots], \quad \text{type} \in \{\text{track, cluster}\}$$

$$y_j = [\text{PID}, p_\text{T}, E, \eta, \phi, q, \ldots], \quad \text{PID} \in \{\text{none, charged hadron, neutral hadron}, \gamma, e^\pm, \mu^\pm\}$$

$$h_i \in \mathbb{R}^{256}$$

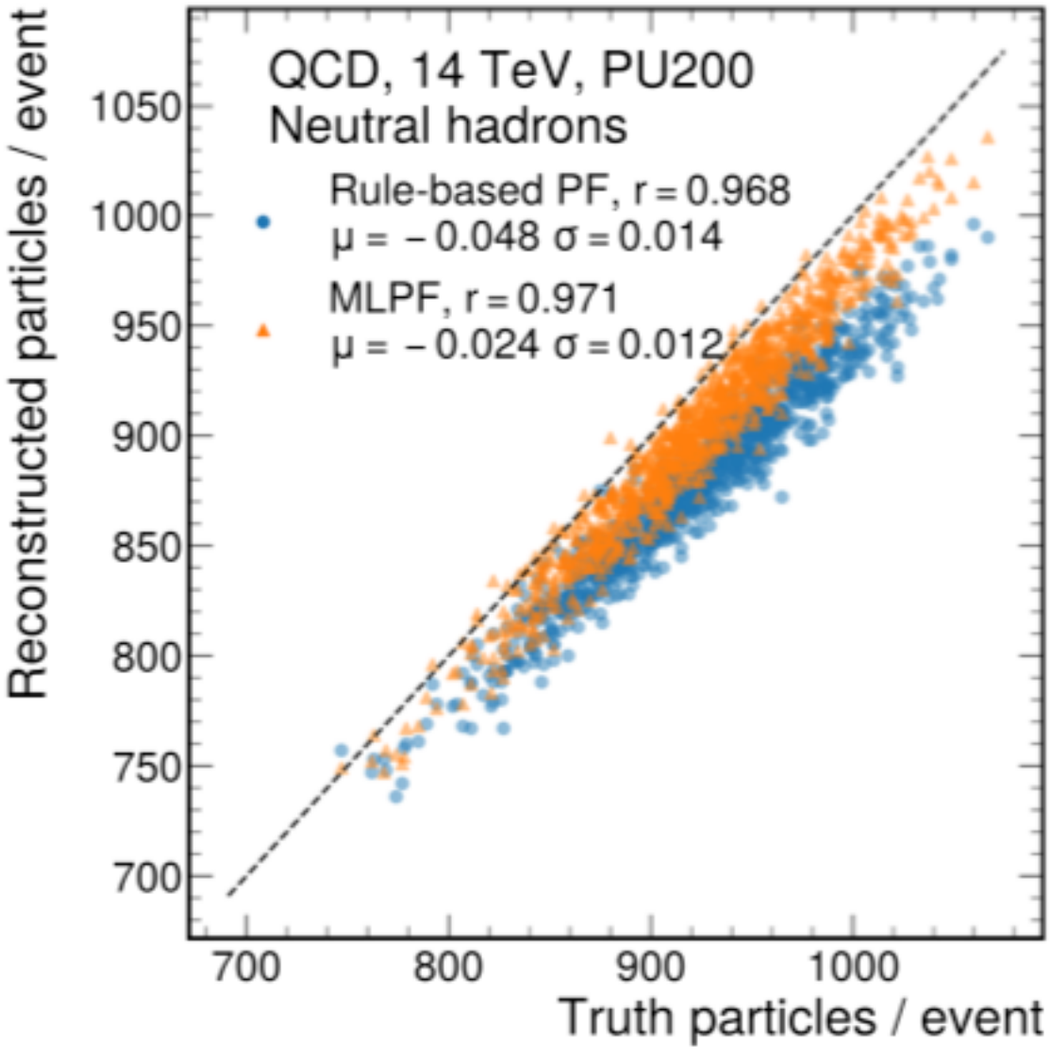
Trainable neural networks: $\mathcal{F}, \mathcal{G}, \mathcal{D}$

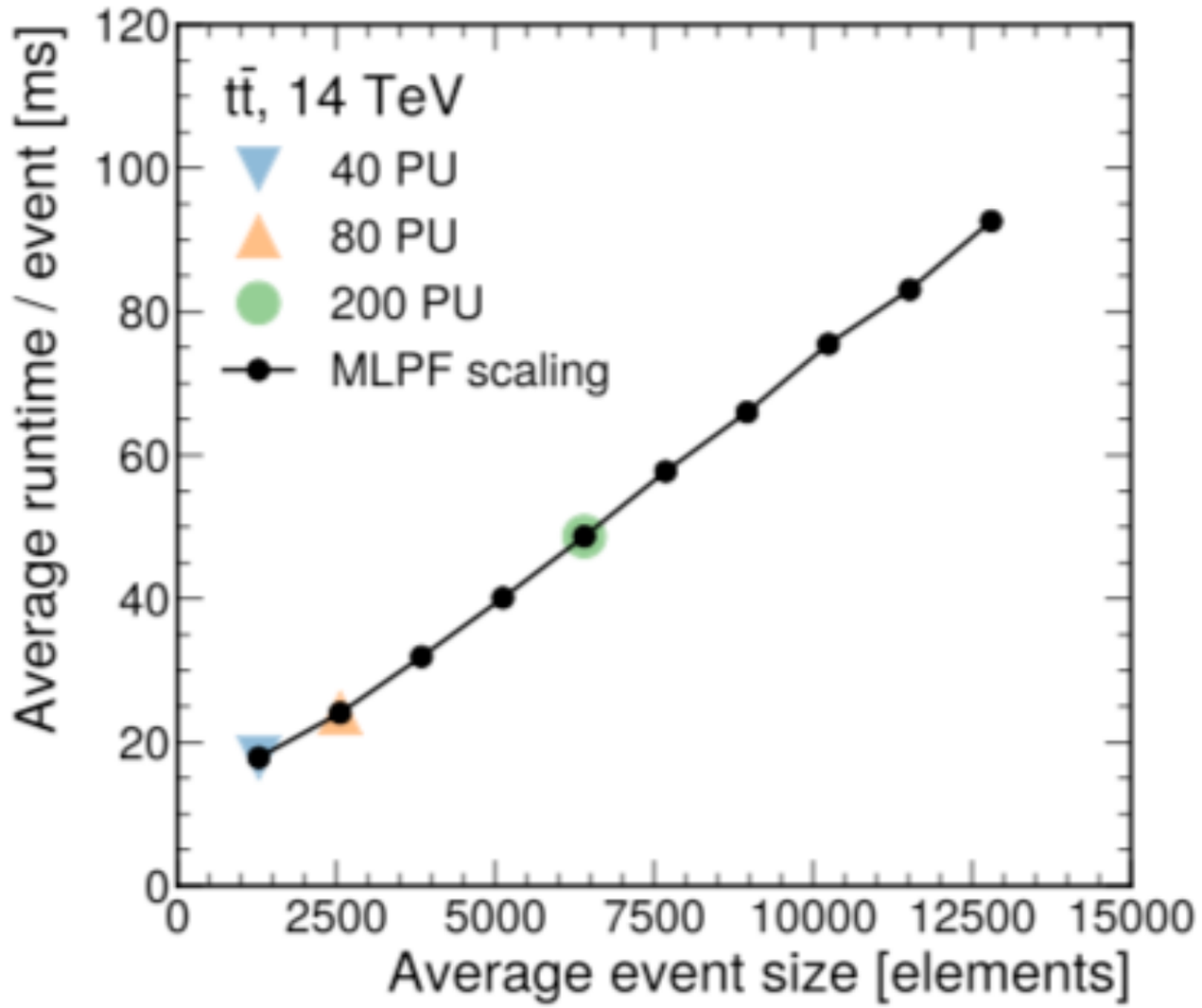
● - track, ■ - calorimeter cluster, ■ - encoded element

■ - target (predicted) particle, ■ - no target (predicted) particle

Performance in simulation

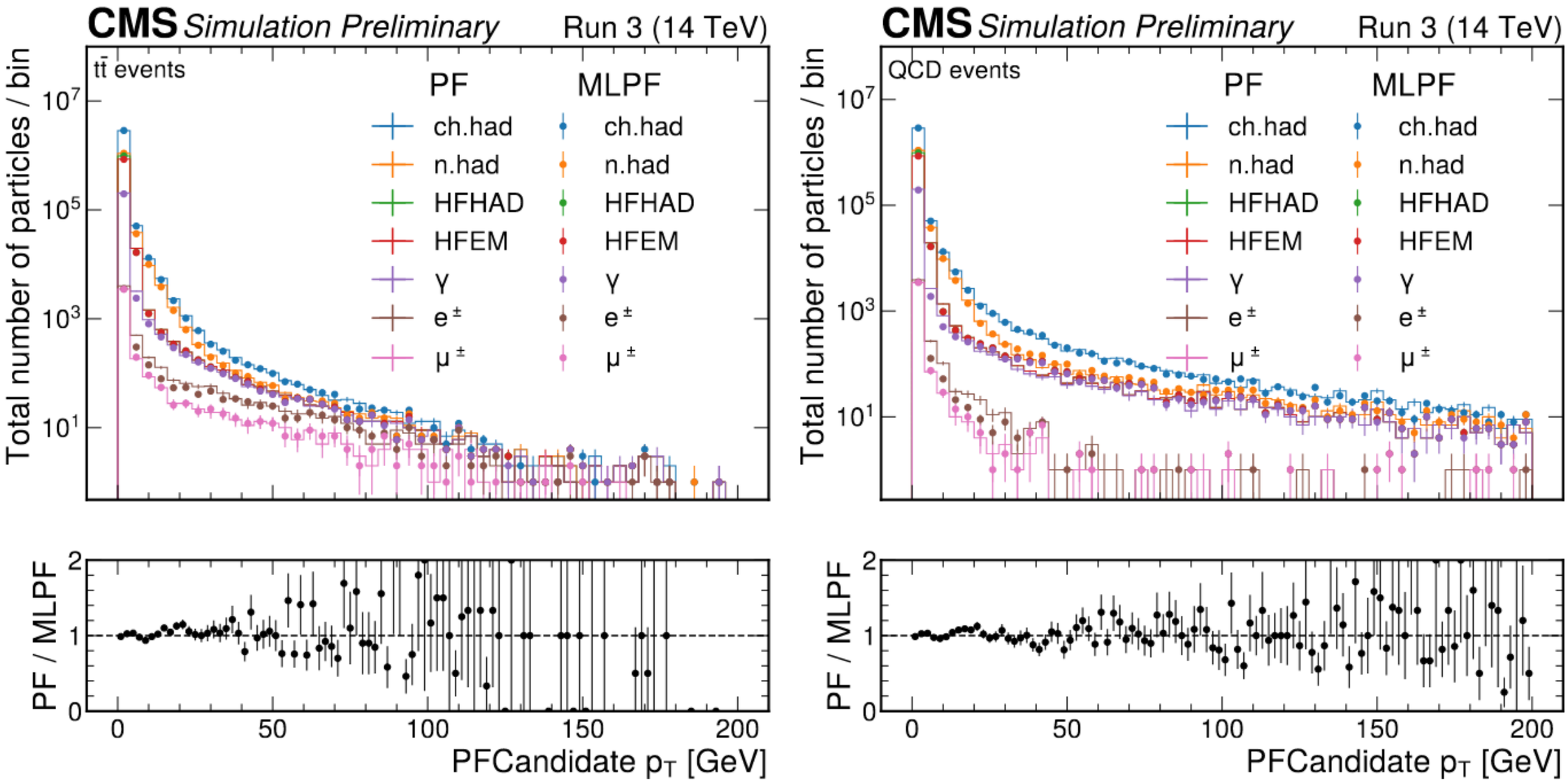
Predicts particle multiplicitly better than the baseline rule-based PF





Runtime (and memory) scale linearly with event size.

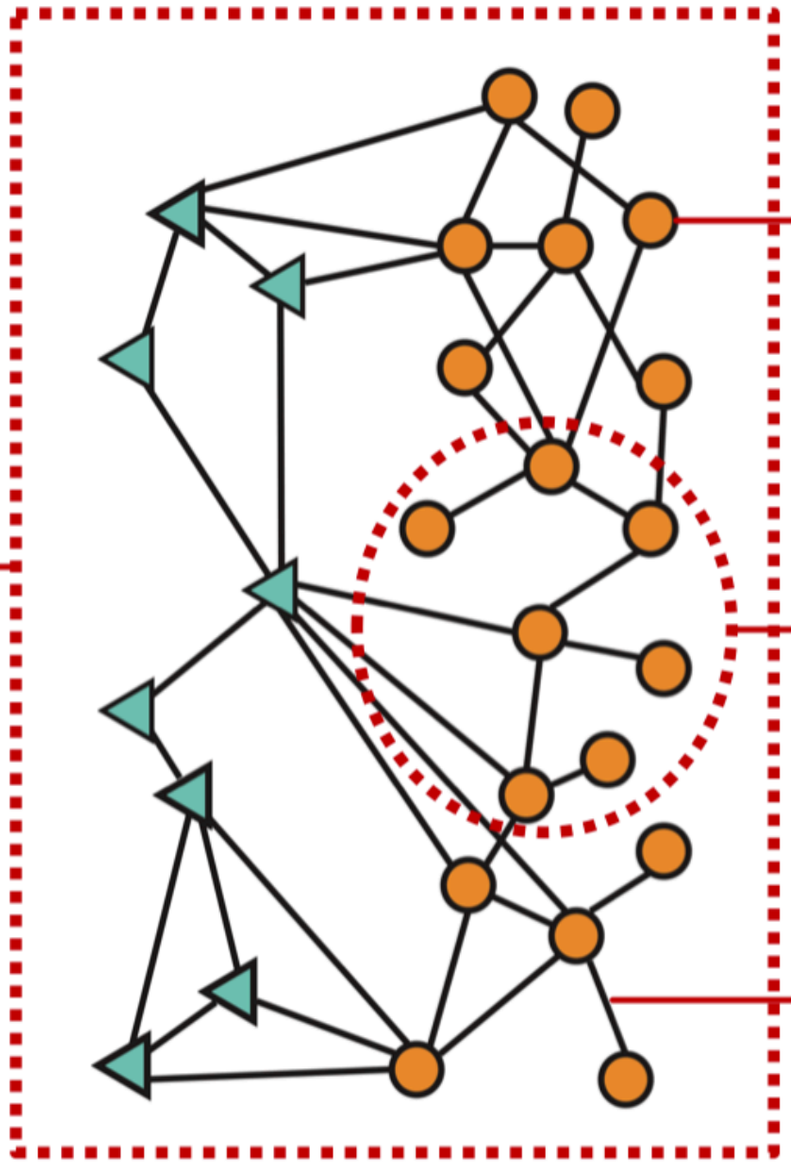
Performance in CMS



https://arxiv.org/pdf/2203.00330.pdf

Recap

Graph problems



Node level

Particle Flow reconstruction, detector hit segmentation

Graph-level prediction, Graph generation

Jet tagging, event tagging

Community (subgraph) level

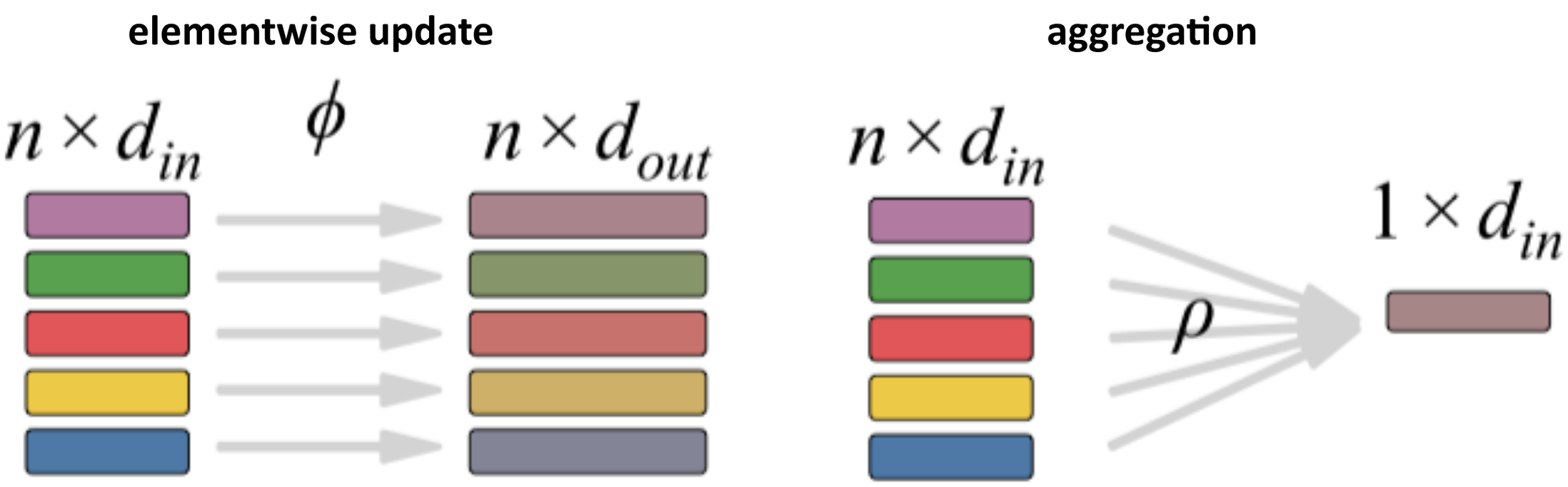
detector hit clustering

Edge-level

Track reconstruction

J. Leskovec et al [2021]

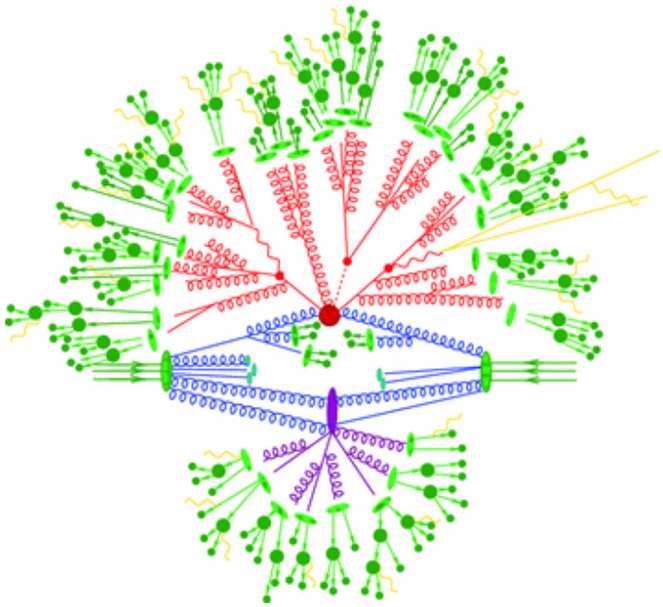
Graph operations



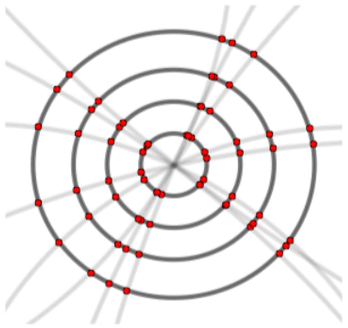
Invariance with respect to permutations!

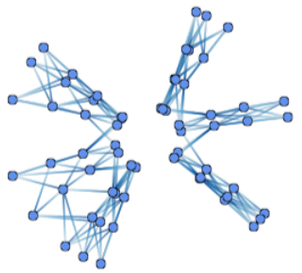
Graph structure

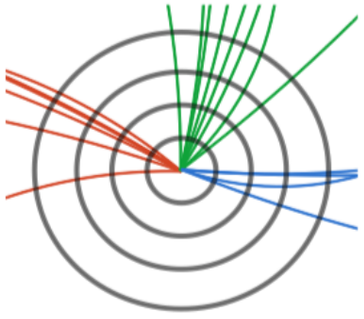
Defined by the process
(but not necessarily observable)

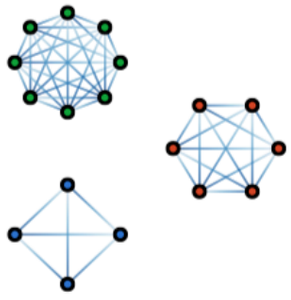


Assumed (static)

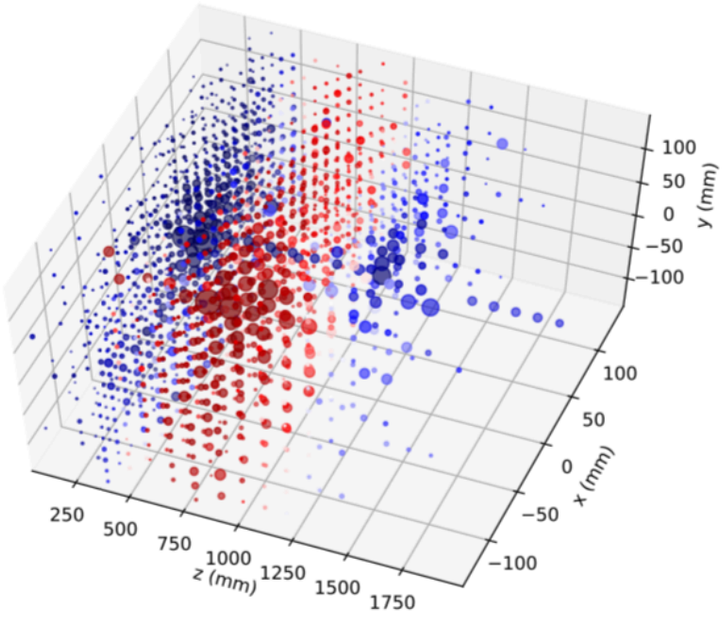


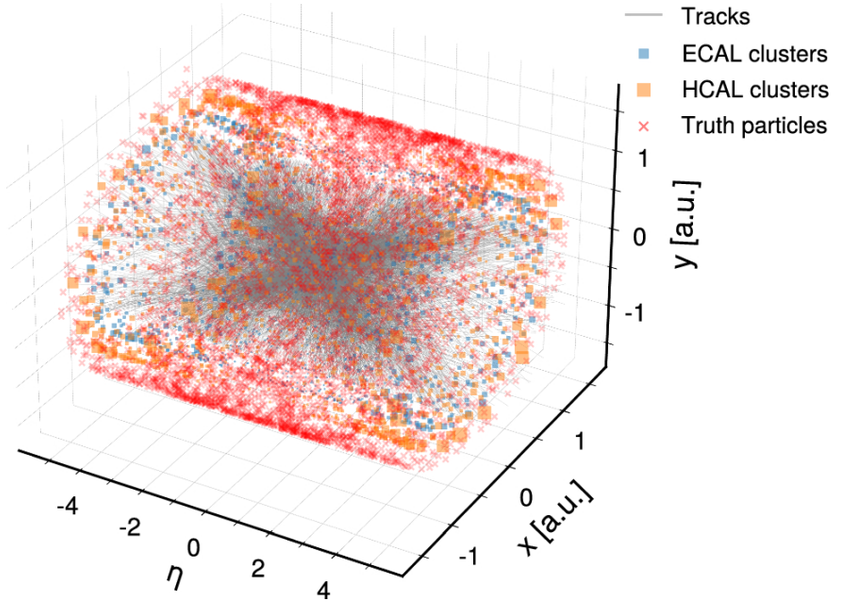


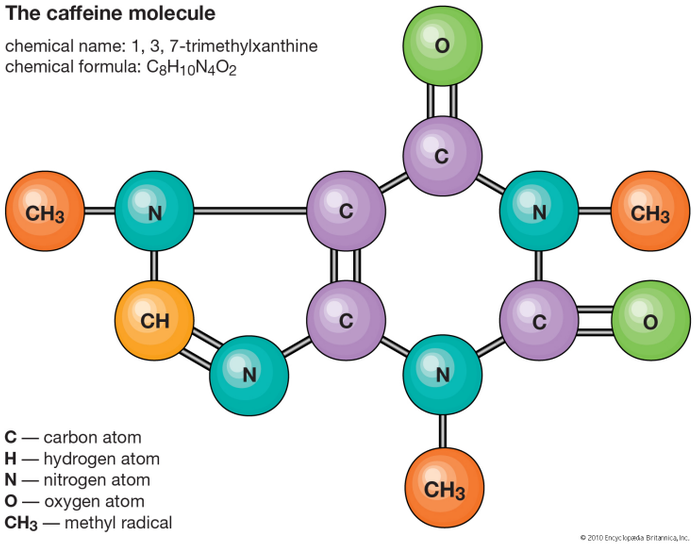




Learned (dynamic)







Advantages/disadvantages

+ Encode physics priors in the graph structure

+ Insensitive to the ordering of inputs

+ Sparse and irregular problem geometries

+ Efficient computation and memory representation

- Support for sparse data structures on GPUs and in DL libraries is not always great (but it's growing)

- It's not always obvious how to best cast the physics problem as a graph problem, i.e. what defines the graph

  - Graph structure may be mismeasured or not known accurately

- Deeper nets often do not perform better

Useful references

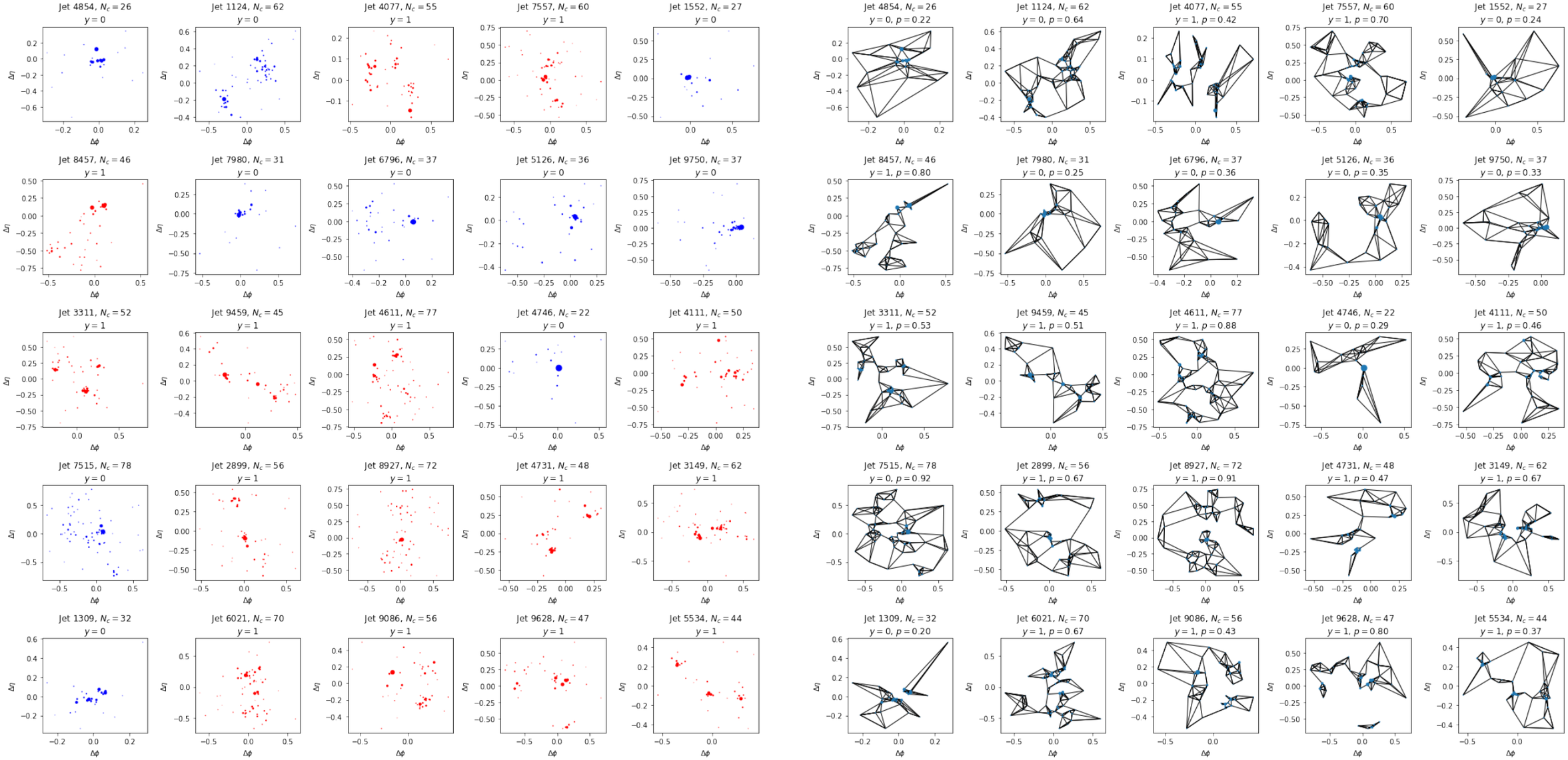
- HEPML Living Review: https://iml-wg.github.io/HEPML-LivingReview/

- ML on Graphs @ Stanford: http://web.stanford.edu/class/cs224w/

- Graph Representation Learning book (WIP): https://www.cs.mcgill.ca/~wlh/grl_book/

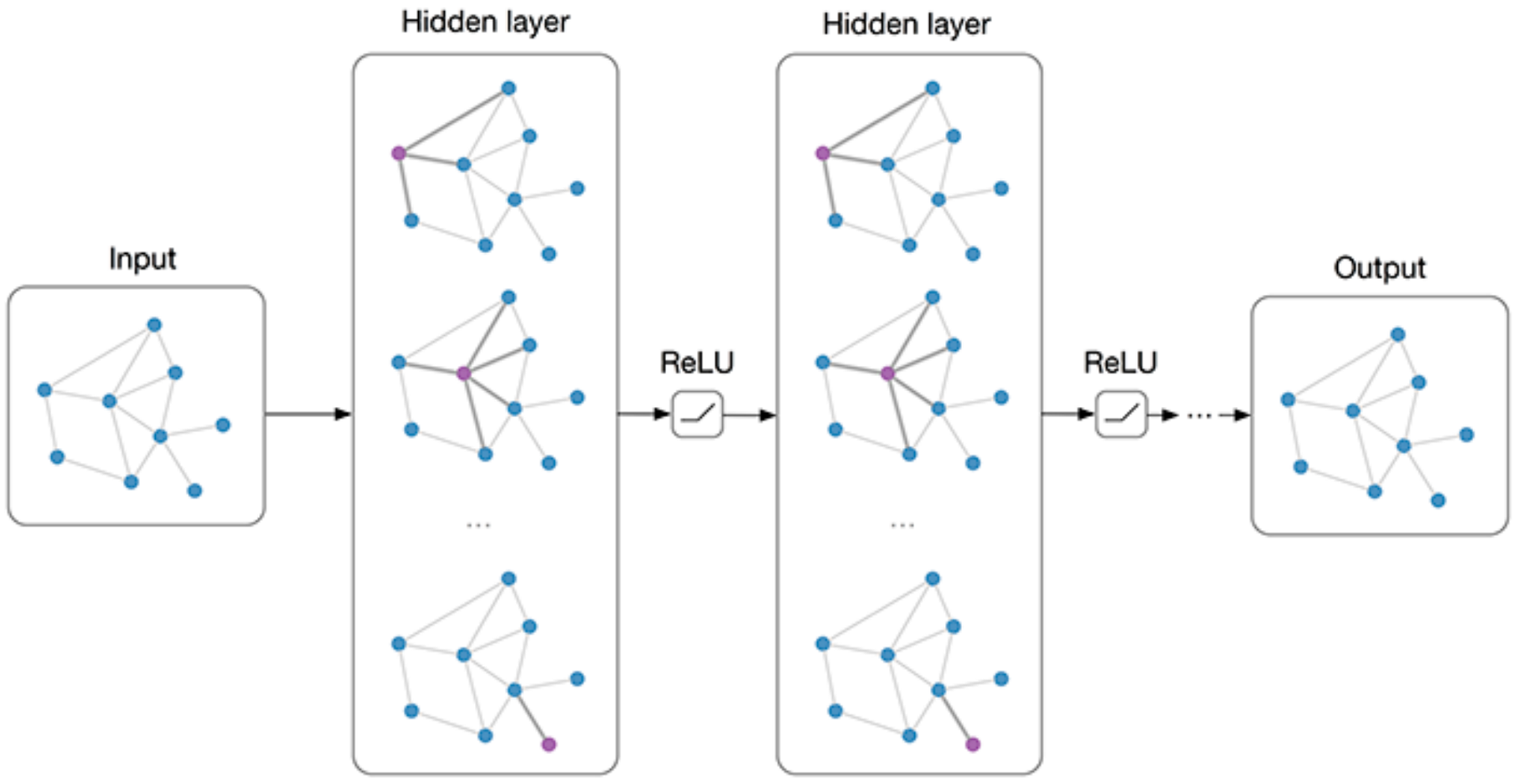
- Graph Neural Networks in Particle Physics: https://arxiv.org/pdf/2203.12852.pdf

Practical exercise



Jupyter notebook: github, colab

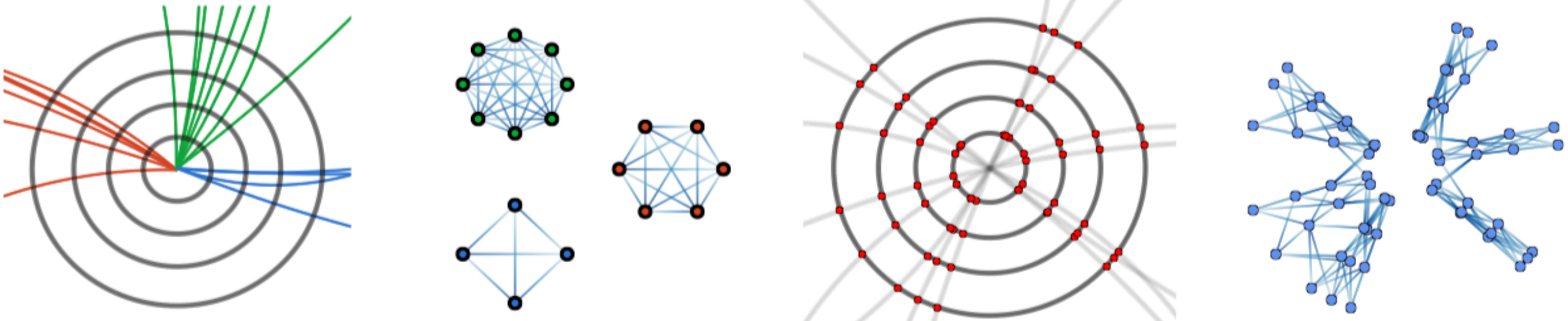
GNNs to Transformers



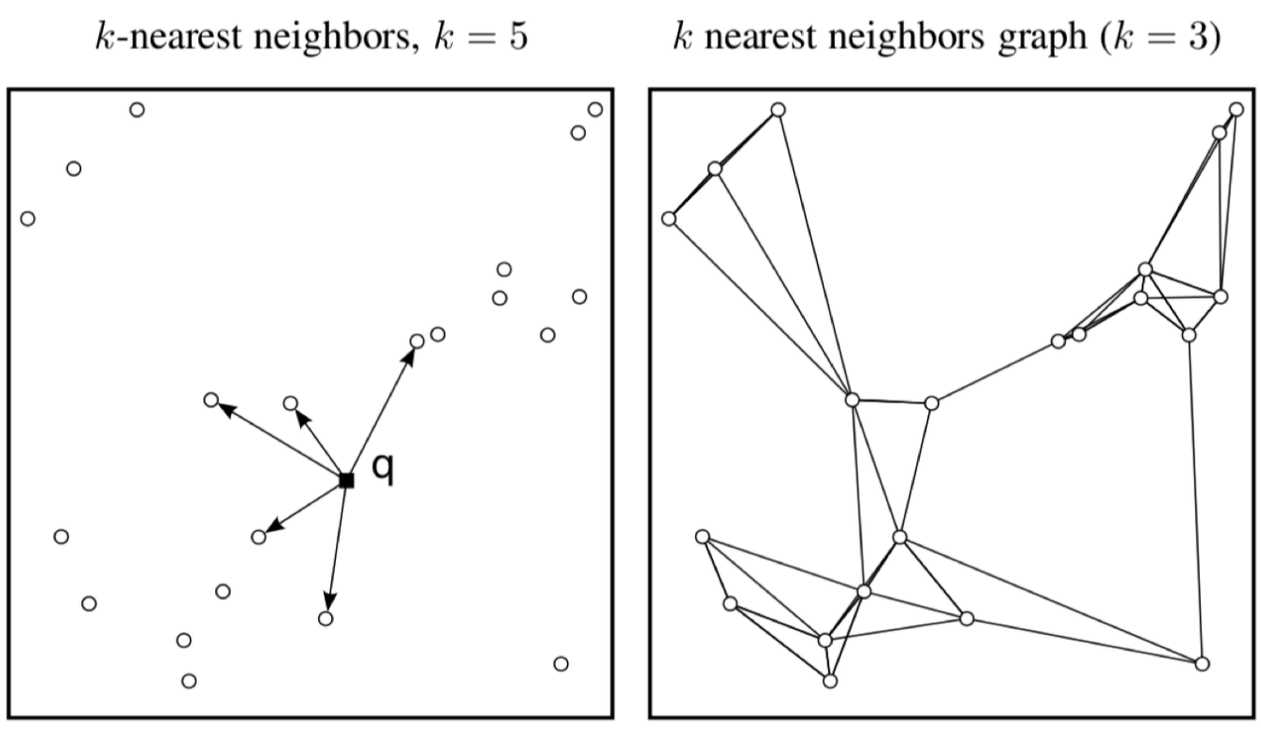
The GNN updates (encodes) each node, given information from nearby nodes, in a learnable way, to minimize an overall objective function.

GNNs need structure

explicit / predefined

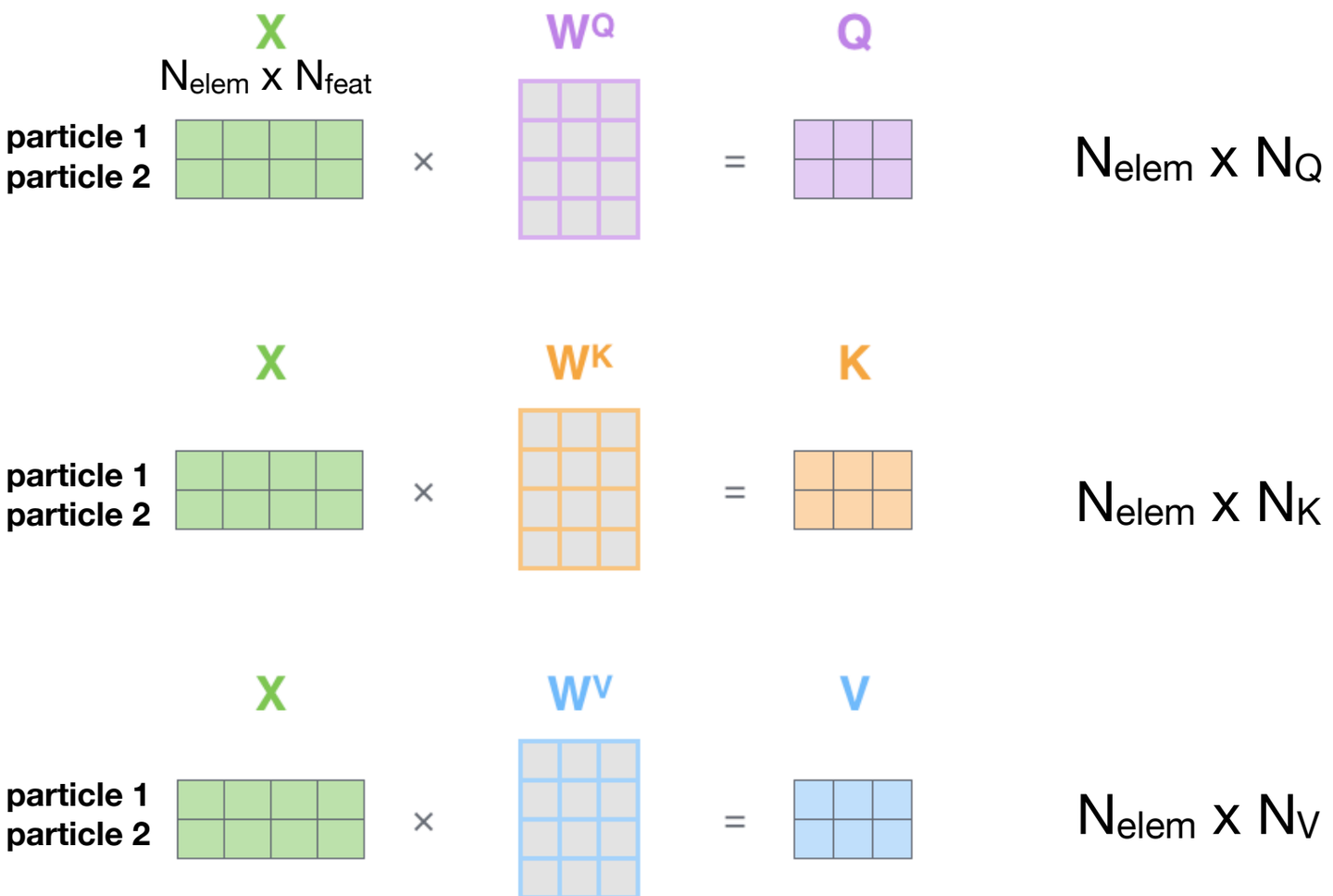


implicit / learnable



Self-attention

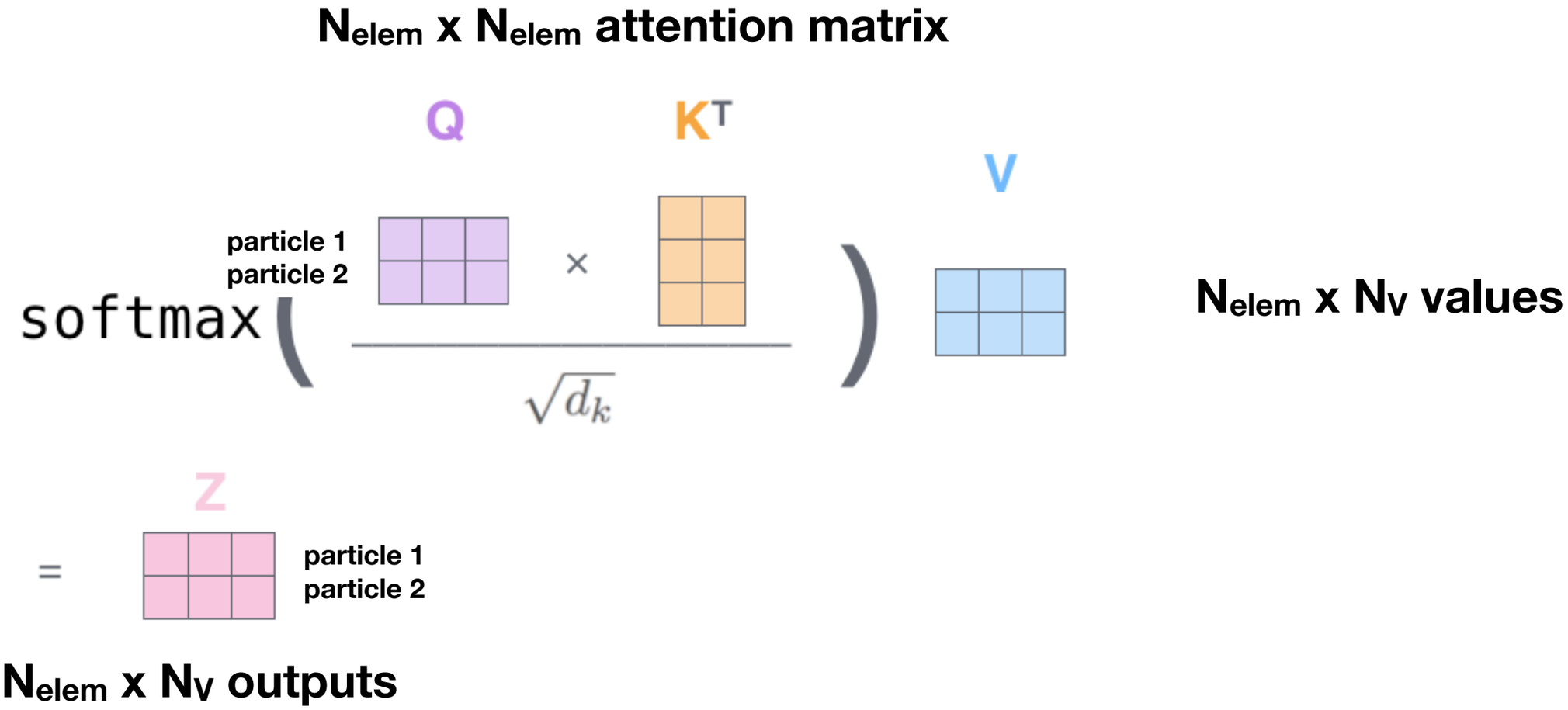
Project each element from input space X to Q,K,V with learnable weights.



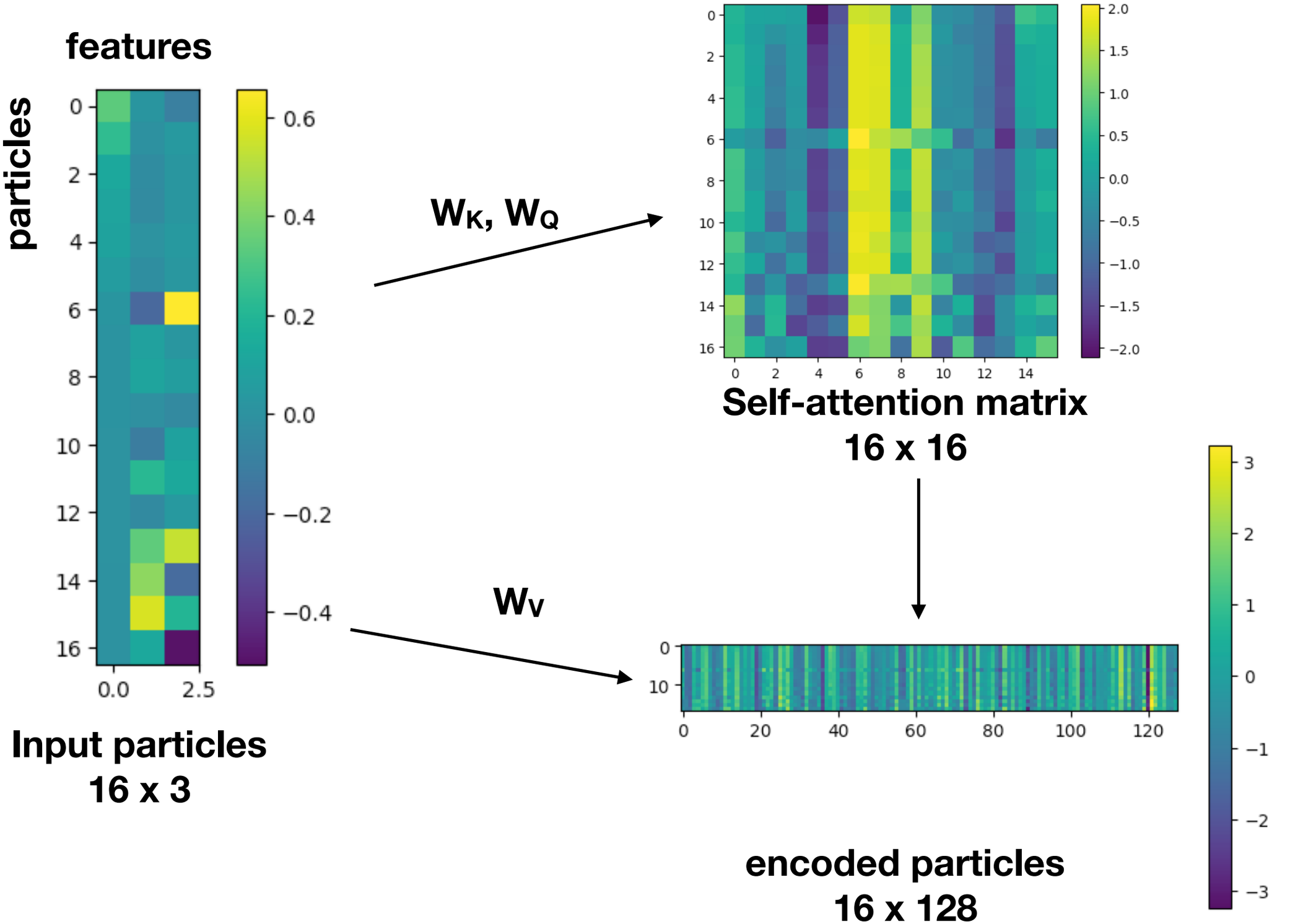
https://arxiv.org/pdf/2108.04253.pdf

https://jalammar.github.io/illustrated-transformer/

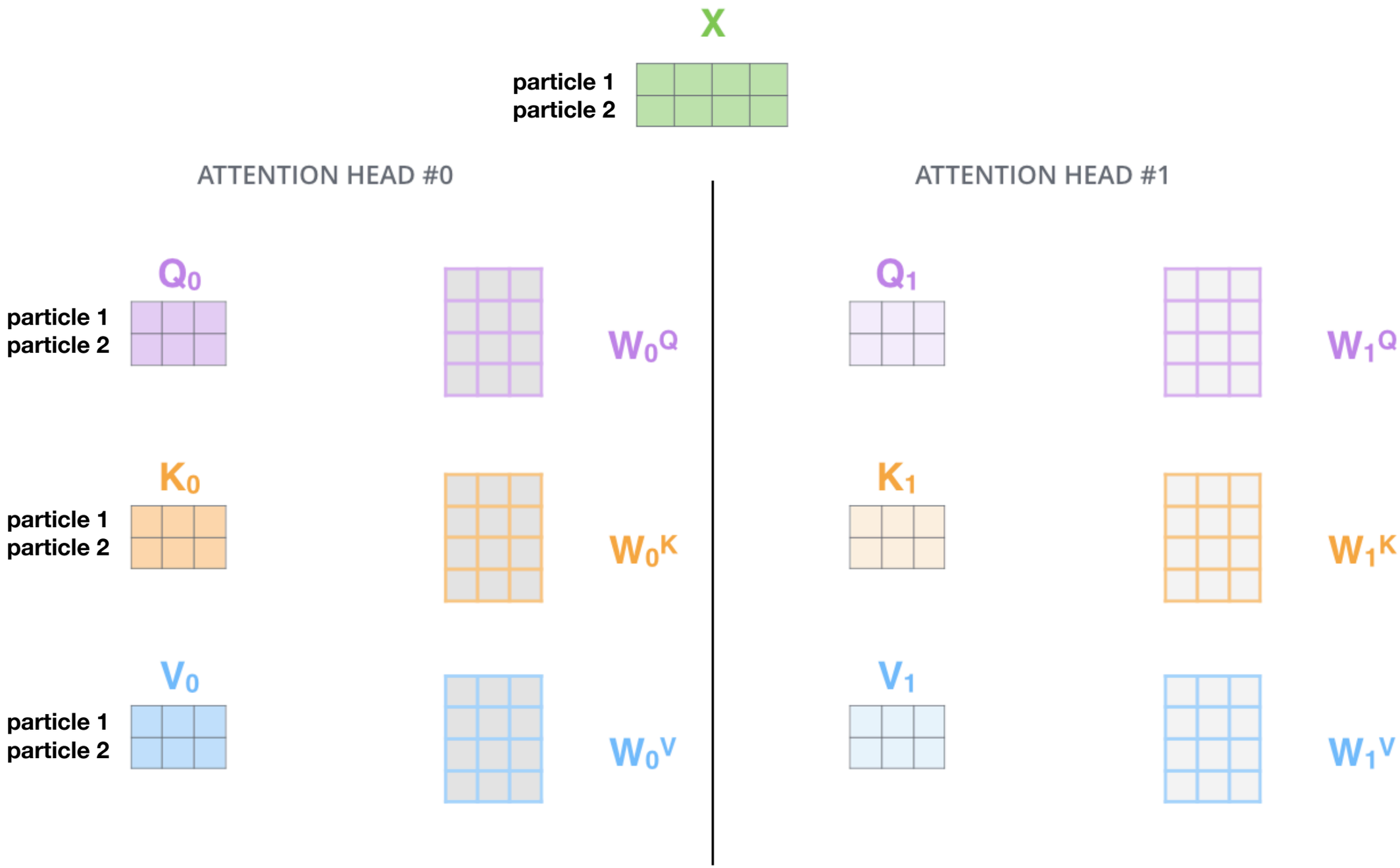
Self-attention outputs



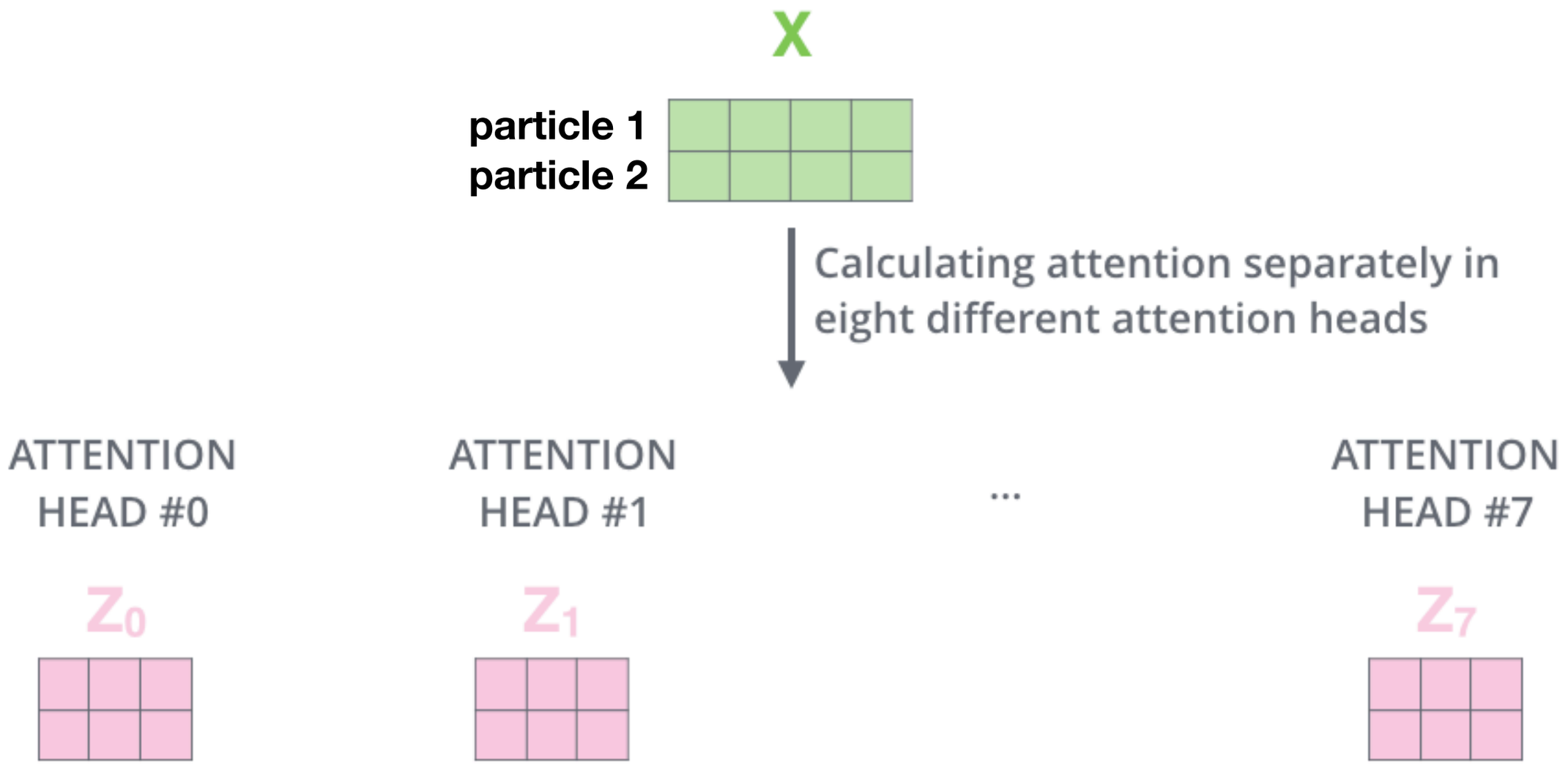
- Compute the matrix product of per-element queries with per-element keys
- Retrieve the corresponding values according to softmax-normalized attention matrix


features
particles
0
2
4
6
8
10
12
14
16
0.0
2.5
0.6
0.4
0.2
0.0
−0.2
−0.4
Input particles
16 x 3
W_K, W_Q
0
2
4
6
8
10
12
14
16
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
Self-attention matrix
16 x 16
W_V
0
10
0
20
40
60
80
100
120
3
2
1
0
−1
−2
−3
encoded particles
16 x 128

Multi-head attention

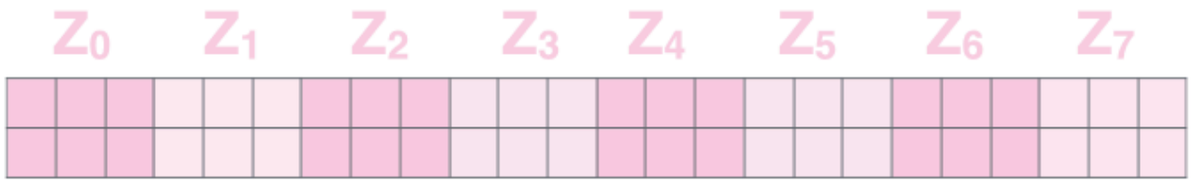


Multi-head outputs



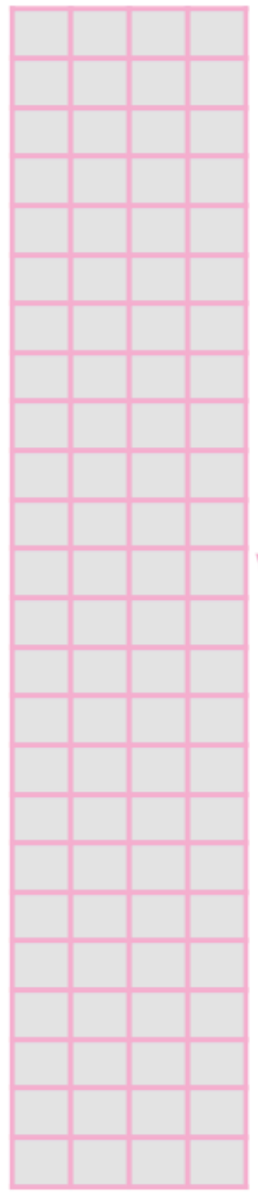
Input to a subsequent task

1) Concatenate all the attention heads



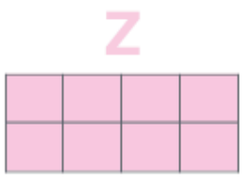
2) Multiply with a weight matrix W^O that was trained jointly with the model

X

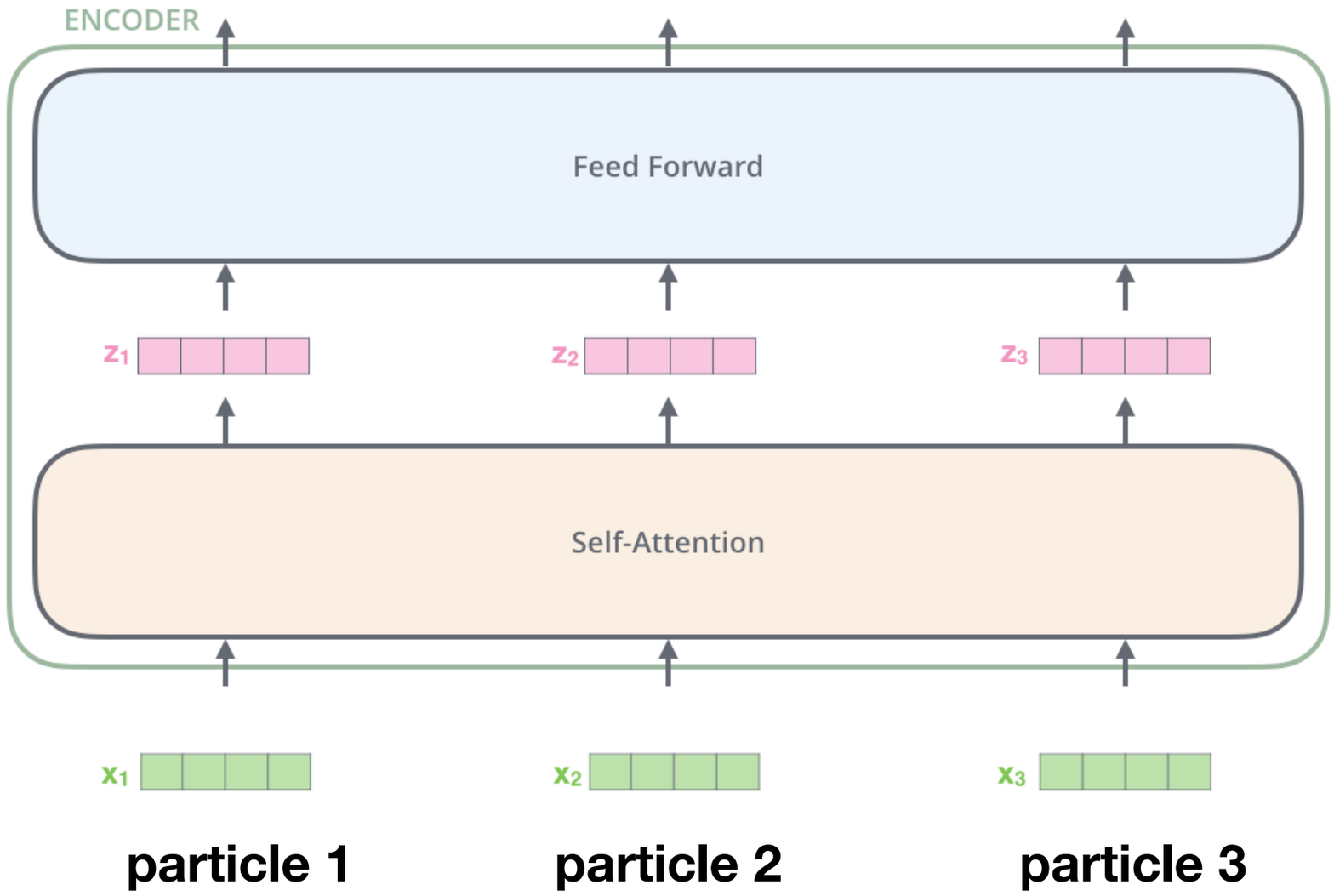


3) The result would be the Z matrix that captures information from all the attention heads. We can send this forward to the FFNN

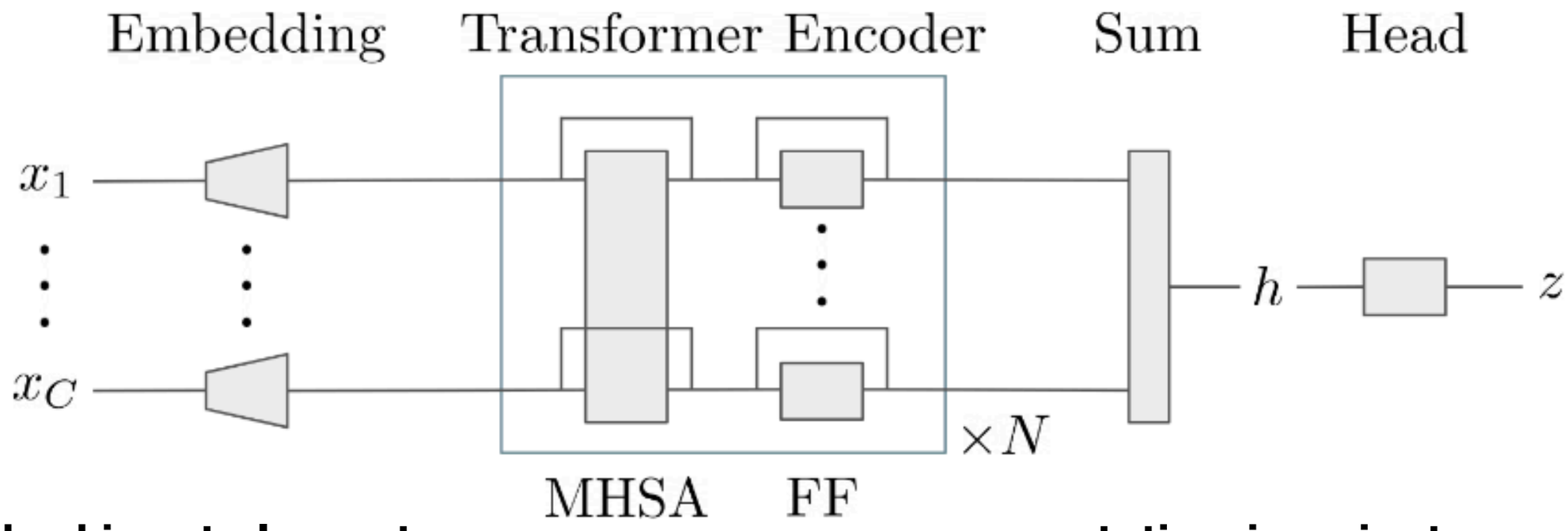
=



Self-attention to encoder



Permutation invariance



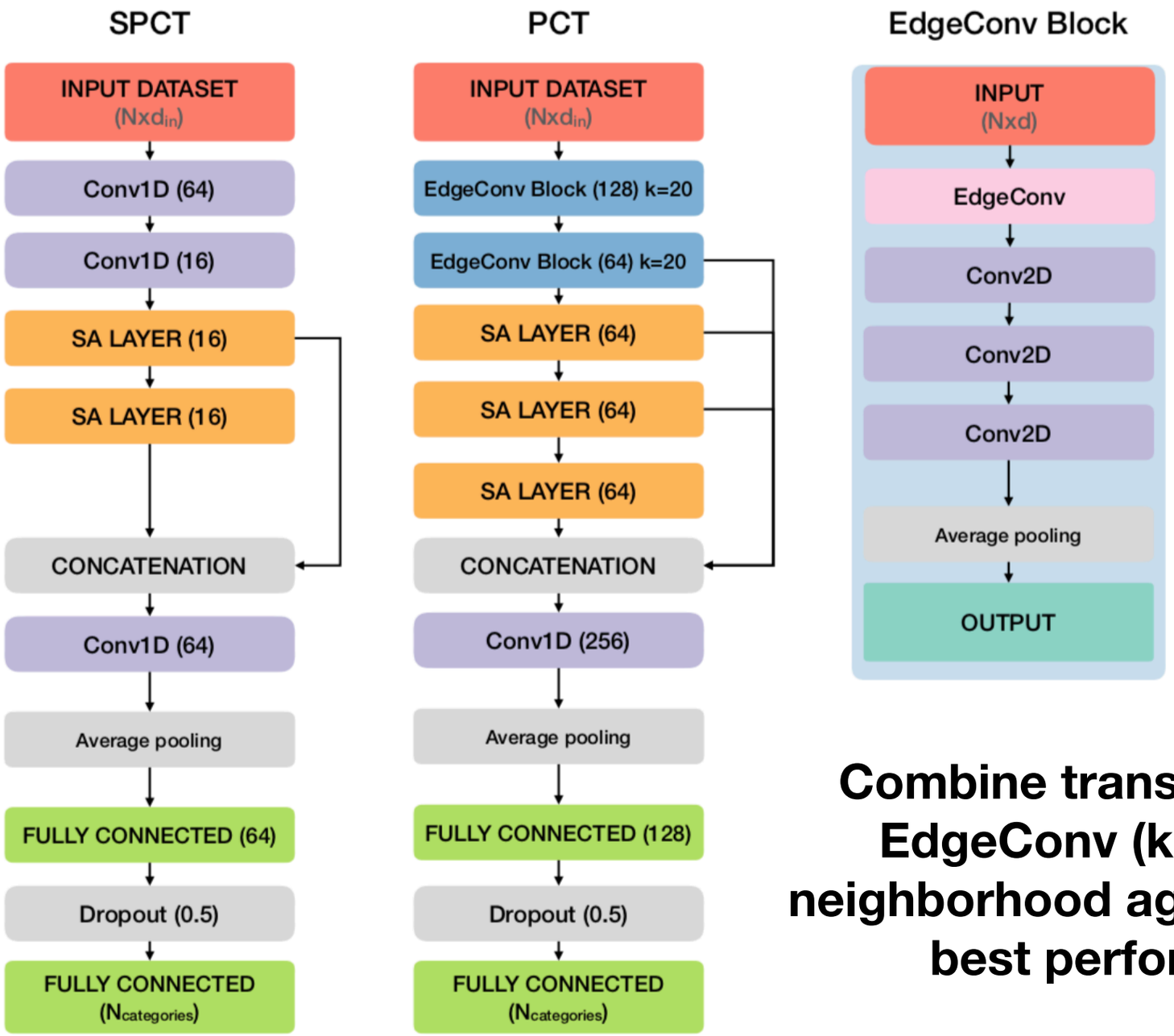
embed input elements with FFN

transform input elements with multi-haded self-attention

permutation-invariant aggregation

https://www.scipost.org/SciPostPhys.12.6.188/pdf

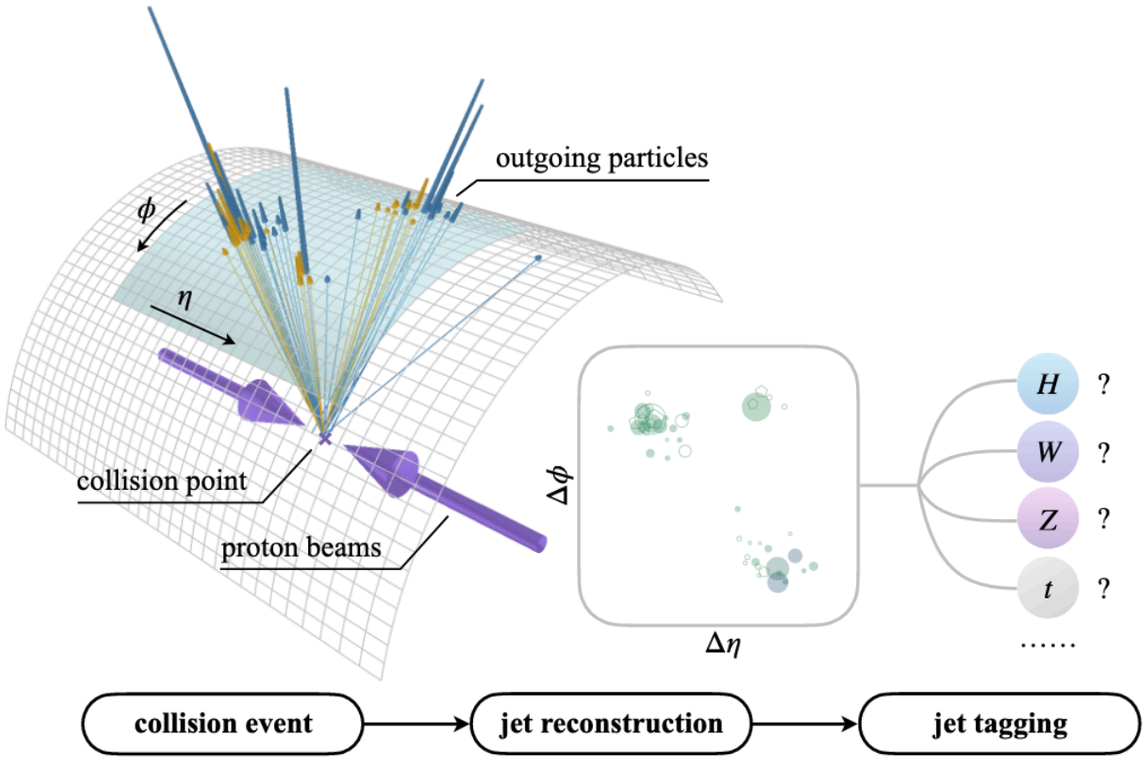
Point cloud transformer



Combine transformer with EdgeConv (kNN-based neighborhood aggregation) for best performance.

https://arxiv.org/abs/2102.05073

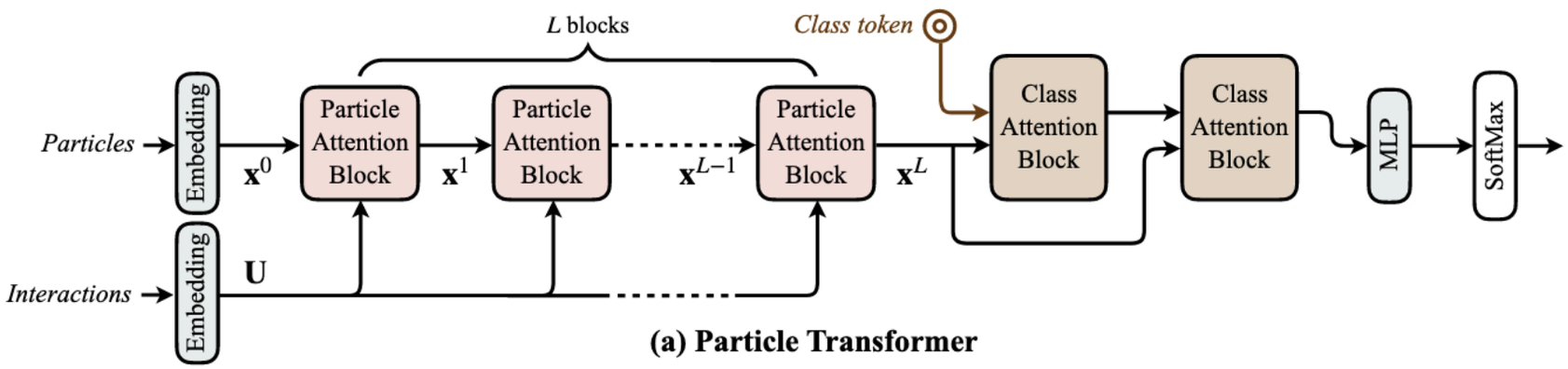
Jet tagging via transformers



Add explicit particle-particle interaction terms.

task. Specifically, for a pair of particles a, b with 4-vectors p_a, p_b, we calculate the following 4 features:

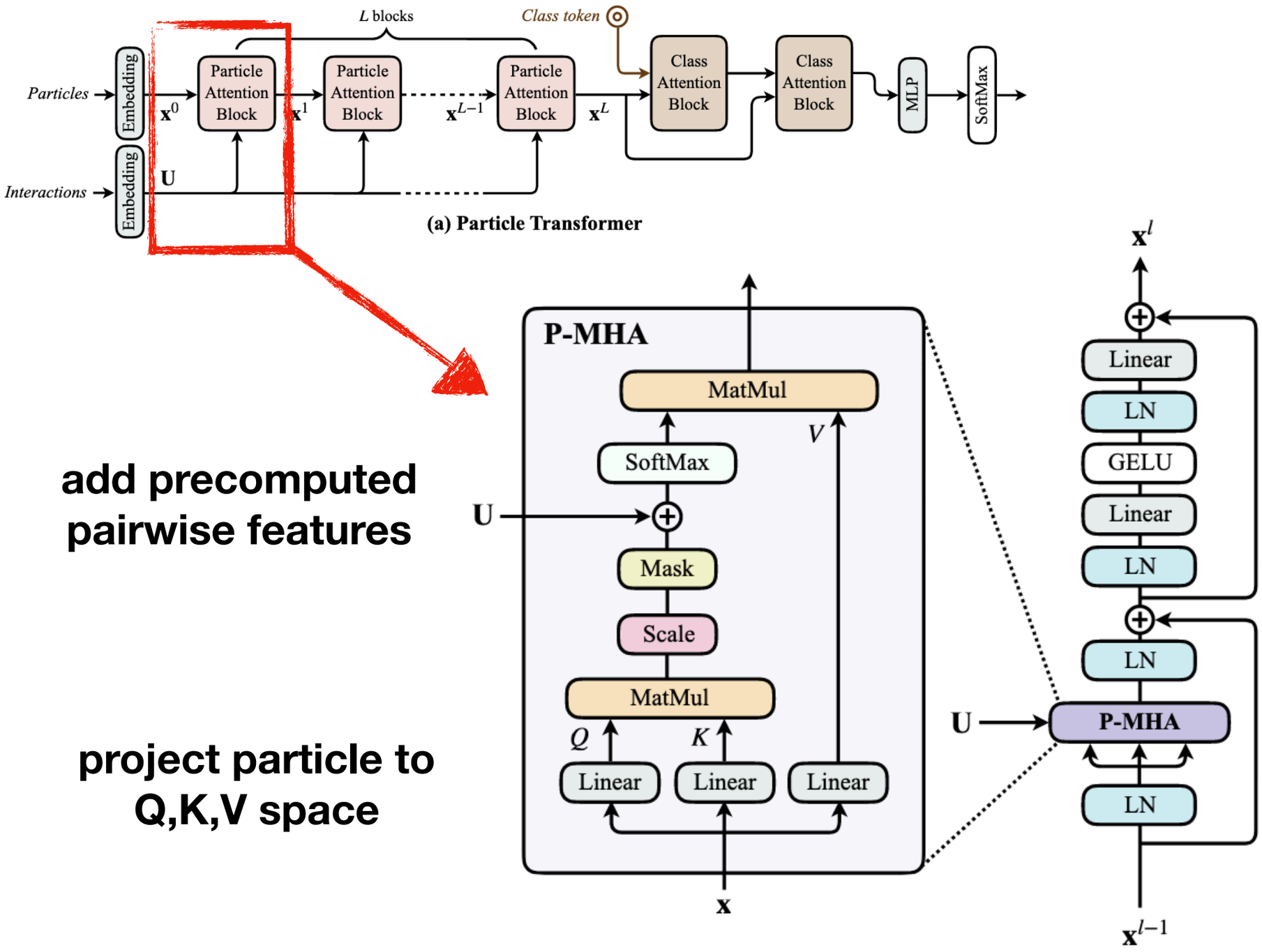
$$\begin{aligned} \Delta &= \sqrt{(y_a - y_b)^2 + (\phi_a - \phi_b)^2}, \\ k_{\mathrm{T}} &= \min(p_{\mathrm{T},a}, p_{\mathrm{T},b})\Delta, \\ z &= \min(p_{\mathrm{T},a}, p_{\mathrm{T},b})/(p_{\mathrm{T},a} + p_{\mathrm{T},b}), \\ m^2 &= (E_a + E_b)^2 - \|\mathbf{p}_a + \mathbf{p}_b\|^2, \end{aligned} \tag{3}$$



(a) Particle Transformer

https://arxiv.org/pdf/2202.03772.pdf

Particle attention block

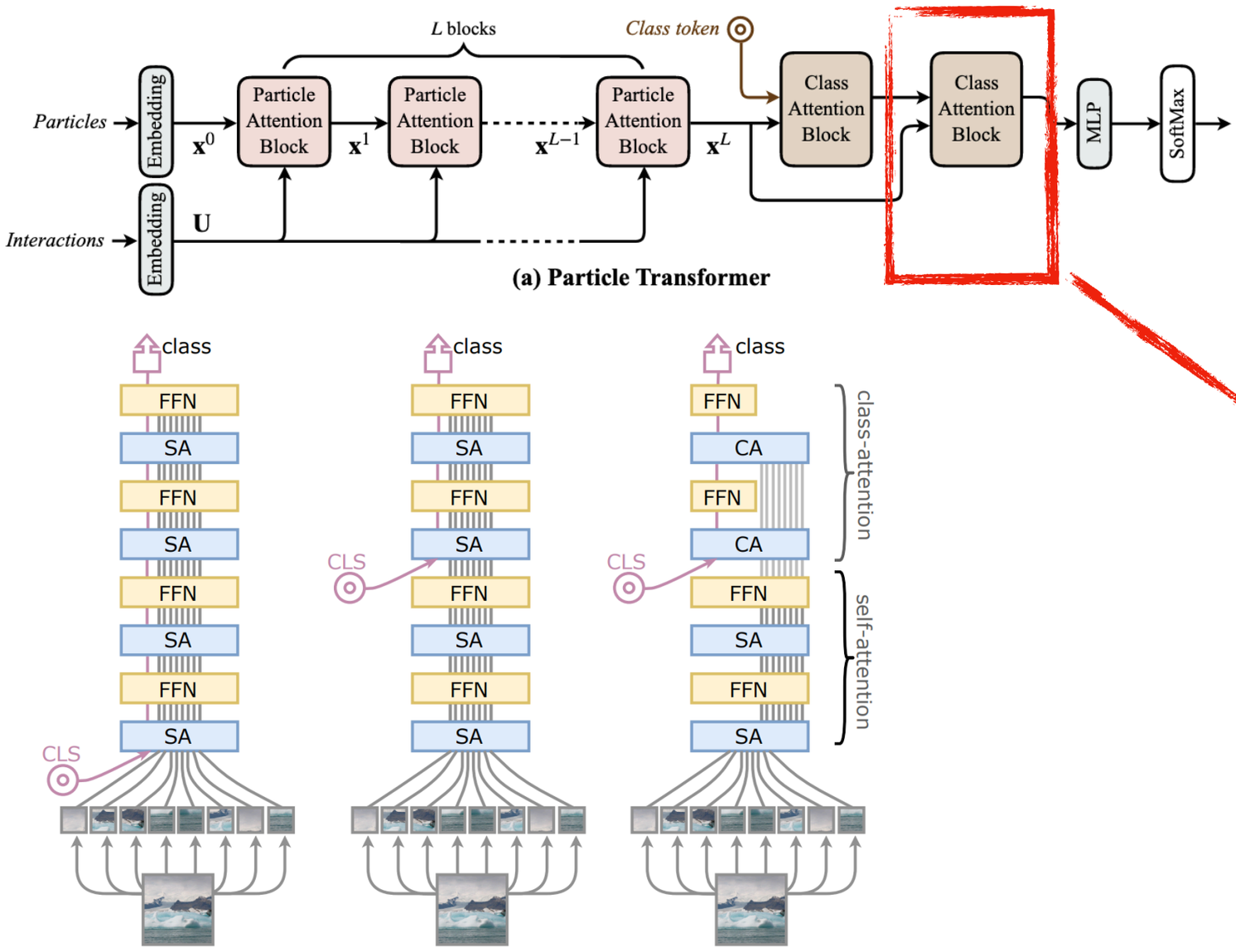


add precomputed pairwise features

project particle to Q,K,V space

encoded particle data

Class attention



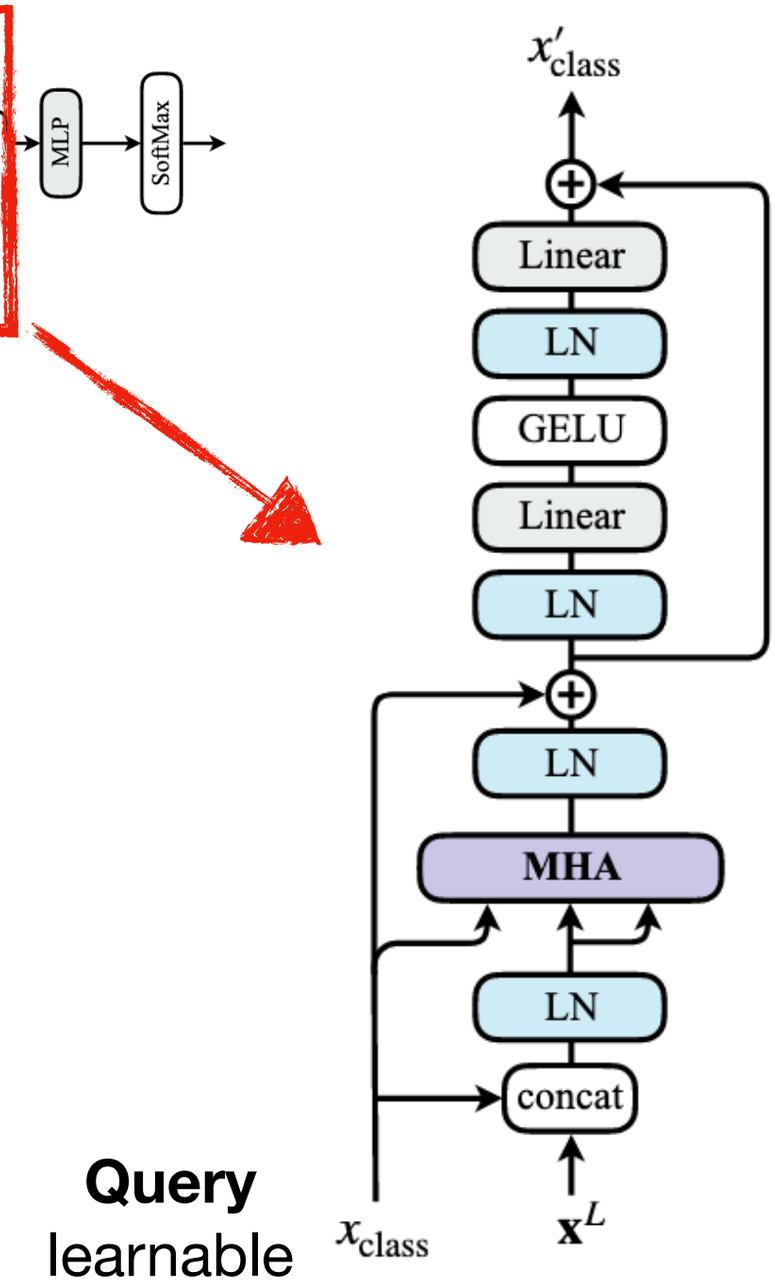
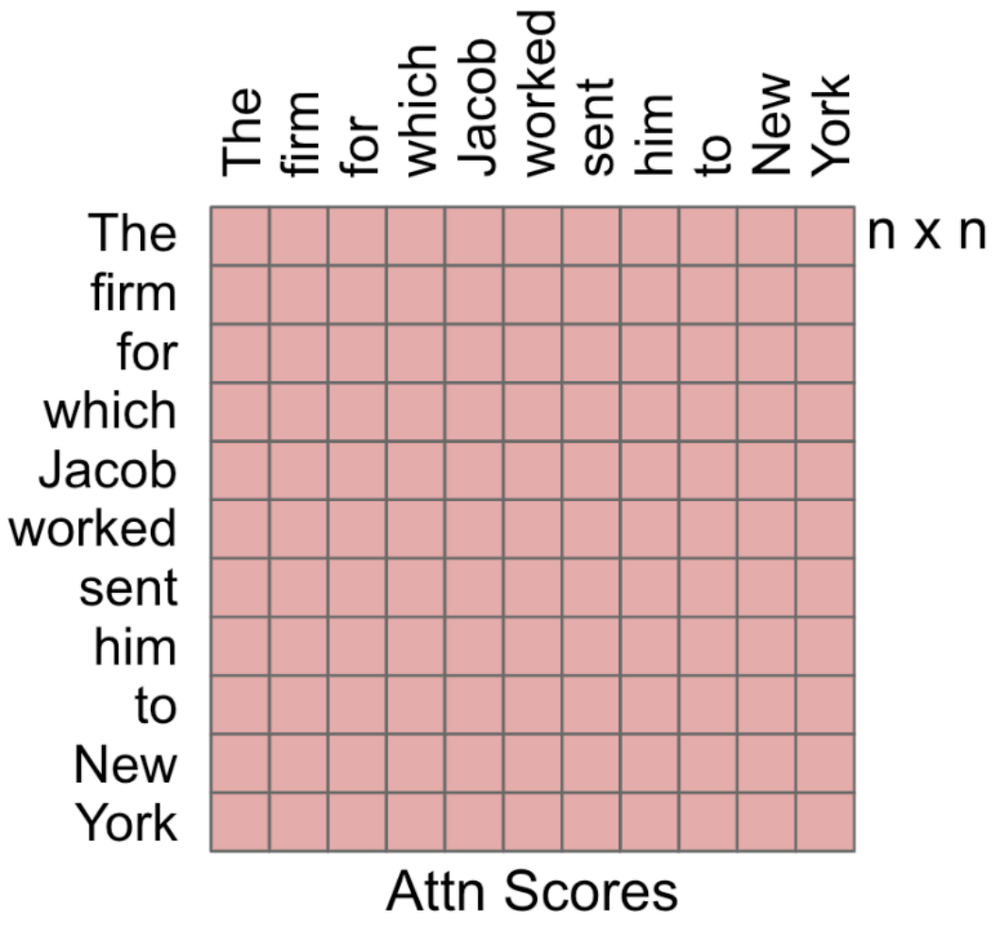
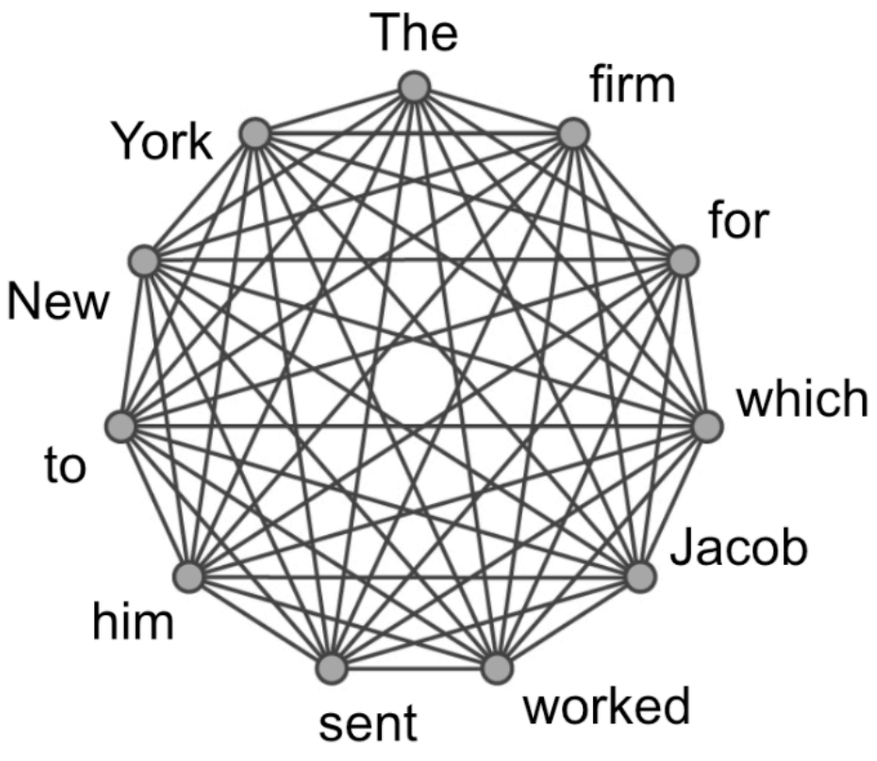


Figure 2: In the ViT transformer (*left*), the class embedding (CLS) is inserted along with the patch embeddings. This choice is detrimental, as the same weights are used for two different purposes: helping the attention process, and preparing the vector to be fed to the classifier. We put this problem in evidence by showing that inserting CLS later improves performance (*middle*). In the **CaiT** architecture (*right*), we further propose to freeze the patch embeddings when inserting CLS to save compute, so that the last part of the network (typically 2 layers) is fully devoted to summarizing the information to be fed to the linear classifier.

Attention as a graph



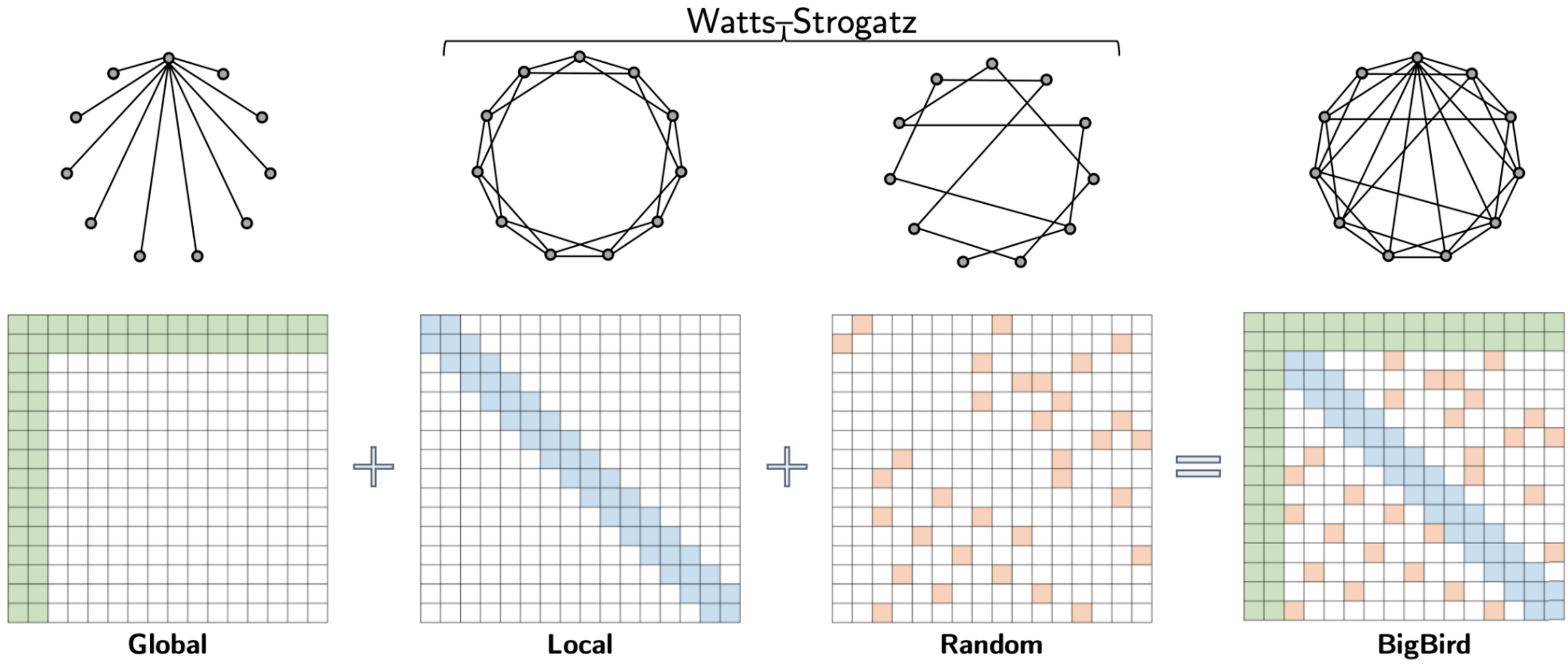


Full self attention ~ all-to-all dense graph.
Naively N^2 time/memory complexity!

https://ai.googleblog.com/2021/03/constructing-transformers-for-longer.html

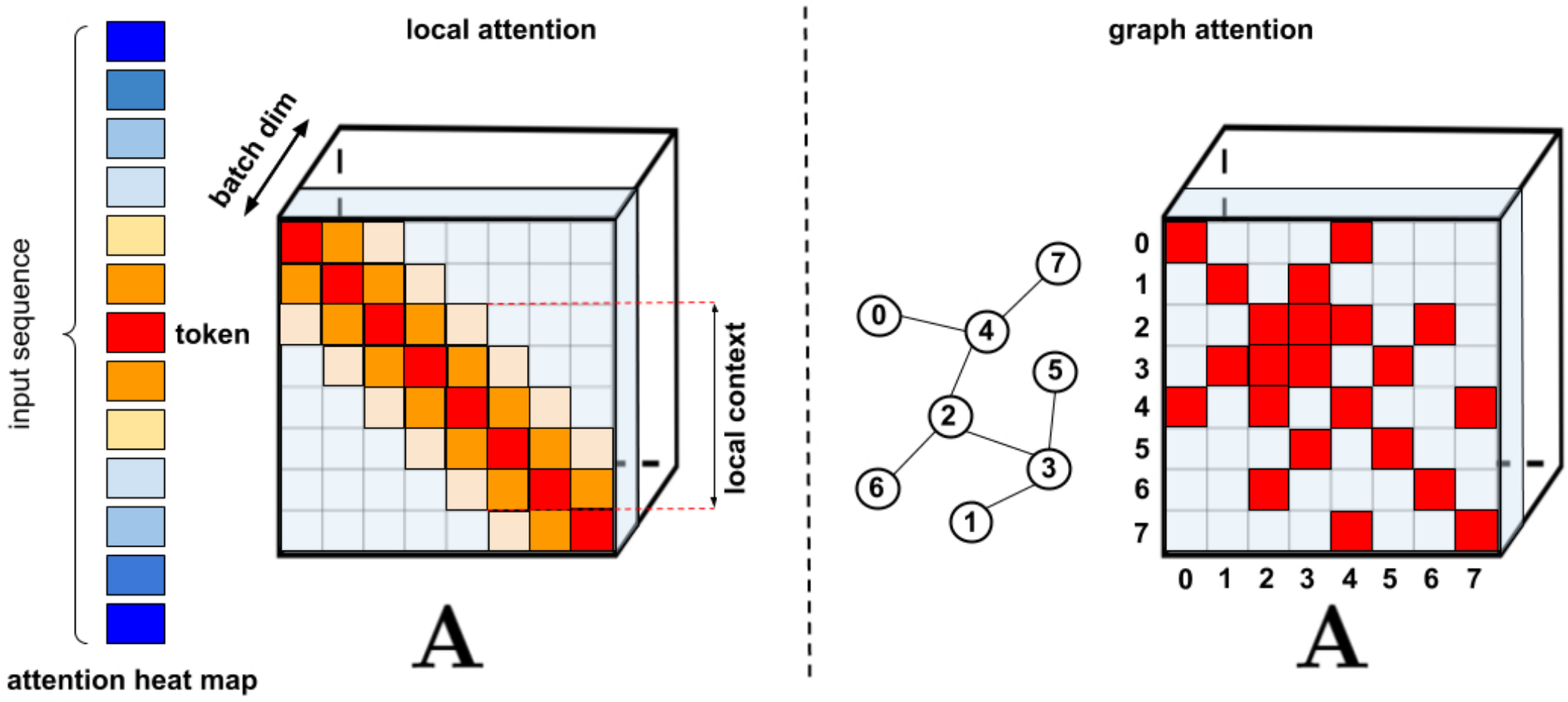
Global vs. local attention

In case the input is an ordered sequence (e.g. a long text, big image), attention can be constrained to local regions.



GNNs vs. Transformers

Most state-of-the-art language processing models use an attention-based "transformer" architecture: a dense attention matrix with elements A_{ij} is computed between input elements x_i. The attention matrix **A** is used to successively transform the input elements.

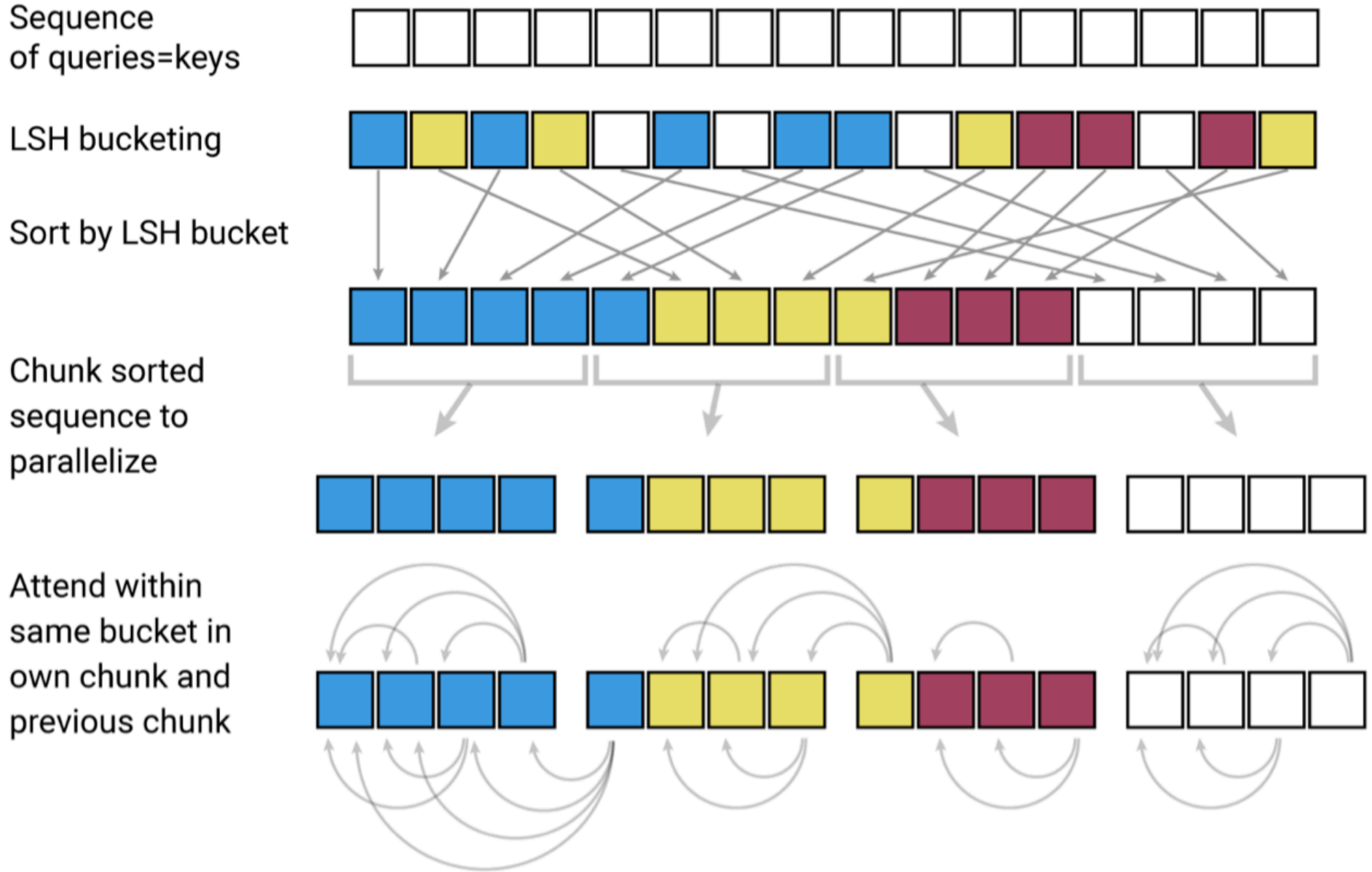


In GNNs, the learned graph adjacency is usually sparse, but is similarly used to propagate information between associated input elements to transform them.

https://ai.googleblog.com/2020/10/rethinking-attention-with-performers.html

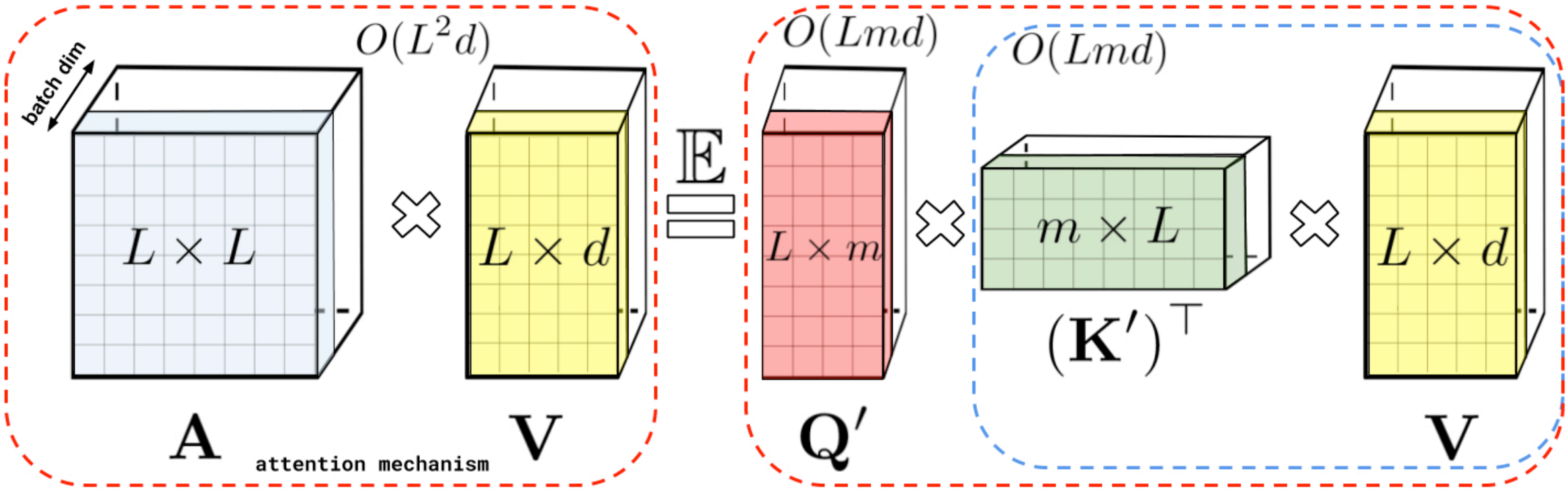
Scalable pairwise operations

Naive kNN graph / attention matrix construction (e.g. *tf.nn.top_k*) scales as $O(N^2)$ with the number of input nodes N.



Kitaev, Nikita, Łukasz Kaiser, and Anselm Levskaya. "Reformer: The efficient transformer." *arXiv preprint arXiv:2001.04451* (2020).

Approximating the attention

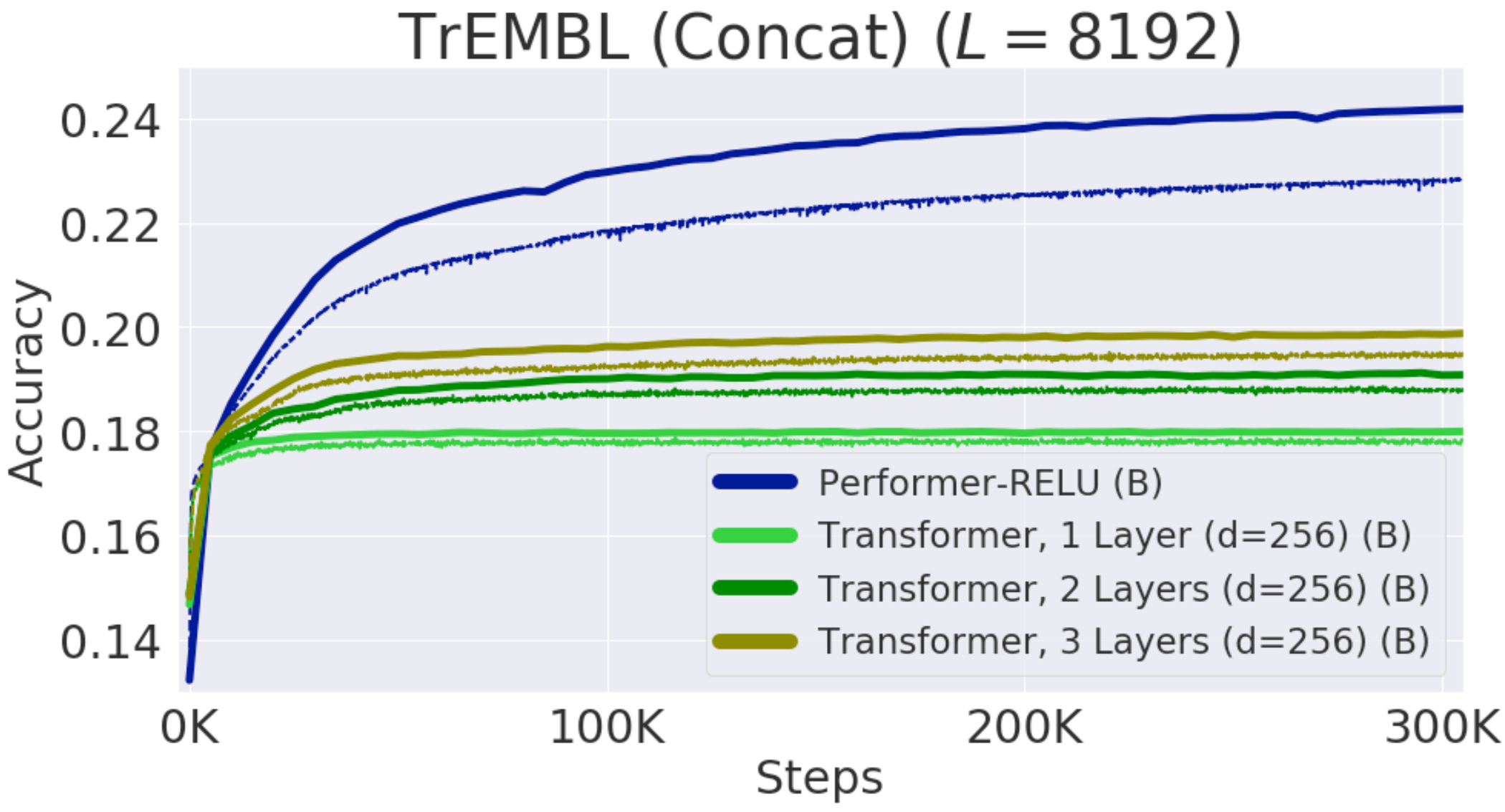


- The hyperparameter m << L of random projections to make for each Q, K tunes how closely the attention mechanism is approximated

- Does not rely on sparsity / structure in the attention matrix

https://ai.googleblog.com/2020/10/rethinking-attention-with-performers.html

Training on large sequences



Domain-agnostic models

To what extent can one use a single model architecture on different data domains?

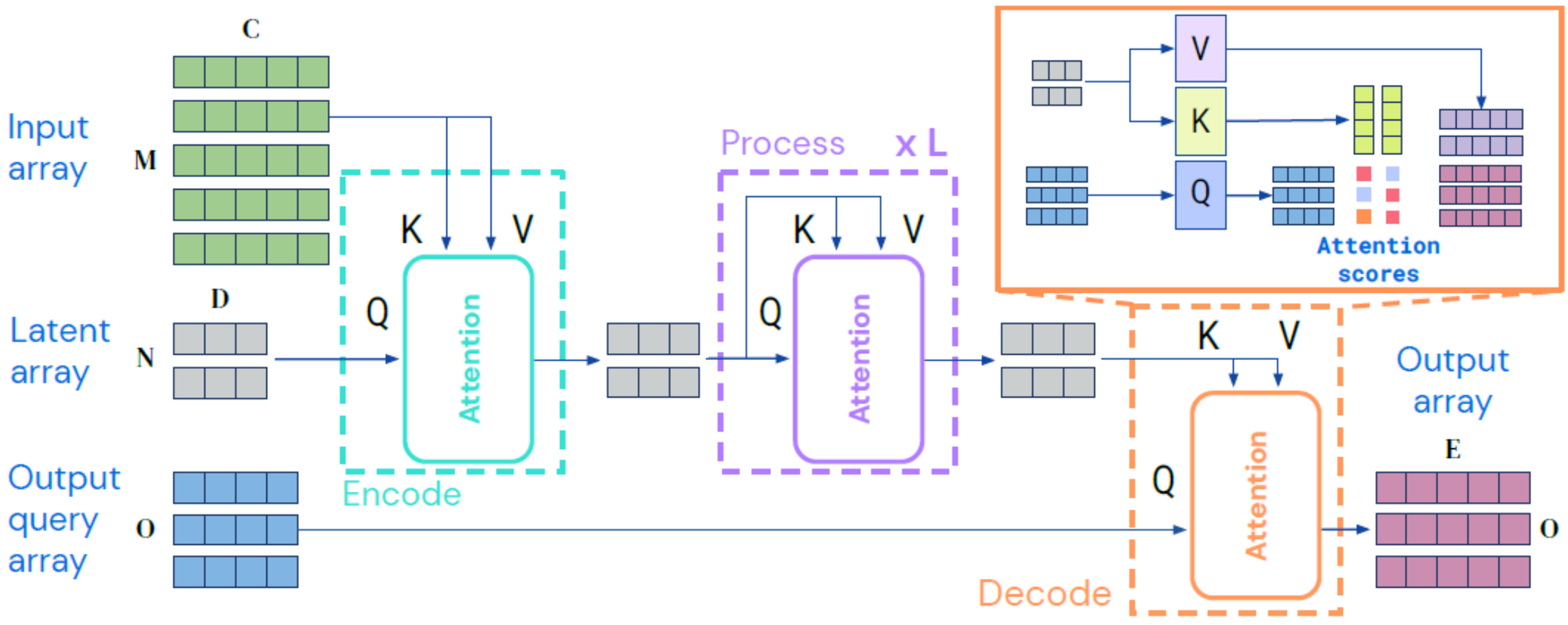


Figure 2: The Perceiver IO architecture. Perceiver IO maps arbitrary input arrays to arbitrary output arrays in a domain agnostic process. The bulk of the computation happens in a latent space whose size is typically smaller than the inputs and outputs, which makes the process computationally tractable even for very large inputs & outputs.

PerceiverIO

https://arxiv.org/abs/2107.14795v3

Summary

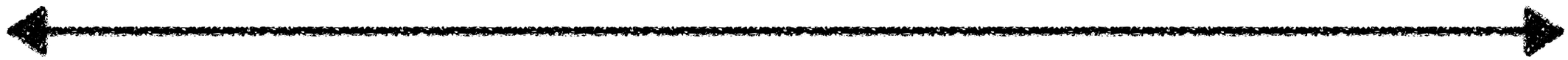
GNN with a fixed neighborhood graph

dynamic graph (e.g. EdgeConv, GarNet)

Point Cloud Transformer

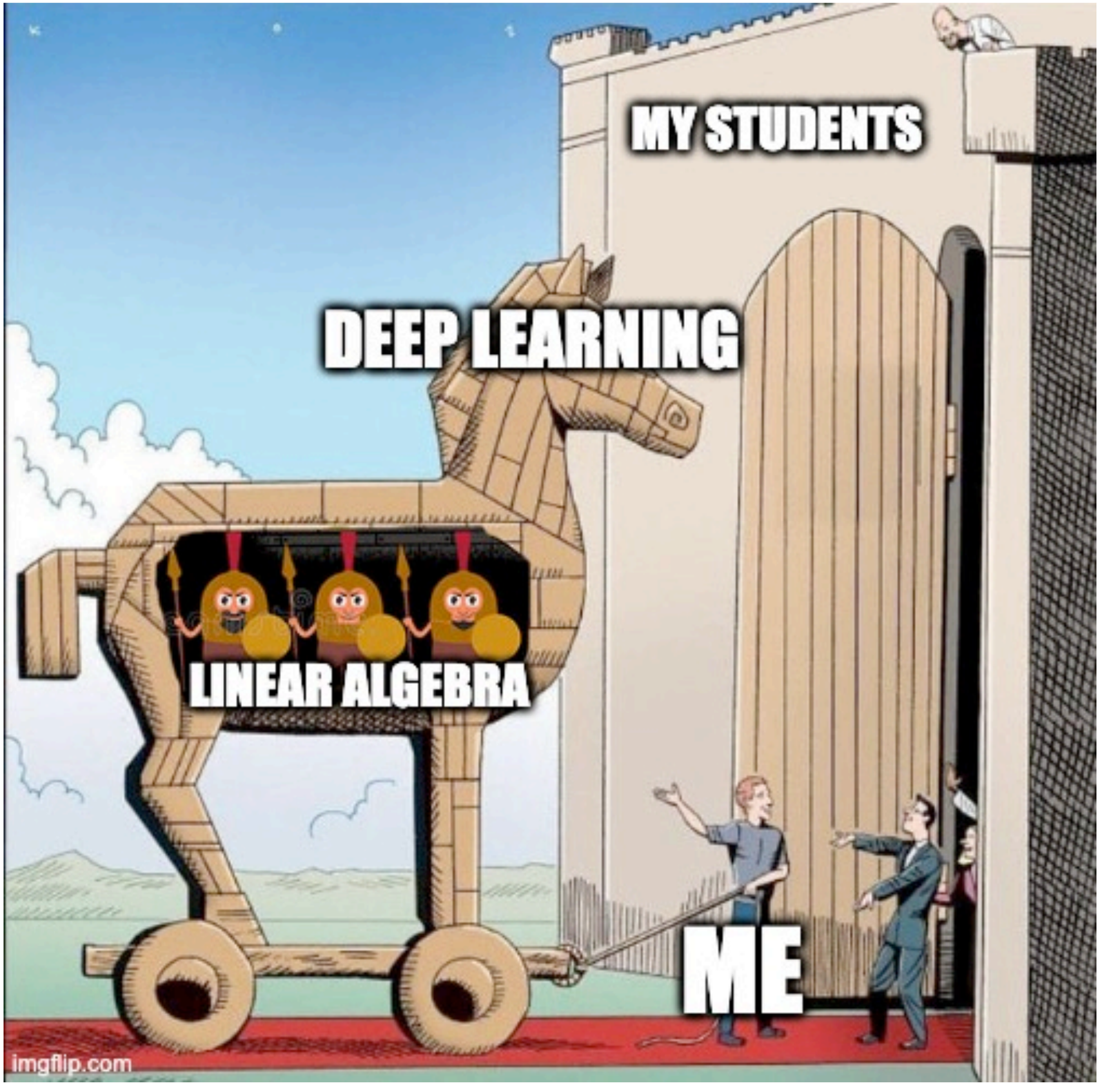
Particle Transformer

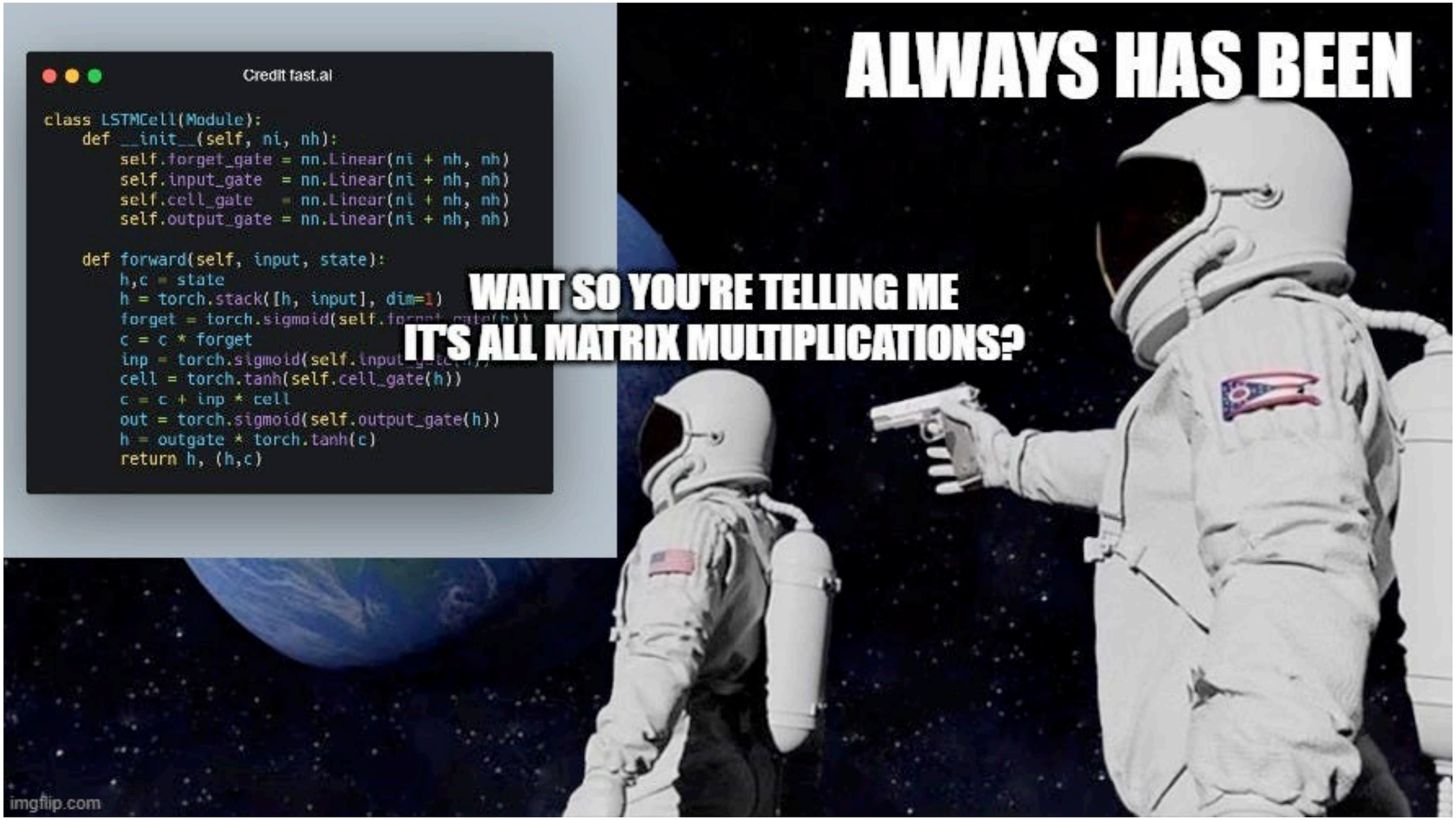
Generic models (e.g. Perceiver-IO)



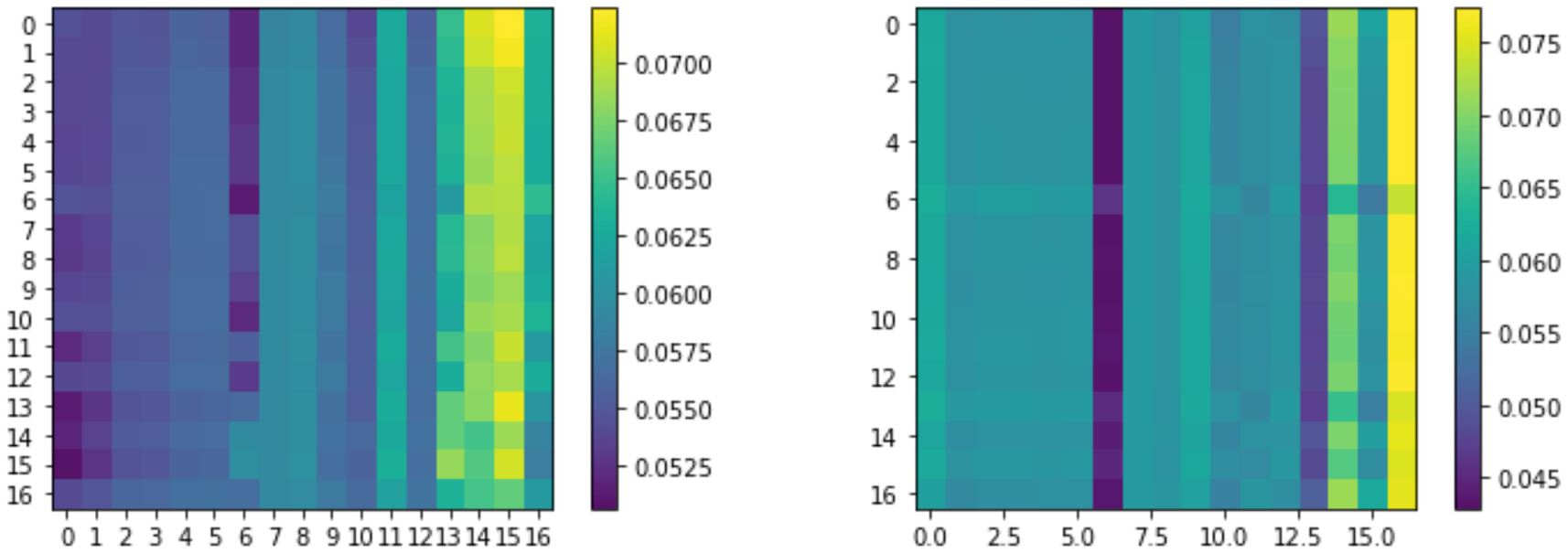
- **more assumed structure**
- **fewer parameters**
- **more scalable on large inputs**

- **less assumed structure**
- **more parameters**
- **less scalable on large inputs**


MY STUDENTS
DEEP LEARNING
LINEAR ALGEBRA
ME
imgflip.com


Credit fast.ai
class LSTMCell(Module):
def __init__(self, ni, nh):
self.forget_gate = nn.Linear(ni + nh, nh)
self.input_gate = nn.Linear(ni + nh, nh)
self.cell_gate = nn.Linear(ni + nh, nh)
self.output_gate = nn.Linear(ni + nh, nh)
def forward(self, input, state):
h,c = state
h = torch.stack([h, input], dim=1)
forget = torch.sigmoid(self.forget_gate(h))
c = c * forget
inp = torch.sigmoid(self.input_gate(h))
cell = torch.tanh(self.cell_gate(h))
c = c + inp * cell
out = torch.sigmoid(self.output_gate(h))
h = outgate * torch.tanh(c)
return h, (h,c)
ALWAYS HAS BEEN
WAIT SO YOU'RE TELLING ME
IT'S ALL MATRIX MULTIPLICATIONS?
imgflip.com

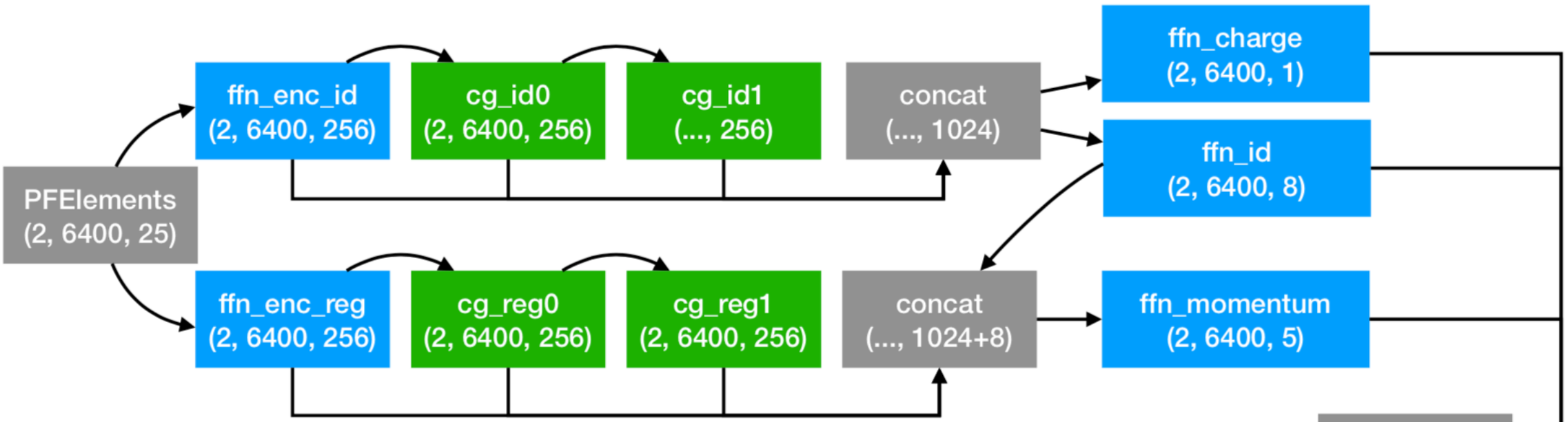
Transformers exercise



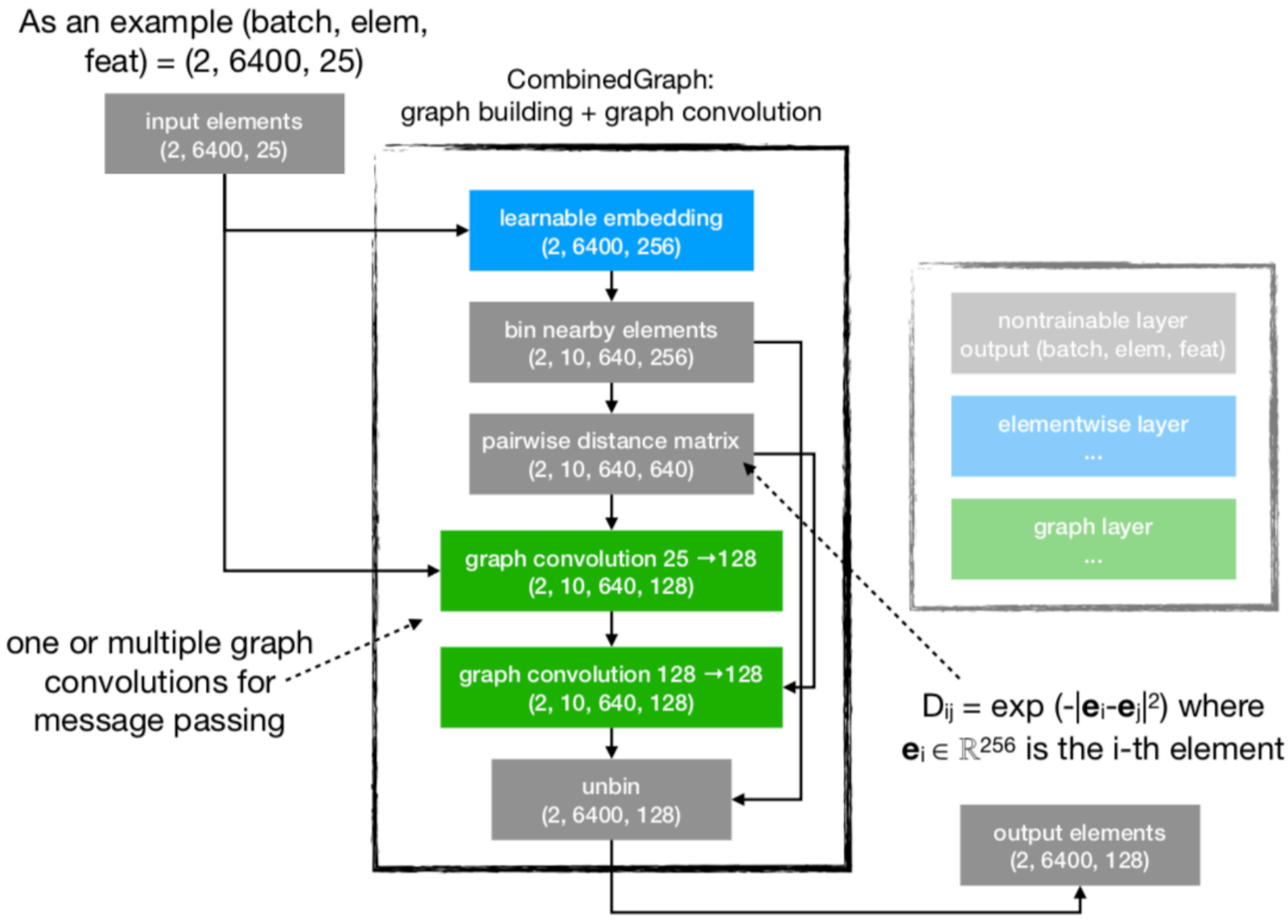
Jupyter notebook: github, colab

Backup

As an example (batches, elements, features) = (2, 6400, 25)

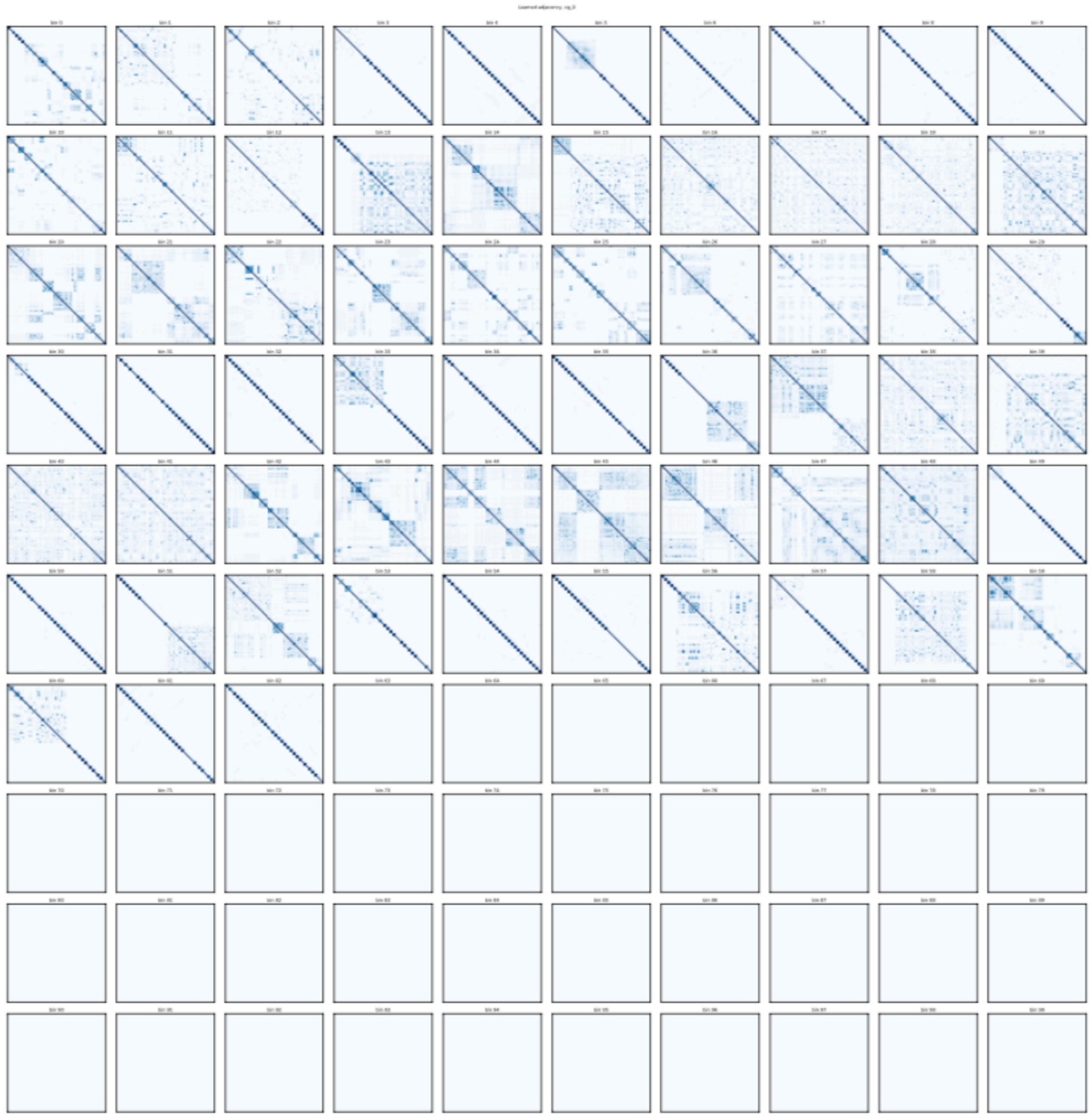


Stacked CombinedGraph (cg) layers, each builds a a new graph in a learnable way and propagates information using graph convolutions.

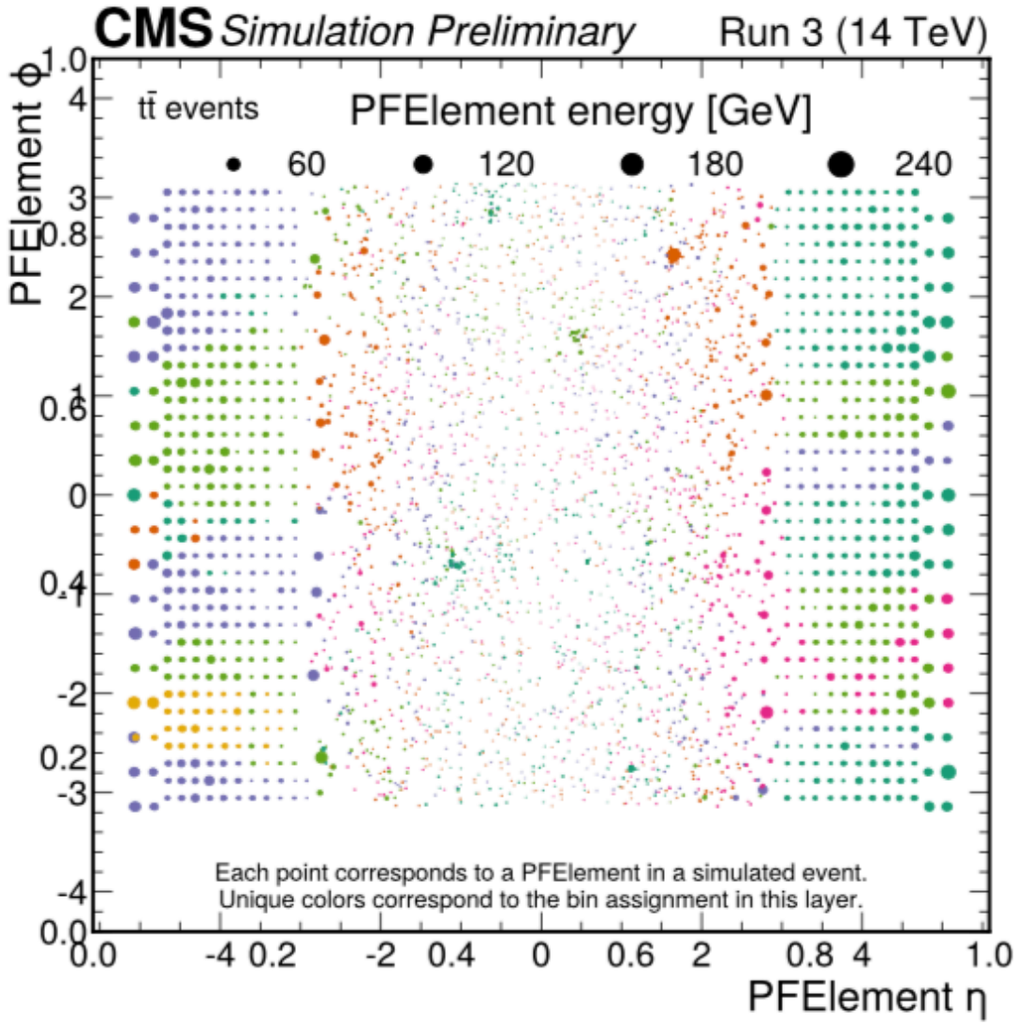


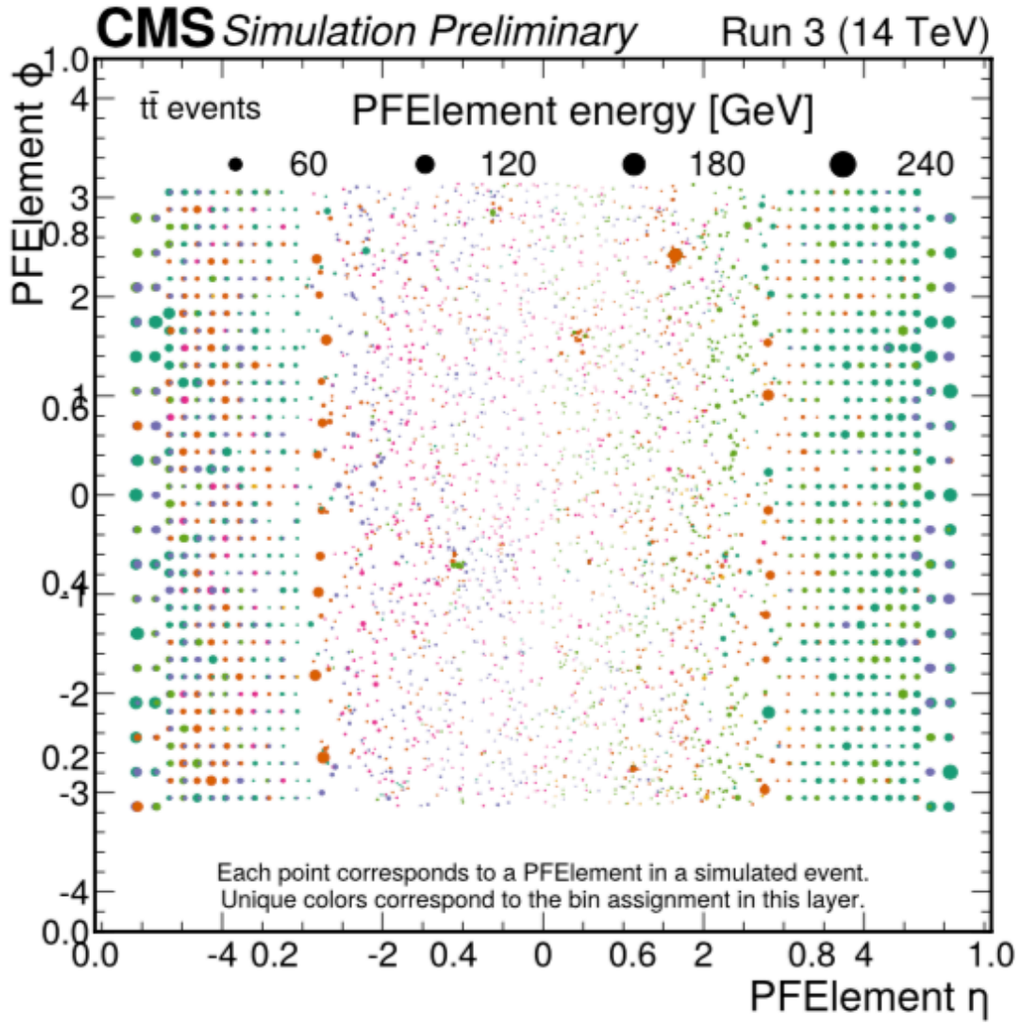
Uses built-in dense matrix, reshape and scatter/gather operations in TF.
Requires batch-mode graphs. No N^2 allocation or computation needed.

One scalable combined graph layer. The input elements are projected into a learnable embedding space. Nearby elements in the embedding space are binned to fixed-size bins. A fully-connected graph is built in each bin, which is used for one or multiple graph convolutions that are used to transform the input elements. Finally, the transformed elements are unbinned.



The learned graph structure in the first layer of the model. Each plot corresponds to a bin where an all-to-all graph adjacency matrix is built between the PFElements in the bin. Emtpy bins are present since for efficiency reasons in GPU training and inference, the model operates on a fixed-size zero-padded input.





The learned binning structure in the first two layers of the model. We show one simulated ttbar event, with each point corresponding to a PFElement in the event. The colors correspond to the assignment of the PFElements into the bins in each layer.